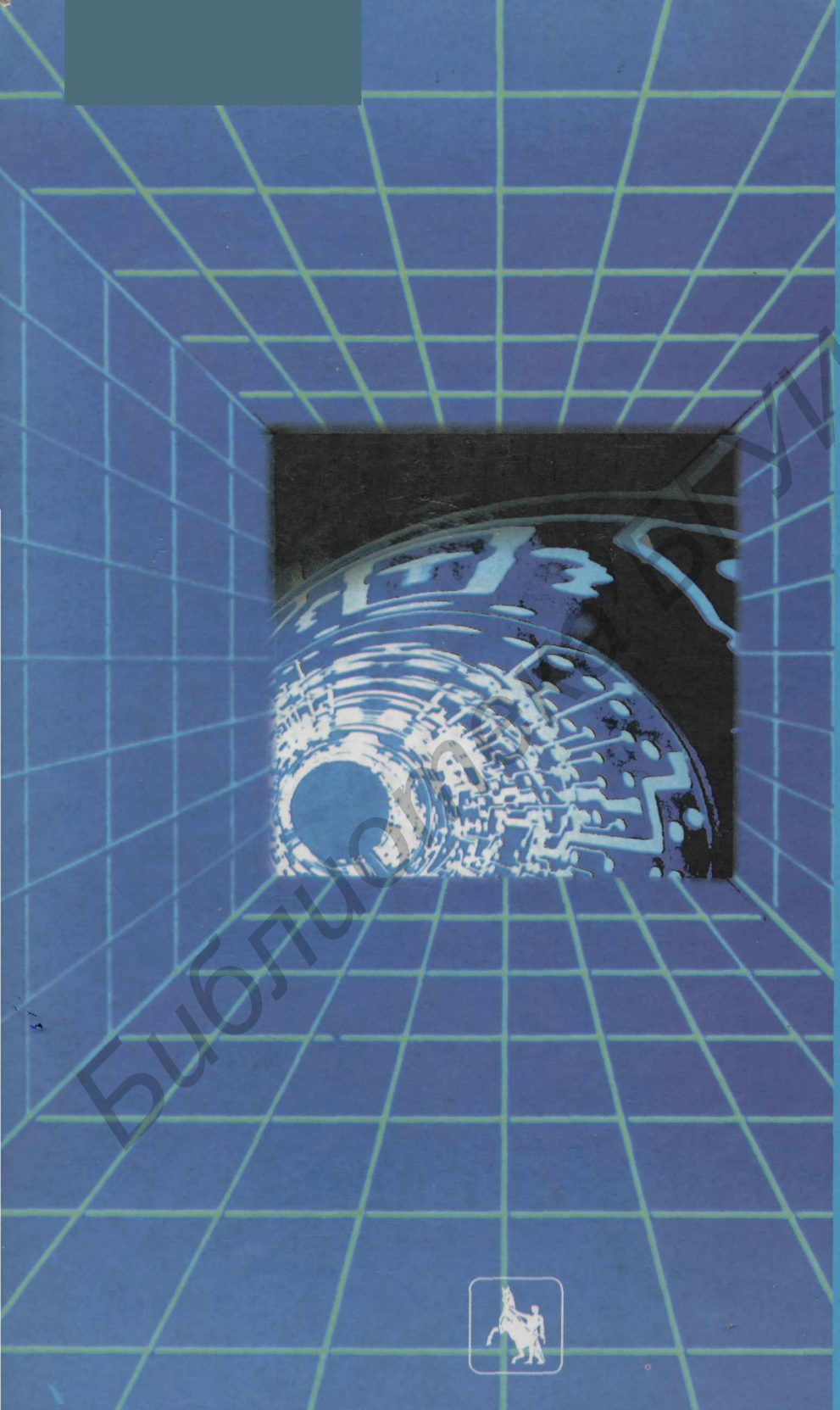
Г. И. Пухальский

ПРОЕКТИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

ВУЗ



УЧЕБНОЕ ПОСОБИЕ ДЛЯ ВУЗОВ

ВУЗ

Г. И. Пухальский

ПРОЕКТИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

Рекомендовано УМО по образованию в области радиотехники, электроники, биомедицинской техники и автоматизации в качестве учебного пособия для межвузовского использования при подготовке специалистов 654200, а также бакалавров и магистров 552500 "Радиотехника"

ПОЛИТЕХНИКА
ИЗДАТЕЛЬСТВО
Санкт-Петербург 2001

УДК 621.396.6
ББК 32.973.2
П90

Федеральная программа книгоиздания России

Пухальский Г. И.

П90 Проектирование микропроцессорных устройств: Учебное пособие для вузов.— СПб.: Политехника, 2001.— 544 с.: ил.

ISBN 5-7325-0557-1

Изложены принципы работы микропроцессоров 8080, 8085, 8086/8088 и арифметического сопроцессора 8087. Подробно описаны программные методы ввода-вывода с квитированием и без квитирования, по прерыванию и по прямому доступу к памяти. Материал проиллюстрирован большим числом примеров и задач.

Приведено описание и применение БИС RAM, EPROM и FIFO различных зарубежных фирм, а также интерфейсных БИС, разработанных фирмой Intel для аппаратной поддержки вышеперечисленных микропроцессоров.

Рассмотрены методы обнаружения и исправления ошибок в оперативных запоминающих устройствах и приведены примеры их практической реализации.

УДК 621.396.6
ББК 32.973.2

ISBN 5-7325-0557-1

9 785732 505573

УЧЕБНОЕ ПОСОБИЕ

Пухальский Геннадий Иванович

ПРОЕКТИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

Оригинал-макет подготовлен автором

Заведующая редакцией *Е. В. Шарова*
Переплет художника *М. Л. Черненко*
Корректор *Н. В. Соловьева*

ЛР № 010292 от 18.08.98.

Сдано в набор 18.05.01. Подписано в печать 01.10.01. Формат издания 70×100 $^{1}/_{16}$. Бумага офсетная. Гарнитура Times New Roman. Печать офсетная. Усл.-печ. л. 43,86. Уч.-изд. л. 44,58. Тираж 3000 экз. Заказ 3690.

Государственное предприятие «Издательство "Политехника"».
191011, Санкт-Петербург, Инженерная ул., 6.

Отпечатано с готовых диапозитивов в ГП «Типография им. П. Ф. Анохина».
185005, г. Петрозаводск, ул. «Правды», 4.

ISBN 5-7325-0557-1

Предисловие

Первая интегральная схема (ИС) была разработана в 1959 г., когда инженерам фирмы *Texas Instruments* удалось расположить на одной подложке кристалла несколько транзисторов и соединить их без проводников. В 1969 г. фирма *Intel*, которой исполнился всего один год, объявила о выпуске оперативного запоминающего устройства (ОЗУ) объемом 1 Кбит, что явилось для того времени большим достижением микроэлектроники в области изготовления ИС.

Идея построения микропроцессора (МП) впервые была высказана японским инженером фирмы "*Бьюсиком*" *Масатоси Сима*. Эта фирма обратилась к фирме *Intel* с заказом на производство набора специализированных ИС для калькуляторов. Для уменьшения их сложности специалист фирмы *Intel Марсиан Хофф* предложил использовать для выполнения арифметических операций 4-разрядный универсальный микропроцессор. Использование для вычислений программного обеспечения вместо "жесткой" логики должно было резко повысить потребность калькулятора в памяти. Проект *Хоффа* получил предпочтение, и фирма *Intel* заключила с фирмой "*Бьюсиком*" контракт на производство универсального МП, получившего имя 4004.

С фирмой *Intel* в начале 1970 г. начал сотрудничать *Федерико Фаджин*, который довел МП от стадии концепции до кремниевого кристалла всего за 9 месяцев, а позже основал фирму *Zilog*. Изготовлен МП 4004 был по *p*-МОП технологии и поступил в продажу в 1971 г.

В 1972 г. фирма *Intel* выпустила на рынок первый 8-разрядный МП 8008. Он требовал 20 ИС поддержки и мог адресовать громадную по тем временам память в 16 Кбайт, что было серьезным шагом вперед по сравнению с МП 4004. В 1973 г. была создана первая универсальная микроЭВМ на основе МП 8008 и вскоре выяснилась недостаточность для многих приложений адресного пространства в 16 Кбайт.

В апреле 1974 г. фирма *Intel* объявила о создании МП 8080 — значительно более совершенного, чем МП 8008. Проект был предложен *Фаджином*, но группу разработки возглавил *Масатоси Сима*, перешедший в фирму *Intel* из фирмы "*Бьюсиком*". Проект нового МП был настолько усовершенствован, что МП 8080 стал действительно полезным вычислительным элементом для широкого круга применений (требовал всего шесть ИС поддержки). При производстве МП 8080 впервые была применена *n*-МОП технология с обогащением (частота двухфазного тактового сигнала не более 2 МГц; три источника питания: +5 В, –5 В и +12 В).

Несколько компаний начали производство микроЭВМ на основе МП 8080. Вначале микроЭВМ были достаточно примитивны, поскольку не было совместимых с ними операционных систем. Преподаватель Высшей школы военно-морских сил *Гарри Килдалл*, создавший в 1976 г. фирму *Digital Research*, разработал в 1975 г. операционную систему *C/PM* (*Control Program for Microcomputers* — управляющая программа для микроЭВМ), ориентированную на МП 8080. Эта операционная система сыграла основную роль в успехе МП 8080 и его архитектуры.

В ответ на успех МП 8080 в фирме *Motorola* инженером *Чаком Педдом* был разработан МП 6800. Фирма *Motorola* впервые представила также серию ИС для периферийных устройств, включавших ИС для параллельного (ИС 6820) и последовательного (ИС 6850) ввода-вывода. Эти ИС обеспечили разработчикам систем включение в компьютеры функций ввода-вывода чрезвычайно простым способом.

В конце 1975 г. *Фаджин* покинул фирму *Intel* и основал новую фирму *Zilog*, которая в 1976 г. объявила о создании 8-разрядного МП *Z*80 (частота однофазного тактового сигнала 2,5 МГц, один источник питания +5 В, встроенная схема регенерации динамической памяти, 176 команд) — значительно улучшенного варианта МП 8080, включавшего весь набор команд

последнего и позволявшего работать с программным обеспечением, написанным для МП 8080. Фирма *Zilog* изготовила также МП, работающие на тактовой частоте 4 МГц, вдвое большей тактовой частоты МП 8080.

В 1976 г. фирма *Intel* объявила о выпуске МП 8085 (частота однофазного тактового сигнала 3 МГц, один источник питания +5 В), но он уступал по производительности и мощности системы команд МП Z80. Система команд МП 8085 отличается от системы команд МП 8080 только двумя новыми командами. Фирма *Intel* для сохранения своих позиций на рынке МП решила выпустить 16-разрядный МП, совместимый "снизу вверх" с МП 8080. Такая совместимость обеспечивала возможность создания простых трансляторов программ, написанных для 8-разрядного МП 8080, в программы для нового 16-разрядного МП 8086, выпущенного на рынок фирмой *Intel* в 1978 г. (*Морс* — один из главных разработчиков МП 8086). Для проектирования мультипроцессорных систем семейство МП 8086 было дополнено сопроцессором 8087 и процессором ввода-вывода 8089, резко повысившими производительность системы в целом по выполнению арифметических операций (8087) и функций ввода-вывода (8089).

Для аппаратной совместимости с МП 8080 в это же время был выпущен МП 8088 с 16-разрядной внутренней и 8-разрядной внешней шинами данных, но имеющий систему команд МП 8086. Средняя производительность МП 8088 всего на 20% ниже производительности МП 8086 благодаря введению в них очереди команд. Первый персональный 16-разрядный компьютер *IBM PC* был выпущен в 1981 г. фирмой *IBM* (*International Business Machines*) на основе МП 8088. На разработку и изготовление этого компьютера потребовался всего один год. Одновременно с этим фирма *Microsoft*, основанная *Биллом Гейтсом*, разработала для него дисковую операционную систему *MS–DOS* (*Microsoft–Disk Operating System*). В качестве внешней памяти в компьютере модели *PC* использовался НГМД (накопитель на гибких магнитных дисках, который изобрел *Йосиро Накамацу*, когда ему было 24 года — все права на изготовление и использование дискеты купила у него корпорация *IBM*). Далее был выпущен персональный компьютер модели *XT*, в котором для хранения файлов использован более скоростной жесткий диск большего объема (винчестер). В следующих моделях (*AT*) персональных компьютеров фирмы *IBM* использованы более совершенные МП фирмы *Intel* 80286, 80386, 80486 и *Pentium* (первый персональный компьютер модели *AT* был изготовлен на МП 80286 в 1984 г.).

При проектировании каждого нового типа МП усилия специалистов фирм направлены как на усовершенствование технологий изготовления ИС (увеличение быстродействия — повышение частоты тактового сигнала, снижение потребляемой мощности), так и на создание функционально более сложных МП. В результате было изготовлено несколько десятков типов МП, производительность которых в несколько раз превосходит производительность первых МП. Кроме того, были разработаны однокристальные микроконтроллеры (микроЭВМ), соединяющие в себе функции МП и некоторых внешних устройств и содержащие на кристалле небольшие по объему ОЗУ и постоянное ЗУ (ПЗУ). Для построения на таких микроконтроллерах простых МП-систем требуется привлекать минимальное число других ИС.

Классификация микропроцессорных ИС:

– однокристальные МП (см. табл. 1); эти МП имеют большой фиксированный набор команд, выполняемых за несколько тактов;

– однокристальные микроконтроллеры (например, 8035, 8048, 8051 и др. фирмы *Intel*); в отношении системы команд принципиальных отличий от однокристальных МП не имеют;

– разрядно-модульные МП; процессор создается из нескольких микропроцессорных секций для получения необходимой разрядности шины данных, и имеется возможность изменять (расширять) систему команд с помощью внешних ИС;

– *RISC*-процессоры (*Reduced Instruction Set Computer* — компьютер с сокращенным набором команд); большинство команд выполняется за один такт;

– транспьютеры — процессоры для параллельной обработки данных; транспьютер спроектирован для работы в мультипроцессорной конфигурации, когда несколько транспьютеров выполняют одну задачу; первый транспьютер *T*414 был разработан фирмой *Inmos* в 1985 г. (построен на *RISC*-процессоре, шина данных и шина адреса 32-разрядные, имеется внутреннее статическое ОЗУ объемом 2 Кбайта и четыре быстрые последовательные линии связи с другими транспьютерами; при частоте синхронизации 5 МГц выполняет 10 млн. команд в секунду).

Некоторые параметры однокристальных МП фирмы *Intel* приведены в нижеследующей таблице.

МП	Отечественный аналог	Число разрядов шины данных	Число разрядов шины адреса	Частота тактового сигнала, МГц	Технология	Год выпуска
4004	–	4	8К×4/1280×4*	0,75	*PMOS*	1971
4040	–	4	12		*PMOS*	1972
8008	–	8	14	0,5	*PMOS*	1972
8080	–	8	16	2	*PMOS*	1973
8080*A*	580ВМ80А	8	16	2,08/2,63/3,125	*NMOS*	1973
8085*A*	–	8	16	3	*NMOS*	1976
8085*AH*	–	8	16	3/5/6	*HMOS*	1976
80*C*85*A*	1821ВМ85А	8	16	3	*CMOS*	1977
8086	1810ВМ86	16	20	5/8/10	*HMOS*	1978
8088	1810ВМ88	16/8	20	5/8	*HMOS*	1979
80186	–	16	20	8/10/12,5	*HMOS*	1983
80188	–	16/8	20	8/10	*HMOS*	1983
80286	–	16	24	6/8	*HMOS*	1983
80386	–	32	32	16/20/25/33	*HMOS*	1985
80486	–	32	32	33/40/60	*HMOS*	1987
Pentium (80586)	–	64	32	100/200	*HMOS*	1993
		8 Кбайт кэш данных, 8 Кбайт кэш команд				

* 8К×4 ПЗУ и 1280×4 ОЗУ. Технологии изготовления МП:
PMOS (*P-Channel Metal Oxide Semiconductor*); ***NMOS*** (*N-Channel Metal Oxide Semiconductor*);
HMOS (*N-Channel high-performance Metal Oxide Semiconductor*);
CMOS (*Complementary Metal Oxide Semiconductor*).

С начала 70-х годов МП, интерфейсные микросхемы и полупроводниковая память произвели революцию в проектировании цифровых систем. Объясняется это программируемостью МП и тем фактом, что однокристальный МП по своим возможностям эквивалентен сотням микросхем с малой и средней степенью интеграции. Микропроцессоры применяют во все большем числе цифровых систем, реализованных ранее на так называемой "жесткой логике", и эта тенденция сохраняется. По мере снижения стоимости МП их используют даже в простых системах типа микроконтроллеров, в которых реализуются не все возможности МП. Причины указанной тенденции станут понятными, если разобраться в основных критериях, которые учитываются при проектировании цифровых систем.

Быстродействие. Основное преимущество системы, построенной на основе "жесткой логики", над МП-системой заключается в том, что она намного быстрее реагирует на входные воздействия. Однако во многих применениях высокого быстродействия не требуется и опреде-

ляющим фактором оказывается гибкость программируемой системы. Кроме того, производительность новых МП увеличивается и расхождение в быстродействии сокращается. Производительность семейства МП фирмы *Intel* за прошедшие годы увеличилась в тысячи раз.

Стоимость. Кроме стоимости собственно микросхем, на стоимость МП-системы влияют следующие факторы: приобретение, хранение и контроль микросхем, оплата монтажа, пайки и внешнего оформления, расходы на приобретение печатных плат, блоков питания, корпусов. Расходы на приобретение микросхем обычно не превышают 10% общей стоимости системы, однако стоимость МП-системы пропорциональна числу микросхем, а не их внутренней сложности. Следовательно, экономичнее применять более дорогие БИС и СБИС, если они заменяют достаточное число микросхем с малой и средней степенью интеграции.

Гибкость. После запуска системы в производство могут потребоваться ее модификации или усовершенствования, что объясняется выявлением по результатам эксплуатации просчетов проектирования или необходимостью введения новых функций. Если система реализована на основе "жесткой логики", ее придется заново проектировать, модифицировать и проверять, на что затрачивается много времени и средств. С другой стороны, благодаря программируемости МП изменения касаются в основном управляющей программы, а не аппаратных средств.

Надежность. Так как интенсивность отказов системы пропорциональна числу микросхем, надежность системы увеличивается по мере роста сложности микросхем и уменьшения их числа. Кроме того, сокращение числа микросхем ведет к уменьшению межсоединений с соответствующим упрощением решения проблем отказов, помех и синхронизации.

Время проектирования. Процесс традиционного проектирования систем на основе "жесткой логики" по своей природе последовательный, т. е. следующий его этап невозможно начинать до завершения предыдущего. Проектирование же МП-системы возможно разделить на разработки аппаратных и программных средств, которые можно вести параллельно.

Эра микропроцессоров началась с тех пор, когда технология позволила реализовать в одной микросхеме все необходимые функции центрального процессора. Совершенствование МП шло параллельно с развитием микроэлектронной технологии, которая позволяла размещать на кристалле все больше и больше логических схем. Хотя по мере усложнения МП растет и его стоимость, она все же остается намного меньше стоимости эквивалентной системы, реализованной на микросхемах с меньшими функциональными возможностями. Кроме уменьшения числа микросхем, необходимых для реализации данной функции, сокращается и общее число контактов, что позволяет уменьшить расходы на монтаж.

Первые МП подходили только для калькуляторов и простых контроллеров, а современные МП имеют встроенные средства для организации мультипроцессорных систем и поддержки мультипрограммирования. Такие МП можно использовать в качестве центральных процессоров сложных компьютеров широкого назначения. Вместе с новыми МП выпускаются разнообразные микросхемы, предназначенные для реализации памяти, интерфейсов и управления шиной.

В данном учебном пособии изложены принципы работы МП, управление памятью и внешними устройствами на примере МП 8080, 8085 и 8086/8087. Приведены примеры проектирования контроллеров на основе этих МП.

Основная цель учебного пособия — максимально сократить время на овладение принципами построения МП-систем как в отношении схемной реализации, так и в отношении программного управления составляющими ее компонентами.

Автор выражает признательность Т. Я. Новосельцевой, любезно предоставившей для этого учебного пособия параграфы 1.10 ÷ 1.12, 2.6 и 3.10. Прилагаемая к учебному пособию дискета с программными пакетами *AVSIM*85 фирмы *Avocet Systems, Inc.* и *Turbo Assembler Version* 3.2 фирмы *Borland International* содержит файлы исходных текстов программ большинства задач, включенных в учебное пособие.

Глава 1

МИКРОПРОЦЕССОРЫ 8080 И 8085

1.1. Трехшинная архитектура микроЭВМ

Любая электронная вычислительная машина (ЭВМ) состоит из пяти основных узлов: арифметического устройства, устройства управления, системной памяти (ОЗУ — оперативного запоминающего устройства и ПЗУ — постоянного запоминающего устройства), устройства ввода и устройства вывода (внешних устройств). Выпускаемые промышленностью однокристальные микропроцессоры (МП) выполняют функции арифметического устройства и устройства управления, что значительно упрощает проектирование микропроцессорных систем (МП-систем): микроконтроллеров и микроЭВМ. Выполняемая микроЭВМ программа должна находиться в системной памяти, а обрабатываемые данные могут находиться как в системной памяти, так и поступать от внешних устройств. Программа состоит из некоторого числа команд, которые МП последовательно читает из памяти и сразу же выполняет. При выполнении команды может потребоваться запись или чтение данных из памяти.

Трехшинная архитектура МП-систем. Принцип построения МП-систем на основе трехшинной архитектуры показан на рис. 1.1 [1]:

DB, *AB*, *CB* (*Data Bus*, *Address Bus*, *Control Bus*) — шины данных, адреса и управления;

CPU (*Central Processing Unit*) — центральное процессорное устройство, в состав которого входят однокристальный МП, шинные драйверы, приемопередатчики, адресные регистры и другие вспомогательные ИС. *CPU* управляет выборкой (чтением) команд из памяти и передачей данных в МП-системе;

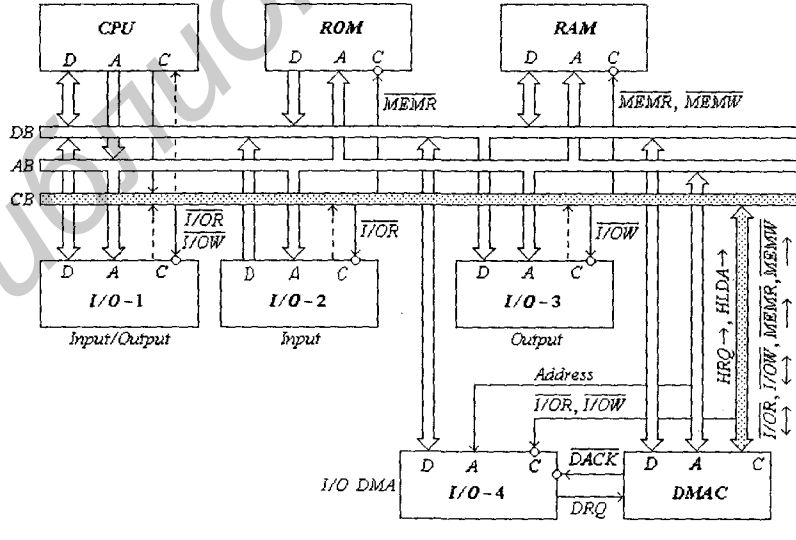


Рис. 1.1. Трехшинная архитектура микроконтроллера

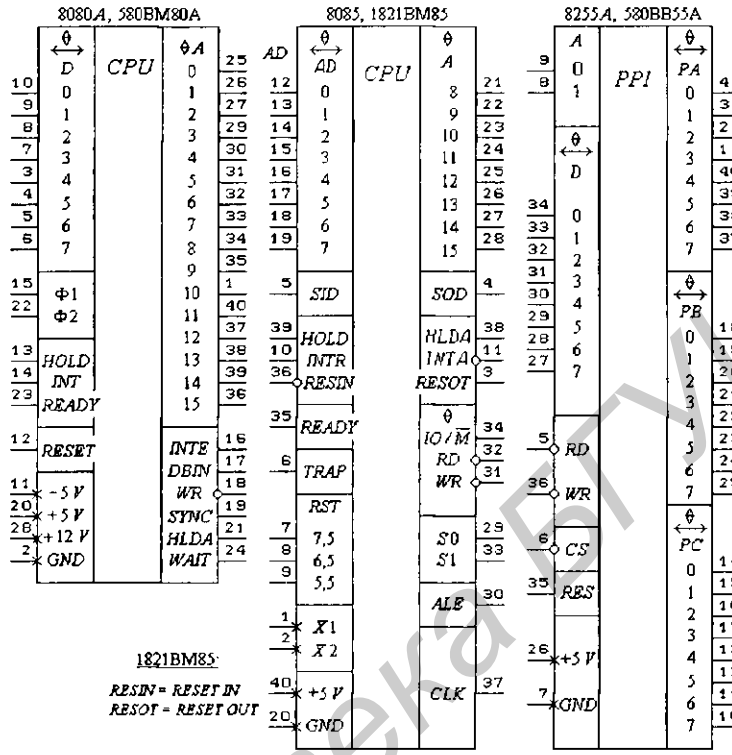


Рис. 1.2. Микропроцессорные БИС

ROM (*Read Only Memory* — память только для чтения) — ПЗУ;

RAM (*Random Access Memory* — память с произвольной выборкой) — ОЗУ;

I/O (*Input/Output*) — устройства ввода-вывода (внешние устройства), которые могут быть и только устройствами ввода (*I/O*–2) или только устройствами вывода (*I/O*–3);

DMAC (*Direct Memory Access Controller*) — контроллер прямого доступа к памяти, представляющий собой специализированный процессор ввода-вывода (*DMAC*, как и МП, может управлять шинами). Если шинами управляет МП, то *DMAC* является ведомым системной шины (представляет для МП обычное устройство ввода-вывода — пассивный режим работы *DMAC*). При переходе *DMAC* в режим ведущего системной шины (активный режим работы *DMAC*) МП предварительно должен перевести свои шины в *Z*-состояние. В активном режиме *DMAC* генерирует адресные сигналы и сигналы управления, обеспечивающие непосредственную передачу данных между системной памятью и внешним устройством *I/O*–4.

На рис. 1.2 приведены условные графические обозначения нескольких микропроцессорных БИС: 8080*A* и 8085*A* фирмы *Intel* (отечественные аналоги 580ВМ80А и 1821ВМ85А) — однокристальные МП с 8-разрядиой шиной данных; 8255*A* фирмы *Intel* (580ВВ55А) — программируемый периферийный интерфейс (*I/O*–1 на рис. 1.1), имеющий три 8-разрядных порта ввода-вывода данных ***PA***, ***PB*** и *PC* [2].

Операнды и их обработка. Объекты, над которыми МП производит операции, называются операндами. Операнды могут находиться как в *ROM*, *RAM*, *I/O*, так и во внутренних регистрах МП, называемых регистрами общего назначения (РОНы). Операнды могут содержать как числовую, так и нечисловую информацию. Наиболее часто МП производит над операндами

операцию пересылки между РОНами (внутри МП) и между МП и *ROM*, *RAM*, *I/O* (внешние пересылки по шине данных *DB*).

Для эффективной работы устройства, изображенного на рис. 1.1, МП должен управлять большим количеством объектов: ячейками памяти *ROM*, ячейками памяти *RAM*, внешними устройствами *I/O* (устройства ввода-вывода), что возможно только при двоичной их адресации. Для уменьшения числа линий передачи данных, как правило, используются двунаправленные приемопередатчики, встроенные в МП (направлением передачи управляет МП).

Операции пересылки не изменяют информацию, содержащуюся в операндах. Их обработка может производиться только в арифметическо-логическом устройстве (АЛУ) микропроцессора. Если операнд представляет собой число, то над ним обычно производятся арифметические действия вычитания или сложения с другим числом. Обработка же операндов, не являющихся числами, обычно заключается в выполнении над ними логических операций И (*AND* — конъюнкция), ИЛИ (*OR* — дизъюнкция), сумма по модулю два (*exclusive OR* — исключающее ИЛИ) и НЕ (*NOT*, *complement Boolean* — отрицание, инверсия, булево дополнение). Операции И, ИЛИ и сумма по модулю два — двухместные, поэтому в них участвуют два операнда. Все логические операции выполняются поразрядно. Например, конъюнкция двух операндов

$$A = a_7a_6a_5a_4a_3a_2a_1a_0 \text{ и } B = b_7b_6b_5b_4b_3b_2b_1b_0,$$

где a_i и b_i — двоичные разряды операндов, выполняется в виде

$$A \,\&\, B = (a_7 \,\&\, b_7)(a_6 \,\&\, b_6) \ldots (a_0 \,\&\, b_0).$$

Системная шина управления *CB*, как правило, формируется с помощью дополнительных аппаратных средств, специально спроектированных для выпускаемых семейств БИС (для серии 580 — системные контроллеры 580ВК28 и 580ВК38, для серии 1810 — контроллер шин 1810ВГ88). Центральное процессорное устройство (*CPU* на рис. 1.1) при использовании серии 580 состоит из ИС 580ВМ80А (МП 8080), 580ГФ24 (генератор тактовых сигналов 8224), 580ВК38 (системный контроллер 8238) и 1533АП5 × 2 (драйверы шины адреса; см. рис. 1.32). В любой МП-системе с раздельной адресацией памяти и *I/O* системная шина управления *CB* содержит линии управления чтением *ROM* и *RAM*, записью данных в *RAM*, вводом данных из *I/O* и выводом данных в *I/O*, обеспечивающие операции пересылки операндов. В микропроцессорной серии 580 для соответствующих системных сигналов управления приняты обозначения:

$\overline{MEMR}$ (*Memory Read*) — чтение команд программы или данных из памяти (*ROM* и *RAM*),

$\overline{MEMW}$ (*Memory Write*) — запись данных в память (*RAM*),

$\overline{I/OR}$ (*I/O Read*) — ввод данных из внешнего устройства *I/O* (чтение данных),

$\overline{I/OW}$ (*I/O Write*) — вывод данных во внешнее устройство *I/O* (запись данных).

Конечно, в МП-системах используются и другие сигналы управления операциями пересылки, например, сигнал *READY* (готовность), выдаваемый памятью или *I/O*, имеющими недостаточное быстродействие (при их готовности к приему или выдаче операндов выдается значение *READY* = 1).

Ввод-вывод по прямому доступу к памяти. При использовании МП пересылки операндов между памятью и *I/O* осуществляются программным способом с помощью команд ввода и вывода (IN *port* и OUT *port*) и команд записи и чтения памяти. Для пересылки операнда требуется последовательное выполнение двух команд, осуществляющих пересылку операнда между *I/O*, МП и памятью по схеме:

$$I/O \rightarrow \text{МП} \rightarrow Memory \text{ или } Memory \rightarrow \text{МП} \rightarrow I/O.$$

Такой способ пересылки требует большого времени, так как используется промежуточное звено (МП). Контроллер прямого доступа к памяти *DMAC* используется для реализации пересылок вида

$$I/O \rightarrow Memory,\ Memory \rightarrow I/O$$

с целью их ускорения. Внешнее устройство *I/O*–4 на рис. 1.1 может пересылать операнды только под управлением *DMAC*, который является специализированным процессором ввода-вывода, осуществляющим непосредственную связь между *I/O* и системной памятью (*RAM* и *ROM*). Для аппаратной поддержки ввода-вывода по прямому доступу к памяти фирмой *Intel* были спроектированы БИС 8257 и 8237*A* (580ВТ57 и 1810ВТ37А) — программируемые контроллеры прямого доступа к памяти.

В пассивном режиме *DMAC* программируется с помощью команд OUT *port* на выполнение операции записи или чтения памяти, т. е. *DMAC* указывается направление передачи данных типа *I/O*–4 → *Memory* или *Memory* → *I/O*–4. При программировании обязательно задается начальный адрес системной памяти A_B (базовый адрес) и размер ΔA пересылаемого блока данных. Взаимодействие *DMAC* с МП осуществляется с помощью сигналов:

DRQ (*DMA Request*) — сигнал запроса *DMA*, поступающий от *I/O*;

HRQ = *HOLD* (*Hold Request*) — сигнал запроса *DMA* (захвата шин), подаваемый от *DMAC* на МП;

HLDA (*Hold Acknowledge*) — сигнал подтверждения захвата шин, выдаваемый МП на *DMAC* с одновременным переводом своих локальных шин данных, адреса и управления в *Z*-состояние;

$\overline{DACK}$ (*DMA Acknowledge*) — сигнал подтверждения *DMA*. Этот сигнал принимает активный уровень (0) при пересылке каждого байта по адресам от предварительно запрограммированного начального адреса A_B до конечного адреса, вычисляемого *DMAC* с помощью указанного при программировании объема пересылаемых данных ΔA.

Для МП 8080*A* и 8085*A* взаимодействие *I/O*, *DMAC* и МП определяется схемой:

$$DRQ = 1 \Rightarrow HRQ = 1 \Rightarrow HLDA = 1 \Rightarrow \overline{DACK} = 0 \ldots \overline{DACK} = 0$$

до окончания заданного объема пересылок. После окончания пересылок *DMAC* устанавливает значение сигнала *HRQ* = 0, в ответ на которое МП выдает значение сигнала *HLDA* = 0, указывающее завершение прямого доступа к памяти. В активном режиме *DMAC* вырабатывает активные уровни пар сигналов управления $\overline{MEMR}$ и $\overline{I/OW}$ (при пересылках данных из памяти в *I/O*) или $\overline{MEMW}$ и $\overline{I/OR}$ (при пересылках данных в память из *I/O*), аналогичные системным сигналам управления *CPU*.

Описанная процедура *DMA*-пересылок инициируется запросом *DMA* сигналом *DRQ* = 1, поступающим от внешнего устройства, являющегося в этом смысле активным. Инициатором пересылок может быть и сам МП. В связи с этим все БИС контроллеров *DMA* выполняются так, что могут воспринимать специальные команды разрешения *DMA*, в ответ на которые *DMAC* выдает сигнал *HRQ* = 1 при *DRQ* ≡ 1.

Единицы измерения памяти. В МП-системах наиболее часто используются следующие единицы измерения памяти:

бит (*bit*) — один двоичный разряд (минимальная единица количества информации в ЭВМ);

полубайт (*nibble*, *half-byte*), тетрада (*tetrad*) — четыре бита (левая или правая половина байта);

байт (*byte*) — 8 бит (наименьшая адресуемая единица данных или памяти ЭВМ),

слово (*word*) — два байта,

1 Кбит = 2^{10} бит = 1024 бит (*kilobit* — килобит),

1 Мбит = 2^{20} бит = $2^{10} \cdot 2^{10}$ бит = 1 048 576 бит (*megabit* — мегабит),

1 Гбит = 2^{30} бит = $2^{10} \cdot 2^{10} \cdot 2^{10}$ бит = 1 073 741 824 бит (*gigabit* — гигабит),

1 Тбит = 2^{40} бит = $2^{10} \cdot 2^{10} \cdot 2^{10} \cdot 2^{10}$ бит = 1 099 511 627 776 бит (*terabit* — терабит).

В микроконтроллерах и компьютерах используются объемы памяти:

64К × 8 бит = 64 Кбайт = $2^6 \cdot 2^{10} \times 8$ бит = $2^{16} \times 8$ бит = 65536 байт (максимальный объем системной памяти для МП 8080А и 8085*А*),

1М × 8 бит = 1 Мбайт = $2^{20} \times 8$ бит (максимальный объем системной памяти для МП 8086),

16М × 8 бит = 16 Мбайт = $2^{24} \times 8$ бит (максимальный объем системной памяти для МП 80286),

128М × 8 бит = 128 Мбайт = $2^{27} \times 8$ бит (объем памяти, требующийся в современных персональных компьютерах для решения сложных задач),

4Г × 8 бит = 4 Гбайт = $2^{32} \times 8$ бит и др.

1.2. Архитектура микропроцессора 8080

Первый в мире однокристальный 8-разрядный МП 8008 был изготовлен фирмой *Intel* в 1972 г., а в 1973 г. эта фирма выпустила 8-разрядный МП 8080. Широкое коммерческое применение нашли версии 8080*А*, 8080*А*-1 и 8080*А*-2 этого МП, различающиеся только максимально допустимой частотой тактовых сигналов (2,083 МГц, 3,125 МГц и 2,632 МГц соответственно; минимальная частота равна 0,5 МГц). Изготовляются эти МП по *NMOS* технологии (*N-Channel Metal Oxide Semiconductor* — *n*-МОП технология). Для питания МП требуется подавать три напряжения: V_{CC} = +5 В (±5%), V_{DD} = +12 В (±5%) и напряжение смещения подложки (*substrate bias*) V_{BB} = –5 В (±5%). Типовые и максимальные значения потребляемых токов составляют:

$$I_{CC\ typ}/I_{CC\ max} = 60/80 \text{ мА},\ I_{DD\ typ}/I_{DD\ max} = 40/70 \text{ мА и } I_{BB\ typ}/I_{BB\ max} = 0{,}01/1 \text{ мА}.$$

Максимальная рассеиваемая мощность равна 1,5 Вт. Выходные сигналы МП характеризуются параметрами:

$$V_{OL\ max} = 0{,}45 \text{ В при } I_{OL} = 1{,}9 \text{ мА и } V_{OH\ min} = 3{,}7 \text{ В при } I_{OH} = -150 \text{ мкА}$$

(уровни сигналов совместимы с ТТЛ ИС). Отечественным аналогом МП 8080*А* является МП 580ВМ80А.

Структурная схема МП 8080*А*. Структурная схема 8-разрядного МП 8080*А* представлена на рис. 1.3. Основными узлами МП являются:

Data Bus Buffer/Latch — буфер шины данных с фиксацией данных, обеспечивающий двунаправленную связь МП с памятью (ОЗУ, ПЗУ) и внешними устройствами *I/O* по 8-разрядной шине данных D_{7-0} (приемопередатчик с *Z*-состоянием выходов). В течение первого такта каждого машинного цикла МП на шину данных выводит слово состояния *SW*;

Instruct. RG (*Instruction Register*) — 8-разрядный регистр памяти команд (инструкций), выбираемых МП из ПЗУ или ОЗУ при выполнении программы;

Instruction Decoder and Machine Cycle Encoding — дешифратор команд (инструкций) и шифратор машинных циклов, преобразующий код машинной команды в последовательность внутренних управляющих сигналов (обычно реализуется на программируемых логических матрицах). Машинный цикл — это некоторое число тактов сигнала синхронизации Φ_2, затрачиваемых на пересылку одного байта данных по внешней шине данных МП D_{7-0}. Команды выполняются за 1 ÷ 5 машинных циклов, а машинные циклы — за 3 ÷ 5 тактов тактового сигнала Φ_2. Самые "быстрые" команды выполняются за 4 такта (один машинный цикл), а самая "медленная" — за 18 тактов (пять машинных циклов);

Timing and Control — устройство синхронизации и управления, обеспечивающее как управление всеми внутренними узлами, так и связь с внешней средой;

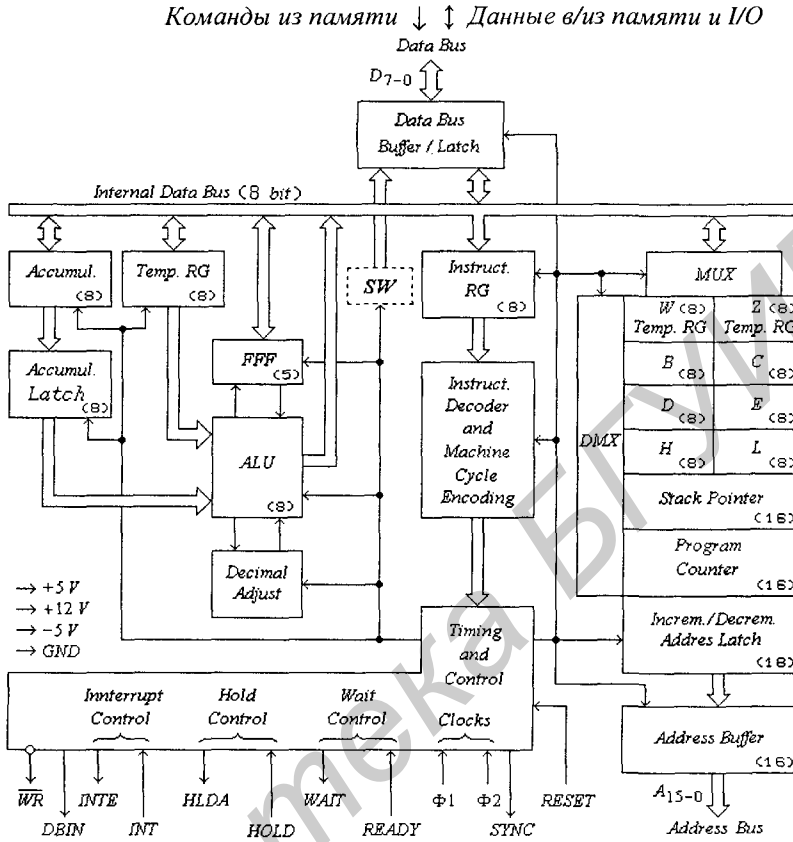


Рис. 1.3. Структурная схема микропроцессора 8080*A*

Internal Data Bus — 8-разрядная внутренняя шина данных, обеспечивающая связь между регистрами памяти МП;

B, *C*, *D*, *E*, *H* и *L* — регистры общего назначения (РОНы), предназначенные для хранения операндов в процессе выполнения программ и имеющие связь с шиной данных D_{7-0} (программно доступны). РОНы могут объединяться в регистровые пары *rp B*, *D* и *H* (регистры *B* и *C*, *D* и *E*, *H* и *L* — *Register Pairs*) для хранения 16-разрядных двоичных чисел (слов) и выполнения операций над ними (*B*, *D* и *H* — старшие байты слов, а *C*, *E* и *L* — младшие байты слов);

W, *Z* и *Temp. RG* (*Temporary Register*) — регистры временного хранения операндов, используемые при внутренних пересылках операндов (программно недоступны);

PC (*Program Counter*) — 16-разрядный программный счетчик, задающий адреса команд программы, выбираемых из памяти;

SP (*Stack Pointer*) — 16-разрядный указатель стека, адресующий специальную область ОЗУ, называемую стеком и используемую для хранения адресов возврата из подпрограмм и операндов при выполнении некоторых команд. Стек представляет собой магазинную память типа *LIFO* (*last-in, first-out* — последним вошел, первым вышел);

Incrementer/Decrementer, Address Latch — блок модификации адреса команды (+1/–1) с его фиксацией в регистре памяти (*Address Latch*), обеспечивающий параллельное выполнение операций вычисления следующего адреса команды программы, выбираемой из памяти, и операций над операндами;

Address Buffer — буфер адреса A_{15-0} с *Z*-состоянием выходов, выдающий адреса памяти и устройств ввода-вывода при выполнении команд программы. Адреса памяти могут поступать как из программного счетчика *PC* и указателя стека *SP*, так и из регистров общего назначения *B* и *C*, *D* и *E*, *H* и *L*;

MUX (*Multiplexer*) — мультиплексор, обеспечивающий под управлением устройства *Timing and Control* коммутацию регистров, выдающих операнды на внутреннюю шину данных (мультиплексор производит выбор источника операнда, указанного в выполняемой команде);

DMX (*Demultiplexer*) — демультиплексор, производящий выбор одного из регистров, указанного в выполняемой команде в качестве устройства назначения (приемника операнда);

ALU (*Arithmetic and Logic Unit*) — арифметическо-логическое устройство, выполняющее арифметические и логические операции над операндами;

Accumulator — буфер аккумулятора, обеспечивающий связь регистра, называемого аккумулятором, с внутренней шиной данных;

Accumulator Latch — аккумулятор, выдающий на *ALU* один из операндов при выполнении арифметических и логических операций и принимающий результат выполненной операции;

Decimal Adjust — схема десятичной коррекции, производящая учет переносов при выполнении команд десятичной арифметики;

FFF (*Flag Flip Flops*) — регистр признаков *F* (регистр флагов *F*), фиксирующий пять результатов выполнения операции в *ALU* (см. рис. 1.14 в § 1.6);

SW (*Status Word*) — устройство формирования слова состояния МП, выдающее на шину данных код операции, которую будет выполнять МП в данном машинном цикле. По этому коду системный контроллер 580ВК28/580ВК38 (см. рис. 1.5) формирует активный уровень (0) одного из системных сигналов управления:

$\overline{MEMR}$ (чтение байта данных из памяти),
$\overline{MEMW}$ (запись байта данных в память),
$\overline{I/OR}$ (чтение байта данных из внешнего устройства),
$\overline{I/OW}$ (запись байта данных во внешнее устройство),
$\overline{INTA}$ (*Interrupt Acknowledge* — подтверждение прерывания).

Сигналы микропроцессора. Для синхронизации работы МП и обеспечения его взаимодействия с внешней средой используются сигналы (8 линий данных, 16 линий адреса и 12 линий управления):

D_{7-0} (*Data Bus*) — двунаправленная шина данных, обеспечивающая связь МП с памятью (*RAM* и *ROM*) и внешними устройствами (*I/O*) при передаче команд программы и данных. Шина данных используется в мультиплексном режиме для последовательной во времени выдачи кода слова состояния *SW* (одновременно с выдачей адресов A_{15-0}) и затем для приема/передачи данных D_{7-0}, т. е. МП имеет совмещенную (мультиплексную) шину управления-данных. Слово состояния *SW* должно фиксироваться во внешнем регистре памяти (из-за мультиплексного режима) для последующей его дешифрации — преобразования в системные сигналы управления памятью и внешними устройствами $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}$ и $\overline{INTA}$. При выполнении команд программы на шину данных может выдаваться содержимое регистров *A* (*Accumulator*), *B*, *C*, *D*, *E*, *H*, *L*, *PC* и регистра признаков *FFF* (*Flag Flip Flops*);

A_{15-0} (*Address Bus*) — шина адреса, обеспечивающая адресацию всех устройств, подключенных к МП. Максимальный объем адресуемой памяти равен 64 Кбайт ($2^{16} \times 8$ бит) или 65536 байт в десятичном представлении. Адресация внешних устройств производится по шине адреса только одним байтом, т. е. в МП-системе можно использовать не более $2^8 = 256$ устройств ввода и 256 устройств вывода. При выполнении команд ввода IN *port* и вывода OUT *port* на шину адреса A_{15-0} МП выдает два одинаковых байта

$$A_7A_6A_5A_4A_3A_2A_1A_0 = A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8 \quad (A_i = A_{i+8},\ i = 0, 1, \ldots, 7),$$

любой из которых можно использовать для адресации внешних устройств. При выполнении команд программы на шину адреса может выдаваться содержимое регистров *PC*, *SP*, *B* и *C*, *D* и *E*, *H* и *L*;

Φ_1, Φ_2 (*Clock Phases*) — сигналы двухфазной синхронизации (тактовые сигналы), подаваемые от генератора 8224 фирмы *Intel* (580ГФ24) и имеющие уровень $V_{OH} \geq 9{,}0$ В;

SYNC (*Synchronizing Signal*) — сигнал синхронизации, указывающий начало каждого машинного цикла (*SYNC* = 1 при выдаче на шину данных слова состояния *SW*);

$\overline{WR}$ (*Write* — запись) и *DBIN* (*Data Bus In* — ввод с шины данных) — сигналы, указывающие внешней среде на выполнение микропроцессором операций записи/вывода ($\overline{WR}$ = 0) или чтения/ввода (*DBIN* = 1). Полное описание этих операций приведено в табл. 1.1. При значении сигнала $\overline{WR}$ = 0 данные $D_{7\text{–}0}$, выданные микропроцессором, имеют истинное значение (данные устойчивы — переходные процессы закончились);

READY (готовность) и *WAIT* (ожидание, или подтверждение состояния ожидания) — сигналы квитирования чтения-записи и ввода-вывода, используемые для синхронизации операций передачи данных между МП и памятью или внешними устройствами, требующими большей длительности активных уровней сигналов управления записью/чтением ($\overline{WR}$ = 0 / *DBIN* = 1), чем их стандартная длительность. Значение сигнала *READY* = 0 должно поступать от памяти или внешних устройств, имеющих недостаточное быстродействие. Значение *READY* = 0 переводит МП в состояние ожидания до тех пор, пока не будет получено значение *READY* = 1 (МП в ответ на значение сигнала *READY* = 0 выдает значение сигнала *WAIT* = 1 — ввод-вывод по готовности подробно описан в § 2.2);

INT (*Interrupt Request* — запрос прерывания) и *INTE* (*Interrupt Enable* — разрешение прерывания) — сигналы квитирования ввода-вывода по прерыванию. Значение сигнала *INTE* указывает состояние внутреннего триггера разрешения прерывания. Если *INTE* = 0, то запросы прерывания *INT* = 1 не воспринимаются. Сброс триггера *INTE* в 0 производится как программным, так и аппаратным способами, а установка в 1 — только программным способом (ввод-вывод по прерыванию подробно описан в § 2.4);

HOLD (запрос захвата шин) и *HLDA* (*Hold Acknowledge* — подтверждение захвата шин) — сигналы квитирования ввода-вывода по прямому доступу к памяти. В ответ на значение сигнала *HOLD* = 1 МП переводит свои шины данных и адреса в *Z*-состояние (МП отключается от шин) и выдает значение сигнала *HLDA* = 1 (ввод-вывод по прямому доступу к памяти кратко описан в § 2.5 и подробно — в § 3.6);

RESET — сигнал системного сброса, подаваемый при включении питания микроЭВМ (*RESET* = 1) и производящий сброс в 0 программного счетчика *PC* (*PC* ← 0 — старт программы производится с адреса 0), триггера *HLDA* и триггера *INTE* (*INTE* ← 0 — прерывания запрещены), которым соответствуют внешние сигналы *HLDA* и *INTE*. Длительность значения сигнала *RESET* = 1 должна быть не менее трех периодов тактового сигнала Φ_2.

Таблица 1.1. **Управление операциями передачи данных**

$\overline{WR}$	*DBIN*	Операция	Комментарий
0	0	Запись данных из МП в *RAM* и вывод (запись данных в *I/O*)	$D_{7\text{–}0}$ МП → *RAM* или *I/O*
0	1	Нет операций	Запрещено (МП никогда не выдает значений $\overline{WR}$ = 0 и *DBIN* = 1)
1	0	Нет операций	Пассивное состояние
1	1	Чтение данных из *RAM*, *ROM* и ввод (чтение данных из *I/O*)	МП ← $D_{7\text{–}0}$ *RAM*, *ROM* или *I/O*

Программный счетчик *PC*. Команды МП 8080*A* состоят из одного, двух или трех байт. Байты команд, составляющих программу, располагаются в памяти последовательно в порядке возрастания адресов. После выполнения очередной команды программный счетчик *PC* указывает, где в памяти расположен первый байт следующей команды выполняемой программы. Устройство управления увеличивает содержимое счетчика *PC* на единицу (инкремент *PC*) всякий раз, когда очередной байт команды передается (читается) из памяти в МП. Если команда состоит из двух или трех байт, то ее выборка происходит за два и три машинных цикла соответственно (машинный цикл — от 3 до 5 тактов сигналов синхронизации Φ_1 и Φ_2, затрачиваемых на пересылку одного байта по внешней шине данных МП). Таким образом, перед началом выборки следующей команды счетчик *PC* уже содержит адрес ее первого байта. Далее при рассмотрении принципа работы МП-системы часто будут использоваться команды на языке ассемблера, полное описание любой из которых приведено в § 1.6 (см. табл. 1.5).

Команды выполняются в той же последовательности, в которой они расположены в памяти. Изменить этот порядок выполнения программист может с помощью команд передачи управления — команд переходов JMP *addr* и команд вызова подпрограмм CALL *addr* и RST *n* ($n = 0 \ldots 7$). Команды JMP *addr* и CALL *addr* во втором и третьем байтах содержат два байта адреса *addr*, который автоматически загружается в программный счетчик *PC* при выполнении этих команд:

$$PC \leftarrow addr = 0000h \ldots \text{FFFF}h,$$

где *h* — указатель 16-ричной системы счисления. При выполнении однобайтовых команд рестарта RST *n* в программный счетчик *PC* автоматически загружается адрес, равный значению $n \times 8$. Например, если выполняется команда RST 5, то в счетчик *PC* загружается адрес

$$5 \times 8 = 40_{10} = 0028_{16} = 0000\ 0000\ 0010\ 1000_2.$$

Принцип работы стека. Для аппаратной поддержки выполнения вложенных подпрограмм идеально подходит магазинная память типа *LIFO* (*last-in, first-out* — последним вошел, первым вышел), называемая стеком (*Stack*). Такая память автоматически обеспечивает последовательное сохранение нескольких адресов возврата из вложенных подпрограмм и последовательное их извлечение в обратном порядке (последний адрес возврата, включенный в стек, извлекается первым).

Для адресации стека в неявном виде используется указатель стека *SP*. Например, при выполнении команды вызова подпрограммы CALL *addr* автоматически последовательно производятся операции:

$SP \leftarrow SP - 1$, $Stack \leftarrow PCh$, $SP \leftarrow SP - 1$, $Stack \leftarrow PCl$ — включение в стек адреса возврата,
$PC \leftarrow addr$ — переход по адресу, по которому расположена первая команда подпрограммы,

где $Stack = M(SP)$ — содержимое ячейки памяти ОЗУ, адресуемой текущим значением указателя стека *SP*; *PCh* — старший байт программного счетчика *PC*; *PCl* — младший байт *PC*; $addr = 0000h \ldots \text{FFFF}h$; адрес возврата — адрес первого байта команды, непосредственно следующей за командой CALL *addr*. В результате выполнения команды CALL *addr* содержимое указателя стека *SP* уменьшится на 2. Если в вызванной подпрограмме имеется другая команда CALL *addr*1 (вложенные подпрограммы), то описанные выше действия будут повторены — в стек будет включен еще один адрес возврата. Число вложений подпрограмм ограничивается только объемом памяти, который можно отвести в МП-системе под стек за счет уменьшения памяти данных.

Каждая подпрограмма должна заканчиваться командой возврата из подпрограммы RET, при выполнении которой автоматически последовательно производятся операции:

$PCl \leftarrow Stack$, $SP \leftarrow SP + 1$, $PCh \leftarrow Stack$, $SP \leftarrow SP + 1$ — извлечение из стека адреса возврата.

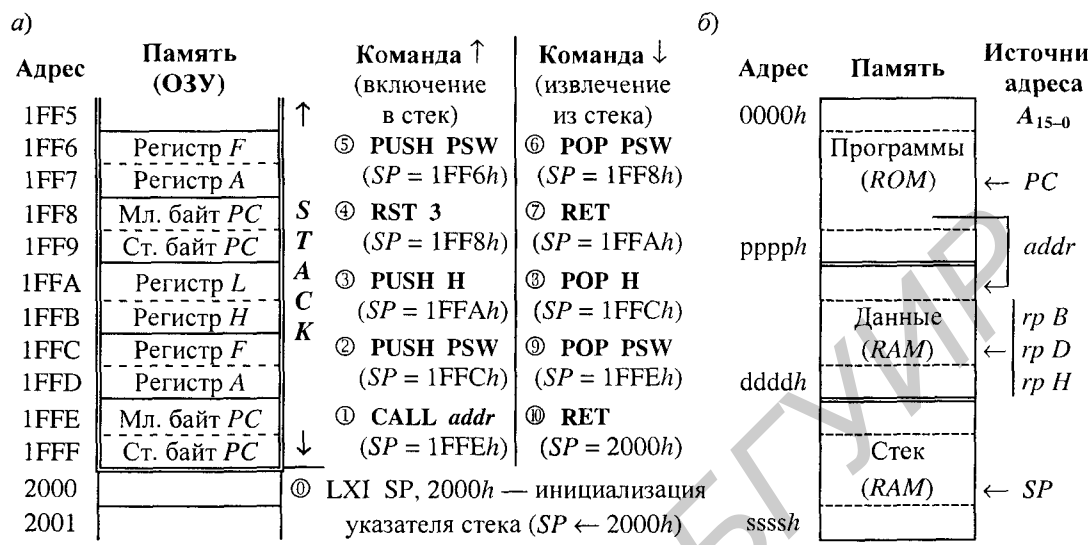


Рис. 1.4. Стек (*а*) и типовое распределение памяти в микроконтроллере (*б*)

В результате выполнения команды RET содержимое указателя стека *SP* увеличится на 2 и выполнение основной программы продолжится с того места, в котором произошло обращение к подпрограмме. Вызов одной и той же подпрограммы может производиться любое число раз из разных мест основной программы. Автоматический декремент при включении байта в стек и инкремент при извлечении байта из стека избавляют программиста от задания адресов ячеек памяти в явном виде.

В МП используются только команды, которые включают и извлекают из стека слова — два байта адреса или данных. Для включения в стек и извлечения из стека слов данных предназначены команды PUSH и POP соответственно. Итак, стек используется только при выполнении команд CALL, RST, RET, PUSH и POP (имеются также команды условного вызова подпрограмм и условного возврата из подпрограмм, аналогичные по выполняемым операциям командам безусловного вызова подпрограмм CALL *addr* и безусловного возврата из подпрограмм RET). При выполнении любой из этих команд МП на шину адреса A_{15-0} выдает содержимое указателя стека *SP*.

Работа стека наглядно представлена на рис. 1.4, *а* — последовательность выполнения команд помечена цифрами ① … ⑩, значения указателя стека *SP* после выполнения очередной команды указаны в скобках. После включения питания МП-системы сразу же следует выполнить инициализацию указателя стека *SP* — командой LXI SP, *d*16 (*d*16 = 0000*h* … FFFF*h*) необходимо записать в указатель стека некоторый адрес, задающий начальную вершину стека (*Top of Stack*).

Указатель стека *SP* в любой момент времени указывает на вершину стека, т. е. на последний элемент, включенный в стек. Содержимое 8-разрядных ячеек памяти стека изменяется только при включении в него новых слов. При извлечении же слов из стека содержимое ячеек памяти не изменяется, как и при чтении данных из ОЗУ.

Программист при разработке программ должен следить за правильным использованием стека — всегда должен соблюдаться баланс между числом команд включения в стек и числом команд извлечения из стека. При выполнении равного числа таких типов команд указатель стека *SP* будет указывать на начальную вершину стека, заданную командой LXI SP, *d*16. В част-

ности, после выполнения всей программы указатель стека *SP* должен указывать на начальную вершину стека.

Ошибки в использовании стека программистом очень трудны для обнаружения. Типичными примерами ошибок являются извлечение из стека данных в неверном порядке, включение в стек или извлечение из него избыточных данных, переполнение стека или потеря данных в стеке. Могут также возникнуть сложности при отладке и документировании программ, оперирующих со стеком — так как элемент данных в стеке не имеет фиксированного в программе адреса, то его содержимое может быть трудно для распознавания.

Распределение памяти в микроконтроллере. Для хранения программ в микроконтроллерах (небольших МП-системах) обычно используется ПЗУ, так как в процессе эксплуатации ни аппаратная часть, ни программное обеспечение микроконтроллеров не изменяется. Кроме того, в ПЗУ могут храниться используемые программами постоянные (неизменяемые) данные, например, числовые константы, таблицы преобразований кодов и др. Для хранения же оперативных (изменяемых) данных должно использоваться ОЗУ, которое необходимо также и для организации стека. Из этого следует, что объем ПЗУ в микроконтроллерах значительно больше объема ОЗУ (например, 60 Кбайт и 4 Кбайта соответственно при использовании всего адресного пространства системной памяти). На рис. 1.4, *б* показан пример типового распределения памяти в микроконтроллере:

0000*h* ... pppp*h* — адреса памяти программ и неизменяемых данных,
(pppp + 1)*h* ... dddd*h* — адреса памяти оперативных данных,
(dddd + 1)*h* ... ssss*h* — адреса стековой памяти,

где pppp*h*, dddd*h* и ssss*h* — некоторые четырехразрядные 16-ричные числа. Конечно, можно использовать и другие распределения памяти, отличающиеся от показанного на рис. 1.4, *б*. В частности, память программ, данных и стека могут быть разделены неиспользуемыми участками адресного пространства. В микроконтроллерах может использоваться не все адресное пространство памяти. В этом случае адрес ssss*h* < FFFF*h*. Для адресации изображенных на рис. 1.4, *б* трех областей памяти используются различные регистры и счетчики:

программный счетчик *PC* для адресации памяти программ;

указатель стека *SP* для адресации стековой памяти;

регистровые пары *rp B*, *D* и *H* для адресации памяти данных; адрес данных *addr* может содержаться и в командах, выбираемых микропроцессором из ПЗУ (при выполнении выбранной из памяти команды, содержащей во втором и третьем байтах операнд *addr*, данный операнд выдается на шину адреса — это так называемая прямая адресация данных).

Адресация данных, находящихся в ПЗУ, ничем не отличается от адресации данных, содержащихся в ОЗУ.

Структурная схема МП-системы. На рис. 1.5 показана структурная схема МП-системы, построенной на основе МП 8080*A*. Центральное процессорное устройство (*CPU*) реализовано на трех кристаллах: БИС 8080*A* — микропроцессор, СИС 8224 (580ГФ24) — генератор тактовых сигналов, СИС 8238 (580ВК38) — системный контроллер (см. § 1.9). СИС 8224 и 8238 изготавливаются по ТТЛШ технологии и обеспечивают высокую нагрузочную способность выходов. Системный контроллер 8238 по принятому коду слова состояния *SW* в определенные моменты времени вырабатывает активный уровень (0) только одного системного сигнала управления $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}$ или $\overline{INTA}$. Системные шины (*System Bus*), сформированные *CPU*, содержат все сигналы, необходимые для подключения к нему памяти (обычно более одной БИС *ROM* и *RAM*) и внешних устройств (обычно более одной БИС *I/O*). У БИС *ROM* вход $\overline{WE}$ (*Write Enable* — разрешение записи) отсутствует.

Часть адресных сигналов из $A_{15\text{–}0}$ подается непосредственно на адресные входы A_i БИС *ROM*, *RAM* и *I/O* ($i = 0 \ldots m$, где $m < 15$ для БИС *ROM*, *RAM* и $m \le 3$ для большинства БИС *I/O*).

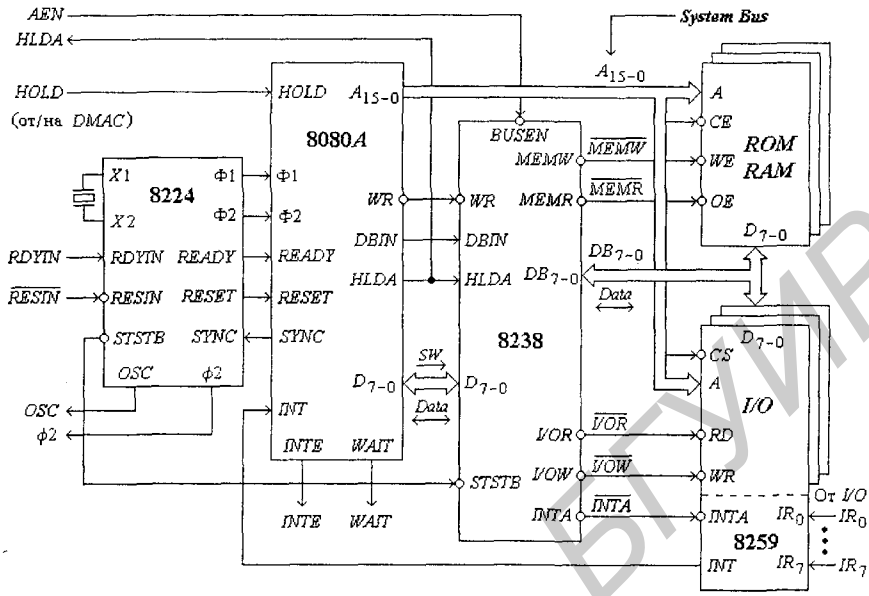


Рис. 1.5. Структурная схема МП-системы на основе МП 8080*A*

Эти адресные сигналы поступают на внутренние дешифраторы БИС для селекции (выбора) одной из 2^{m+1} ячеек памяти в *ROM* и *RAM* или одного из 2^{m+1} регистра памяти во внешнем устройстве *I/O*. Остальные адресные сигналы A_j ($j = m + 1 \ldots 15$ для *ROM*, *RAM* и $j = m + 1 \ldots 7$ для *I/O*) подаются на внешние дешифраторы, выходные сигналы которых используются для селекции (выбора) одной из нескольких БИС *ROM*, *RAM* и *I/O*. Эти сигналы подаются на входы выбора кристалла $\overline{CE}$ и $\overline{CS}$ (*Chip Enable*, *Chip Select* — разрешение кристалла) для включения только одной БИС.

Передача данных между МП и памятью происходит только при совпадении во времени активных уровней (0) адресного сигнала $\overline{CE}$ и сигнала управления $\overline{MEMR}$ или $\overline{MEMW}$. Передача данных между МП и внешними устройствами *I/O* происходит только при совпадении во времени активных уровней (0) адресного сигнала $\overline{CS}$ и сигнала управления $\overline{I/OR}$ или $\overline{I/OW}$. Если активные уровни сигналов $\overline{CE}$ и $\overline{MEMR}$ какой-либо БИС памяти не совпадают во времени, то ее выходы данных $D_{7\text{-}0}$ находятся в *Z*-состоянии. Если активные уровни сигналов $\overline{CS}$ и $\overline{I/OR}$ какой-либо БИС *I/O* не совпадают во времени, то ее выходы данных $D_{7\text{-}0}$ находятся в *Z*-состоянии. При правильно выполненной дешифрации адресных сигналов в каждый момент времени может быть включена (выбрана) только одна БИС.

Если в МП-системе ввод-вывод по прямому доступу к памяти не используется, то следует положить $HOLD \equiv 0$ и $AEN \equiv 0$, т. е. входы *HOLD* и $\overline{BUSEN}$ (*Bus Enable* — разрешение шины) необходимо подключить к земле.

Если в МП-системе использованы достаточно быстродействующие память и внешние устройства, то следует задать $READY \equiv 1$, что обеспечивается подключением входа *RDYIN* (*Ready Input*) к источнику питания $V_{CC} = +5$ В.

Ввод-вывод по прерыванию в стандартных МП-системах обеспечивается контроллером прерываний 8259/8259*A*, что и отражено на рис. 1.5 (описание этого контроллера см. в § 3.5). Работа контроллера прерываний кратко описывается схемой:

$$IR_m \text{ ⎍ (при } INTE = 1) \Rightarrow INT \text{ ⎍} \Rightarrow INTE \text{ ⎴}, \quad \overline{INTA} = \text{⎍⎴⎍⎴⎍⎴} \Rightarrow \text{CALL } addr\_m.$$

Команды CALL *addr_m* поступают в МП из контроллера прерываний 8259 по шине данных $D_{7\text{-}0}$. Чтение трехбайтовых команд CALL *addr_m* из контроллера прерываний безадресное (адресный сигнал выбора кристалла $\overline{CS}$ может иметь произвольное значение), так как чтение команд CALL *addr_m* выполняется сигналом $\overline{INTA}$, который больше нигде не используется. Это единственный случай, когда команды в МП поступают не из памяти, а из внешнего устройства — в данном случае из БИС 8259 (можно и самостоятельно разработать на ИС средней степени интеграции какой-либо простейший контроллер прерываний).

1.3. Машинные циклы

Команды, выбираемые МП из памяти, выполняются под воздействием тактовых сигналов Φ_1 и Φ_2, частота которых определяет производительность микроконтроллера, построенного на основе МП 8080. В МП используется микропрограммное управление для реализации большого числа команд различной сложности. В зависимости от типа команды для ее выполнения требуется вполне определенное число тактов. Микропрограммный автомат, управляющий выполнением машинных команд, обычно реализуется на программируемой логической матрице, занимающей значительную площадь на кристалле МП. Чем больше команд может выполнять МП, тем сложнее система микропрограммного управления, что ограничивает расширение его функциональных возможностей из-за трудности расположения на кристалле дополнительных устройств.

Машинные циклы. Идеальной с точки зрения производительности МП была бы реализация выполнения каждой команды за один такт. Однако, при большом числе различных команд, выполняемых МП, это потребовало бы значительного увеличения площади кристалла, отводимой под схему управления (в настоящее время разработаны МП с сокращенной системой команд — *RISC*-процессоры, большинство команд которых выполняется за один такт). В традиционных МП для упрощения схемы управления при большом числе команд их выполнение разделено на микрооперации, каждая из которых выполняется за один такт. Определенное число тактов группируется в машинный цикл, в течение которого выполняется одно обращение к шине данных для приема или выдачи одного байта во внешнюю среду.

Число обращений к шине данных, необходимое для выборки команды из памяти и ее выполнения, определяет число машинных циклов и время выполнения команды. В МП используются 1-, 2- и 3-байтовые команды. В первом машинном цикле (M_1) всегда производится выборка (чтение) из памяти первого байта команды, в котором содержится код операции (КОП — *Opcode*). Декодировав первый байт команды, МП "узнает", сколько байт она содержит. Выполнив чтение из памяти всех байт команды, МП исполняет ее. Если при исполнении команды требуется передача по системной шине данных одного, двух или четырех (команда XTHL — см. табл. 1.5) байт данных, то число машинных циклов выполнения команды будет больше числа байт в команде на 1, 2 или 4 цикла. Команды в зависимости от их типа выполняются за 1 ÷ 5 машинных циклов, каждый из которых выполняется за 3 ÷ 5 тактов. За один машинный цикл могут выполняться только те однобайтные команды, КОП которых задает операции только над внутренними объектами МП. Простые команды выполняются за 4 такта, самая сложная команда (XTHL) — за 18 тактов [3].

На рис. 1.6 показаны временные диаграммы, поясняющие выполнение двухбайтовой команды OUT *port* (OUT — мнемоника первого байта команды вывода данных из аккумулятора *A* во внешнее устройство, адрес которого задан вторым байтом команды: *port* = 00*h* ÷ FF*h*). Машинный код первого байта команды (КОП) равен D3*h* = 11010011*b* (*h* и *b* — указатели 16-ричной и двоичной систем счисления). Для выполнения данной команды требуется три машинных цикла — два цикла для выборки двух байт команды из памяти и один цикл для ее ис-

полнения — вывода байта данных во внешнее устройство. В начале каждого машинного цикла МП по шине данных выдает слово состояния *SW*, указывающее, какую операцию будет производить МП по шине данных (чтение памяти, запись в память, ввод или вывод данных в устройства ввода-вывода).

Рассмотрим процессы, протекающие в МП при выполнении команды OUT *port*. По адресу, находящемуся в программном счетчике *PC*, МП после окончания выполнения предыдущей команды переходит к выборке из памяти первого байта команды OUT *port*. В первом такте машинного цикла M_1 МП выдает на шину адреса A_{15-0} адрес первого байта команды, находящейся в памяти. Одновременно с адресом МП по шине данных D_{7-0} выдает слово состояния $SW = 10100010$ — код выборки команды, которое значением сигнала $\overline{STSTB} = 0$ записывается в регистр памяти системного контроллера 8238 (см. рис. 1.5). Выдача слова состояния *SW* сопровождается значением сигнала $SYNC = 1$, по которому генератор тактовых сигналов 8224 и вырабатывает значение сигнала $\overline{STSTB} = 0$. Системный контроллер дешифрирует код слова состояния *SW* и выдает активный уровень сигнала чтения памяти $\overline{MEMR} = 0$, устанавливающий на шине данных значение $D_{7-0} = DI$ (*Data Input*) — содержимое ячейки памяти. Сказанное наглядно описывается схемой:

$$SYNC = 1 \Rightarrow \overline{STSTB} = 0 \Rightarrow \overline{MEMR} = 0 \quad \text{(последовательность изменения сигналов)}$$
от МП на 8224 — от 8224 на 8238 — от 8238 на память

$$SW \Rightarrow Reg\ 8238 \Rightarrow \text{МП} \leftarrow \text{КОП} \quad \text{(последовательность передач по шине данных)}$$
из МП в 8238 — фиксация *SW* в *Reg* — из памяти в МП

Код первого байта команды, поступивший по шине данных D_{7-0} в регистр *Instruct. RG* (см. рис. 1.3), подается на дешифратор команды и шифратор машинных циклов (***Instruction Decoder and Machine Cycle Encoding***), который сообщает схеме управления МП, сколько байт содержится в выбираемой команде. Далее МП выполняет машинный цикл M_2 для выборки из памяти второго байта команды OUT *port* (при этом также выдается слово состояния $SW = 10100010$).

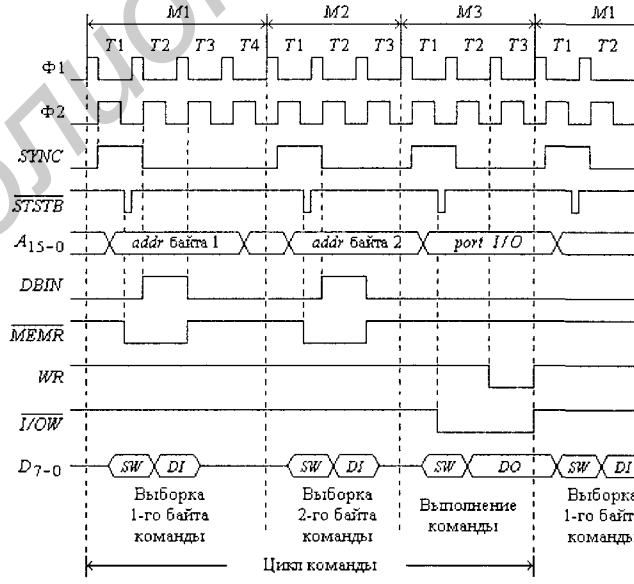


Рис. 1.6. Временны́е диаграммы выполнения команды OUT *port* в МП 8080

Для исполнения команды OUT *port* требуется один дополнительный машинный цикл M_3. В первом такте этого цикла МП выдает на шину адреса A_{15-0} значения

$$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8 = A_7A_6A_5A_4A_3A_2A_1A_0 = port$$

(*port* — число, прочитанное из памяти в машинном цикле M_2), а на шину данных — код слова состояния $SW = 00010000$ (см. табл. 1.15 в § 1.9), по которому системный контроллер 8238 формирует активный уровень сигнала управления $\overline{I/OW} = 0$, поступающий на вход записи $\overline{WR}$ внешнего устройства (см. рис. 1.5) в момент времени, когда МП выводит на шину данных D_{7-0} из аккумулятора A байт данных $DO = A$ (*DO — Data Output*; рис. 1.6). После завершения выполнения текущей команды МП автоматически переходит к выборке первого байта следующей команды. Этот процесс продолжается до выполнения всей программы, заданной оператором. Таким образом, МП представляет собой цифровой детерминированный автомат.

Сигналы квитирования передачи данных *READY*, *INT* и *HOLD*, показанные на рис. 1.3, анализируется микропроцессором в определенных тактах машинных циклов.

Сигнал готовности *READY*. Этот сигнал анализируется в такте T_2 каждого машинного цикла. Если будет обнаружено значение $READY = 0$, то МП между тактами T_2 и T_3 вводит целое число тактов ожидания T_W (см. рис. 2.8 и 2.9). Такты ожидания T_W вводятся до тех пор, пока сигнал *READY* не изменится с 0 на 1. В тактах ожидания T_W сигналы на всех шинах МП сохраняют те же значения, что и в такте T_2 (изменяется только сигнал *WAIT* с 0 на 1). Этим самым достигается увеличение длительности активных уровней системных сигналов управления $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$ и $\overline{I/OW}$.

Сигнал запроса прерывания *INT*. Значение сигнала запроса прерывания $INT = 1$ записывается во внутренний триггер запроса прерывания в последнем такте последнего машинного цикла текущей команды при условии, что установлены значения сигналов $INTE = 1$ (прерывания разрешены) и $HOLD = 0$ (нет запроса прямого доступа к памяти). Микропроцессор, полностью завершив выполнение текущей команды, переходит к выполнению машинного цикла "Подтверждение прерывания" — цикл M_1. В такте T_1 этого цикла МП сбрасывает в 0 внутренний триггер *INTE* (дальнейшие прерывания запрещаются) и на шину данных D_{7-0} выдает код слова состояния $SW = 00100011$, по которому системный контроллер 8238 вырабатывает активный уровень сигнала $\overline{INTA}$ (см. рис. 1.5). Внутренний триггер запроса прерывания сбрасывается в 0 в такте T_2 машинного цикла M_1. Способы аппаратной реализации чтения сигналом $\overline{INTA}$ команд вызова подпрограмм RST *n* и CALL *addr* см. в § 2.4.

При значении сигнала $INTE = 1$ запрос прерывания $INT = 1$ воспринимается микропроцессором также в состоянии его останова, которое наступает после выполнения команды HLT (при этом выдается код слова состояния $SW = 00101011$, по которому также вырабатывается активный уровень сигнала $\overline{INTA}$).

Сигнал запроса прямого доступа к памяти *HOLD*. Этот сигнал анализируется в тактах T_2 и T_W каждого машинного цикла. Если будет обнаружено значение $HOLD = 1$ при условии, что значение сигнала готовности $READY = 1$, то МП прерывает работу после завершения текущего машинного цикла выполняемой команды (шины данных и адреса МП переводятся в Z-состояние), выдав значение сигнала подтверждения захвата шин $HLDA = 1$. При выполнении операций чтения памяти ($\overline{MEMR} = 0$) или ввода ($\overline{I/OR} = 0$) значение $HLDA = 1$ выдается в такте T_3 по положительному фронту тактового сигнала Φ_2. При выполнении же операций записи в память ($\overline{MEMW} = 0$) или вывода ($\overline{I/OW} = 0$) значение $HLDA = 1$ выдается в такте, следующим за тактом T_3, также по положительному фронту тактового сигнала Φ_2.

Запрос прямого доступа к памяти $HOLD = 1$ воспринимается микропроцессором также в состоянии его останова, которое наступает после выполнения команды HLT (при переходе в состояние останова МП выдает значение сигнала $WAIT = 1$ и переводит шины данных и адреса в Z-состояние).

1.4. Микропроцессор 8085

В результате усовершенствования архитектуры микропроцессоров фирма *Intel* в 1976 г. разработала новый 8-разрядный МП — 8085*A*, полностью совместимый с программным обеспечением МП 8080*A*. Изготовляется этот МП по *NMOS* технологии (*N-Channel Metal Oxide Semiconductor* — *n*-МОП технология). В отличие от МП 8080*A* для МП 8085*A* требуется только один источник питания V_{CC} = +5 В (±10%), причем генератор тактового сигнала встроен в МП. Максимальная частота тактового сигнала равна 3 МГц, а минимальная — 0,5 МГц.

Далее были изготовлены МП 8085*AH*, 8085*AH*-1 и 8085*AH*-2 по *HMOS* технологии (*N-Channel high-performance Metal Oxide Semiconductor* — высококачественная *n*-МОП технология), различающиеся только максимально допустимой частотой тактовых сигналов (3, 6 и 5 МГц соответственно). Максимальные значения токов потребления равны:

$$I_{CC\,max} = 135 \text{ мА для 8085}AH\text{, 8085}AH\text{-2 и } I_{CC\,max} = 200 \text{ мА для 8085}AH\text{-1.}$$

Максимальная рассеиваемая мощность составляет 1,5 Вт. Выходные сигналы МП характеризуются параметрами:

$$V_{OL\,max} = 0{,}45 \text{ В при } I_{OL} = 2 \text{ мА и } V_{OH\,min} = 2{,}4 \text{ В при } I_{OH} = -400 \text{ мкА.}$$

Выпускается также МП 80*C*85*A* — *CMOS* вариант МП (*Complementary Metal Oxide Semiconductor* — КМОП), аналогом которого является отечественный МП 1821ВМ85А с максимально допустимой частотой тактового сигнала 3,6 МГц. Электрические параметры МП 1821ВМ85А:

$$I_{CC} = 100 \text{ мА при } V_{CC} = +5{,}5 \text{ В; } V_{OL\,max} = 0{,}4 \text{ В при } I_{OL} = 2 \text{ мА и } V_{CC} = +4{,}5 \text{ В;}$$
$$V_{OH\,min} = 3 \text{ В при } I_{OH} = -1{,}2 \text{ мА и } V_{CC} = +4{,}5 \text{ В.}$$

Структурная схема МП 8085. Система команд МП 8085*A* совпадает, за исключением двух новых команд SIM и RIM, с системой команд МП 8080*A* [4]. Структурная схема 8-разрядного МП 8085*A* изображена на рис. 1.7. Основные отличия МП 8085*A* от МП 8080*A*:

1) генератор тактового сигнала *CLK* (*Clock*) и системный контроллер встроены в МП (включены все возможности, которые обеспечивают СИС 8224 и 8228/8238);

2) добавлены четыре входа запросов векторных прерываний *RST* 5.5, *RST* 6.5, *RST* 7.5 и *TRAP*;

3) добавлены порты последовательного ввода (*SID*) и вывода (*SOD*) данных;

4) использована мультиплексная шина адреса-данных AD_{7-0};

5) предусмотрена возможность построения минимальной МП-системы всего на трех БИС (см. § 3.9) — 8085*AH*/1821ВМ85А (*CPU*), 8155*H*/1821РУ55 (*RAM/IO*) и 8755*A*/1821РФ55 (*EPROM/IO*) — с возможностью расширения до большой системы с использованием стандартных внешних устройств и памяти.

Большинство функциональных узлов, приведенных на рис. 1.7, аналогичны по назначению одноименным узлам МП 8080*A* (см. рис. 1.3). Новые функциональные узлы имеют назначение:

Data/Address Buffer — 8-разрядный двухканальный мультиплексор и 8-разрядный приемопередатчик;

Interrupt Control — схема управления прерываниями;

Serial I/O Control — схема управления последовательными вводом и выводом данных.

Сигналы МП. Сигналы A_{15-8}, *READY, HOLD* и *HLDA*, а также сигналы *INTR*, $\overline{RESET\ IN}$ и *CLK OUT* (*INT*, *RESET* и φ_2 в МП 8080*A*) имеют то же назначение, что и у МП 8080*A*. Сигналы МП 8085*A* выполняют следующие функции:

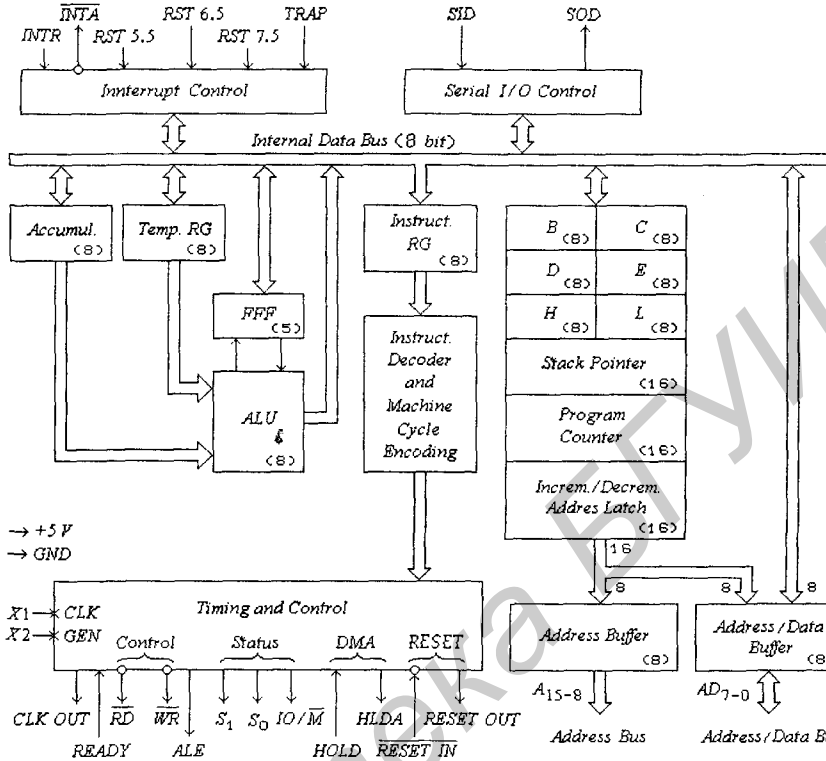


Рис. 1.7. Структурная схема микропроцессора 8085*A*

A_{15-8} (*Address Bus*) — старший байт адреса памяти или 8-разрядный адрес внешнего устройства. Сигналы A_{15-8} устанавливаются в *Z*-состояние в режимах *Hold* (прямой доступ к памяти) и *Halt* (останов МП), а также во время сброса ($\overline{RESET\ IN} = 0$);

AD_{7-0} (*Multiplexed Address/Data Bus*) — мультиплексная шина адреса-данных. В первом такте машинного цикла по шине AD_{7-0} выдается младший байт адреса памяти A_{7-0} или 8-разрядный адрес внешнего устройства A_{7-0}. В течение второго и третьего тактов машинного цикла шина AD_{7-0} становится шиной данных для приема или передачи байта данных D_{7-0};

ALE (*Address Latch Enable*) — сигнал фиксации младшего байта адреса A_{7-0} во внешнем регистре (*ALE* = ⎍ в первом такте каждого машинного цикла). Истинное значение адреса гарантировано по отрицательному (заднему) фронту *ALE*. Сигнал *ALE* можно использовать также для фиксации сигналов состояния S_1 и S_0 во внешнем регистре. Сигнал *ALE* никогда не переводится в *Z*-состояние;

S_1, S_0 и $IO/\overline{M}$ (*Machine Cycle Status*) — сигналы состояния машинного цикла, указывающие тип операции, которую будет выполнять МП в текущем машинном цикле (табл. 1.2). Сигналы S_1 и S_0 становятся истинными в начале машинного цикла и остаются стабильными в течение всего цикла. Детальная информация о машинном цикле, содержащаяся в этих сигналах, может быть использована для управления МП-системами, но обычно сигналы S_1 и S_0 игнорируются — для управления МП-системами более пригодны сигналы $\overline{RD}$ и $\overline{WR}$ из-за оптимизации временно́го положения их активных уровней (см. рис. 1.9);

$IO/\overline{M}$ (*IO/Memory*) — сигнал, указывающий на обращение МП к внешнему устройству ($IO/\overline{M} = 1$) или к памяти ($IO/\overline{M} = 0$);

Таблица 1.2. **Типы машинных циклов**

Тип машинного цикла	*Status*			*Control*			Сигнал МП 8080*A*
	$IO/\overline{M}$	S_1	S_0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$	
Выборка кода операции (*Opcode fetch*)	0	1	1	0	1	1	$\overline{MEMR} = 0$
Чтение памяти (*Memory read*)	0	1	0	0	1	1	$\overline{MEMR} = 0$
Запись в память (*Memory write*)	0	0	1	1	0	1	$\overline{MEMW} = 0$
Чтение *I/O* (*I/O read* — ввод)	1	1	0	0	1	1	$\overline{I/OR} = 0$
Запись в *I/O* (*I/O write* — вывод)	1	0	1	1	0	1	$\overline{I/OW} = 0$
Подтверждение прерывания (*Acknowledge of INTR*)	1	1	1	1	1	0	$\overline{INTA} = 0$
Холостые циклы шины (*Bus Idle*):							
DAD	0	1	0	1	1	1	—
Acknowledge of RST 5.5/6.5/7.5 *and TRAP*	1	1	1	1	1	1	—
HLT	*Z*	0	0	*Z*	*Z*	1	—
Сигнал *RESET* = 1	*Z*	×	×	*Z*	*Z*	1	—
Сигнал *HOLD* = 1	*Z*	×	×	*Z*	*Z*	1	—

Примечание: *Z* — *Z*-состояние; × — безразличное значение; команда *DAD* выполняется за три машинных цикла, второй и третий из которых холостые — нет обращения к шине данных, поэтому для исключения противоречия с ранее введенным определением машинного цикла можно считать, что команда *DAD* выполняется за один машинный цикл.

$\overline{RD}$ (*Read*) — сигнал чтения *I/O* или памяти (табл. 1.2). Значение $\overline{RD} = 0$ указывает, что шина данных доступна для передачи данных;

$\overline{WR}$ (*Write*) — сигнал записи данных в память или *I/O*. Истинные значения данных гарантированы на заднем фронте сигнала $\overline{WR}$ = ⊔ (табл. 1.2);

INTR (*Interrupt Request*) — сигнал запроса прерывания высоким уровнем (*INTR* = 1), поступающий от внешнего устройства (обычно от контроллера прерываний 580ВН59). В ответ на значение *INTR* = 1 микропроцессор выдает значение сигнала $\overline{INTA} = 0$ для ввода из внешнего устройства в МП команды RST или CALL вызова подпрограммы обработки прерывания. Сигнал *INTR* полностью соответствует сигналу *INT* в МП 8080*A*;

$\overline{INTA}$ (*Interrupt Acknowledge*) — выходной сигнал подтверждения прерывания, подаваемый на внешнее устройство или контроллер прерываний 580ВН59 в ответ на запрос *INTR* = 1;

RST 5.5, *RST* 6.5, *RST* 7.5 (*Restart Interrupts*) — входы маскируемых запросов прерывания для вызова подпрограмм, расположенных по фиксированным адресам 5.5×8 = 2C*h*, 6.5×8 = 34*h* и 7.5×8 = 3C*h* (команды RST *n*.5, где *n* = 5, 6 и 7, автоматически вставляются внутри МП). Управление индивидуальным маскированием этих запросов прерывания производится двумя командами: RIM (*Read Interrupt Mask* — чтение маски прерываний; см. рис. 1.16) и SIM (*Set Interrupt Mask* — установка маски прерываний; см. рис. 1.15). Чувствительность входов: сигналы *RST* 5.5 и *RST* 6.5 запрашивают прерывание высоким уровнем, а сигнал *RST* 7.5 — положительным фронтом. Запрос прерывания *RST* 7.5 запоминается во внутреннем триггере, даже когда запросы прерываний замаскированы (вызов подпрограммы обработки прерываний запрещен). Значение сигнала $\overline{RESET\ IN} = 0$ маскирует все входы *RST* *n*.5;

TRAP (ловушка) — вход запроса немаскированного прерывания положительным фронтом с последующим удержанием высокого уровня до завершения выполнения текущей команды. В ответ на значение *TRAP* = 1 внутри МП автоматически вставляется команда RST 4.5 — ад-

рес передачи управления равен $4.5 \times 8d = 24h$. Запрос прерывания по входу *TRAP* не может быть запрещен ни программным, ни аппаратным способом. Высокий уровень сигналов запроса прерываний *TRAP*, *RST* 5.5, *RST* 6.5 и *INTR* должны сохраняться до их выборки в последнем такте последнего машинного цикла текущей команды. Если прерывания разрешены, то МП после завершения текущей команды выполняет соответствующую команду RST и автоматически запрещает все маскируемые прерывания. Запрос прерывания *TRAP* имеет наивысший приоритет. Приоритеты остальных входов запросов прерывания уменьшаются в последовательности *RST* 7.5, *RST* 6.5, *RST* 5.5 и *INTR*. Эта система приоритетов не принимает во внимание приоритет подпрограммы, выполнение которой было начато запросом прерывания с более высоким приоритетом. Так, запрос прерывания по входу *RST* 5.5 может прервать выполнение подпрограммы RST 7.5, если командой EI разрешить прерывания до завершения подпрограммы RST 7.5;

$\overline{RESET\ IN}$ — входной сигнал системного сброса, автоматически подаваемый при включении питания МП-системы ($\overline{RESET\ IN} = 0$) и производящий сброс в 0 программного счетчика ($PC = 0000h$ — старт программы производится с адреса 0), триггера *HLDA* и триггера разрешения прерывания (прерывания запрещаются). Для автоматического сброса вход $\overline{RESET\ IN}$ подключается к *RC*-цепи (см. рис. 1.10). В МП сигнал $\overline{RESET\ IN}$ подается на триггер Шмитта, что обеспечивает задержку отмены сброса относительно момента включения питания. При включении питания длительность значения $\overline{RESET\ IN} = 0$ должна быть не мене 10 мс после того, как напряжение питания достигнет номинального значения. В установившемся режиме работы МП эта длительность должна быть не менее трех периодов тактового сигнала *CLK*. Шины данных, адреса и управления во время сброса находятся в Z-состоянии;

RESET OUT — выходной сигнал системного сброса (реакция на входной сигнал $\overline{RESET\ IN}$), привязанный к тактовому сигналу МП. Длительность активного уровня (1) этого сигнала равна целому числу периодов тактового сигнала *CLK*;

*X*1, *X*2 — входы для подключения кварцевого резонатора, параллельного *LC*-контура или *RC*-цепи, задающих частоту сигнала внутреннего генератора (рис. 1.8, *а*). Частота внутреннего тактового сигнала МП вдвое меньше частоты сигнала на входе *X*1. При допуске частоты ±10% можно использовать параллельный резонансный *LC*-контур:

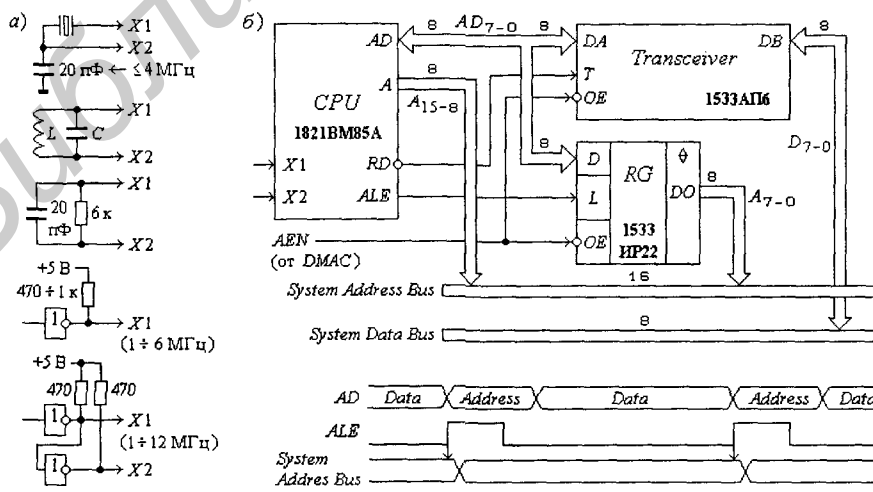


Рис. 1.8. Схема формирования шин адреса и данных

$$f = \frac{1}{2\pi\sqrt{L(C + C_{INT})}},$$

где C_{INT} — паразитная емкость входов $X1$ и $X2$ ($C_{INT} \approx 15$ пФ); $C \geq 2C_{INT}$. Использование LC-контура не рекомендуется для частот выше 5 МГц. Сигнал от внешнего генератора подается на вход $X1$ или на входы $X1$ и $X2$ через логические элементы;

CLK (*Clock*) — выходной тактовый сигнал МП. Частота сигнала *CLK* в два раза меньше частоты сигнала на входе $X1$;

SID (*Serial Input Data Line*), *SOD* (*Serial Output Data Line*) — вход и выход канала последовательной передачи данных для простого последовательного интерфейса. Ввод одного разряда данных по линии *SID* в разряд D_7 аккумулятора производится командой RIM (см. рис. 1.16), а вывод разряда D_7 аккумулятора на линию *SOD* — командой SIM (см. рис. 1.15). Выход *SOD* сбрасывается в 0 сигналом $\overline{RESET\ IN}$.

Сигналы *READY*, *HOLD* и *HLDA* имеют то же самое назначение, что и в МП 8080*A* (см. § 1.2). Сигналы A_{15-8}, AD_{7-0}, $IO/\overline{M}$, $\overline{RD}$ и $\overline{WR}$ находятся в *Z*-состоянии во время выполнения прямого доступа к памяти (*HOLD* = 1), в течение системного сброса ($\overline{RESET\ IN} = 0$) и при останове МП ($S_1 = S_0 = 0$). Все прерывания, кроме *TRAP*, запрещаются командой DI и разрешаются командой EI. Частота внутреннего тактового сигнала и внешнего сигнала *CLK* в два раза меньше частоты кварцевого резонатора.

Особенности работы МП. Разделение шины адреса-данных AD_{7-0} на шину младшего байта адреса A_{7-0} и шину данных D_{7-0} показано на рис. 1.8, *б*. Выдача в первом такте каждого машинного цикла адресных сигналов A_{7-0} сопровождается активным уровнем сигнала *ALE* = 1, который загружает и фиксирует младший байт адреса в 8-разрядном асинхронном потенциальном регистре 1533ИР22. Управление приемопередатчиком (*Transceiver*) производится сигналами: *AEN* — управление *Z*-состоянием шин адреса и данных (сигнал *AEN* поступает от контроллера прямого доступа к памяти) и $\overline{RD}$ — управление направлением передачи данных (вместо сигнала $\overline{RD}$ можно использовать и сигнал $\overline{WR}$).

На рис. 1.9 показаны временны́е диаграммы, поясняющие работу МП при выполнении команды OUT *port* (*PCh* и *PCl* — старший и младший байты адреса; *DI* и *DO* — вводимые в МП и выводимые из МП данные). Эта команда выполняется за три машинных цикла: M_1 — чтение первого байта команды (кода операции), M_2 — чтение второго байта команды (значения *port* — адреса внешнего устройства), M_3 — выполнение команды (вывод байта из аккумулятора во внешнее устройство по адресу *port*, введенному в МП во втором машинном цикле).

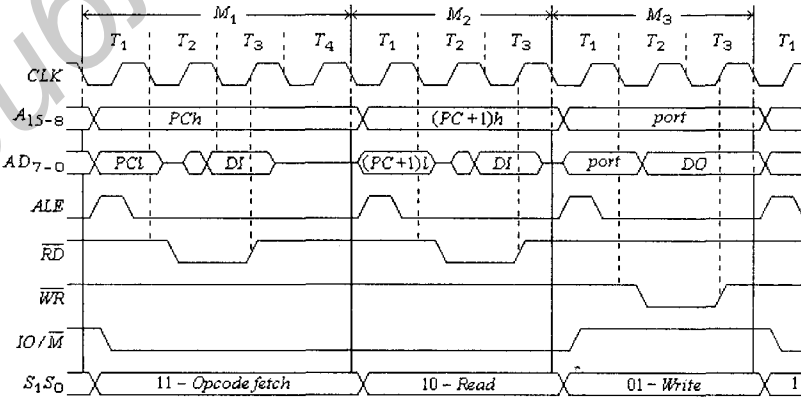


Рис. 1.9. Временны́е диаграммы выполнения команды вывода

Обычно машинный цикл состоит из трех тактов T_1, T_2 и T_3, за исключением машинного цикла выборки кода операции, состоящего из четырех тактов (выборка КОП из памяти) или шести тактов (выборка КОП из *I/O* при подтверждении прерывания).

Из табл. 1.2 для значений сигналов состояния и управления $IO/\overline{M}$, $\overline{RD}$, $\overline{WR}$ (аргументы) и системных сигналов управления $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}$ (функции) следует, что системные сигналы управления памятью и внешними устройствами описываются функциями:

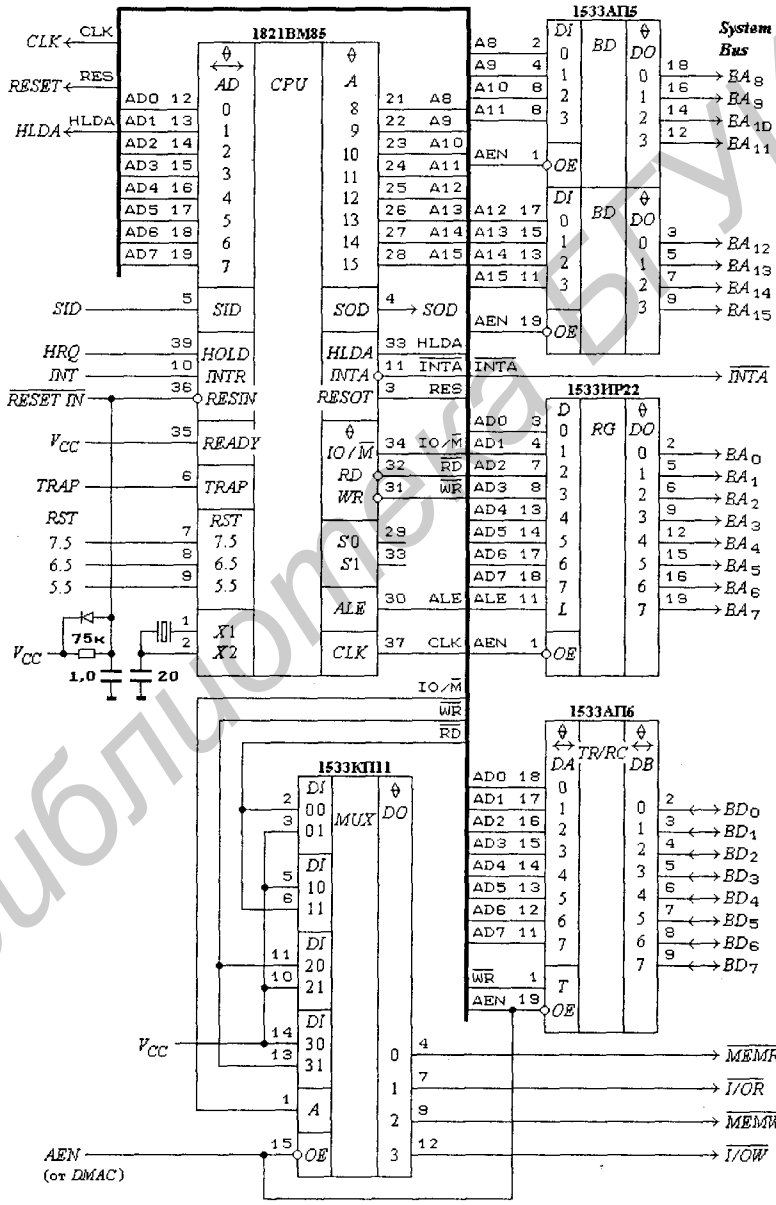


Рис. 1.10. Принципиальная схема центрального процессорного устройства

$$\overline{MEMR} = \overline{RD \cdot IO / \overline{M}}, \quad \overline{MEMW} = \overline{WR \cdot IO / \overline{M}}, \quad \overline{I/OR} = \overline{RD \cdot IO / \overline{M}}, \quad \overline{I/OW} = \overline{WR \cdot IO / \overline{M}}$$

(эти сигналы аналогичны соответствующим сигналам МП 8080*A*). На рис. 1.10 изображена принципиальная схема центрального процессорного устройства с тремя буферированными системными шинами (*System Bus*) — мультиплексор 1533КП11 реализует вышеприведенные функции. Действительно, из функций

$$DO_j = \begin{cases} DI_{j,0}\overline{A} \vee DI_{j,1}A, \text{ если } \overline{OE} = AEN = 0, \\ Z\text{ - состояние, если } \overline{OE} = AEN = 1, \end{cases}$$

описывающих мультиплексор 1533КП11 [5], например, следует, что

$$\overline{MEMR} = DI_{0,0}\overline{A} \vee DI_{0,1}A = \overline{RD} \cdot \overline{IO / \overline{M}} \vee 1 \cdot IO / \overline{M} = \overline{RD} \vee IO / \overline{M} = \overline{RD \cdot IO / \overline{M}}.$$

Значение сигнала $AEN = 1$ вырабатывается контроллером прямого доступа к памяти (*DMAC*) для отключения центрального процессорного устройства (*CPU*) от системных шин. Если ввод-вывод по прямому доступу к памяти не используется, то следует задать значение сигнала $AEN \equiv 0$.

Сигнал *INT* поступает от контроллера прерываний 8259/8259*A*, а сигнал $\overline{INTA}$ подается только на этот контроллер для чтения трехбайтовой команды CALL *addr*. Для автоматической подачи при включении питания значения сигнала сброса $\overline{RESET\,IN} = 0$ используется *RC*-цепь (75 кОм и 1 мкФ). Кроме того, этот сигнал можно подавать от клавиши системного сброса.

Конфигурации МП-систем, построенных на основе МП 8085 с использованием специальных БИС (8155/8156, 8755/8355 и 8185), будут рассмотрены в § 3.9.

1.5. Формат команд микропроцессоров 8080 и 8085

Команды МП 8080*A*/8085*A* представляются одним, двумя и тремя байтами (рис. 1.11). Многобайтные команды размещаются в последовательных ячейках памяти (адрес первого байта команды является адресом всей команды). Чем больше байт в команде, тем больше времени затрачивается для ее выборки из памяти.

Код операции (КОП) всегда находится в первом байте команды. Второй и третий байты команды используются для задания операндов (данных, портов устройств ввода-вывода и адресов памяти). Так как КОП задается только одним байтом, то всего может быть не более $2^8 = 256$ различных команд. Все множество КОП по числу байт в команде делится на четыре подмножества (рис. 1.12). В МП 8080*A* для представления КОП не используется 12 кодов, а в МП 8085*A* — 10 кодов.

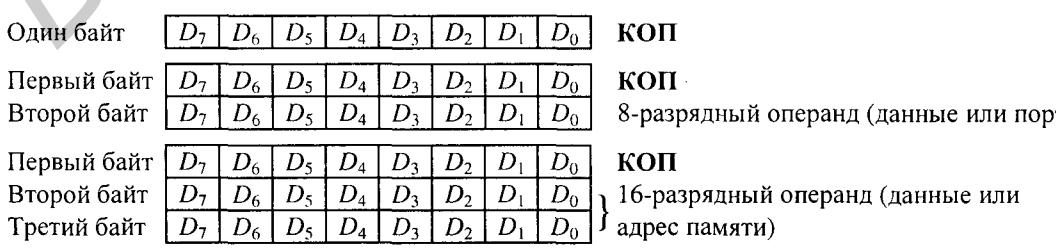


Рис. 1.11. Форматы команд

1 байт — 200/202	2 байта — 18
3 байта — 26	Нет — 12/10

Рис. 1.12. Типы команд

D7	D6	D5	D4	D3	D2	D1	D0
0	1	D	D	D	S	S	S

Рис. 1.13. Адресация регистров

Таблица 1.3. **Коды регистров и памяти**

D S	D S	D S	Имя регистра
1	1	1	*A*
0	0	0	*B*
0	0	1	*C*
0	1	0	*D*
0	1	1	*E*
1	0	0	*H*
1	0	1	*L*
1	1	0	*M*

Таблица 1.4. **Кодирование регистровых пар**

Разряды D_5	Разряды D_4	Регистровая пара (*rp*)	Регистры
0	0	*B*	*B* и *C*
0	1	*D*	*D* и *E*
1	0	*H*	*H* и *L*
1	1	*SP* или *PSW*	*PSW* = *A* и *F*

Примеры:

МК = $00D_5D_4$0001 для команд LXI *rp*, *d*16
МК = $11D_5D_4$0101 для команд PUSH *rp*
МК = $11D_5D_4$0001 для команд POP *rp*

Кроме адресации памяти и устройств ввода-вывода, выполняемой по шине адреса $A_{15\text{-}0}$, необходима адресация РОНов и регистровых пар. Эта адресация производится частью разрядов КОП *DDD* и *SSS* (*D* — *Destination* — получатель, *S* — *Source* — источник), коды которых приведены в табл. 1.3. Пример машинного кода (МК) команды MOV *r*1, *r*2, где r_i = A, B, C, D, E, H и L, показан на рис. 1.13. Код D_7D_6 = 01 задает операцию MOV — пересылку содержимого регистра *r*2 (код *SSS* — источник) в регистр *r*1 (код *DDD* — получатель). В двухоперандных командах первым (слева) всегда указывается получатель. Из рис. 1.13 следует, что машинные коды всех команд передачи данных типа MOV *r*1, *r*2 задаются значениями 40*h* ... 7F*h*, за исключением МК = 76*h*, которому должна была бы соответствовать команда MOV M, M — команд, оперирующих с двумя операндами в памяти нет [3, 4].

Задача. Найти машинный код команды MOV E, C. *Решение*: *DDD* = 011 для регистра *E* и *SSS* = 001 для регистра *C*, поэтому МК = 01 011 001 = 59*h* (см. табл. 1.6).

Значения *DDD* = 110 и *SSS* = 110 указывают операции с операндом, находящимся в памяти, когда его адрес находится в регистрах *H* (старший байт) и *L* (младший байт). В двухоперандных командах только один операнд может находиться в памяти, т. е. могут использоваться только команды пересылки вида MOV M, *r*2 и MOV *r*1, M. Для хранения 16-разрядных операндов 8-разрядные РОНы объединяются в три 16-разрядные регистровые пары: *rp B* (регистры *B* и *C*), *rp D* (регистры *D* и *E*) и *rp H* (регистры *H* и *L*). В обозначении регистровой пары указывается регистр, в котором находится старший байт операнда. Адресация регистровых пар производится двумя разрядами D_5D_4 в коде операции (табл. 1.4). Код 11 в табл. 1.4 адресует указатель стека *SP* (16-разрядный операнд) или *PSW* в зависимости от типа команды.

1.6. Система команд микропроцессоров 8080 и 8085

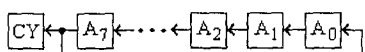
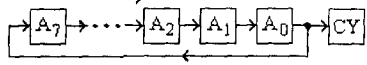
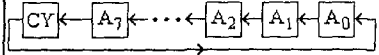
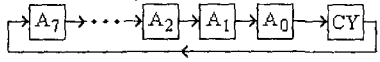
Мнемоника команд на языке ассемблера и выполняемые ими операции представлены в табл. 1.5. Операндами могут являться 8-разрядные регистры общего назначения МП с именами A, B, C, D, E, H и L (*r*, *r*1, *r*2), 16-разрядные регистровые пары B, D и H (*rp*), указатель стека SP, слово состояния процессора PSW (*Processor Status Word* — содержимое аккумулятора A и

регистра признаков F), 8-разрядные ячейки памяти *M*, 8- и 16-разрядные данные *d*8 и *d*16, которым пользователем могут присваиваться символические имена, 8-разрядный адрес *port* внешнего устройства, 16-разрядный адрес памяти *addr*.

Таблица 1.5. **Команды микропроцессоров 8080*A* и 8085*A***

Команда		Описание операции		*B/M*	T^{80}	T^{85}	*F*
Группа команд передачи данных							
MOV	*r*1, *r*2	*r*1 ← *r*2	{*Move*}	1/1	5	4	–
MOV	*r*, M	*r* ← *M*(*rp* H)		1/2	7	7	–
MOV	M, *r*	*M*(*rp* H) ← *r*		1/2	7	7	–
MVI	*r*, *d*8	*r* ← *d*8	{*Move immediate*}	2/2	7	7	–
MVI	M, *d*8	*M*(*rp* H) ← *d*8		2/3	10	10	–
LXI	*rp*, *d*16	*rp* ← *d*16, *rp* = B, D, H или SP	{*Load immediate*}	3/3	10	10	–
LDA	*addr*	A ← *M*(*addr*)	{*Load direct accumulator*}	3/4	13	13	–
STA	*addr*	*M*(*addr*) ← A	{*Store direct accumulator*}	3/4	13	13	–
LHLD	*addr*	L ← *M*(*addr*), H ← *M*(*addr* + 1)	{*Load H and L direct*}	3/5	16	16	–
SHLD	*addr*	*M*(*addr*) ← L, *M*(*addr* + 1) ← H	{*Store H and L direct*}	3/5	16	16	–
LDAX	*rp*	A ← *M*(*rp*), *rp* = B или D	{*Load accumulator indirect*}	1/2	7	7	–
STAX	*rp*	*M*(*rp*) ← A, *rp* = B или D	{*Store accumulator indirect*}	1/2	7	7	–
XCHG		H ↔ D, L ↔ E	{*Exchange*}	1/1	4	4	–
Группа команд арифметических операций							
ADD	*r*	A ← A + *r*	{*Add*}	1/1	4	4	+
ADI	*d*8	A ← A + *d*8	{*Add immediate*}	2/2	7	7	+
ADD	M	A ← A + *M*(*rp* H)		1/2	7	7	+
ADC	*r*	A ← A + *r* + CY	{*Add with carry*}	1/1	4	4	+
ADC	M	A ← A + *M*(rp H) + CY		1/2	7	7	+
ACI	*d*8	A ← A + *d*8 + CY	{*Add with carry immediate*}	2/2	7	7	+
SUB	*r*	A ← A – *r*	{*Subtract*}	1/1	4	4	+
SUI	*d*8	A ← A – *d*8	{*Subtract immediate*}	2/2	7	7	+
SUB	M	A ← A – *M*(*rp* H)		1/2	7	7	+
SBB	*r*	A ← A – *r* – CY	{*Subtract with borrow*}	1/1	4	4	+
SBI	*d*8	A ← A – *d*8 – CY	{*Subtract immediate*}	2/2	7	7	+
SBB	M	A ← A – *M*(*rp* H) – CY		1/2	7	7	+
INR	*r*	*r* ← *r* + 1	{*Increment*}	1/1	5	4	Δ
INR	M	*M*(*rp* H) ← *M*(*rp* H) + 1		1/3	10	10	Δ
DCR	*r*	*r* ← *r* – 1	{*Decrement*}	1/1	5	4	Δ
DCR	M	*M*(*rp* H) ← *M*(*rp* H) – 1		1/3	10	10	Δ
INX	*rp*	*rp* ← *rp* + 1, *rp* = B, D, H или SP	{*Increment rp*}	1/1	5	6	–
DCX	*rp*	*rp* ← *rp* – 1, *rp* = B, D, H или SP	{*Decrement rp*}	1/1	5	6	–
DAD	*rp*	*rp* H ← *rp* H + *rp*, *rp* = B, D, H или SP	{*Double precision add*}	1/1	10	10	∇
DAA		Десятичная коррекция	{*Decimal adjust accumulator*}	1/1	4	4	+
Группа команд логических операций							
ANA	*r*	A ← A & *r*	{*And register with accumulator*}	1/1	4	4	*
ANI	*d*8	A ← A & *d*8	{*And immediate with accumulator*}	2/2	7	7	*
ANA	M	A ← A & *M*(*rp* H)	{*And memory with accumulator*}	1/2	7	7	*
XRA	*r*	A ← A ⊕ *r*	{*Exclusive or register with accumulator*}	1/1	4	4	#
XRI	*d*8	A ← A ⊕ *d*8	{*Exclusive or immediate with accumulator*}	2/2	7	7	#
XRA	M	A ← A ⊕ *M*(*rp* H)	{*Exclusive or memory with accumulator*}	1/2	7	7	#

Продолжение табл. 1.5

Команда	Описание операции	B/M	T^{80}	T^{85}	F
ORA r	A ← A ∨ r {Or register with accumulator}	1/1	4	4	#
ORI d8	A ← A ∨ d8 {Or immediate with accumulator}	2/2	7	7	#
ORA M	A ← A ∨ M(rp H) {Or memory with accumulator}	1/2	7	7	#
CMP r	A – r {Compare}	1/1	4	4	+
CPI d8	A – d8 } Z = 1 при A = r, d8, M; CY = 1 при A < r, d8, M {Compare immediate}	2/2	7	7	+
CMP M	A – M(rp H)	1/2	7	7	+
RLC	 {Rotate accumulator left in carry}	1/1	4	4	∇
RRC	 {Rotate accumulator right in carry}	1/1	4	4	∇
RAL	 {Rotate accumulator left through carry}	1/1	4	4	∇
RAR	 {Rotate accumulator right through carry}	1/1	4	4	∇
CMA	A ← $\overline{A}$ {Complement accumulator}	1/1	4	4	–
CMC	CY ← $\overline{CY}$ {Complement carry}	1/1	4	4	∇
STC	CY ← 1 {Set carry}	1/1	4	4	∇
Группа команд передачи управления					
JMP addr	PC ← addr {Jump}	3/3	10	10	–
Jcond addr	PC ← addr {Jump conditional}	3/3	10	7/10 [1]	–
PCHL	PC ← HL {Load Program Counter with H and L}	1/1	5	6	–
CALL addr	SP ← SP – 1, M(SP) ← PCh, {Call}	3/5	17	18	–
Ccond addr	SP ← SP – 1, M(SP) ← PCl; PC ← addr {Call conditional}	3/3/5 [2]	11/17 [1]	9/18 [1]	–
RET	PCl ← M(SP), SP ← SP + 1, {Return}	1/3	10	10	–
Rcond	PCh ← M(SP), SP ← SP + 1 {Return conditional}	1/1/3 [2]	5/11 [1]	6/12 [1]	–
RST n	SP ← SP – 1, M(SP) ← PCh, {Restart}; SP ← SP – 1, M(SP) ← PCl; PC ← 8 × n (n = 0 … 7)	1/3	11	12	–
Группа команд управления стеком, вводом-выводом и состоянием МП					
PUSH rp	SP ← SP – 1, M(SP) ← rph, SP ← SP – 1, M(SP) ← rpl {Push}	1/3	11	12	–
PUSH PSW	SP ← SP – 1, M(SP) ← A, SP ← SP – 1, M(SP) ← F	1/3	11	12	–
POP rp	rpl ← M (SP), SP ← SP + 1, rph ← M(SP), SP ← SP + 1 {Pop}	1/3	10	10	–
POP PSW	F ← M(SP), SP ← SP + 1, A ← M(SP), SP ← SP + 1	1/3	10	10	+
XTHL	L ↔ M(SP),H ↔ M(SP+1) {Exchange Top of Stack with H and L}	1/5	18	16	–
SPHL	SP ← HL {Load Stack Pointer with H and L}	1/1	5	6	–
IN port	A ← I/O(port) {Input}	2/3	10	10	–
OUT port	I/O(port) ← A {Output}	2/3	10	10	–
EI [3]	INTE = 1 после следующей команды {Enable Interrupts}	1/1	4	4	–
DI [3]	INTE = 0 после данной команды {Disable Interrupts}	1/1	4	4	–
HLT	Останов процессора; PC ← PC + 1 {Halt}	1/1	7	5	–
NOP	Пустая операция {No-operation}	1/1	4	4	–
SIM [4]	Установка маски прерываний {Set Interrupt Mask}	1/1	–	4	–
RIM [4]	Чтение маски прерываний {Read Interrupt Mask}	1/1	–	4	–

Примечание: [1] *X* из *X/Y* при невыполнении условия *cond* и *Y* при выполнении условия *cond*; [2] *X* из *Z/X/Y* при невыполнении условия *cond* и *Y* при выполнении условия *cond*; [3] в МП 8085*A* вместо триггера *INTE* данные команды управляют флагом (триггером) *IE* (см. рис. 1.16); [4] только для МП 8085*A*.

Таблица 1.6. **Машинные коды команд МП 8080*A* и 8085*A***

МК	×0	×1	×2	×3	×4	×5	×6	×7	МК
0×	NOP	LXI B, *d*16	STAX B	INX B	INR B	DCR B	MVI B, *d*8	RLC	0×
1×	—	LXI D, *d*16	STAX D	INX D	INR D	DCR D	MVI D, *d*8	RAL	1×
2×	RIM	LXI H, *d*16	SHLD *addr*	INX H	INR H	DCR H	MVI H, *d*8	DAA	2×
3×	SIM	LXI SP, *d*16	STA *addr*	INX SP	INR M	DCR M	MVI M, *d*8	STC	3×
4×	MOV B, B	MOV B, C	MOV B, D	MOV B, E	MOV B, H	MOV B, L	MOV B, M	MOV B, A	4×
5×	MOV D, B	MOV D, C	MOV D, D	MOV D, E	MOV D, H	MOV D, L	MOV D, M	MOV D, A	5×
6×	MOV H, B	MOV H, C	MOV H, D	MOV H, E	MOV H, H	MOV H, L	MOV H, M	MOV H, A	6×
7×	MOV M, B	MOV M, C	MOV M, D	MOV M, E	MOV M, H	MOV M, L	HLT	MOV M, A	7×
8×	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	8×
9×	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	9×
A×	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	A×
B×	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	B×
C×	RNZ	POP B	JNZ *addr*	JMP *addr*	CNZ *addr*	PUSH B	ADI *d*8	RST 0	C×
D×	RNC	POP D	JNC *addr*	OUT *port*	CNC *addr*	PUSH D	SUI *d*8	RST 2	D×
E×	RPO	POP H	JPO *addr*	XTHL	CPO *addr*	PUSH H	ANI *d*8	RST 4	E×
F×	RP	POP PSW	JP *addr*	CP *addr*	DI	PUSH PSW	ORI *d*8	RST 6	F×
МК	×0	×1	×2	×3	×4	×5	×6	×7	МК

Символическое обозначение типа $r1 \leftarrow r2$ означает передачу содержимого регистра $r2$ в регистр $r1$. Адреса памяти *M* и внешних устройств *I/O* указываются в круглых скобках, например: *M*(*addr*), *M*(*rp*), *I/O*(*port*) (*addr* — адрес памяти, *port* — адрес внешнего устройства, *rp* — содержимое регистровой пары B, D или H, являющееся адресом памяти). Адресация некоторых операндов производится в неявном виде самим кодом операции команды. Двухбайтовые операнды и адреса при их описании иногда маркируются буквами *l* (*low* — младший байт операнда или адреса) и *h* (*high* — старший байт). В табл. 1.5 дано описание команд: их назначение, число байт и машинных циклов (*B*/*M*), число тактов *T* выполнения команды (T^{80}, T^{85}) и воздействие на флаги (*F*). Машинные коды команд (МК) приведены в табл. 1.6 (например, МК команды MOV D, B равен $01010000_2 = 50_{16}$ — см. рис. 1.13).

Регистр признаков. При выполнении некоторых команд используются биты (флаги *F*) регистра признаков *F* (узел *FFF* на рис. 1.3). Содержимое регистра признаков может изменяться командами, так как в нем фиксируются некоторые результаты выполнения команд. Формат регистра признаков *F* показан на рис. 1.14.

МК	×8	×9	×A	×B	×C	×D	×E	×F	МК
0×	—	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C, *d8*	RRC	0×
1×	—	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E, *d8*	RAR	1×
2×	—	DAD H	LHLD *addr*	DCX H	INR L	DCR L	MVI L, *d8*	CMA	2×
3×	—	DAD SP	LDA *addr*	DCX SP	INR A	DCR A	MVI A, *d8*	CMC	3×
4×	MOV C, B	MOV C, C	MOV C, D	MOV C, E	MOV C, H	MOV C, L	MOV C, M	MOV C, A	4×
5×	MOV E, B	MOV E, C	MOV E, D	MOV E, E	MOV E, H	MOV E, L	MOV E, M	MOV E, A	5×
6×	MOV L, B	MOV L, C	MOV L, D	MOV L, E	MOV L, H	MOV L, L	MOV L, M	MOV L, A	6×
7×	MOV A, B	MOV A, C	MOV A, D	MOV A, E	MOV A, H	MOV A, L	MOV A, M	MOV A, A	7×
8×	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A	8×
9×	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A	9×
A×	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A	A×
B×	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A	B×
C×	RZ	RET	JZ *addr*	—	CZ *addr*	CALL *addr*	ACI *d8*	RST 1	C×
D×	RC	—	JC *addr*	IN *port*	CC *addr*	—	SBI *d8*	RST 3	D×
E×	RPE	PCHL	JPE *addr*	XCHG	CPE *addr*	—	XRI *d8*	RST 5	E×
F×	RM	SPHL	JM *addr*	EI	CM *addr*	—	CPI *d8*	RST 7	F×
МК	×8	×9	×A	×B	×C	×D	×E	×F	МК

D7	*D6*	*D5*	*D4*	*D3*	*D2*	*D1*	*D0*
S	*Z*	0	*AC*	0	*P*	1	*CY*

Рис. 1.14. Регистр признаков *F*

Флаг *Z* (*zero* — нуль) идентифицирует нулевой результат выполнения в АЛУ команды:

$$Z = \begin{cases} 0 - \text{результат операции не равен 0}, \\ 1 - \text{результат операции равен 0}. \end{cases}$$

Флаг *CY* (*Carry* — перенос) идентифицирует перенос/заем, возникающий при выполнении в АЛУ операций над операндами:

$$CY = \begin{cases} 0 - \text{нет переноса (заема) из разряда } D_7, \\ 1 - \text{есть перенос (заем) из разряда } D_7. \end{cases}$$

Флаг P (*Parity* — паритет) идентифицирует четное или нечетное число единиц, содержащихся в байте результата выполнения команды (*Even/Odd Parity* — четный/нечетный паритет):

$$P = \begin{cases} 0 - \text{число единиц в байте нечетное}, \\ 1 - \text{число единиц в байте четное} \end{cases}$$

($P = \overline{A_7 \oplus A_6 \oplus A_5 \oplus A_4 \oplus A_3 \oplus A_2 \oplus A_1 \oplus A_0}$, A_i — разряды аккумулятора, $i = 0 \ldots 7$).

Флаг S (*Sign* — знак) идентифицирует знак числа результата выполнения команд арифметических операций в дополнительном коде:

$$S = \begin{cases} 0 - \text{разряд } D_7 = 0 \text{ (число положительное)}, \\ 1 - \text{разряд } D_7 = 1 \text{ (число отрицательное)} \end{cases}$$

(состояние флага S изменяют также команды логических операций). Основные сведения о дополнительном коде чисел смотри в конце этого параграфа.

Флаг AC (*Auxiliary Carry* — вспомогательный перенос) идентифицирует наличие переноса из разряда D_3 в разряд D_4 при выполнении команд арифметических операций:

$$AC = \begin{cases} 0 - \text{нет переноса из разряда } D_3, \\ 1 - \text{есть перенос из разряда } D_3. \end{cases}$$

Этот флаг используется при выполнении команд десятичной арифметики, когда числа представляются в упакованном двоично-десятичном коде, например: $95_{10} = 1001\ 0101_2$ — один байт представляет двухразрядное десятичное число. В тетрадах байта нельзя использовать числа от A*h* = 1010_2 до F*h* = 1111_2. Десятичную коррекцию числа выполняет команда DAA (см. *задачу* 1).

Воздействие команд на флаги регистра признаков указано в графе F табл. 1.5 символами: –, +, Δ, ∇, * и #. Значения флагов, обозначенных этими символами, приведены в табл. 1.7 ("–" — команда не изменяет значения флага, "+" — команда изменяет флаг, "0" — флаг устанавливается в состояние 0).

Таблица 1.7. **Флаги**

F	*S*	*Z*	*AC*	*P*	*CY*
–	–	–	–	–	–
+	+	+	+	+	+
Δ	+	+	+	+	–
∇	–	–	–	–	+
*	+	+	+	+	0
#	+	+	0	+	0

Флаг переноса *CY* позволяет организовать программным способом сложение, вычитание, умножение и деление чисел, для представления которых используется любое число байт. Четыре флага (*Z*, *CY*, *S* и *P*) используются командами передачи управления для осуществления условных переходов, условных вызовов подпрограмм и условных возвратов из подпрограмм. Все команды по их назначению делятся на пять групп.

Группа команд передачи данных. Эти команды не изменяют содержимого регистра признаков *F*, так как производят только пересылку данных из одного устройства (регистра или памяти) в другое, не изменяя их содержимого. Примеры записи команд передачи данных:

```
MOV     B, H        ; B ← H
MOV     A, M        ; A ← M(rp H) — адрес памяти M в регистрах H и L
XCHG                ; H ↔ D, L ↔ E — обмен содержимым rp H и rp D
```

Код операции команды (мнемоника MOV, XCHG и др.) должны отделяться одним или большим числом пробелов от операндов B, H, A, M и др. В двухоперандных командах слева указывается операнд назначения, а после запятой — источник операнда.

Группа команд арифметических операций. Эти команды преобразуют операнды и изменяют флаги регистра признаков *F* (см. табл. 1.5 и 1.7). Группа содержит команды сложения, вычитания (двухоперандные команды), инкрементирования и декрементирования (одноопе-рандные команды), а также специальную команду DAA десятичной коррекции результата арифметической операции десятичной арифметики. В двухоперандных командах один из операндов, как правило, находится в аккумуляторе A и результат операции помещается в аккумулятор. Примеры команд:

```
ADD   D    ; A ← A + D — сложение содержимых аккумулятора A и регистра D
SBB   H    ; A ← A – H – CY — вычитание из содержимого аккумулятора A
           ;      содержимого регистра H с заемом CY (CY = Borrow — заем)
INR   D    ; D ← D + 1 — увеличение на 1 содержимого 8-разрядного регистра D
INX   D    ; DE ← DE + 1 — увеличение на 1 содержимого 16-разрядной
           ;      регистровой пары rp D
DAD   B    ; HL ← HL + BC — операция сложения с двойной точностью
           ;      (сложение двух 16-разрядных чисел, находящихся в rp H и rp B)
```

При выполнении операции сложения в десятичной арифметике одним байтом данных должно быть представлено двухразрядное десятичное число от 00 до 99 (упакованный формат — для каждой десятичной цифры используется одна тетрада). Перенос из младшей тетрады $D_{3\text{-}0}$ в старшую $D_{7\text{-}4}$ фиксируется в бите *AC* регистра признаков *F*, который используется для коррекции результата сложения двоичных байт в АЛУ с целью получения десятичного значения суммы чисел.

Задача 1. Вычислить сумму двухразрядных десятичных чисел, находящихся в регистрах *D* и *L*, и сохранить ее в регистровой паре *rp B* (в регистре *C* сумму и в регистре *B* перенос — 0 или 1). *Решение*:

```
MOV   A, L
ADD   D        ; A ← L + D
DAA            ; Десятичная коррекция
MOV   C, A     ; C ← L + D
MVI   A, 0     ; A ← 0
ADC   A        ; A ← 0 + 0 + CY
MOV   B, A     ; B ← CY = 0 или 1
```

Группа команд логических операций. Двухоперандные логические операции производятся поразрядно. Например, команды ANA E, ORA E и XRA E производят преобразования:

$$A_7A_6A_5A_4A_3A_2A_1A_0 \leftarrow (A_7{*}E_7)(A_6{*}E_6)(A_5{*}E_5)(A_4{*}E_4)(A_3{*}E_3)(A_2{*}E_2)(A_1{*}E_1)(A_0{*}E_0),$$

где $A_7A_6A_5A_4A_3A_2A_1A_0$ и $E_7E_6E_5E_4E_3E_2E_1E_0$ — операнды, а операция * означает одну из операций: & (конъюнкция), $\vee$ (дизъюнкция) и $\oplus$ (сумма по модулю два). Логическая операция инвертирования всех разрядов содержимого аккумулятора *A* выполняется командой CMA:

$$A_7A_6A_5A_4A_3A_2A_1A_0 \leftarrow \overline{A_7}\,\overline{A_6}\,\overline{A_5}\,\overline{A_4}\,\overline{A_3}\,\overline{A_2}\,\overline{A_1}\,\overline{A_0}.$$

Задача 2. Проинвертировать нечетные разряды байта в ячейке памяти по адресу D57A. *Решение*:

```
LDA  0D57Ah  ; Загрузка в аккумулятор A содержимого ячейки памяти M(D57A)
XRI  0AAh    ; AAh = 10101010b — инвертирование при значении 1 разряда этого числа
STA  0D57Ah  ; Запись в ячейку памяти M(D57A) результата операции
```

Команды сравнения CMP (CMP — *Compare* — сравнение) 8-разрядных операндов выполняются с помощью операции вычитания без фиксирования результата в аккумуляторе *A*, т. е. сами операнды при выполнении этих команд не изменяются. Это позволяет производить последовательные многократные сравнения содержимого аккумулятора *A* с различными операндами. Изменение флагов *Z* и *CY* этими командами в регистре признаков используется командами условной передачи управления для изменения порядка выполнения команд программы.

Команды сдвига операндов на *один* разряд влево RLC, RAL и вправо RRC, RAR производятся над содержимым аккумулятора с участием флага *CY* регистра признаков (см. табл. 1.5). Данные операции позволяют реализовать, в частности, программное выполнение операций умножения и деления многобайтных чисел в соответствии с их традиционными алгоритмами.

Задача 3. В регистре *E* поменять местами младшую и старшую тетрады. *Решение*:

```
MOV  A, E   ; A = A7A6A5A4A3A2A1A0 ← E
RRC         ; A ← A0A7A6A5A4A3A2A1
RRC         ; A ← A1A0A7A6A5A4A3A2
RRC         ; A ← A2A1A0A7A6A5A4A3
RRC         ; A ← A3A2A1A0A7A6A5A4
MOV  E, A   ; E ← A (здесь вместо команды сдвига вправо RRC можно
            ;        использовать и команду сдвига влево RLC)
```

Группа команд передачи управления. Выборка команд программы производится из последовательных ячеек памяти в порядке возрастания их адресов при выполнении всех команд, кроме команд передачи управления (JMP, J*cond*, PCHL и CALL, C*cond*, RST). Данные команды применяются для изменения последовательности выборки команд из памяти. Некоторые из этих команд используют флаги регистра признаков *F* для условных передач управления — при выполнении условия, зафиксированного во флаге регистра признаков *F*, производится передача управления по адресу, указанному в команде, а при невыполнении условия производится переход к выполнению следующей команды программы.

Таблица 1.8. **Условия передачи управления**

Cond	Результат операции для передачи управления	Команды условных передач управления		
NZ	Не нуль ($Z = 0$)	JNZ,	CNZ,	RNZ
Z	Нуль ($Z = 1$)	JZ,	CZ,	RZ
NC	Нет переноса ($CY = 0$)	JNC,	CNC,	RNC
C	Есть перенос ($CY = 1$)	JC,	CC,	RC
PO	Нечетность ($P = 0$)	JPO,	CPO,	RPO
PE	Четность ($P = 1$)	JPE,	CPE,	RPE
P	Плюс ($S = 0$)	JP,	CP,	RP
M	Минус ($S = 1$)	JM,	CM,	RM

В табл. 1.5 были приведены команды условных передач управления. Они содержат мнемонику *cond* (*condition* — условие), которая заменяется в командах на языке ассемблера на мнемонику в соответствии с табл. 1.8.

Команды JMP *addr* и J*cond addr* (команды безусловных и условных переходов) задают переход на выполнение команды, расположенной по адресу *addr*, который содержится во втором и третьем байтах команд (прямая адресация переходов). Эти команды обеспечивают ветвление программ. Команда безусловного перехода PCHL производит загрузку содержимого *rp H* в программный счетчик *PC*, что приводит к выборке следующей команды, находящейся по адресу, задаваемому содержимым регистров *H* и *L* (косвенно-регистровая адресация переходов).

Команды CALL *addr*, C*cond* *addr* (трехбайтовые команды) и RST *n* (однобайтовые команды) используются для вызова подпрограмм. Команды CALL *addr* и C*cond* *addr* передают управление по адресу *addr*, который содержится во втором и третьем байтах этих команд. Команды RST *n* передают управление по адресу $addr = n \times 8$ ($n = 0, 1, \ldots, 7$), который автоматически вычисляется внутри МП. Эти команды можно использовать также для организации векторных прерываний (см. § 2.4). Команды рестарта RST *n* передают управление только по фиксированным адресам:

0000*h*, 0008*h*, 0010*h*, 0018*h*, 0020*h*, 0028*h*, 0030*h*, 0038*h*.

Для наиболее часто вызываемых подпрограмм целесообразно использовать однобайтные команды RST *n* вместо трехбайтовых команд CALL *addr*. Команды вызова подпрограмм используют стек для включения в него адреса возврата:

SP ← SP – 1, *M*(SP) ← PC*h*, SP ← SP – 1, *M*(SP) ← PC*l* — включение в стек адреса возврата;
PC ← *addr* (для команд CALL) и PC ← 8 × *n* (для команд RST) — передача управления

(адрес возврата — это находящийся в программном счетчике *PC* адрес команды, непосредственно следующей за командой вызова подпрограммы).

Команды возврата из подпрограмм RET и R*cond* заканчивают выполнение подпрограммы и передают управление на команду, непосредственно следующую за командой, вызвавшей эту подпрограмму: PC*l* ← *M*(SP), SP ← SP + 1, PC*h* ← *M*(SP), SP ← SP + 1 — извлечение из стека адреса возврата.

Выполнение подпрограммы обычно прекращается по команде RET безусловного возврата из подпрограмм, однако для выхода из подпрограммы можно использовать и команды J*cond* условных переходов (см. *задачу* 9 в § 1.8 на с. 81). Подпрограммы могут содержать и команды условного возврата R*cond*, досрочно прекращающих их выполнение при реализации заданного условия. Но в любом случае подпрограмма должна заканчиваться командой безусловного выхода RET.

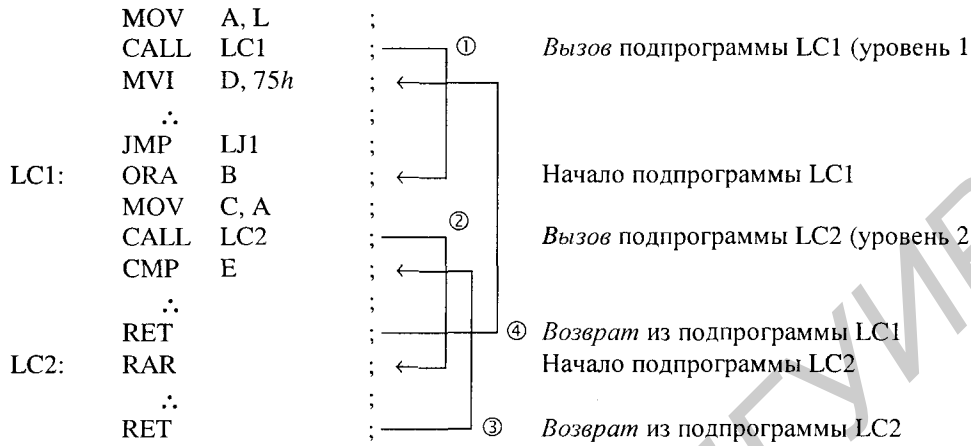
Пример 1:

```
      MOV   A, L    ; ①
      CALL  LC1     ; ──┐        Вызов подпрограммы (LC1 — символическое имя
      MVI   D, 75h  ; ←─┼──┐         адреса вызова подпрограммы)
        ∴           ;   │  │
      JMP   LJ1     ;   │  │
LC1:  ORA   B       ; ←─┘  │     Начало подпрограммы
      MOV   C, A    ;      │
        ∴           ;      │
      RZ            ; ─────②     Условный возврат из подпрограммы
        ∴           ;      │
      RET           ; ─────②     Безусловный возврат из подпрограммы (и ее конец)
```

Адрес передачи управления при вызове подпрограмм на языке ассемблера указывается метками, например, LC1 — символическое имя адреса передачи управления. Метка должна заканчиваться двоеточием (в некоторых ассемблерах двоеточие можно и не писать — его заменяет пробел). Ассемблер при трансляции программы в машинные коды вычисляет адреса меток и подставляет их двухбайтовые численные значения в команды вызова подпрограмм, содержащие символические имена этих адресов.

Подпрограммы могут иметь любое число уровней вложенности. От их числа зависит минимальный размер стека.

Пример 2:



Эта программа имеет один уровень вложенности подпрограмм — подпрограмма LC2 вложена в подпрограмму LC1 (цифрами ①, ②, ③ и ④ помечен порядок выполнения передач управления).

Группа команд управления стеком, вводом-выводом и состоянием МП. Напомним, что стеком называется область ОЗУ, отводимая для адресации указателем стека *SP*. Стек представляет собой память магазинного типа *LIFO* (*last-in, first-out* — последним вошел, первым вышел), работающую без явной адресации данных (в командах не нужно указывать адрес операнда). Адресация памяти с помощью независимых регистров *SP* и *PC* значительно увеличивает возможности управления памятью.

Команды PUSH PSW, PUSH *rp* и POP PSW, POP *rp* включают в стек два байта, содержащихся в регистре *PSW* = *A* и *F* или регистровой паре *rp B*, *D* и *H*. Адрес памяти содержится в указателе стека *SP*, который автоматически дважды декрементируется при выполнении команды PUSH и дважды инкрементируется при выполнении команды POP:

SP ← SP – 1, *M*(SP) ← A, SP ← SP – 1, *M*(SP) ← F — PUSH PSW;
SP ← SP – 1, *M*(SP) ← *rph*, SP ← SP – 1, *M*(SP) ← *rpl* — PUSH *rp*;
F ← *M*(SP), SP ← SP + 1, A ← *M*(SP), SP ← SP + 1 — POP PSW;
rpl ← *M* (SP), SP ← SP + 1, *rph* ← *M*(SP), SP ← SP + 1 — POP *rp*.

Команды ввода IN *port* и вывода OUT *port* обеспечивают связь МП с внешними устройствами при использовании адресного пространства *I/O*, независимого от адресного пространства памяти.

Команды EI и DI управляют разрешением и запретом приема запросов прерываний от внешних устройств *I/O*.

Команда HLT устанавливает режим останова МП, выйти из которого он может только при запросе прерывания от внешних устройств при условии, что предварительно командой EI было установлено значение сигнала *INTE* = 1 в МП 8080*A* или флага *IE* в МП 8085*A* (см. рис. 1.16).

Системы команд МП 8080*A* и 8085*A* различаются только двумя командами SIM и RIM, которых нет в МП 8080*A*.

Команда SIM (*Set Interrupt Mask*) устанавливает маску прерываний, используя содержимое аккумулятора, который должен быть предварительно загружен. Назначение разрядов аккумулятора показано на рис. 1.15: *Mk*.5 = 1 — запрет прерываний по входу *RST k*.5 (*k* = 5, 6, 7); *MSE* = 1 (*Mask Set Enable*) — разрешение установки маски; *R*7.5 = 1 — сброс триггера запроса

прерывания, устанавливаемого в 1 положительным фронтом сигнала *RST* 7.5 (сброс этого триггера производится и автоматически при подтверждении прерывания *RST* 7.5); *SOD* (*Serial Output Data*) — разряд данных для вывода по последовательному каналу; *SOE* (*Serial Output Enable*) — разрешение вывода разряда D_7 аккумулятора в триггер *SOD* последовательного канала связи. Если разряд *MSE* = 0, то маска не устанавливается, т. е. разряды *MSE* и *SOE* используются для разделения операций установки маски и последовательного вывода данных. Значение сигнала $\overline{RESET\ IN} = 0 \Rightarrow SOD = 0$, *R*7.5 = 0 и *Mk*.5 = 1.

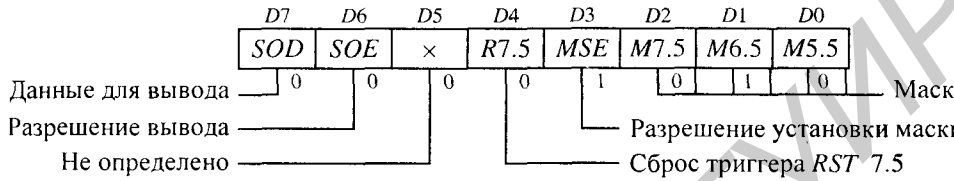


Рис. 1.15. Назначение разрядов аккумулятора для команды SIM

Задача 4. Разрешить обработку запросов прерываний по входам *RST* 7.5, *RST* 5.5 и запретить по входу *RST* 6.5. Вывести значение разряда 0 регистра *B* в последовательный канал связи *SOD*. *Решение*:

```
MVI   A, 0Ah    ; A ← 0Ah = 0000 1010
SIM             ; Установка маски прерываний
EI              ; Разрешение приема незамаскированных запросов прерываний
MOV   A, B      ; A ← B7B6B5B4B3B2B1B0
RRC             ; A ← A0A7A6A5A4A3A2A1 = B0B7B6B5B4B3B2B1
ANI   80h       ; A ← B0B7B6B5B4B3B2B1 & 1000 0000 = B0000 0000
ORI   40h       ; A ← B0000 0000 ∨ 0100 0000 = B0100 0000 (SOE ← 1)
SIM             ; SOD ← B0
```

На рис. 1.15 указаны значения 0 и 1 разрядов аккумулятора, удовлетворяющие условиям поставленной задачи — перед выполнением команды SIM в аккумулятор следует загрузить число 0A*h*.

Команда RIM (*Read Interrupt Mask*) загружает в аккумулятор данные, относящиеся к прерываниям и последовательному каналу ввода. На рис. 1.16 показано содержимое аккумулятора после выполнения этой команды. Байт содержит информацию о состояниях маски (разряды D_2, D_1 и D_0), флага разрешения прерывания (разряд D_3), информацию о входах аппаратных прерываний *RST* 7.5, 6.5 и 5.5, находящихся в состоянии ожидания обслуживания (разряды D_6, D_5 и D_4), и значение сигнала на входе *SID* последовательного канала (разряд D_7).

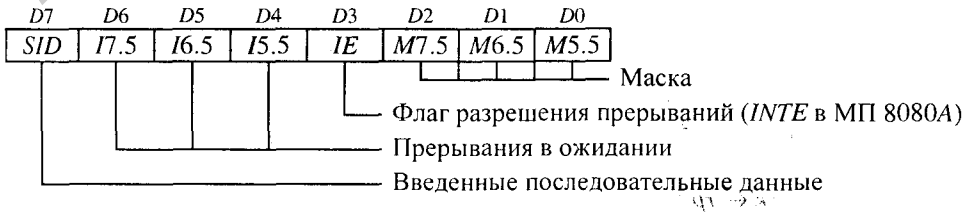


Рис. 1.16. Содержимое аккумулятора после выполнения команды RIM

Триггер *RST* 7.5 устанавливается в состояние 1 положительным фронтом сигнала *RST* 7.5 независимо от значения маски. Значения флага разрешения прерываний *IE* задаются командами EI (флаг *IE* устанавливается в 1) и DI (флаг *IE* устанавливается в 0). Сигнал системного сброса ($\overline{RESIN} = 0$) устанавливает значение *IE* = 0.

Задача 5. Установить маску для входа *RST* 7.5, оставив маскирование остальных входов *RST* неизменным. *Решение*:

```
RIM          ; Чтение маски прерываний
ANI   7      ; Установка в 0 пяти старших разрядов аккумулятора
ORI   0Ch    ; MSE ← 1 и M7.5 ← 1
SIM          ; Установка новой маски
```

Запрос прерывания по входу *TRAP* — немаскируемый, т. е. запросы прерывания воспринимаются независимо от значений флага разрешения прерываний *IE*. Если необходимо восстановить состояние флага *IE*, который был до прерывания *TRAP* (при переходе к обслуживанию запроса прерывания по любому входу флаг *IE* сбрасывается), то при обработке прерывания *TRAP* в вызванной подпрограмме должна быть выполнена команда RIM. В этом случае первая команда RIM загружает в разряд D_3 аккумулятора значение *IE*, которое было до прерывания *TRAP*. Для восстановления состояния флага *IE* необходимо выполнить команду EI, если будет получено значение $D_3 = 1$. Все последующие команды RIM дают текущее состояние флага разрешения прерываний *IE*.

Примеры программ. Качество программ оценивается двумя параметрами: объемом требуемой памяти и временем ее выполнения. Удовлетворить обоим этим требованиям обычно не представляется возможным. Поэтому приходится выбирать в качестве основного один из этих параметров.

Задача 6. Произвести взаимный обмен содержимого регистровых пар *B* и *D*. *Решение*:

```
XCHG         ; rp H ← rp D (H ← D, L ← E)
MOV   D, B   ; D ← B
MOV   E, C   ; E ← C
MOV   B, H   ; B ← H, равное исходному значению D
MOV   C, L   ; C ← L, равное исходному значению E
```

Вариант 1.
Программа требует 5 байт памяти и выполняется за 20 тактов (МП 8085*A*), при этом требуется, чтобы регистровая пара *H* не была занята.

```
 ∴
PUSH  B      ; Включение в стек содержимого rp B
PUSH  D      ; Включение в стек содержимого rp D
POP   B      ; Извлечение из стека (rp B ← rp D)
POP   D      ; Извлечение из стека (rp D ← rp B)
```

Вариант 2.
Программа требует 4 байта памяти и выполняется за 48 тактов.

Задача 7. В регистре *F* проинвертировать флаги *Z* и *P*, а флаги *AC* и *CY* сбросить в 0. *Решение*:

```
PUSH  PSW    ; Stack ← PSW = AF
POP   B      ; B ← A, C ← F
MOV   A, C
XRI   44h    ; Z ← Z̄, P ← P̄
ANI   0EEh   ; AC ← 0, CY ← 0
MOV   C, A   ; C = new F
PUSH  B      ; Stack ← rp B
POP   PSW    ; A ← B, регистр F ← C = new F
```

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		*Регистр*
	S	*Z*	0	*AC*	0	*P*	1	*CY*		*F*
	0	1	0	0	0	1	0	0	=	44*h*
	1	1	1	0	1	1	1	0	=	EE*h*

Задача 8. Начиная с адреса памяти 2000*h*, над A5E*h* байтами данных выполнить преобразование

$$D_7D_6D_5D_4D_3D_2D_1D_0 \rightarrow 0000D_7D_6D_5D_4$$

(требуется байты данных преобразовать в старшие тетрады с перемещением их на местоположение младших тетрад). *Решение*:

```
        LXI   H, 2000h   ; HL ← 2000h — начальный адрес ячеек памяти
        LXI   B, 0A5Eh   ; BC ←1A50h — объем массива преобразуемых данных
LM1:    MOV   A, M       ; A ← M(rp H) = D7D6D5D4D3D2D1D0 — чтение памяти
        RLC              ; A ← D6D5D4D3D2D1D0D7
        RLC              ; A ← D5D4D3D2D1D0D7D6
        RLC              ; A ← D4D3D2D1D0D7D6D5
        RLC              ; A ← D3D2D1D0D7D6D5D4
        ANI   0Fh        ; A ← 0000D7D6D5D4, 0Fh = 00001111
        MOV   M, A       ; M(rp H) ← A = 0000D7D6D5D4
        INX   H          ; HL ← HL +1
        DCX   B          ; BC ← BC – 1 (команда DCX не воздействует на флаги)
        MOV   A, C       ; A ← C
        ORA   B          ; A ← C ∨ B (C ∨ B = 0 только при C = 0 и B = 0),
        JNZ   LM1        ;              команда ORA воздействует на флаг Z
```

Задача 9. Вычислить сумму 4-разрядных упакованных десятичных чисел (двухбайтовых), находящихся в памяти: *M*(*AmS*+2, *AmS*+1, *AmS*) = *M*(*Am*2+1, *Am*2) + *M*(*Am*1+1, *Am*1) — в скобках указаны символические имена адресов памяти байтов слагаемых и суммы. *Решение*:

```
LHLD  Am2      ; H ← M(Am2+1), L ← M(Am2) — два байта второго слагаемого
XCHG           ; D ← M(Am2+1), E ← M(Am2)
LXI   H, AmS   ; HL ← AmS — адрес младшего байта суммы
LDA   Am1      ; A ← M(Am1) — младший байт первого слагаемого
ADD   E        ; A ← M(Am2) + M(Am1) — сумма младших байтов слагаемых
DAA            ; Десятичная коррекция суммы младших байтов в аккумуляторе
MOV   M, A     ; M(AmS) ← M(Am2) + M(Am1) (десятичная сумма младших байтов)
LDA   Am1+1    ; A ← M(Am1+1) — старший байт первого слагаемого
ADC   D        ; A ← M(Am2+1) + M(Am1+1) + CY — сумма старших байтов слагаемых
DAA            ; Десятичная коррекция суммы старших байтов в аккумуляторе
INX   H        ; HL ← AmS + 1
MOV   M, A     ; M(AmS+1) ← M(Am2+1) + M(Am1+1) + CY
MVI   A, 0     ; A ← 0
ADC   A        ; A ← CY
INX   H        ; HL ← AmS + 2
MOV   M, A     ; M(AmS+2) ← CY
```

На языке ассемблера можно использовать символические имена операндов, конкретные значения которых задаются директивами ассемблера (см. § 1.8).

Дополнительный код. Для выполнения сложных арифметических вычислений в МП-системах наиболее часто используется дополнительный код чисел (*Two's Complement*). Если число *Y'* может иметь любой знак, то его дополнительный код

$$[Y']_{Д} = \begin{cases} 0Y, & \text{если } Y' \geq 0, \\ 1W, & \text{если } Y' < 0, \end{cases}$$

где $\overline{Y} = |\,Y'\,| = y_{n-1} \ldots y_1 y_0$ — модуль числа Y', $W = \overline{\overline{Y} + 1} = 2^n - Y$ — дополнение числа Y до 2^n, $\overline{Y} = \overline{y}_{n-1} \ldots \overline{y}_1 \overline{y}_0$, $W = w_{n-1} \ldots w_1 w_0$. Старший разряд дополнительного кода задает знак числа: 0 — число положительное, 1 — число отрицательное. Таким образом, дополнительный код положительного числа равен самому числу (добавление в старший разряд цифры 0 не изменяет числа), а дополнительный код отрицательного числа $[-Y]_д = 1W$, где $W = \overline{Y} + 1$. По дополнительному коду $1W$ отрицательного числа $-Y$ легко найти его модуль: $Y = \overline{W} + 1$.

Примеры десятичных чисел, представимых одним байтом в дополнительном коде:

$$
\begin{array}{ll}
[+0]_д = 0000\ 0000, & [-0]_д = 0000\ 0000, \\
[+1]_д = 0000\ 0001, & [-1]_д = 1111\ 1111, \\
[+2]_д = 0000\ 0010, & [-2]_д = 1111\ 1110, \\
\quad\therefore & \quad\therefore \\
[+126]_д = 0111\ 1110, & [-127]_д = 1000\ 0001, \\
[+127]_д = 0111\ 1111, & [-128]_д = 1000\ 0000.
\end{array}
$$

Использование дополнительного кода для арифметических вычислений основывается на *теореме*: дополнительный код арифметической суммы S' двух двоичных чисел X' и Y' любых знаков равен арифметической сумме дополнительных кодов чисел, т. е.

$$[S']_д = [X' + Y']_д = [X']_д + [Y']_д,$$

причем при сложении чисел одинакового знака разрядная сетка не должна переполняться. Если числа $X' > 0$, $Y' > 0$ и $|\,X'\,| + |\,Y'\,| \geq 2^n$, то происходит потеря значения $+2^n$ и изменение знака остатка суммы на противоположный (переполнение разрядной сетки). Если же $X' < 0$ и $Y' < 0$, то потеря значения -2^n и изменение знака остатка суммы на противоположный происходит при $|\,X'\,| + |\,Y'\,| \geq 2^n + 1$ (переполнение разрядной сетки).

Материал по кодам, используемым для выполнения арифметических операций сложения и вычитания чисел, подробно изложен в § 6.9 книги [5].

При использовании дополнительного кода для решения практических задач следует учитывать, что диапазоны представимых в дополнительном коде чисел определяются соотношениями:

$-128 \ldots +127$ или $-80h \ldots +7Fh$ для однобайтовых чисел,

$-32768 \ldots +32767$ или $-8000h \ldots +7FFFh$ для двухбайтовых чисел,

$-8388608 \ldots +8388607$ или $-800000h \ldots +7FFFFFh$ для трехбайтовых чисел,

$-2147483648 \ldots +2147483647$ или $-80000000h \ldots +7FFFFFFFh$ для четырехбайтовых чисел,

$-2^{n-1} \ldots +(2^{n-1} - 1)$ для n-разрядного дополнительного кода (старший разряд — знак числа).

Диапазоны представимых чисел очень легко запоминаются и записываются в 16-ричной системе счисления.

Выше при определении дополнительного кода была использована логическая операция отрицания. Однако, на основании выражения $W = 2^n - Y$ дополнительный код отрицательного числа Y можно получить и с помощью команд SUB и SBB (SUI и SBI), выполняющих вычисление значения $0 - Y = [-Y]_д$, где число Y может быть представлено любым числом байтов. Эти команды автоматически выполняют заем из числа 2^8 при выполнении байтовой операции вычитания и из числа $2^{8 \times m}$ при выполнении m-байтовой операции вычитания. Результат последнего заема (значение флага переноса/заема *CY*), выходящий за рамки разрядной сетки, игнорируется. Понятно, что такой способ вычисления дополнительного кода предпочтительнее, чем его вычисление с привлечением логической операции отрицания. Аналогично вычисляется и модуль числа: $Y = 0 - [-Y]_д$.

Задача 10. Найти дополнительный код отрицательного числа $-7EF9h > -8000h$ двумя рассмотренными выше способами (преобразуемое число представимо двумя байтами). *Решение*:

```
; Вычисление дополнительного кода с использованием логической операции отрицания
        MVI     A, 0F9h     ; A ← F9h
        CMA                 ; A ← Ā = 06h
        MOV     E, A        ; E ← 06h
        MVI     A, 7Eh      ; A ← 7Eh
        CMA                 ; A ← Ā = 81h
        MOV     D, A        ; D ← 81h, DE = 8106h
        INX     D           ; DE ← 8106h + 1 = 8107h = [–7EF9]д
; Вычисление дополнительного кода с использованием арифметической операции вычитания
        SUB     A           ; A ← A – A = 0
        SUI     0F9h        ; A ← 0 – F9h = 07h, CY ← 1
        MOV     E, A        ; E ← 07h
        MVI     A, 0        ; A ← 0
        SBI     7Eh         ; A ← 0 – 7Eh – CY = 81h
        MOV     D, A        ; D ← 81h, DE = [–7EF9]д = 0 – 7EF9h = 8107h
```

Типы чисел. Из вышеизложенного следует, что в МП 8080*A*/8085*A* можно использовать три типа представления исходных чисел, над которыми производятся арифметические операции:

1) целые числа без знака — все разряды байта и слова определяют величину числа; одним байтом представимы десятичные числа из диапазона 0 … 255 (00*h* … FF*h*) и двумя байтами (словом) — из диапазона 0 … 65535 (0000*h* … FFFF*h*);

2) целые числа со знаком — числа представлены в дополнительном коде (старший разряд байта и слова задает знак числа); одним байтом представимы десятичные числа из диапазона –128 … +127 и двумя байтами (словом) — из диапазона –32768 … +32767 (–8000*h* … 7FFF*h*);

3) упакованные *BCD*-числа (*Binary Coded Decimal* — код 8–4–2–1) — в одном байте содержится двухразрядное десятичное число от 00 до 99*d* (для каждой десятичной цифры используется одна тетрада с весами разрядов 8–4–2–1).

Независимо от формата представления чисел *ALU* при выполнении команд сложения и вычитания производит над ними операции как над двоичными числами. Поэтому для получения правильного результата при использовании упакованных *BCD*-чисел нужно выполнять команду десятичной коррекции DAA, а при использовании двоичных чисел необходимо следить за правильной их интерпретацией.

Команды условных передач управления JP *addr*, JM *addr* (команды условных переходов), CP *addr*, CM *addr* (команды условных вызовов подпрограмм), RP и RM (команды условных возвратов из подпрограмм) имеет смысл использовать только в том случае, если флаг *S* (знак) фиксирует результат выполнения операций над операндами, представленными в дополнительном коде. Неправильное использование флага *S* может привести к непредсказуемым результатам.

При вычислении разности $S' = X - Y$ двух положительных m-байтовых чисел X и Y с помощью команд вычитания разность S' получается в дополнительном коде $[S']_д = [X - Y]_д$, если число $X \le +(2^{n-1} - 1)$ и разность $-2^{n-1} \le X - Y$ ($n = 8 \times m$). Из этого следует, что допустимы значения числа $Y \le X + 2^{n-1}$, хотя для представления отрицательного числа $-Y$ в дополнительном коде требуется выполнение более жесткого условия: $Y \le 2^{n-1}$. Так, для двухбайтовых чисел должны выполняться условия

$$X \le +7FFFh \text{ и } Y \le X + 8000h \text{ вместо условий } X \le +7FFFh \text{ и } Y \le +8000h.$$

Задача 11. Вычислить разность $S' = X - Y$ и ее модуль для положительных m-байтовых чисел X и Y в памяти, адресам младших байтов которых присвоены имена *AOper*1 и *AOper*2. Число байт m задано именем *Count*. Разность S' поместить по адресам операнда Y, а модуль — по адресам операнда X (считаем, что условия $X \le +(2^{n-1} - 1)$ и $-2^{n-1} \le X - Y$ соблюдаются). *Решение*:

```
; Вычисление разности двух m-разрядных чисел
      LXI    D, AOper1  ; DE ← AOper1 (адрес младшего байта числа X)
      LXI    H, AOper2  ; HL ← AOper2 (адрес младшего байта числа Y)
      MVI    C, Count   ; C ← Count (число байт в представлении чисел)
      XRA    A          ; CY ← 0
LM1:  LDAX   D          ; A ← M(rp D)
      SBB    M          ; A ← M(rp D) – M(rp H) – CY
      MOV    M, A       ; M(rp H) ← M(rp D) – M(rp H)
      STAX   D          ; M(rp D) ← M(rp D) – M(rp H)
      INX    D          ; Команды INX D, INX H и DCR C не изменяют
      INX    H          ;       значения флага переноса CY
      DCR    C
      JNZ    LM1
      ANI    80h        ; A ← sS××× ×××× & 1000 0000 = sS000 0000, sS — разряд знака
      JZ     LM2        ; Переход, если разность положительная
; Вычисление модуля разности чисел
      LXI    D, AOper1  ; DE ← AOper1 (адрес младшего байта числа X)
      LXI    H, AOper2  ; HL ← AOper2 (адрес младшего байта числа Y)
      MVI    C, Count   ; C ← Count (число байт в представлении чисел)
      XRA    A          ; CY ← 0
LM3:  MVI    A, 0       ; A ← 0
      SBB    M          ; A ← 0 – M(rp H) – CY
      STAX   D          ; M(rp D) ← 0 – M(rp H)
      INX    D          ; Команды INX D, INX H и DCR C не изменяют значения
      INX    H          ;       флага переноса CY
      DCR    C
      JNZ    LM3
LM2:  ∴                  ; Вычислен дополнительный код [S']д = [X – Y]д и модуль | S' |
```

Если, например, числа X = 7D5A*h* $\le$ +7FFF*h* и $Y = X + 8000h$ = FD5A*h*, то дополнительный код разности $[S']_д = [X - Y]_д = 8000h$ и модуль разности $|S'| = 8000h$ — получены правильные результаты. Если же числа X = +7D5A*h* и $Y = X + 8000h + 1$ = FD5B*h* $> X + 8000h$, то для разности S' будет получен результат: S' = 7FFF*h* (число положительное), который не является дополнительным кодом разности. В обоих случаях по окончании вычисления разности устанавливается значение флага переноса $CY = 1$, свидетельствующее о заеме значения 2^{16}. При использовании представления чисел в дополнительном коде (числа со знаком) значение этого разряда игнорируется — это основное преимущество вычислений в дополнительном коде (результаты вычислений никогда не выходят за границы правильно выбранной разрядной сетки, состоящей из целого числа байт).

Если в *задаче* 11 считать, что используются числа без знака, то правильный результат получается при любых значениях чисел X и Y, но при этом флаг переноса CY будет являться расширением разрядной сетки результата. Так, если числа X = +7D5A*h* и Y = FD5B*h*, то разность $S' = 17FFFh = -2^{16} + 7FFFh$. Возникающий при вычитании заем $CY = 1$ с весом 2^{16} имеет отри-

цательный знак. По существу, число 17FFF*h* представляет собою расширенный дополнительный код разности, но такой код невозможно использовать при вычислениях, так как он выходит за рамки разрядной сетки, которая может содержать только целое число байт.

Для демонстрации особенностей использования некоторых команд МП приведем другой, хотя и менее эффективный, способ решения *задачи* 11 (файл 1#06_11@.asm на дискете):

```
; Вычисление разности двух m-разрядных чисел
        LXI   D, AOper1   ; DE ← AOper1 (адрес младшего байта числа X)
        LXI   H, AOper2   ; HL ← AOper2 (адрес младшего байта числа Y)
        MVI   C, Count    ; C ← Count (число байт в представлении чисел)
        XRA   A           ; CY ← 0
        PUSH  PSW         ; Включение в стек флага переноса CY
LM1:    POP   PSW         ; Извлечение из стека флага переноса CY
        LDAX  D           ; A ← M(rp D)
        SBB   M           ; A ← M(rp D) – M(rp H) – CY
        PUSH  PSW         ; Включение в стек флага переноса CY и флага знака S
        MOV   M, A        ; M(rp H) ← M(rp D) – M(rp H)
        STAX  D           ; M(rp D) ← M(rp D) – M(rp H)
        INX   D
        INX   H
        DCR   C
        JNZ   LM1
        POP   PSW         ; Извлечение из стека флага знака S (знака разности чисел)
        JP    LM2         ; Переход, если разность положительная
; Вычисление модуля разности чисел
        LXI   D, AOper1   ; DE ← AOper1 (адрес младшего байта числа X)
        LXI   H, AOper2   ; HL ← AOper2 (адрес младшего байта числа Y)
        MVI   C, Count    ; C ← Count (число байт в представлении чисел)
        XRA   A           ; CY ← 0
LM3:    MVI   A, 0        ; A ← 0
        SBB   M           ; A ← 0 – M(rp H) – CY
        STAX  D           ; M(rp D) ← 0 – M(rp H)
        INX   D           ; Команды INX  D, INX  H и DCR  C не изменяют
        INX   H           ;       значения флага переноса CY
        DCR   C
        JNZ   LM3
LM2:    ...               ; Вычислен дополнительный код [S']д = [X – Y]д и модуль | S' |
```

В данной программе первая команда PUSH PSW и последняя команда POP PSW выполняются по одному разу, а команды POP PSW и PUSH PSW, находящиеся в цикле, по $m = count$ раз. Таким образом, число включений в стек равно числу извлечений из стека, что и требуется для его нормальной работы. Нарушение баланса между числом включений в стек и числом извлечений из стека является трудно обнаружимой ошибкой (особенно при использовании вложенных циклов). В этой программе можно использовать команду условного перехода JP по знаку разности чисел $S' = X - Y$, так как предполагается, что ее значения не выходят за границы допустимых чисел, представимых дополнительным кодом.

Последняя программа содержит на три команды больше, чем предыдущая и выполняется медленнее из-за стековых команд PUSH PSW и POP PSW, требующих по два обращения к памяти.

Переполнение разрядной сетки. При арифметических вычислениях с использованием дополнительного кода возникает проблема переполнения разрядной сетки при сложении чисел, имеющих одинаковый знак. Установить программным способом появление переполнения можно с помощью анализа знаковых разрядов исходных чисел и суммы.

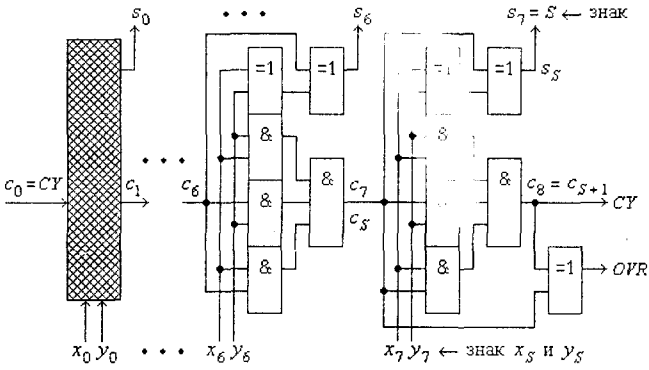


Рис. 1.17. Схема сумматора с обнаружением переполнения

Из приведенной выше теоремы для дополнительного кода следует, что переполнение описывается переключательной функцией

$$OVR = s_S \cdot \bar{x}_S \cdot \bar{y}_S \vee \bar{s}_S \cdot x_S \cdot y_S,$$

которую и необходимо реализовать программным способом (*OVR* — *Overflow* — переполнение; x_S, y_S и s_S — знаковые разряды чисел *X'*, *Y'* и суммы *S'*).

Аппаратная реализация этой функции показана на рис. 1.17 на примере простейшего сумматора с последовательным переносом [5]:

$$OVR = c_S \oplus c_{S+1}.$$

Используя соотношения для функций $s_S = x_S \oplus y_S \oplus c_S$ и $c_{S+1} = x_S \cdot y_S \vee x_S \cdot c_S \vee y_S \cdot c_S$ (см. рис. 1.17), легко показать эквивалентность этих двух представлений функции переполнения. Функция *OVR* реализована аппаратно на кристаллах МП, только начиная с МП 8086 (флаг *OF — Overflow Flag*).

Задача 12. Вычислить сумму *m*-байтовых чисел *X'* и *Y'* любого знака, представленных в дополнительном коде, с обнаружением переполнения разрядной сетки. Результат поместить по адресам операнда *Y*. Число байт *m*, используемых для представления чисел, и адреса младших байтов дополнительных кодов чисел *X'* и *Y'* имеют символические имена *count*, *oper*1 и *oper*2 соответственно. *Решение*:

```
; Анализ возможности переполнения
      LXI   D, oper1 + count – 1  ; DE ← адрес старшего байта числа X'
      LXI   H, oper2 + count – 1  ; HL ← адрес старшего байта числа Y'
      LDAX  D                     ; A ← M(rp D) = xS××× ××××, где xS — знаковый разряд числа X'
      ANI   80h                   ; A ← xS××× ×××× & 1000 0000 = xS000 0000
      MOV   B, A                  ; B ← xS000 0000, xS — знак числа X'
      XRA   M                     ; A ← xS000 0000 ⊕ yS××× ×××× = (xS ⊕ yS)××× ×××× = G××× ××××
      ANI   80h                   ; A ← G××× ×××× & 1000 0000 = G000 0000, G = xS ⊕ yS = 0 при xS = yS
      RLC                         ; A ← 0000 000G  (G = 0 — возможно переполнение)
      ORA   B                     ; A ← 0000 000G ∨ xS000 0000 = xS000 000G
      MOV   B, A                  ; B ← xS000 000G, xS — знак числа X'
; Вычисление суммы дополнительных кодов двух m-разрядных чисел
      LXI   D, oper1              ; DE ← адрес младшего байта числа X
      LXI   H, oper2              ; HL ← адрес младшего байта числа Y
      MVI   C, count              ; C ← число байт в представлении чисел
      XRA   A                     ; CY ← 0
L1:   LDAX  D                     ; A ← M(rp D)
      ADC   M                     ; A ← M(rp D) + M(rp H)
```

```
        MOV   M, A      ; M(rp H) ← M(rp D) + M(rp H)
        INX   D
        INX   H
        DCR   C
        JNZ   L1
; Анализ на предмет наличия или отсутствия переполнения и принятие решения
        ANI   80h       ; A ← s_S××× ×××× & 1000 0000 = s_S000 0000
        XRA   B         ; A ← s_S000 0000 ⊕ x_S000 000G = (s_S ⊕ x_S)000 000G
        CPI   80h
        JNZ   L2        ; Переход, если нет переполнения
          ∴             ; Принятие решения: например, вывод сообщения о переполнении и
          ∴             ;       необходимости увеличения разрядной сетки чисел
L2:       ∴             ; Продолжение программы — переполнения не было
```

Обнаружение переполнения полезно использовать на стадии отладки программ.

1.7. Адресация данных и переходов

Микропроцессор в каждом машинном цикле может выдавать на шину адреса только одно из значений 0000 … FFFF*h*, которое адресует один из операндов, находящихся в памяти. Поэтому в двухоперандных командах второй операнд всегда должен находиться во внутренних регистрах МП, адресуемых частью разрядов КОП (см. рис. 1.13). В МП 8080*A*/8085*A* используются четыре режима адресации данных (*непосредственный*, *прямой*, *регистровый* и *косвенно-регистровый*) и два метода адресации переходов (*прямой* и *косвенно-регистровый*).

Непосредственная адресация данных. При такой адресации 8-разрядные (*d*8) или 16-разрядные (*d*16) данные представляются вторым или вторым и третьим байтами команды. Примеры команд:

```
MVI   C, 0E7h   ; C ← E7h
ANI   51d       ; A ← A & 51 (указатель десятичной системы счисления d можно опускать)
ADI   11h       ; A ← A + 11h
LXI   SP, 2000h ; SP ← 2000h — инициализация стека
```

Вместо чисел в программах, написанных на языке ассемблера, можно указывать символические имена непосредственных операндов, значения которых определяются директивами ассемблера.

Прямая адресация данных. В этом случае адрес операнда находится во втором и третьем байтах команды. Эти два байта задают адрес ячейки памяти, в которой находится операнд. *Примеры* команд:

```
LDA   23ABh     ; A ← M(23AB), где адрес памяти addr = 23Abh
STA   0CF3Dh    ; M(CF3D) ← A, где адрес памяти addr = CF3Dh
LHLD  0ED5Ah    ; L ← M(ED5A), H ← M(ED5B)
SHLD  0ED5Ah    ; M(ED5A) ← L, M(ED5B) ← H
```

Формально к методу прямой адресации можно причислить и команды ввода и вывода IN *port* и OUT *port*, выполняющие операции A ← *I*/*O*(*port*) и *I*/*O*(*port*) ← A, так как адрес операнда находится во втором байте команды, но, как правило, внешние устройства не используются для хранения оперативных данных с последующим их использованием в вычислительном процессе (для этого целесообразно использовать оперативную память).

Регистровая адресация данных. При такой адресации в коде команды адресуются регистры или регистровые пары, в которых хранятся операнды (см. табл. 1.3 и 1.4, рис. 1.13). Примеры команд:

```
MOV   E, B    ; E ← B
ADC   C       ; A ← A + C + CY
ORA   L       ; A ← A ∨ L
INR   D       ; D ← D + 1 — инкремент 8-разрядного числа
DCX   D       ; rp D ← rp D – 1 — декремент 16-разрядного числа
LXI   H, d16  ; rp H ← d16, т. е. H ← d16_h, L ← d16_l (h — high, l — low)
```

В двухоперандных командах для каждого операнда может использоваться свой режим адресации данных. Так, в последней команде имеют место регистровая и непосредственная адресации операндов. Достоинство регистровой адресации — команды однобайтовые (меньше места занимают в памяти программ).

Косвенно-регистровая адресация данных. В этом случае адрес операнда, расположенного в памяти *M*, определяется содержимым одной из 16-разрядных регистровых пар *B*, *D* или *H*. Примеры команд:

```
LDAX  B       ; A ← M(rp B), адрес памяти addr_h = B, addr_l = C (h — high, l — low)
STAX  D       ; M(rp D) ← A, адрес памяти addr_h = D, addr_l = E
MOV   M, A    ; M(rp H) ← A, адрес памяти addr_h = H, addr_l = L
INR   M       ; M(rp H) ← M(rp H) + 1
```

В командах LDAX и STAX (команды передачи данных) для адресации памяти можно использовать только регистровые пары *B* и *D*, причем вторым из операндов всегда является аккумулятор *A*. Регистровая пара *H* имеет более универсальное применение — используется в командах передачи данных, командах арифметических и логических операций для косвенно-регистровой адресации операндов, расположенных в памяти и обозначаемых в командах символом M.

Неявная и стековая адресации данных. Описанные выше четыре метода адресации операндов требуют указания в командах операнда или его адреса в явном виде. Неявная и стековая адресация не требуют этого и поэтому не входят в перечень режимов адресации данных — наличие в МП неявной и стековой адресации операндов подразумевается по умолчанию.

Примерами команд с неявной адресацией могут служить команды циклического сдвига (в неявном виде подразумевается, что операндом является содержимое аккумулятора). Это же относится и к таким командам, как CMA, CMC, STC и др. (см. табл. 1.5).

Стековая адресация означает, что содержимое указателя стека *SP* является адресом данных, причем этот адрес в неявном виде подразумевается в кодах операций однобайтовых команд. Оперируют эти команды с двумя байтами данных (словами). В командах PUSH PSW и PUSH *rp* источником операндов являются регистровые пары *A* и *F*, *B* и *C*, *D* и *E*, *H* и *L*, а получателем — ячейки памяти *M*(*SP* – 1) и *M*(*SP* – 2). Декремент указателя стека *SP* при выполнении этих команд производится автоматически. В командах POP PSW и POP *rp* источником операндов являются ячейки памяти *M*(*SP*) и *M*(*SP* + 1), а получателем — пары регистров *A* и *F*, *B* и *C*, *D* и *E*, *H* и *L*. При выполнении этих команд инкремент указателя стека *SP* производится автоматически с установкой окончательного его значения *SP* + 2.

Достоинство стековой адресации: команды однобайтовые с выполнением операции пересылки двух 8-разрядных операндов. Данные команды часто используются для временного сохранения содержимого регистровых пар в памяти при нехватке регистров общего назначения для оперативного хранения операндов в процессе решения задачи. Для инициализации стека

используется команда LXI SP, *d*16. Первой операцией всегда должно быть включение слова в стек и только затем его извлечение. Необходимый объем памяти стека определяется программистом, исходя из решаемых задач.

Прямая адресация переходов. Если адрес перехода содержится в самой команде переходов, то такая адресация называется *прямой*. В командах переходов JMP *addr* и J*cond addr* используется прямая адресация — адрес перехода содержится во втором и третьем байтах команд. При прямой адресации в программах адреса переходов указываются метками.

Косвенно-регистровая адресация переходов. При такой адресации в коде операции команды передачи управления указывается регистровая пара, содержимое которой загружается в программный счетчик *PC* с потерей предыдущего его значения. Имеется только одна команда с *косвенно-регистровой адресацией переходов* PCHL, загружающая содержимое *rp H* в программный счетчик *PC*. Например, переход по адресу 25E8*h* можно выполнить с помощью команд:

```
LXI     H, 25E8h   ; HL ← 25E8h (HL — регистровая пара rp H)
PCHL               ; PC ← HL
```

Команду PCHL удобно использовать для организации системы переходов или вызова подпрограмм по фиксированным адресам, записанным предварительно в таблицу, хранящуюся в памяти, или при вычислении адресов переходов по какому-либо алгоритму.

Задача 1. По любому двоичному коду *m* = 0000 0000 … 1111 1111, поступившему в регистр *E*, выполнить переход по адресу, записанному в 512-байтовой таблице адресов передачи управления, начальному адресу которой присвоено символическое имя *a_tabl*. *Решение*:

```
LXI     H, a_tabl  ; HL ← a_tabl (начальный адрес таблицы адресов передачи управления)
MVI     D, 0       ; D ← 0
DAD     D          ; HL ← HL + DE = a_tabl + m
DAD     D          ; HL ← HL + DE = a_tabl + m × 2
MOV     A, M       ; Чтение адреса передачи управления из таблицы a_tabl
INX     H
MOV     H, M
MOV     L, A       ; HL = addr_m
PCHL               ; PC ← addr_m
  ∴

; Таблица адресов addr_m передачи управления, определенная в сегменте данных:

; Адрес   m    addr_m      Комментарий

; a_tabl   0    A5C7h   a_tabl — символическое имя начального адреса
;   +2     1    2F76h           таблицы адресов передачи управления
;   +4     2    1799h   addr_m — двухбайтовые адреса передачи управления, которые
;   +6     3    EB3Dh       должны быть заданы директивами определения данных
;    ∴     ∴      ∴         (см. § 1.8 и файл 1#07_01.asm на дискете)
```

Например, если содержимое регистра *E* = 02*h*, то переход будет осуществлен по адресу 1799*h*. Описанный метод преобразования двоичного кода *m* в адрес перехода весьма эффективен для вызова подпрограмм обслуживания нескольких внешних устройств по общему запросу прерывания. Для этого в вызванной подпрограмме сначала производится ввод номера *m* внешнего устройства из приоритетного шифратора запросов прерывания [5], а затем вычисление адреса перехода на нужную часть подпрограммы и ее выполнение.

1.8. Директивы ассемблера

Текст исходной программы состоит из операторов ассемблера, каждый из которых занимает отдельную строку этого текста. Различают два типа операторов: *инструкции* и *директивы*. Инструкции при трансляции преобразуются в машинные коды МП. Директивы ассемблера, или *псевдокоманды* языка ассемблера, не транслируются в машинные коды, а только управляют процессом ассемблирования (трансляции). Программа-ассемблер интерпретирует и обрабатывает операторы один за другим, генерируя последовательность из машинных кодов команд МП и байтов данных. Запись инструкции на языке ассемблера состоит из четырех полей в свободном формате: поля метки, поля кода операции (КОП), поля операндов и поля комментария:

[Метка:] КОП [Операнд1[, Операнд2]] [; Комментарий]

(элементы, указанные в квадратных скобках, могут отсутствовать). Метка — это идентификатор, присваиваемый первому байту того оператора, в котором она появляется, а КОП — это мнемоническое обозначение команд МП. Для директив ассемблера эти поля называются полем имени, полем директивы, полем выражения и полем комментария соответственно. Операнды могут описываться выражениями, составленными из чисел и символических имен с помощью специальных операторов языка ассемблера.

Пробелы вводятся произвольно (свободный формат), но минимум один пробел должен быть после кода операции. В двухоперандных командах операнды должны быть разделены запятой, а поле комментария должно начинаться с точки с запятой.

Операнды, представляющие собой числа, должны обязательно начинаться с цифры (например, 0A800*h*), а символические имена операндов — с буквы. Для символических имен нельзя использовать зарезервированные имена, например, имена РОНов, мнемонику команд и директив. Для обозначения систем счисления служат буквы:

B или *b* — двоичная система счисления,

O или *Q* (*o* или *q*) — восьмеричная система счисления,

D или *d* — десятичная система счисления,

H или *h* — 16-ричная система счисления

(по умолчанию принимается десятичная система счисления, т. е. указатель *d* можно опускать). В тексте программы на языке ассемблера можно использовать как прописные, так и строчные буквы — ассемблер их не различает.

***ASCII*-коды.** Текст исходной программы на языке ассемблера подготавливается с помощью обычного текстового редактора, имеющегося в дисковой операционной системе *MS-DOS* (*Microsoft Disk Operating System*) для *PC IBM*. Все символы такого текста представлены в *ASCII*-кодах (*American Standard Code for Information Interchange* — американский стандартный код для информационного обмена), используемых для семиразрядного кодирования символов (табл. 1.9). Стандарт *ASCII* утвержден *ISO* (*International Standards Organization*) — международной организацией по стандартизации, основанной в 1946 г и включающей более 70 национальных организаций по стандартизации в области телекоммуникаций. Восьмой бит символа предназначен для контроля четности. В компьютерах используется расширенный (8-разрядный) код *ASCII*, в котором первые 128 кодов совпадают со стандартным кодом *ASCII*, а следующие 128 комбинаций используются для кодирования национальных алфавитов, спецсимволов и псевдографики.

Пробел имеет *ASCII*-код 20*h*, а управляющим символам, невидимым в текстовом редакторе, приписаны *ASCII*-коды 00*h* ÷ 1F*h*. Например, в текстах постоянно используются *ASCII*-коды 0D*h* = 13*d* — возврат каретки (*Carriage Return*) и 0A*h* = 10*d* — перевод строки (*Line Feed*). При выводе текста на дисплей эти *ASCII*-коды используются в подпрограммах, выполняющих сдвиги курсора в левую позицию экрана и вниз на одну позицию соответственно.

Таблица 1.9. **Таблица символов *ASCII***

D H	*S*	*D H*	*S*	*D H*	*S*	*D H*	*S*	*D H*	*S*	*D H*	*S*	*D H*	*S*	*D H*	*S*
000 00	*Управляющие символы*	016 10	*Управляющие символы*	032 20		048 30	0	064 40	@	080 50	P	096 60	`	112 70	p
001 01		017 11		033 21	!	049 31	1	065 41	A	081 51	Q	097 61	a	113 71	q
002 02		018 12		034 22	"	050 32	2	066 42	B	082 52	R	098 62	b	114 72	r
003 03		019 13		035 23	#	051 33	3	067 43	C	083 53	S	099 63	c	115 73	s
004 04		020 14		036 24	$	052 34	4	068 44	D	084 54	T	100 64	d	116 74	t
005 05		021 15		037 25	%	053 35	5	069 45	E	085 55	U	101 65	e	117 75	u
006 06		022 16		038 26	&	054 36	6	070 46	F	086 56	V	102 66	f	118 76	v
007 07		023 17		039 27	'	055 37	7	071 47	G	087 57	W	103 67	g	119 77	w
008 08		024 18		040 28	(	056 38	8	072 48	H	088 58	X	104 68	h	120 78	x
009 09		025 19		041 29	)	057 39	9	073 49	I	089 59	Y	105 69	i	121 79	y
010 0A		026 1A		042 2A	*	058 3A	:	074 4A	J	090 5A	Z	106 6A	j	122 7A	z
011 0B		027 1B		043 2B	+	059 3B	;	075 4B	K	091 5B	[	107 6B	k	123 7B	{
012 0C		028 1C		044 2C	,	060 3C	<	076 4C	L	092 5C	\	108 6C	l	124 7C	|
013 0D		029 1D		045 2D	-	061 3D	=	077 4D	M	093 5D	]	109 6D	m	125 7D	}
014 0E		030 1E		046 2E	.	062 3E	>	078 4E	N	094 5E	^	110 6E	n	126 7E	~
015 0F		031 1F		047 2F	/	063 3F	?	079 4F	O	095 5F	_	111 6F	o	127 7F	

Примечание: *D — Decimal* (десятичное число), *H — Hexadecimal* (16-ричное число), *S — Symbol* (символ).

Управляющим символам также приписаны стандартные образы, используемые при необходимости для их визуализации. Например, *ASCII*-кодам 0D*h* и 0A*h* соответствуют образы ♪ и ◙ соответственно. Цифры # = 0 … 9 имеют *ASCII*-коды 30*h* … 39*h*, т. е. их *ASCII*-код = # + 30*h*, что и используется для их взаимного преобразования программным способом.

В компьютерах *ASCII*-коды используются для передачи данных по каналам связи, при вводе данных с клавиатуры и выводе текста на дисплей, т. е. для информационного обмена между МП и внешними устройствами. Следует четко представлять себе различие между двоичными кодами, которыми должна быть представлена разрабатываемая программа, чтобы она могла быть выполнена МП-системой (микроконтроллером, персональным компьютером), и соответствующими им *ASCII*-кодами (табл. 1.10).

Таблица 1.10. **Коды команд**

Мнемоника команд	Двоичный код команд	*ASCII*-коды команд
RAL	00010111 = 17*h*	31*h* 37*h* = 00110001 00110111
HLT	01110110 = 76*h*	37*h* 36*h* = 00110111 00110110
RET	11000001 = C1*h*	43*h* 31*h* = 01000011 00110001

Представление информации в *ASCII*-кодах используется в текстовых документах, при передаче ее по каналам связи и др. 16-ричное представление двоичных кодов их не изменяет и используется исключительно для удобства восприятия двоичных кодов человеком.

Ассемблирование, компоновка и отладка программ. Разработчиками программного обеспечения на языке ассемблера введена следующая терминология:

ассемблер (*assembler*) — *а*) программа или техническое средство, выполняющее ассемблирование; б) система программирования, включающая язык ассемблера и транслятор с этого языка;

двухпроходной ассемблер (*two-pass assembler*) — ассемблер, выполняющий трансляцию исходной программы, написанной на языке ассемблера и называемой *исходным модулем*, за два прохода: при первом проходе формируется таблица соответствия символов, при втором — выполняется собственно трансляция с языка ассемблера в машинные коды МП;

ассемблирование (*assemblage*) — *компиляция программы* с языка ассемблера; подготовка программы на машинном языке путем замены символических имен операций на машинные коды, а символических адресов — на абсолютные или относительные адреса, а также включение библиотечных программ и генерация последовательностей команд путем указания конкретных параметров в макрокомандах;

компиляция (*compilation*) — *трансляция программы* на язык, близкий к машинному; трансляция программы, составленной на исходном языке, в *объектный модуль*;

компоновка (*linking*) — процесс построения *загрузочного модуля* из объектных модулей, полученных в результате раздельной трансляции соответствующих исходных модулей;

компоновщик (*linker*) — программа, выполняющая компоновку единой программы из независимо транслированных программ;

макроассемблер (*macro assembler*) — транслятор с языка ассемблера, включающий средства определения и использования макрокоманд;

отладчик (*debugger*) — программа, предназначенная для анализа поведения другой программы, обеспечивающая ее трассировку, останов в указанных точках или при выполнении указанных условий, просмотр и изменение ячеек памяти, регистров МП и команд программы;

моделирующий отладчик (*simulation debugger*) — программа, предназначенная для отладки программ некоторой ЭВМ на другой ЭВМ с использованием модели последней.

Файлу исходного модуля присваивается имя *name*.asm (*name* — имя, составленное не более чем из восьми допустимых символов). Для разработки программного обеспечения микроконтроллеров, построенных на основе МП 8080/8085 фирмы *Intel* и Z80 фирмы *Zilog*, можно использовать, например, программный пакет *AVSIM*85 фирмы *Avocet Systems, Inc.* В этот пакет входят, в частности, программы:

*avmac*85.exe (*Macro Assembler*) — ассемблер, транслирующий исходный модуль *name*.asm в машинные коды и генерирующий файлы *name*.obj, *name*.prn и *name*.xrf. Файл *name*.obj (объектный модуль) используется компоновщиком *avlink*.exe для создания загрузочного модуля *name*.hex для моделирующего отладчика. Файл *name*.prn — листинг результата трансляции с указанием всех обнаруженных ошибок, помогающий исправить текст исходного модуля (листинг содержит и текст исходной программы). Файл *name*.xrf содержит карту перекрестных ссылок;

avlink.exe (*Linker*) — компоновщик, создающий из одного или нескольких объектных модулей загрузочный модуль *name*.hex и генерирующий файлы *name*.map и *name*.sym. Модуль *name*.hex используется моделирующим отладчиком *avsim*85.exe для его выполнения на персональном компьютере *IBM PC*. Файл *name*.map содержит карту распределения памяти, используемой создаваемой программой. Файл *name*.sym содержит описание используемых в программе сегментов (начальных и конечных адресов сегмента программного кода, сегмента данных, сегмента стека и сегмента внешних устройств). Модуль *name*.hex, кроме того, необходим

для генерации командой *load*.com исполняемого модуля *name*.com при использовании дисковой операционной системы *CP/M* (*Control Program for Microcomputers* — управляющая программа для микрокомпьютеров, построенных на основе МП 8080/8085/Z80) фирмы *Digital Research*, созданной *Г. Килдэлом* в 1976 г.;

*avsim*85.exe (*Simulation Debugger*) — моделирующий отладчик, позволяющий выполнять программу на персональных компьютерах типа *IBM PC* частями и в пошаговом режиме для обнаружения неуловимых другими средствами ошибок в разрабатываемой программе (доступно для анализа — выводится на экран — содержимое всех регистров, флагов и используемой памяти);

hexform.exe — преобразователь загрузочного модуля *name*.hex, выполнимого на персональном компьютере *IBM PC* только с помощью моделирующего отладчика *avsim*85.exe, в двоичный модуль *name*.bin, непосредственно используемый для программирования *ROM* микроконтроллеров, построенных на основе МП 8080/8085 фирмы *Intel*.

Объектный модуль *name*.obj генерируется при запуске из командной строки компьютера команды

*avmac*85[.exe] *name*[.asm] [*xr*] [*SM* [*SC*] [*SI*]]

(квадратными скобками отмечены необязательные элементы). При этом создаются также файлы *name*.prn и *name*.xrf, но при отсутствии параметра *xr* файл *name*.xrf не создается. Назначение параметров *SM* (*ShowMacs*), *SC* (*ShowComments*) и *SI* (*ShowIncs*) будет рассмотрено ниже.

Загрузочный модуль *name*.hex генерируется при запуске из командной строки компьютера команды

avlink[.exe] *name*[.hex] = *name*[.obj] [*–sy*]

(создаются также файлы *name*.map и *name*.sym, но при отсутствии параметра *sy* файл *name*.sym не генерируется, и отладчик будет работать неправильно). В файле *name*.hex информация представлена в *ASCII*-кодах. Компоновщик может объединять несколько объектных файлов, например:

avlink.exe *name*.hex = *name*1.obj *name*2.obj *name*3.obj *–sy*

(загрузочный модуль *name*.hex генерируется из трех объектных файлов).

Указание: для генерируемого файла (здесь файл *name*.hex) и файла-источника (здесь файл *name*.obj) всегда можно использовать разные имена *name*, но в дальнейшем, если это возможно, для упрощения конструкций команд будут использоваться одинаковые имена. При использовании разных имен со временем можно также забыть, каким файлом *name*.asm они порождены. В последней команде в качестве имени *name* предпочтительнее взять имя одного из трех файлов, являющегося главным файлом.

На рис. 1.18 показано содержимое загрузочного модуля 1#08_03.hex (1 — номер главы, 08 — номер параграфа, 03 — номер задачи; все файлы с именами ×#××_××.asm находятся на прилагаемой к учебному пособию дискете). Этот файл получен как результат решения *задачи* 3 (см. ниже). Подчеркнутым шрифтом выделен программный код (два сегмента, занимающих 30 и 10 байт памяти), полужирным шрифтом — адреса сегментов, полужирным подчеркнутым шрифтом — адрес старта, указываемый в строке директивы END (см. табл. 1.11).

:1E**003C**00F53A0008E680F640303A0008E68E3A00083717E25400E6FE320008F1FBC914
:01**1800**005F88
:0A**0100**03100103E0B30FBC3070165
:00**0100**01FE

1. Рамкой обведены данные, используемые программой.
2. Адрес старта необходим только для генерации исполняемого модуля в операционной системе *CP/M*.

Рис. 1.18. Загрузочный модуль 1#08_03.hex для моделирующего отладчика

Для запуска отладчика из командной строки компьютера необходимо выполнить команду

*avsim*85[.exe] *par*1*FLname*.cmd [–*par*2]

(но перед этим должен быть создан файл *name*.hex, который, собственно и использует отладчик для прогона программы). Параметр *par*1 = *a*, *b*, *c* или *d* задает конфигурацию МП-системы: *a* — МП 8085 или 8080, *b* — 8085 + 8155, *c* — 8085 + 8355, *d* — 8085 + 8155 + 8355 (БИС 8155 и 8355 описаны в § 3.9), а параметр *FL* (*File Load*) — загрузку командного файла *name*.cmd автоматической настройки экрана отладчика. Командный файл *name*.cmd — текстовый файл параметров отладчика, подготавливаемый пользователем. Параметр *par*2 = *d*0, *d*1, *c*0 или *c*1 настраивает отладчик на используемый тип дисплея (если этот параметр не задан, то устанавливается черно-белый режим работы дисплея).

Например, если *name*.cmd = *ivanov*.cmd, *par*1 = *a* и *par*2 = c1, то следует выполнить команду

*avsim*85.exe *a*fl*ivanov*.cmd –*c*1

(словосочетание *par*1*FLname* пишется слитно). Пример простейшего командного файла *ivanov*.cmd для автоматической настройки отладчика:

```
LAivanov
D1A800h
←0100
```

(назначение подобных команд будет рассмотрено в подразделе **Отладчик фирмы *Avocet Systems, Inc.*** — стр. 92).

Для преобразования загрузочного модуля *name*.hex в двоичный модуль *name*.bin, который можно использовать для программирования *ROM* микроконтроллеров, необходимо выполнить команду

hexform[.exe] *name*[.bin] = *name*[.hex] [–*range*(*addr*1, *addr*2)] [–*fill*(*value*)]

(по умолчанию генерируется файл *name*.bin). Начальный и конечный адреса программы или ее некоторой части, для которой создается файл *name*.bin, задает параметр *range*(*addr*1, *addr*2) — диапазон, область. Параметр *fill*(*value*) указывает, каким числом *value* = 00*h* ... FF*h* (или 0 ... 255) нужно заполнить неиспользованные программой участки памяти в диапазоне адресов *addr*1 ... *addr*2. По умолчанию (если параметры не заданы) заполнение производится числом 00*h*, а диапазон равен 0000*h* ... *addr*, где *addr* — максимальный адрес ячейки памяти, используемой программой под программный код или данные.

При проектировании микроконтроллеров широко применяются *EPROM* (*Erasable Programmable ROM*), или *UV-EPROM* (*Ultra-Violet EPROM*), фирмы *Intel* — стираемые ультрафиолетовыми лучами многократно программируемые ПЗУ. В исходном состоянии сигналы на всех выходах *EPROM* равны 1 и должны программироваться только нулевые значения разрядов данных. Поэтому заполнение неиспользуемых частей памяти следует производить числами FF*h* = 1111 1111. Это позволяет, если потребуется в дальнейшем, запрограммировать в свободные ячейки памяти добавочное программное обеспечение.

Если задать значения *addr*1 = 003C*h*, *addr*2 = 010B*h* и *value* = FF*h*, то при запуске из командной строки команды

hexform.exe 1#08_03.bin = 1#08_03.hex –*range*(3C*h*, 10B*h*) –*fill*(FF*h*)

будет получен файл 1#08_03.bin, содержимое которого показано на рис. 1.19. Если же параметры не задать, то файл 1#08_03.bin будет содержать 2049 байт по адресам 0000*h* ... 0800*h* с заполненными числом 00*h* неиспользуемыми участками памяти (по адресу 0800*h* в программе расположен байт данных).

```
|← 003Ch = addr1                Файл 1#08_03.bin                        0059h →|
|F5 3A 00 08 E6 80 F6 40 30 3A 00 08 E6 8E 3A 00 08 37 17 E2 54 00 E6 FE 32 00 08 F1 FB C9 FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF 31 00 10 3E 0B 30 FB C3 07 01 FF FF|
      |← 0100h              0109h →|       |← 010Ch = addr2
```

Рис. 1.19. Модуль 1#08_03.bin для программирования *EPROM* фирмы *Intel*

Так как файл 1#08_03.bin двоичный, то в текстовом редакторе его содержимое имеет вид:

```
                        ⌡: ◙цА÷@0: ◙цО: ◙7↕тТ ц▄2 ◙ё√╓
1 ►>♂0√├•☺
```

(большинство кодов невидимы, например, 1111 1111 = FF*h*). Содержимое файла 1#08_03.bin, представленное на рис. 1.19, получено переводом его содержимого в *ASCII*-коды специальной программой.

Директивы ассемблера. Ассемблеры разных фирм имеют много общего — их можно считать диалектами одного языка ассемблера. Поэтому, однажды изучив ассемблер одной фирмы, легко перейти к использованию ассемблера другой фирмы и даже ассемблера для другого типа МП. В табл. 1.11 приведены основные директивы ассемблера фирмы *Avocet Systems, Inc.* Освоения этих директив уже достаточно для разработки простых программ.

В ассемблере фирмы *Avocet Systems, Inc.* можно использовать и не стандартные указатели систем счисления: $ — 16-ричная, @ — восьмеричная и % — двоичная системы счисления. Эти символы указываются перед числами, например,

$0D7 = 0D7*h*, @47 = 47*q*, %1011 = 1011*b*.

Таблица 1.11. **Основные директивы ассемблера версии 2.02 фирмы *Avocet Systems, Inc.***

Метка Имя	*КОП* Директива	*Операнды* Выражение	*Комментарий ← Формат инструкций* *Комментарий ← Формат псевдокоманд*
[*Label*1[:]] [1]	org	100*h*	; *Origin* — начало (*Label*1 = 100*h*)
*Name*1[:]	equ	0A800*h*	; *Equate* — приравнять (*Name*1 = A800*h*)
*Name*2[:]	equ	(40*h*∗3 – 40*h*)/2	; *Name*2 = 40*h* (в выражении использованы операторы: ∗, –, /)
[*Label*2[:]]	db	58, 0E2*h*, 99	; *Define Byte* — определить байт (запись в память трех байт)
[*Label*3[:]]	db	'Novoselzeva'	; В память записывается 11 символов в кодах *ASCII*: 4E … 61
[*Label*4[:]]	dw	1998*h*, 0D4C1*h*	; *Define Word* — определить слово. В четыре ячейки памяти ; байты записываются в последовательности: 98 19 C1 D4
[*Label*5[:]]	ds	*k*	; *Define Space* — определить (зарезервировать) *k* байт памяти
	defseg	*N_seg*[, *Attribute*] [2]	; *Define Segment* — определить сегмент
	seg	*N_seg*	; Открытие сегмента *N_seg* после директивы defseg. Сегмент ; закрывается автоматически при открытии другого сегмента
	end	[*AddrStart*]	; Конец программы

Примечание: [1] квадратными скобками помечены необязательные элементы (если метка *Label* имеется, то ей присваивается указанный в директиве или вычисленный транслятором *адрес*, а не *значение* первого *операнда*, определенного в этой директиве); [2] наиболее часто используются атрибуты (*Attribute*) START = *Address*, CLASS = *Code*, *Data* или *IOspace*.

В дальнейшем для улучшения читаемости программ при написании их текстов будем придерживаться следующих правил:

команды МП и метки для команд переходов записываются только прямыми прописными буквами,

директивы ассемблера — только прямыми строчными буквами,

символические имена констант, переменных и сегментов — с использованием курсивных и прописных, и строчных букв, причем первая буква должна быть прописной.

Далее будут описаны не все, о только наиболее часто используемые директивы, которых достаточно для разработки программ с приемлемой эффективностью. Все описанные здесь директивы и операторы поясняются примерами их использования. Порядок описания директив будет определяться как методическими соображениями, так и последовательностью появления директив в примерах программ (указатель директив см. в табл. 4.22, с. 441).

Директивы сегментирования памяти. С помощью этих директив производится распределение памяти для различных блоков программы (данных, стека, устройств ввода-вывода и кода — команд МП).

Директивы определения DEFSEG (*Define Segment*) и *открытия* SEG (*Segment*) *сегментов* предназначены для размещения данных, стека и программного кода по определенным адресам памяти. Эти директивы имеют форматы:

DEFSEG *Name_seg* [, *Attribute*]
SEG *Name_seg*

(атрибуты могут отсутствовать). При определении сегмента директивой DEFSEG и его открытии директивой SEG должно использоваться одно и то же имя сегмента *Name_seg*. Закрытие сегмента происходит автоматически при открытии другого сегмента. Определяется сегмент директивой DEFSEG только один раз, а открываться директивой SEG может любое число раз в разных местах программы (внутри других сегментов). При трансляции все части сегмента (сегменты, имеющие одно и то же имя) объединяются в непрерывную область данных в последовательности их появления в программе.

Наиболее важными и часто используемыми являются два атрибута START и CLASS, задающие начальный адрес сегмента (START = *address*, где *address* — число или символическое имя числа) и принадлежность к определенной группе сегментов (CLASS = *Code* — программный код, CLASS = *Data* — данные, CLASS = *IOspace* — внешние устройства). *Пример*:

DEFSEG *Ivanov*, START = 200*h*, CLASS = *Code*

(первый байт программного кода будет расположен по адресу 200*h*).

Имена сегментов *Name_seg* = *Code*, *Data* и *IOspace* в ассемблере зарезервированы для открытия сегментов директивой SEG с нулевого адреса (постоянных сегментов) без предварительного их определения директивой DEFSEG. Начальные адреса этих сегментов (0000*h* — для памяти и 00*h* — для внешних устройств) не могут быть изменены, так как сегменты не определяются директивой DEFSEG. Например, директивой SEG *IOspace* открывается сегмент, первому порту ввода-вывода которого будет присвоен адрес 00*h*. Хотя и используются одинаковые имена *Code*, *Data* и *IOspace* для сегментов и классов, ассемблер различает их по контексту.

Компоновщик объединяет сегменты *Code* и *Data* в *непрерывную область*, располагая сегмент *Data* после сегмента *Code* независимо от их взаимного расположения в тексте исходной программы, причем сегмент *Code* будет начинаться с адреса 0000*h*.

Директива END задает конец программы — все, что написано после директивы END, транслятором игнорируется. Эта директива имеет формат

END [*AddrStart*]

(параметр *AddrStart* может быть меткой или числом, задающим абсолютный адрес — записывается в файл *name*.hex).

Адрес старта необходим только для генерации исполняемого модуля *name*.com в операционной системе *CP/M*, причем требуется выполнять условие *AddrStart* ≥ 100*h*. Для микроконтроллеров специального назначения, построенных на основе МП 8080/8085, модули *name*.com не нужны — в них операционная система *CP/M* не используется, а требуется только генерировать модули *name*.bin для последующего программирования *ROM*.

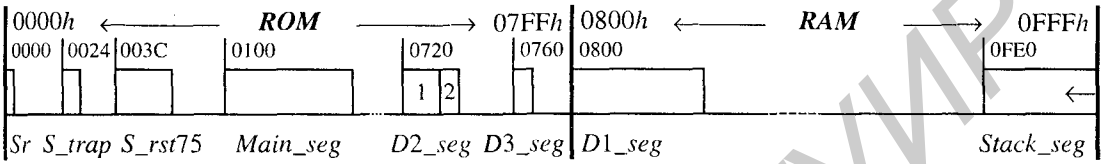


Рис. 1.20. Размещение сегментов в памяти

Задача 1. Для простейшего микроконтроллера, построенного на основе МП 8085*A*, с объемом памяти 4 Кбайта (*ROM* — 2 Кбайта по адресам 0000*h* ... 07FF*h* и *RAM* — 2 Кбайта по адресам 0800*h* ... 0FFF*h*) произвести ее распределение по сегментам

Sr — для одной команды JMP 100*h*, обеспечивающей переход в начало сегмента *Main_seg* после включения питания (*ROM*),

S_trap — для подпрограммы обработки запроса немаскируемого прерывания *TRAP*, вызываемой по адресу *addr* = 8 × 4,5 = 36*d* = 0024*h* (*ROM*),

*S_rst*75 — для подпрограммы обработки запроса маскируемого прерывания *RST* 7.5, вызываемой по адресу *addr* = 8 × 7,5 = 60*d* = 003C*h* (*ROM*),

Main_seg — для основной программы, начинающейся с адреса 100*h* (*ROM*),

*D*1*_seg* — для оперативных данных, начиная с адреса 0800*h* (резервирование 48 байт в *RAM*),

D2_seg — для таблицы преобразования кодов, начиная с адреса 0720*h* (*ROM*),

D3_seg — для таблицы констант, начиная с адреса 0740*h* (*ROM*),

Stack_seg — для стека размером 32 байта (*RAM*)

в соответствии с рис. 1.20. Для внешних устройств задать сегмент по имени *IO_seg* с адреса 30*h*. *Решение*:

```
; Метка  КОП      Операнд                 Комментарий
; Имя    Директива Выражение               Комментарий
        seg       Code          ; Открытие сегмента Code, используемого в качестве сегмента Sr
        JMP       MAIN          ; Переход на начало основной программы
        defseg    S_trap, start = 24h  ; Определение сегмента S_trap (4,5 × 8 = 36d = 24h)
        seg       S_trap        ; Открытие сегмента S_trap
        PUSH      PSW           ; Начало подпрограммы обработки прерывания по входу TRAP
        JMP       TRAP          ; Переход для продолжения подпрограммы
        defseg    S_rst75, start = 3Ch  ; Определение сегмента S_rst75 (7,5 × 8 = 60d = 3Ch)
        seg       S_rst75       ; Открытие сегмента S_rst75
        PUSH      PSW           ; Начало подпрограммы обработки прерывания RST 7.5
          ∴                     ; Основная часть подпрограммы обработки прерывания RST 7.5
        POP       PSW
        EI                      ; Флаг IE ← 1 (прерывания разрешены)
        RET                     ; Конец подпрограммы обработки прерывания RST 7.5
```

```
                                ; Data Segment
Ram     equ   800h              ; Ram = 0800h — начальный адрес RAM
Rsz     equ   800h              ; Rsz = 0800h — объем памяти RAM (RAM Size) 2K × 8 бит
Ssz     equ   20h               ; Ssz = 20h — размер стека (Stack Size)
        defseg D1_seg, start = Ram, class = Data          ; Определение сегмента D1_seg
        seg   D1_seg                                      ; Открытие сегмента D1_seg
Tn      ds    30h               ; С адреса Tn = 0800h резервируется место для 48 байт данных
        defseg D2_seg, start = 720h, class = Data         ; Определение сегмента D2_seg
        seg   D2_seg                                      ; Открытие сегмента D2_seg
T7s     db    40h, 79h, 24h, 30h, 19h, 12h, 2, 78h, 0, 10h, 8, 3, 46h, 21h, 6, 0Eh
        db    7Fh, 3Fh, 4Eh, 9, 48h, 0Ch, 11h, 47h, 41h, 1Bh, 71h
; С адреса T7s = 0720h в память будет записано 27 байт: 40 79 24 30 и т. д.
        defseg D3_seg, start = 760h, class = Data         ; Определение сегмента D3_seg
        seg   D3_seg                                      ; Открытие сегмента D3_seg
Tct     db    28, 2, 11101b, 10h ; Запись в память с адреса Tct = 0760h байтов: 1C 02 1D 10
                                ; Stack Segment
        defseg Stack_seg, start = Ram + Rsz – Ssz, class = Data ; Определение Stack_seg
        seg   Stack_seg                                   ; Открытие сегмента Stack_seg
        ds    Ssz               ; Резервирование 32 байт под стек
                                ; I/O Segment
        defseg IO_seg, start = 30h, class = IOspace       ; Определение сегмента IO_seg
        seg   IO_seg                                      ; Открытие сегмента IO_seg
CSled   ds    1                 ; CSled = 30h — адрес порта внешнего устройства 1
CShs    ds    1                 ; CShs = 31h — адрес порта внешнего устройства 2
CSct    ds    1                 ; CSct = 32h — адрес порта внешнего устройства 3
                                ; Code Segment
        defseg Main_seg, start = 100h, class = Code       ; Определение сегмента Main_seg
        seg   Main_seg                                    ; Открытие сегмента Main_seg
TRAP:   PUSH  H                 ; Продолжение подпрограммы обработки прерывания TRAP
          ∴                     ; Основная часть подпрограммы обработки прерывания TRAP
        POP   H
        POP   PSW
        EI                      ; Флаг IE ← 1 (прерывания разрешены)
        RET                     ; Конец подпрограммы обработки прерывания TRAP
MAIN:   LXI   H, Ram            ; HL ← 800h — начальный адрес RAM
          ∴                     ; Программа тестирования RAM
        LXI   SP, Ram + Rsz     ; SP ← 1000h — инициализация указателя стека
        MVI   A, 0Bh      ; A ← 0Bh = 0000 1011 (A3 = MCE = 1, A2 = M7.5 = 0 — см. рис. 1.15)
        SIM                     ; Разрешение прерывания по входу RST 7.5
        EI                      ; Общее разрешение прерывания
          ∴                     ; Основная программа
        seg   D2_seg            ; Открытие сегмента D2_seg внутри сегмента кода
String  db    'Novoselzeva', 0  ; В память записывается 12 символов в кодах ASCII: 4E 6F ... 00
        seg   Main_seg          ; Открытие сегмента Main_seg и закрытие сегмента D2_seg
          ∴                     ; Основная программа
        end                     ; Конец программы
```

В этой программе имеется восемь сегментов, шесть из которых расположены в ПЗУ, а два в ОЗУ (рис. 1.20). Сегмент данных *D2_seg* открывается два раза. Символическим именам *T7s* и

String, расположенным в сегменте *D2_seg*, транслятор присвоит адреса 0720*h* и 073B*h*, так как первая часть сегмента *D2_seg* займет область памяти с адресами 0720*h* ... 073A*h* (*77s* = 0720*h*, *String* = 073B*h*). Символические имена переменных можно использовать в программе в качестве операндов. Например, команда LXI H, *String* + 5 загрузит в регистровую пару *rp H* число 0740*h* = 073B*h* + 5 (здесь использован арифметический оператор языка ассемблера "+" для вычисления суммы).

Любой из сегментов, в том числе и сегмент кода (*Code Segment*), может отсутствовать. При необходимости сегменты могут быть добавлены, например, сегменты для обработки запросов прерываний по входам *RST* 5.5 и *RST* 6.5, а также сегменты данных.

Если начало сегмента *D3_seg* задать равным 0740*h*, то он будет перекрыт сегментом *D2_seg*, так как его размер равен 39*d* байтам. Перекрытие сегментов обнаруживает компоновщик, который выдает при этом сообщение:

WARNING: Segment D2_SEG overwrites Memory.

Любое предупреждение (*Warning*) программист может игнорировать, считая, что для решения задачи он поступает правильно. Сообщения же ERROR, выдаваемые транслятором при обнаружении ошибок в программе, программист должен непременно исправить для продолжения дальнейшей работы. Решение большого числа задач на языке ассемблера для МП 8080/8085 можно найти в [6, 7].

Задача 2 (файл 1#08_02.asm). По условиям *задачи* 9 из § 1.6 (с. 41), используя постоянные сегменты *Code* и *Data*, написать программу для прогона ее в отладчике. Определить сегмент стека размером 32 байта, хотя он и не будет использован. *Решение*:

```
        seg     Data        ; Открытие сегмента Data
Am1     db      95h, 93h    ; Am1 = 001Dh, M(001D) = 95h
Am2     dw      8595h       ; Am2 = 001Fh, M(001F) = 95h
Ams     ds      3           ; Ams = 0021h
        ds      20h         ; Резервирование 32 байт для стека
Stack:                      ; Транслятор метке Stack присваивает адрес по ее положению в программе
        seg     Code        ; Открытие сегмента Code
        LXI     SP, Stack   ; SP ← Stack = 0044h
        LHLD    Am2         ; H ← M(Am2+1), L ← M(Am2) — два байта второго слагаемого
        XCHG                ; D ← M(Am2+1), E ← M(Am2)
          ∴                 ; Как и в задаче 9 из § 1.6
        INX     H           ; HL ← Ams + 2
        MOV     M, A        ; M(Ams+2) ← CY
        HLT                 ; Команда HLT (останов МП) использована
        end                 ;       для визуализации конца сегмента Code в отладчике
```

На рис. 1.21 показано размещение сегментов в памяти после обработки объектного модуля компоновщиком — постоянные сегменты объединяются в непрерывную область.

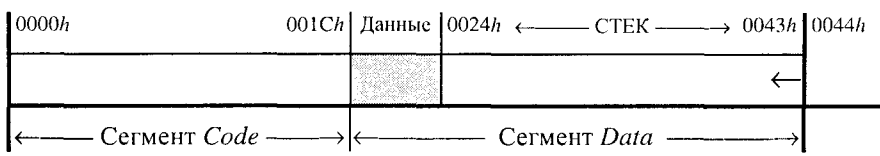


Рис. 1.21. Размещение сегментов в памяти

Директива ORG используется внутри сегментов для задания адреса следующих за нею данных (вводит пустые промежутки в непрерывном массиве данных). В один и тот же сегмент директива может входить любое число раз. Директива ORG (*origin* — начало) имеет формат:

[Label:] ORG *address*

(адрес *address* должен быть не меньше, чем адрес первого элемента смещаемого массива данных). Например, если вторую часть сегмента *D2_seg* в *задаче* 1 оформить в виде:

```
        seg  D2_seg          ; Открытие сегмента D2_seg внутри сегмента кода
        org  720h + 20h      ; 20h — смещение данных относительно начала сегмента 0720h
String  db   'Novoselzeva', 0 ; В память записывается 11 символов в кодах ASCII и 0
        seg  Main_seg        ; Открытие сегмента Main_seg и закрытие сегмента D2_seg,
```

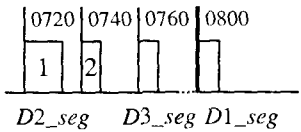


Рис. 1.22. Сегменты данных

то эта часть сегмента *D2_seg* будет смещена на пять байт в сторону старших разрядов (рис. 1.22), так как размер первой части сегмента *D2_seg* равен 27 байтам. Эти пять байт будут принадлежать сегменту *D2_seg*, но их значения не будут определены (зарезервированные байты). Если смещение задать меньше 27 байт, то произойдет перекрытие первой части сегмента *D2_seg* ее второй частью с замещением элементов первой части элементами второй части сегмента *D2_seg*.

Директиву ORG можно не использовать, заменив ее директивой DEFSEG определения нового сегмента *D4_seg*:

```
        defseg D4_seg, start = 740h, class = Data          ; Определение сегмента D4_seg
        seg    D4_seg          ; Открытие сегмента D4_seg внутри сегмента кода
String  db     'Novoselzeva', 0 ; В память записывается 11 символов в кодах ASCII и 0
        seg    Main_seg        ; Открытие сегмента Main_seg и закрытие сегмента D4_seg
```

В постоянных сегментах *Code*, *Data* и *IOspace* директиву ORG использовать нельзя.

Задача 3 (файл 1#08_03.asm). Реализовать программным способом синхронный цифровой автомат, изображенный на рис. 1.23 и представляющий собой генератор псевдослучайной последовательности (ПСП) длиною 255 символов (см. стр. 247 в книге [5]). Цифровой автомат представляет собой счетчик по *mod* 255, выполненный на 8-разрядном сдвигающем регистре с обратной связью, функция возбуждения триггера Q_0 которого $D_0 = Q_7 \oplus Q_3 \oplus Q_2 \oplus Q_1$ — линейная функция. Вывод ПСП выполнить на последовательный выход *SOD* МП 8085*A* по прерыванию с использованием входа запроса прерывания *RST* 6.5.

Программу написать для микроконтроллера, содержащего *EPROM* 6К × 8 бит по адресам 0000*h* ÷ 17FF*h* и *RAM* 2К × 8 бит по адресам 1800*h* ÷ 1FFF*h*. *Решение*:

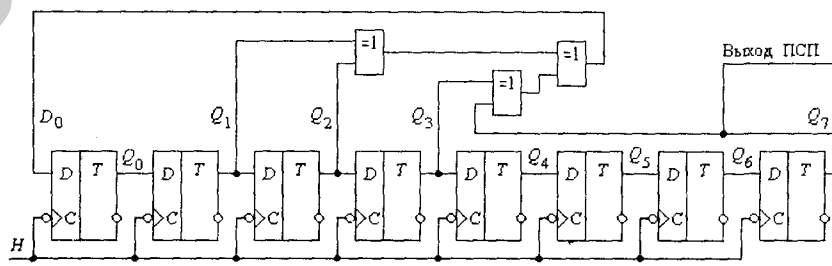


Рис. 1.23. Схема формирования псевдослучайной последовательности

Адрес	МК	*Метка* Имя	*КОП* Директива	*Операнд* Выражение	*Комментарий* Комментарий
В файле		$Allpublic			; Объявление символических имен доступными из других модулей
1#08_03.obj		$title(1#08_03.asm)			; На каждой странице листинга
адреса		$subtitle(*Generator PSP length* 255 *symbol*)			; печатается две строки заголовка
рассчитаны		$paginate			; Разбиение листинга на страницы по 60 строк
без учета		$pagewidth=78			; Задание в строке листинга 78 символов
смещений			defseg	*S_rst*65, start = 34*h*	; 6,5 × 8 = 52*d* = 34*h*
0034*h* и 0100*h*			seg	*S_rst*65	; Открытие сегмента *S_rst*65
		; *Подпрограмма обработки прерывания RST* 6.5			
0000	F5		PUSH	PSW	; 12
0001	3A 0000		LDA	*Ram*	; 13 ; A ← *M*(1800)
0004	E6 80		ANI	80*h*	; 07 ; A ← $A_7$000 0000, *SOD* = A_7
0006	F6 40		ORI	40*h*	; 07 ; A ← $A_7$100 0000, *SOD* = A_7, *SOE* = 1
0008	30		SIM		; 04 ; Вывод разряда A_7 на выход *SOD* МП 8085*A*
0009	3A 0000		LDA	*Ram*	; 13 ; A ← *M*(1800)
000C	E6 8E		ANI	8E*h*	; 07 ; A ← $A_7$000 $A_3A_2A_1$0, флаг *P* = $A_7 \oplus A_3 \oplus A_2 \oplus A_1$
000E	3A 0000		LDA	*Ram*	; 13 ; A ← *M*(1800)
0011	37		STC		; 04 ; CY ← 1, флаг *P* не изменяется
0012	17		RAL		; 04 ; A ← $A_6A_5A_4A_3$ $A_2A_1A_0$1, флаг *P* не изменяется
0013	E2 0018		JPO	LR	; 10 ; LR — *Label* в подпрограмме *RST*
0016	E6 FE		ANI	0FE*h*	; 07 ; A ← $A_6A_5A_4A_3$ $A_2A_1A_0$0
0018	32 0000	LR:	STA	*Ram*	; 13 ; *M*(1800) ← A
001B	F1		POP	PSW	; 10
001C	FB		EI		; 04 ; Флаг *IE* ← 1 (прерывания разрешены)
001D	C9		RET		; 10 ; Возврат в основную программу
				; Всего: 138 тактов	**; *Data Segment***
=0800		*Ram*	equ	1800*h*	; *Ram* = 1800*h* — начальный адрес *RAM*
=0800		*Rsz*	equ	800*h*	; *Rsz* = 800*h* — объем *RAM* (*RAM Size*)
=0020		*Ssz*	equ	20*h*	; *Ssz* = 20*h* — размер стека (*Stack Size*)
			defseg	*Psp_seg*, start = *Ram*, class = *Data*	
			seg	*Psp_seg*	; Открытие сегмента *Psp_seg*
0000	5F		db	5F*h*	; *M*(*Ram*) = *M*(1800) — байт оперативных данных
			defseg	*Stack_seg*, start = *Ram* + *Rsz* – *Ssz*, class = *Data*	**; *Stack Segment***
			seg	*Stack_seg*	
0000	(0020)		ds	*Ssz*	
			defseg	*Main_seg*, start = 100*h*, class = *Code*	**; *Code Segment***
			seg	*Main_seg*	; Открытие сегмента *Main_seg*
0000	31 1000		LXI	SP, *Ram* + *Rsz*	; *SP* = 2000*h* — начальное значение указателя стека
0003	3E 0B		MVI	A, 0D*h*	; A ← 0D*h* = 0000 1101 (A_3 = *MCE* = 1, A_1 = *M*6.5 = 0)
0005	30		SIM		; Разрешение прерывания по входу *RST* 6.5
0006	FB		EI		; Общее разрешение прерывания
0007	C3 0007	LM:	JMP	LM	; МП из этого цикла выходит только по запросу
			end		; прерывания по входу *RST* 6.5

Столбцы Адрес (относительные адреса) и МК (машинные коды) взяты из листинга 1#08_03.prn, полученного при трансляции программы в машинные коды для получения объектного модуля 1#08_03.obj. Подчеркнутые операнды в столбце МК, как и относительные ад-

реса, рассчитаны без учета смещения 003*Ch*. Вся программа занимает в памяти 40 байт, причем 10 из них заняты программой инициализации микроконтроллера после включения питания. Программный цикл JMP LM в общем случае отсутствует (здесь же введен с целью прогона программы в отладчике, как независимой программы), так как микроконтроллер может параллельно с выводом ПСП решать и другие задачи, кроме генерации ПСП. Если же этот цикл используется, то команды PUSH PSW и POP PSW из подпрограммы обработки запроса прерывания по входу *RST* 7.5 можно исключить.

На рис. 1.24 показана генерируемая (строка за строкой) программным способом ПСП, период которой состоит из 255 символов 0 и 1. Максимальное значение частоты вывода символов 0 и 1 определяется только временем выполнения подпрограммы обработки запроса прерывания по входу *RST* 7.5 — затрачивается 138 тактов.

```
Выход МП     5Fh — начальное значение байта данных
   SOD  ←01011111 10111100 11011101 11001010 10010100 01001011 01000110 01110011
         11000110 11000010 00101110 10111101 10111110 00011010 01101011 01101010
         00001001 11011001 00100110 00000111 01001000 11100010 00000010 11000111
         10100001 11111110 01000010 10011111 01010101 11000001 10001010 1100110
```

Рис. 1.24. Псевдослучайная последовательность

Директивы управления листингом. Эти директивы должны начинаться с символа "$" и располагаться в самом начале программы. Описание этих директив дано в поле комментария предыдущей программы. Пробелы перед символами "(" и "=", а также после символа "=" не допускаются.

Директивы $TITLE и $SUBTITLE выводят в начале каждой страницы листинга две строки, заключенные в скобки в тексте исходного модуля (см. предыдущую программу). Начальные строки листинга 1#08_03.prn после трансляции исходного модуля 1#08_03.asm будут содержать информацию:

```
Avocet 8085/Z80 Assembler v2.02, #01235   Chip=8085            3/24/100  11:07:06
1#08_03.asm                                                                Page 1
Generator PSP length 255 symbol                                     ↑
           1  $allpublic                        Месяц/Число/Год  Часы/Минуты/Секунды
           2  $title(1#08_03.asm)                         (год 100 вместо 2000)
           3  $subtitle(Generator PSP length 255 symbol)
           4  $paginate
           5  $pagewidth=78
           6          defseg  S_rst65, start = 34h   ; 6,5 × 8 = 52d = 34h
           7          seg     S_rst65                ; Открытие сегмента S_rst65
0000& F5   8          PUSH    PSW
```

Директивы $PAGINATE и $PAGEWIDTH задают число строк в странице и число символов в строке листинга.

Указание: Программный пакет *AVSIM*85 фирмы *Avocet Systems, Inc.* предназначен для использования нерусскоязычными пользователями, поэтому текст в директивах $TITLE и $SUBTITLE и комментарии на русском языке в листинге будут искажены из-за использования транслятором 7-разрядных *ASCII*-кодов (см. табл. 1.9). Прописные буквы русского алфавита А, К и Н вообще нельзя использовать, так как они преобразуются в такие управляющие символы, которые нарушают работу внутреннего текстового редактора программного пакета (русские буквы А, К и Н при необходимости можно заменить латинскими A, K и H). В России программный пакет *AVSIM*85 фирмы *Avocet Systems, Inc.* был выпущен на *CD-ROM* (*Compact Disk*

read-only Memory — нестираемая память, или память только для чтения, на компакт-дисках). Этот пакет содержит мощный отладчик, настраиваемый на конфигурации микроконтроллеров, содержащих специально разработанные для МП 8085 интегральные схемы 8155 и 8355.

Директивы определения типа МП. Эти директивы должны начинаться с символа "$" и располагаться в самом начале программы.

Директивы $CHIP(8085) и $CHIP(Z80) указывают, для какого типа МП написан исходный модуль. По умолчанию считается, что исходный модуль подготовлен для МП 8080/8085 — директиву Chip(8085) можно не использовать. Если в программе *задачи* 3 указать директиву Chip(Z80), то транслятор выдаст сообщение:

ERROR 31: Instruction not implemented in Z80

(мнемоника команд программы не соответствует мнемонике команд МП Z80). Если перед скобкой дать пробел, то транслятор выдаст сообщение:

ERROR 38: Syntax error

(нарушены правила написания директивы).

Директивы определения данных. Эти директивы используются для определения данных в памяти: байт и слов. Символическим именам данных ставятся в соответствие адреса памяти этих данных.

Директивы DB (*Define Byte* — определить байт) и DW (*Define Word* — определить слово) имеют формат (в отличие от других директив эти директивы генерируют объектный код):

```
[Name1[:]]   DB   43, 0DCh, 39, 10110b, 'IBM PC', 0Ah, 0Dh ; 12 байт
[Name2[:]]   DW   1998h, 0D4C1h, 43                        ; 3 слова (6 байт)
```

Для определения относительного адреса программных элементов в сегментах данных и кода ассемблер использует счетчик адресов. Текущее значение адреса обозначается символом $ (слово) и его можно использовать в директивах. Имени *Name*, если оно имеется, присваивается адрес первого байта массива данных. С этого адреса в смежные ячейки памяти записываются байты или слова. Для слов в первую ячейку памяти помещается младший байт слова, а в следующую (с большим адресом) — старший байт слова. Введенные имена можно использовать в качестве непосредственных операндов в командах МП. Символ и последовательность символов (символьная строка), заключенная в апострофы (например, 'j' и '8085/Z80'), транслируются в их *ASCII*-коды (6Ah и 38 30 38 35 2F 5A 38 30) в соответствии с табл. 1.9.

Пример 1 (наиболее полезное применение значений $ см. в файле 2&04_02.asm):

```
Alpha   db   95h, 93h, 'IBM PC', 10110b, 36  ; Начало сегмента данных      ; 10 байт
        db   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0Ah, 0Bh, 0Ch, 0Dh, 0Eh, 0Fh   ; 16 байт
Beta:   dw   $, 1998h, 0D4C1h ; $ = 001Ah = 26d — относительный адрес Beta ;  6 байт
Gamma:  ds   18h          ; Зарезервировать, не инициализируя, 24 байта в памяти
        LDA  Alpha + 3    ; A ← M(Alpha + 3) = 42h = ASCII-код B  (Gamma = Alpha + 32)
        STA  Gamma + 1    ; M(Gamma + 1) ← 42h
        LXI  D, Alpha     ; DE ← адрес байта 95h
        LDAX D            ; A ← M(Alpha) = 95h (в круглых скобках указан адрес памяти)
        LXI  H, Alpha + 5 ; HL ← адрес пробела между словами IBM и PC
        ADD  M            ; A ← 95h + 20h = B5h
        LXI  B, Beta + 2  ; BC ← адрес слова 1998h
        LXI  H, Gamma + 7
        MOV  M, A         ; M(Gamma + 7) ← B5h
        MVI  H, Beta      ; Ошибка (то же самое и для имени Gamma)
```

При обнаружении ошибок в использовании типов данных компоновщик выдает сообщение (считаем, что START = 100*h* — начальный адрес сегмента кода):

WARNING: (Abs Addr = 0116H) Singles byte values out of range.

Директива DS (*Define Space* — определить область памяти) используется для резервирования ячеек памяти под оперативные данные и стек (синонимы DEFS и RMB). Эта директива имеет формат:

[*Name*[:]] DS *value*

(*value* — число или символическое имя, определяющее число). Этой директивой резервируется в памяти число ячеек памяти, заданное параметром *value*, без задания их начальных значений.

Пример 2:

```
Gamma:  ds    10h     ; Зарезервировать, не инициализируя, 16 байт в памяти
Delta   equ   10      ; Delta = 10d
        ds    Delta   ; Зарезервировать, не инициализируя, 10 байт в памяти
Addr$   equ   $       ; $ — текущее значение счетчика адресов
```

Директивы EQU (*equate* — приравнять) и TEQ (*temporary equate*) не определяют элемент данных в памяти, а лишь вводят символические имена для чисел (констант), с которыми удобнее оперировать, чем с числами, так как в имена всегда можно вложить содержательный смысл. Директива EQU имеет формат:

Name[:] EQU *value*

(*value* — число или символическое имя, определяющее число).

Пример 3:

```
Beta:   equ   10101b    ; Beta = 15h ; Начало сегмента кода
Gamma:  equ   5AF2h     ; Gamma = 5AF2h
        MVI   A, Beta   ; A ← 15h
        LXI   D, Beta   ; DE ← 0015h
Addr1   equ   $         ; Addr1 = $ = 0005h — текущее значение счетчика адресов
        LXI   H, Gamma  ; HL ← 5AF2h
Addr2   teq   $         ; Addr2 = $ = 0008h — текущее значение счетчика адресов
        MVI   C, Gamma  ; ERROR 30: Operand out of range
                        ; ERROR 22: Byte value not in the range –128 ... +255
```

Если использовать директиву *Delta* equ *Beta*, то *Delta* = 15*h*.

Если какая-либо константа используется в программе большое число раз, а затем потребуется изменить ее значение, то это значительно проще сделать одной директивой EQU, чем изменять числа по всей программе. В последнем случае велика также вероятность ошибки при внесении в программу изменений (в больших программах некоторые константы могут использоваться десятки раз).

На рис. 1.25 представлена принципиальная схема памяти объемом 16К × 8 бит, содержащая 8 БИС объемом по 2К × 8 бит: 14К × 8 бит *EPROM* 573РФ2 (*EPROM* 2716 фирмы *Intel*) и 2К × 8 бит *RAM* (БИС адресуются разрядами шины адреса A_{10-0}, а значит, каждая БИС содержит 2^{11} = 2048 ячеек памяти). При выполнении команд программы селекция одной из восьми БИС производится разрядами адреса A_{15-11} с помощью дешифратора, выполненного на микросхеме 1533ИД7 (*SN74ALS*138 фирмы *Texas Instruments*). В табл. 1.12 представлены диапазоны адресов всех БИС, адресуемых сигналами МП A_{15-0}.

Если в микроконтроллере использовать такую память, то в программе *задачи* 3 достаточно изменить значение символического имени *Ram* только в одной строке

Ram equ 3800*h*,

вместо того чтобы изменять значение операнда *Ram* = 800*h* в семи строках. Размер стека *Ssz* также легко изменяется с помощью директивы EQU.

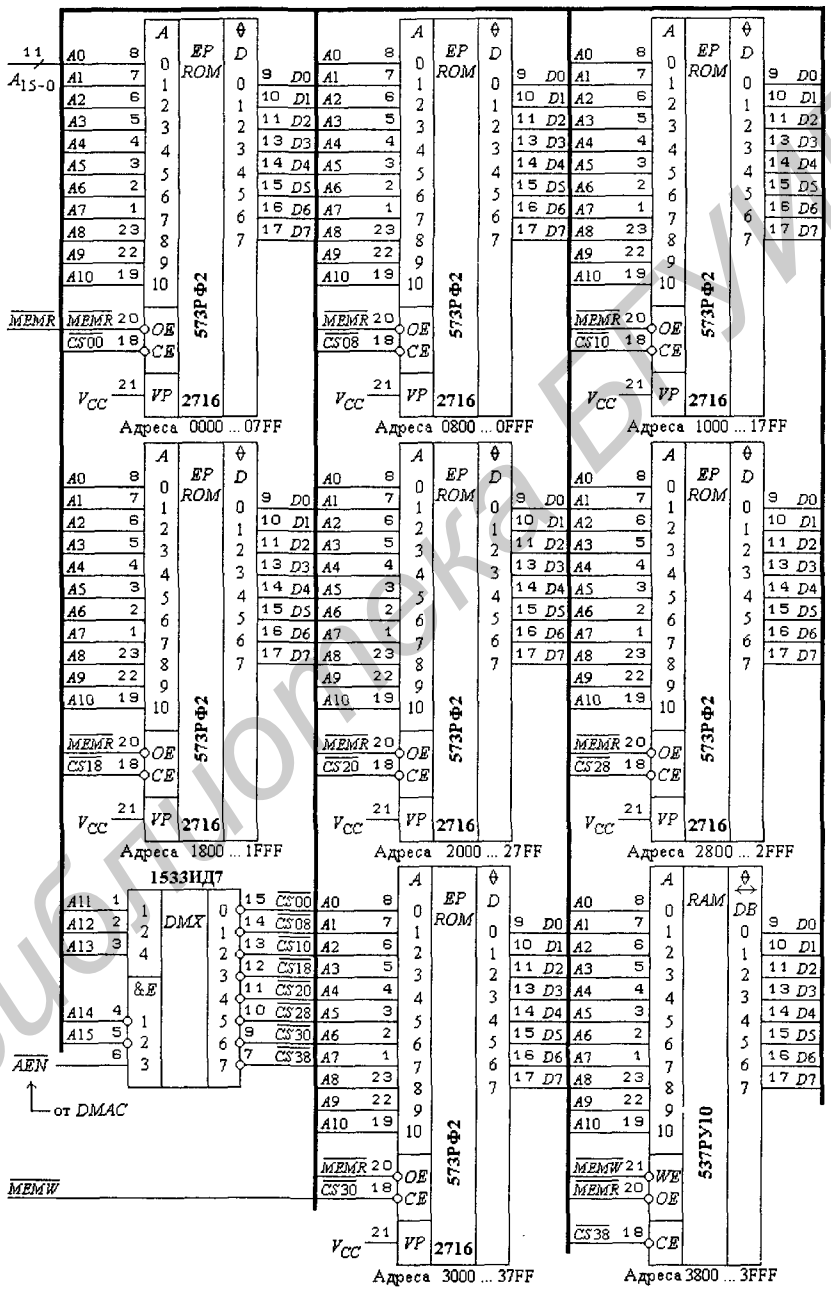


Рис. 1.25. Принципиальная схема памяти объемом 16К × 8 бит

Таблица 1.12. **Адреса памяти БИС по 2К × 8 бит с общим объемом 16К × 8 бит**

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Адреса памяти	Объем памяти
0	0	0	0	0	×	×	×	×	×	×	×	×	×	×	×	0000*h* ÷ 07FF*h*	*EPROM* 2К × 8 бит
0	0	0	0	1	×	×	×	×	×	×	×	×	×	×	×	0800*h* ÷ 0FFF*h*	*EPROM* 2К × 8 бит
0	0	0	1	0	×	×	×	×	×	×	×	×	×	×	×	1000*h* ÷ 17FF*h*	*EPROM* 2К × 8 бит
0	0	0	1	1	×	×	×	×	×	×	×	×	×	×	×	1800*h* ÷ 1FFF*h*	*EPROM* 2К × 8 бит
0	0	1	0	0	×	×	×	×	×	×	×	×	×	×	×	2000*h* ÷ 27FF*h*	*EPROM* 2К × 8 бит
0	0	1	0	1	×	×	×	×	×	×	×	×	×	×	×	2800*h* ÷ 2FFF*h*	*EPROM* 2К × 8 бит
0	0	1	1	0	×	×	×	×	×	×	×	×	×	×	×	3000*h* ÷ 3FFF*h*	*EPROM* 2К × 8 бит
0	0	1	1	1	×	×	×	×	×	×	×	×	×	×	×	3800*h* ÷ 3FFF*h*	*RAM* 2К × 8 бит

Примечание: ××× ×××× ×××× — значения от 000 0000 0000 до 111 1111 1111.

Директивой EQU символическому имени в разных местах программы нельзя присваивать различные численные значения. Это можно делать с помощью директивы TEQ. Формат директивы TEQ такой же, что и директивы EQU:

Name[:] TEQ *value*

(*value* — число или символическое имя, определяющее число).

Операторы ассемблера. В командах в качестве непосредственных операндов *d*8 и *d*16 можно использовать не только числа и символические имена, но и выражения, составленные из чисел и символических имен с помощью специальных операторов.

Арифметические операторы. Для обозначения этих операторов используются символы: +, –, *, / и MOD. Арифметические операторы имеют форматы:

+Expression (унарный плюс — сохраняет знак и значение выражения *Expression*),

*Expression*1 + *Expression*2 (сложение — вычисляет сумму двух выражений),

–Expression (унарный минус — изменяет знак выражения, т. е. вычисляет дополнительный код),

*Expression*1 – *Expression*2 (вычитание — вычисляет разность двух выражений),

*Expression*1 * *Expression*2 (умножение — вычисляет произведение двух выражений),

*Expression*1 / *Expression*2 (деление — вычисляет частное от деления двух выражений),

*Expression*1 MOD *Expression*2 (вычисляет остаток от деления двух выражений).

Пример 4 (использование арифметических операторов):

```
        SUI   +0D4h             ; A ← A – D4h
        MVI   E, 180 + 90       ; Ошибка: 180 + 90 = 270d = 10Eh > 255d
        MVI   A, –2             ; A ← FEh (дополнительный код числа –2)
Alpha:  equ   9                 ; Директива ассемблера: имени alpha присвоить значение 9
        MVI   E, –Alpha         ; E ← F7h (дополнительный код числа –9)
        MVI   L, –0A7h          ; Ошибка: –0A7h < –128d
        MVI   B, 5 – 7          ; B ← FEh (дополнительный код числа –2)
        LXI   H, 5555h – 7777h  ; HL ← DDDEh (дополнительный код числа –2222h)
        MVI   C, 70*5           ; Ошибка: 70*5 = 350d = 15Eh > 255d
        CPI   30*5              ; A – 96h, так как 30*5 = 150d = 96h
        ORI   303/5   ; A ← A ∨ 3Ch, так как 303/5 = 60d = 3Ch (частное от деления чисел)
        ANI   17 mod 5          ; A ← A & 02, так как 17 mod 5 = 2
        MVI   E, 177 mod (Alpha/2)   ; E ← 1, так как 177 mod 4 = 1
```

При обнаружении ошибок транслятор выдает сообщение:

ERROR 30: Operand out of range
ERROR 22: Byte value not in the range –128 … +255

Операторы сдвига. Для этих операторов используются обозначения: SHL и SHR. Операторы сдвига имеют форматы:

Expression SHL *count* (сдвиг влево операнда *Expression* на число разрядов, равное *count*, с вводом значения 0 в младший разряд при каждом сдвиге);

Expression SHR *count* (сдвиг вправо операнда *Expression* на число разрядов, равное *count*, с вводом значения 0 в старший разряд при каждом сдвиге, причем младшие разряды теряются).

Операторы сдвига SHL и SHR производят умножение и деление операнда *Expression* на число 2^{count} соответственно. Значение *Expression* $\times 2^{count}$ при сдвиге влево байта не должно выходить за пределы значений –128 … +255.

Пример 5 (использование операторов сдвига):

```
MVI   C, 16h shl 3      ; C ← B0h =1011 0000 (16h = 0001 0110)
ADI   0D5h shr 2        ; A ← A + 35h = A + 0011 0101 (D5h = 1101 0101)
LXI   D, 5B74h shl 4    ; DE ← 05B7h
MVI   A, 36h shl 3      ; Ошибка
```

ERROR 30: Operand out of range
ERROR 22: Byte value not in the range –128 … +255

Логические операторы. Для этих операторов используются обозначения: NOT, AND, OR и XOR. Логические операторы имеют форматы:

NOT *Expression* (инвертирование разрядов выражения *Expression*),
*Expression*1 AND *Expression*2 (поразрядная конъюнкция двух выражений),
*Expression*1 OR *Expression*2 (поразрядная дизъюнкция двух выражений),
*Expression*1 XOR *Expression*2 (поразрядная сумма по модулю два двух выражений).

Данные операторы выполняют такие же операции, что и соответствующие команды МП. Значение NOT *Expression* для байтов не должно выходить за пределы значений –128 … +255.

Пример 6 (использование логических операторов):

```
Alpha   equ   7Ah
Beta    equ   5Eh
        CPI   not Alpha          ; A – 85h (7Ah = 0111 1010)
        MVI   C, not –Alpha      ; C ← 79h (0 – 7Ah = 86h = 1000 0110)
        ADI   Beta and Alpha     ; A ← A + 5Ah (5Ah = 0101 1110 & 0111 1010)
        ANI   Beta or 0C2h       ; A ← A & DEh (DEh = 0101 1110 ∨ 1100 0010)
        LXI   H, 5E63h xor 7B84h ; HL ← 25E7h
        MVI   E, not 8Ah         ; Ошибка
```

ERROR 30: Operand out of range
ERROR 22: Byte value not in the range –128 … +255

Операторы типа переменной. Для этих операторов используются обозначения: LOW и HIGH. Операторы типа переменной имеют форматы:

LOW *Expression* (выделение младшего байта слова *Expression*),
HIGH *Expression* (выделение старшего байта слова *Expression*).

Пример 7 (использование операторов типа переменной):

```
Gamma   equ   0C95Eh
        MVI   E, low Gamma       ; E ← 5Eh
        XRI   high Gamma         ; A ← A ⊕ C9h
```

Операторы отношений. Для этих операторов используются обозначения: EQ (*equally* — равно), NE (*no equally* — не равно), LT (*less* — меньше), GT (*greater* — больше), LE (*less or equally* — меньше или равно) и GE (*greater or equally* — больше или равно). Операторы отношений имеют форматы:

*Expression*1 EQ *Expression*2 (если *Expression*1 = *Expression*2, то всему выражению присваивается значение *true* и значение *false* в противном случае);

*Expression*1 NE *Expression*2 (если *Expression*1 ≠ *Expression*2, то всему выражению присваивается значение *true* и значение *false* в противном случае);

*Expression*1 LT *Expression*2 (если *Expression*1 < *Expression*2, то всему выражению присваивается значение *true* и значение *false* в противном случае);

*Expression*1 GT *Expression*2 (если *Expression*1 > *Expression*2, то всему выражению присваивается значение *true* и значение *false* в противном случае);

*Expression*1 LE *Expression*2 (если *Expression*1 ≤ *Expression*2, то всему выражению присваивается значение *true* и значение *false* в противном случае);

*Expression*1 GE *Expression*2 (если *Expression*1 ≥ *Expression*2, то всему выражению присваивается значение *true* и значение *false* в противном случае).

Значению *true* (истина) соответствует число FFFF*h* (дополнительный код числа –1), а значению *false* (ложь) — число 0000*h*. Операторы отношений используются в условных директивах, управляющих трансляцией программы, для проверки условия, определяющего, какой из двух следующих за директивой блоков программы следует транслировать.

В командах МП использовать операторы отношений в качестве операндов нет необходимости, но если это по каким-либо причинам сделано, то следует знать, что логическим значениям *true* и *false* при представлении их байтами/словами будут соответствовать числа FF*h*/FFFF*h* и 00*h*/0000*h*.

Пример 8 (использование операторов отношений):

```
Gamma   equ   5Eh                ; Gamma = 5Eh
        LXI   B, Gamma ne 4Ah    ; BC ← ►5Eh ne 4Ah◄ = FFFFh (true)
        LXI   B, Gamma eq 7Fh    ; BC ← ►5Eh eq 7Fh◄ = 0000h (false)
        LXI   D, 1 ge 0FFFFh     ; DE ← ►1 ge FFFFh◄ = 0000h (false)
        LXI   H, –1 eq 0FFFFh    ; HL ← ►–1 eq 0FFFFh◄ = FFFFh (true)
        MVI   D, 1 lt 0FFFFh     ; D ← ►1 lt 0FFFFh◄ = FFh (true)
        MVI   E, 1 ge –1         ; E ← ►1 ge –1◄ = 00h (false)
```

Символами ► … ◄ обозначены значения, присваиваемые транслятором выражениям.

Порядок старшинства операторов. Порядок старшинства операторов определяет последовательность, в соответствии с которой будет вычисляться выражение. Операторы, имеющие большее старшинство, будут выполняться раньше операторов, имеющих меньшее старшинство. В выражениях можно использовать также оператор ( ) — круглые скобки, в которые заключаются отдельные части выражения. Этот оператор изменяет порядок старшинства других операторов — часть выражения, заключенная в круглые скобки выполняется первой. В табл. 1.13 рассмотренные выше операторы расположены в порядке уменьшения старшинства.

Операторы, стоящие в одной строке, имеют одинаковое старшинство и последовательно выполняются в порядке слева направо при их использовании в одном выражении. Круглые скобки можно использовать несколько раз для выделения необходимого числа частей выражения. *Пример*:

$$177 \bmod 9/2 = 03 \quad \text{и} \quad 177 \bmod (9/2) = 01,$$

т. е. значение 177 mod 9/2 = (177 mod 9)/2 = 6/2 = 03, а значение 177 mod (9/2) = 177 mod 4 = 01.

Таблица 1.13. **Порядок старшинства операторов**

№	Операторы	№	Операторы
1	( )	6	EQ, NE, LT, LE, GT, GE
2	HIGH, LOW	7	NOT
3	+ (унарный плюс), – (унарный минус)	8	AND
4	*, /, MOD, SHL, SHR	9	OR, XOR
5	+ (сложение), – (вычитание)		

Использование операторов в ассемблерных программах позволяет многие вычисления переложить на транслятор и тем самым значительно уменьшить как объем памяти, занимаемый программой, так и время выполнения программы. В частности, перевод чисел в дополнительный код очень просто выполняется оператором унарный минус.

Пример 9 (использование унарных операторов):

```
Oper1   db   +43h, –43h, –80h, +39, –7Eh, –127  ; ⇒ Memory ← 43 BD 80 27 82 81
Oper2:  dw   –8000h, –7FFFh, +7FFFh, –3C9Bh     ; ⇒ Memory ← 8000 8001 7FFF C365
Oper3   db   –81h   ; ⇒ Memory ← 7Fh            ERROR 22: Byte value not in the range –128 ... +255
Alpha:  equ  –3C9Bh          ; ⇒ Alpha = C365h = [–3C9Bh]д
Beta    teq  –80h            ; ⇒ Beta = FF80h = [–0080h]д
Gamma   teq  –81h            ; ⇒ Gamma = FF7Fh = [–0081h]д
Delta   equ  –8001h          ; ⇒ Delta = 7FFFh ≠ [–8001h]д
        LDA  Oper1 + 2       ; A ← M(Oper1+2) = 80h = [–80h]д
        LXI  D, Oper2 + 6    ; DE ← адрес слова –3C9Bh
        LXI  H, Alpha + 5    ; HL ← C36Ah = [–3C9Bh + 5]д
        LXI  B, Beta + 2     ; BC ← FF82h = [–80h + 2]д
        LXI  H, Gamma + 7    ; HL ← FF86h = [–81h + 5]д
        MVI  H, Beta         ; H ← 80h (младший байт [–0080h]д)
        MVI  H, Alpha        ; A ← 65h
                             ;
```

ERROR 30: Operand out of range
ERROR 22: Byte value not in the range –128 ... +255

Для константы *Delta*, представленной словом, транслятор не выводит сообщения об ошибке перевода отрицательного числа в дополнительный код. Это делается только для байтовых констант и переменных (например, для переменной *Oper*3). Программист сам должен следить за принадлежностью чисел, представленных словами, диапазону допустимых чисел, если хочет получить их дополнительный код. Первая ошибка в приведенных примерах не связана с переводом отрицательного числа в дополнительный код. Здесь ошибка заключена в разной размерности операндов — получатель байтовый, а источник двухбайтовый (слово).

Примеры программ. Для получения практических навыков программирования на языке ассемблера рекомендуется внимательно разобрать следующие задачи.

Задача 4 (файл 1#08_04.asm). Реализовать программным способом комбинационную схему, изображенную на рис. 1.26 и представляющую собой преобразователь 8-разрядных двоичных чисел в 8-разрядный код Грея. Комбинационная схема описывается функциями [8]

$$\alpha_p = \begin{cases} x_p \oplus x_{p+1}, \text{ если } p = 0,1, \ \ldots, \ 6; \\ x_7 \oplus 0 = x_7, \text{ если } p = 7. \end{cases}$$

и генерирует для двоичных чисел из диапазона 0 ÷ 255 код Грея в соответствии с табл. 1.14.

Программу конвертора написать для микроконтроллера, содержащего *EPROM* 8К × 8 бит по адресам 0000*h* ÷ 1FFF*h* и *RAM* 2К × 8 бит по адресам 2000*h* ÷ 27FF*h*. *Решение*:

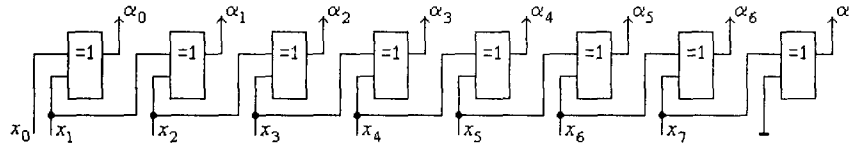


Рис. 1.26. Схема формирования 8-разрядного кода Грея

```
; Табличное преобразование 8-разрядного двоичного кода в код Грея
Ram   equ    2000h        ; Ram = 2000h — начальный адрес RAM          Data Segment
      defseg Gray_seg, start = Ram, class = Data
      seg    Gray_seg
      db     4, 5, 6, 7, 2Ch, 0EAh, 2Dh, 0FBh ; Эти двоичные числа преобразовать в код Грея
Conv  ds     8            ; Резервирование 8 байт для преобразований
Gray  ds     100h         ; Резервирование 256 байт для таблицы кода Грея
      defseg Main_seg, start = 100h, class = Code                  Code Segment
      seg    Main_seg
; Копирование преобразуемых двоичных чисел
      LXI    H, Ram       ; HL ← 2000h — начальный адрес двоичных чисел
      LXI    D, Conv      ; DE ← 2008h — начальный адрес области копирования
      MVI    B, 8         ; B ← 8 — количество преобразуемых чисел
LM0:  MOV    A, M         ; A ← M(HL)
      STAX   D            ; M(DE) ← A
      INX    H
      INX    D
      DCR    B
      JNZ    LM0
; Генерация таблицы 8-разрядных двоичных чисел
      LXI    H, Gray      ; HL ← 2010h — начальный адрес таблицы двоичных чисел
      SUB    A            ; A ← 0
      MOV    B, A         ; B ← 0
LM1:  MOV    M, A         ; M(HL) ← A
      INX    H
      INR    A
      DCR    B
      JNZ    LM1
; Генерация таблицы 8-разрядного кода Грея
      LXI    H, Gray      ; HL ← 2010h — начальный адрес таблицы кода Грея
      MVI    B, 0
LM2:  SUB    A            ; CY ← 0
      MOV    A, M         ; A ← M(HL) — двоичное число из таблицы
      MOV    C, A
      RAR                 ; A7A6A5A4A3A2A1A0 ← 0A7A6A5A4A3A2A1
      XRA    C            ; A7A6A5 ... A0 ← (A7 ⊕ 0)(A6 ⊕ A7)(A5 ⊕ A6) ... (A0 ⊕ A1)
      MOV    M, A         ; M(HL) ← A — запись кода Грея в таблицу
      INX    H
      DCR    B
      JNZ    LM2
```

```
; Преобразование восьми двоичных чисел в код Грея
      LXI   D, Conv    ; DE ← 2008h — начальный адрес преобразуемых чисел
      MVI   B, 8       ; B ← 8 — количество преобразуемых чисел
LM3:  LXI   H, Gray    ; HL ← 2010h — начальный адрес таблицы кода Грея
      LDAX  D          ; A ← M(DE) — преобразуемое двоичное число
      ADD   L
      MOV   L, A
      MVI   A, 0
      ADC   H
      MOV   H, A       ; HL ← HL + M(DE) — адрес кода Грея в таблице
      MOV   A, M       ; A ← M(HL) — код Грея из таблицы
      STAX  D          ; M(DE) ← A — преобразованное в код Грея двоичное число
      INX   D
      DCR   B          ;              Результат преобразования:
      JNZ   LM3        ;     4 → 6, 5 → 7, 6 → 5, 7 → 4,
      end              ;     2Ch → 3Ah, EAh → 9Fh, 2Dh → 3Bh, FBh → 86h
```

Таблица 1.14. **Взаимное преобразование 8-разрядных двоичных чисел и кода Грея**

B	Код Грея		*B*	Код Грея		*B*	Код Грея		*B*	Код Грея	
00	00	0000 0000	40	60	0110 0000	80	C0	1100 0000	C0	A0	1010 0000
01	01	0000 0001	41	61	0110 0001	81	C1	1100 0001	C1	A1	1010 0001
02	03	0000 0011	42	63	0110 0011	82	C3	1100 0011	C2	A3	1010 0011
03	02	0000 0010	43	62	0110 0010	83	C2	1100 0010	C3	A2	1010 0010
04	06	0000 0110	44	66	0110 0110	84	C6	1100 0110	C4	A6	1010 0110
05	07	0000 0111	45	67	0110 0111	85	C7	1100 0111	C5	A7	1010 0111
06	05	0000 0101	46	65	0110 0101	86	C5	1100 0101	C6	A5	1010 0101
07	04	0000 0100	47	64	0110 0100	87	C4	1100 0100	C7	A4	1010 0100
08	0C	0000 1100	48	6C	0110 1100	88	CC	1100 1100	C8	AC	1010 1100
09	0D	0000 1101	49	6D	0110 1101	89	CD	1100 1101	C9	AD	1010 1101
0A	0F	0000 1111	4A	6F	0110 1111	8A	CF	1100 1111	CA	AF	1010 1111
0B	0E	0000 1110	4B	6E	0110 1110	8B	CE	1100 1110	CB	AE	1010 1110
0C	0A	0000 1010	4C	6A	0110 1010	8C	CA	1100 1010	CC	AA	1010 1010
0D	0B	0000 1011	4D	6B	0110 1011	8D	CB	1100 1011	CD	AB	1010 1011
0E	09	0000 1001	4E	69	0110 1001	8E	C9	1100 1001	CE	A9	1010 1001
0F	08	0000 1000	4F	68	0110 1000	8F	C8	1100 1000	CF	A8	1010 1000
10	18	0001 1000	50	78	0111 1000	90	D8	1101 1000	D0	B8	1011 1000
11	19	0001 1001	51	79	0111 1001	91	D9	1101 1001	D1	B9	1011 1001
12	1B	0001 1011	52	7B	0111 1011	92	DB	1101 1011	D2	BB	1011 1011
13	1A	0001 1010	53	7A	0111 1010	93	DA	1101 1010	D3	BA	1011 1010
14	1E	0001 1110	54	7E	0111 1110	94	DE	1101 1110	D4	BE	1011 1110
15	1F	0001 1111	55	7F	0111 1111	95	DF	1101 1111	D5	BF	1011 1111
16	1D	0001 1101	56	7D	0111 1101	96	DD	1101 1101	D6	BD	1011 1101
17	1C	0001 1100	57	7C	0111 1100	97	DC	1101 1100	D7	BC	1011 1100
18	14	0001 0100	58	74	0111 0100	98	D4	1101 0100	D8	B4	1011 0100
19	15	0001 0101	59	75	0111 0101	99	D5	1101 0101	D9	B5	1011 0101
1A	17	0001 0111	5A	77	0111 0111	9A	D7	1101 0111	DA	B7	1011 0111
1B	16	0001 0110	5B	76	0111 0110	9B	D6	1101 0110	DB	B6	1011 0110
1C	12	0001 0010	5C	72	0111 0010	9C	D2	1101 0010	DC	B2	1011 0010

Продолжение табл. 1.14

B	Код Грея		B	Код Грея		B	Код Грея		B	Код Грея	
1D	13	0001 0011	5D	73	0111 0011	9D	D3	1101 0011	DD	B3	1011 0011
1E	11	0001 0001	5E	71	0111 0001	9E	D1	1101 0001	DE	B1	1011 0001
1F	10	0001 0000	5F	70	0111 0000	9F	D0	1101 0000	DF	B0	1011 0000
20	30	0011 0000	60	50	0101 0000	A0	F0	1111 0000	E0	90	1001 0000
21	31	0011 0001	61	51	0101 0001	A1	F1	1111 0001	E1	91	1001 0001
22	33	0011 0011	62	53	0101 0011	A2	F3	1111 0011	E2	93	1001 0011
23	32	0011 0010	63	52	0101 0010	A3	F2	1111 0010	E3	92	1001 0010
24	36	0011 0110	64	56	0101 0110	A4	F6	1111 0110	E4	96	1001 0110
25	37	0011 0111	65	57	0101 0111	A5	F7	1111 0111	E5	97	1001 0111
26	35	0011 0101	66	55	0101 0101	A6	F5	1111 0101	E6	95	1001 0101
27	34	0011 0100	67	54	0101 0100	A7	F4	1111 0100	E7	94	1001 0100
28	3C	0011 1100	68	5C	0101 1100	A8	FC	1111 1100	E8	9C	1001 1100
29	3D	0011 1101	69	5D	0101 1101	A9	FD	1111 1101	E9	9D	1001 1101
2A	3F	0011 1111	6A	5F	0101 1111	AA	FF	1111 1111	EA	9F	1001 1111
2B	3E	0011 1110	6B	5E	0101 1110	AB	FE	1111 1110	EB	9E	1001 1110
2C	3A	0011 1010	6C	5A	0101 1010	AC	FA	1111 1010	EC	9A	1001 1010
2D	3B	0011 1011	6D	5B	0101 1011	AD	FB	1111 1011	ED	9B	1001 1011
2E	39	0011 1001	6E	59	0101 1001	AE	F9	1111 1001	EE	99	1001 1001
2F	38	0011 1000	6F	58	0101 1000	AF	F8	1111 1000	EF	98	1001 1000
30	28	0010 1000	70	48	0100 1000	B0	E8	1110 1000	F0	88	1000 1000
31	29	0010 1001	71	49	0100 1001	B1	E9	1110 1001	F1	89	1000 1001
32	2B	0010 1011	72	4B	0100 1011	B2	EB	1110 1011	F2	8B	1000 1011
33	2A	0010 1010	73	4A	0100 1010	B3	EA	1110 1010	F3	8A	1000 1010
34	2E	0010 1110	74	4E	0100 1110	B4	EE	1110 1110	F4	8E	1000 1110
35	2F	0010 1111	75	4F	0100 1111	B5	EF	1110 1111	F5	8F	1000 1111
36	2D	0010 1101	76	4D	0100 1101	B6	ED	1110 1101	F6	8D	1000 1101
37	2C	0010 1100	77	4C	0100 1100	B7	EC	1110 1100	F7	8C	1000 1100
38	24	0010 0100	78	44	0100 0100	B8	E4	1110 0100	F8	84	1000 0100
39	25	0010 0101	79	45	0100 0101	B9	E5	1110 0101	F9	85	1000 0101
3A	27	0010 0111	7A	47	0100 0111	BA	E7	1110 0111	FA	87	1000 0111
3B	26	0010 0110	7B	46	0100 0110	BB	E6	1110 0110	FB	86	1000 0110
3C	22	0010 0010	7C	42	0100 0010	BC	E2	1110 0010	FC	82	1000 0010
3D	23	0010 0011	7D	43	0100 0011	BD	E3	1110 0011	FD	83	1000 0011
3E	21	0010 0001	7E	41	0100 0001	BE	E1	1110 0001	FE	81	1000 0001
3F	20	0010 0000	7F	40	0100 0000	BF	E0	1110 0000	FF	80	1000 0000

Двоичным числам B и $B + 1$ всегда соответствуют коды Грея (зеркальные, или рефлексные, коды), характеризующиеся изменением только одного разряда. Коды Грея находят применение в электронно-механических датчиках физических величин, например, в датчиках перемещения стрелки весов, что позволяет исключить большие ошибки при одновременном переключении нескольких двоичных разрядов, которые возникают в датчиках с двоичной шкалой измерения.

Задача 5 (файл 1#08_05.asm). Вычислить произведение двух 4-разрядных десятичных чисел, переведенных в двоичный код, с использованием алгоритма

$$Oper1 \times Oper2 = Oper1 \times (B_H \cdot 2^8 + B_L) = Oper1 \times B_H \cdot 2^8 + Oper1 \times B_L,$$

где операнд *Oper*2 = $B_H B_L$ (B_H — старший байт, B_L — младший байт), и результат представить в упакованном двоично-десятичном коде. *Решение*:

```
            seg     Code
            JMP     MAIN
Ram         equ     800h            ; Ram = 0800h — начальный адрес RAM        Data Segment
Rsz         equ     800h            ; Rsz = 800h  — объем RAM (RAM Size)
Ssz         equ     20h             ; Ssz = 20h — размер стека (Stack Size)
            defseg  Prod_seg, start = Ram, class = Data
            seg     Prod_seg
AOper1:     dw      9599d           ; 9599d = 257Fh  }
AOper2:     dw      9976d           ; 9976d = 26F8h  }  Oper1 × Oper2 = 95759624d
AProd:      ds      8               ; AProd = 0804h
            defseg  Stack_seg, start = Ram + Rsz – Ssz, class = Data        Stack Segment
            seg     Stack_seg
            ds      Ssz
            defseg  Main_seg, start = 100h, class = Code                   Code Segment
            seg     Main_seg
; Подпрограмма вычисления произведений Oper1×B_H и Oper1×B_L
Mult:       LXI     H, 0            ; HL ← 0000h          Начало подпрограммы CALL Mult
            MVI     C, 8            ; C ← 08h — задание восьми циклов
LC2:        DAD     H               ; HL ← HL × 2
            RAL
            JNC     LC1
            DAD     D               ; HL ← HL + DE
            ACI     0
LC1:        DCR     C
            JNZ     LC2
            RET                     ; Конец подпрограммы CALL Mult
; Транслятор преобразует десятичные числа в двоичные: 9599d = 257Fh, 9976d = 26F8h
MAIN:       LXI     SP, Ram + Rsz   ; SP ← 1000h — инициализация стека
            LDA     AOper2          ; A ← M(0802h) = F8h — первый байт числа Oper2
            LHLD    AOper1          ; HL ← M(0801, 0800h) = 257Fh — число Oper1
            XCHG                    ; DE ← 257Fh (D ← 25h, E ← 7Fh)
            CALL    Mult            ; HL = 5308h, A = 24h, AHL = 257F×F8 = 245308h
            SHLD    AProd           ; M(0805, 0804h) ← 5308h
            STA     AProd + 2       ; M(0806h) ← 24h
            LDA     AOper2 + 1      ; A ← M(0803h) = 26h — второй байт операнда Oper2
            CALL    Mult            ; HL = 90DAh, A = 05h, AHL = 257F×26 = 590DAh
            XCHG                    ; DE ← 90DAh
            LHLD    AProd + 1       ; HL ← M(0806, 0805h) = 2453h
            DAD     D               ; HL ← HL + DE = 2453h + 90DAh = B52Dh
            ACI     0               ; A ← A + CY = 05 + CY
            SHLD    AProd + 1       ; M(0806, 0805h) ← B52Dh
            STA     AProd + 3       ; M(0807h) ← A = 05, M(0807 … 0804h) = Oper1×Oper2
; Преобразование двоичного кода в упакованный двоично-десятичный код
            LXI     H, 0            ; HL ← 0000h
; 080Bh … 0808h — адреса ячеек памяти конечного результата Prod = 95 75 96 24
```

```
        SHLD   AProd + 4      ; M(0809, 0808h) ← 0000h
        SHLD   AProd + 6      ; M(080B, 080Ah) ← 0000h
        MVI    C, 20h         ; C ← 20h — задание 32 циклов
LM1:    LXI    H, AProd       ; HL ← 0804h
        MVI    B, 4           ; B ← 04h — задание четырех циклов
        XRA    A              ; A ← 0
LM2:    MOV    A, M           ; A ← M(rp H)
        RAL
        MOV    M, A           ; M(rp H) ← A
        INX    H
        DCR    B
        JNZ    LM2
        LXI    H, AProd + 4   ; HL ← 0808h
        MVI    B, 4           ; B ← 04h — задание четырех циклов
LM3:    MOV    A, M           ; A ← M(rp H)
        ADC    A
        DAA
        MOV    M, A           ; M(rp H) ← A
        INX    H
        DCR    B
        JNZ    LM3
        DCR    C
        JNZ    LM1
        LHLD   AProd + 4      ; HL ← M(0808h)
        XCHG                  ; DE ← M(0808h)
        LHLD   AProd + 6      ; HL ← M(080Ah)
        end
```

Использованный алгоритм вычисления двухбайтовых двоичных чисел наглядно может быть представлен схемой:

$$\begin{array}{rcl} & 2\,4\,5\,3\,0\,8\;h & = Oper1 \times B_L = 257\mathrm{F}h \times \mathrm{F}8h, \\ + \quad 0\,5\,9\,0\,\mathrm{D}\,\mathrm{A}\;\;\;\; h & = Oper1 \times B_H = 257\mathrm{F}h \times 26h, \\ \hline 0\,5\,\mathrm{B}\,5\,2\,\mathrm{D}\,0\,8\;h & = Oper1 \times Oper2 = 257\mathrm{F}h \times 26\mathrm{F}8h. \end{array}$$

Задача 6 (файл 1#08_06.asm). Выполнить сортировку чисел по возрастанию в массиве, расположенном по адресам от *Abeg* до *Aend* – 1 (*Abeg* < *Aend* – 1). Решение основано на последовательном многократном сравнении двух смежных чисел x и y и их перестановке при $x > y$.

Решение:

```
Ram     equ    800h           ; Ram = 0800h — начальный адрес RAM          Data Segment
Rsz     equ    800h           ; Rsz = 800h  — объем RAM (RAM Size)
Ssz     equ    20h            ; Ssz = 20h — размер стека (Stack Size)
        defseg Sort_seg, start = Ram, class = Data
        seg    Sort _seg
        db     'Novoselzeva', 0 ; ASCII-коды имени Novoselzeva = 4E 6F 76 6F 73 65 6C 7A
Sort    ds     12             ;                  65 76 61 00 — массив для сортировки
        defseg Stack_seg, start = Ram + Rsz – Ssz, class = Data      Stack Segment
        seg    Stack_seg
        db     'Stack ! Stack ! ' ; 16 байт = Ssz/2 (визуализация стека в отладчике)
```

```
        ds      Ssz/2           ; Ssz/2 = 16 байт
        defseg  Main_seg, start = 100h, class = Code                    Code Segment
        seg     Main_seg
        LXI     SP, Ram + Rsz   ; SP ← 1000h — инициализация стека
; Копирование сортируемых чисел
        LXI     H, Ram          ; HL ← 0800h — начальный адрес сортируемых чисел
        LXI     D, Sort         ; DE ← 080Ch — начальный адрес области копирования
        MVI     B, 12           ; B ← 0Ch — количество сортируемых чисел
LM:     MOV     A, M            ; A ← M(HL)
        STAX    D               ; M(DE) ← A
        INX     H
        INX     D
        DCR     B
        JNZ     LM
; Программа сортировки
        LXI     H, Sort         ; HL ← Sort = 080Ch
        LXI     B, Sort – Ram   ; BC ← Sort – Ram = ΔA — размер массива
L0:     PUSH    B
        PUSH    H
        MVI     E, 0  ; Сброс флага перестановок чисел: E = 0 — не было перестановок чисел
; при сравнении всех соседних пар чисел x и y массива ΔA, E = 1 — были перестановки чисел
L1:     MOV     A, M            ; A ← x
        INX     H
        CMP     M               ; Сравнение двух соседних чисел в памяти x и y
        JC      L2
        JZ      L2
        MOV     D, M            ; D ← y
        MOV     M, A            ; Перестановка двух соседних чисел
        DCX     H
        MOV     M, D
        MVI     E, 1            ; Установка значения флага E = 1
        INX     H
L2:     DCX     B
        MOV     A, C
        ORA     B
        JNZ     L1
        POP     H
        POP     B
        ORA     E
        JNZ     L0
        end
```

Вычисление объема массива чисел ΔA, подлежащих сортировке, можно сделать и в программе, чтобы не использовать команды PUSH и POP:

```
L0:     LXI     H, Ram  ; Начало вычисления объема массива чисел ΔA, подлежащих
        LXI     B, Sort ;    сортировке, в предположении, что объем массива может
        MOV     A, C    ;    потребовать и двухбайтового представления
        SUB     L
```

```
MOV     C, A
MOV     A, B
SBB     H
MOV     B, A        ; BC ← ΔA = Sort – Ram (вычисление закончено)
```

Директивы макрокоманд. Эти директивы позволяют повторяющимся блокам программы, различающимися только некоторыми параметрами, присвоить имя с перечнем формальных параметров (*dummy or formal parameter*), а затем любое число раз использовать только имя с указанием фактических параметров (*actual parameter*) вместо повторения текста всего блока в разных местах программы. Транслятор же вместо имен блоков подставит все команды МП, входящие в этот блок, и введет фактические параметры — транслятор по имени блока создаст его макрорасширения (*Macro Expansion*). Определенные директивами макрокоманд блоки программы называются *макрокомандами* (*Macro Command, Macro Instruction*), или *макросами* (*Macros*). Оформление программ, содержащих макрокоманды, имеет вид:

```
Name_M  %MACRO  Par1, Par2, Par3, Par4, … ; Директива %MACRO (Macro Definition)
        %LOCAL  Name1, Name2, Name3, … ; Директива %LOCAL (Definition Local Name)
          ∴①            ; ① — операторы ассемблера (инструкции и директивы)
        [%EXITM]        ; Директива прекращения макрогенерации (Exit Macro Generation)
          ∴②            ; ② — операторы ассемблера (инструкции и директивы)
        %ENDM           ; Директива конца макроопределения (End Macro Definition)
          ∴                             ; Основная программа
        Name_M  Mg11, Mg12, Mg13, … ; Макрорасширение 1 (Macro Generation)
          ∴                             ; Основная программа
        Name_M  Mg21, Mg22, Mg23, … ; Макрорасширение 2 (Macro Expansion 2)
          ∴                             ; Основная программа
```

Директива %MACRO связывает с именем макрокоманды *Name_M* некоторое число формальных параметров *Par*1, *Par*2, *Par*3, … . Вызов макрокоманды производится строкой, содержащей имя макрокоманды *Name_M* и такое же число фактических параметров (допустимое число параметров ограничено только длиной строки операторов ассемблера, равной 120 символам). Транслятор имеет генератор макрорасширений, который формальные параметры *Par*# (# = 1, 2, …) заменяет фактическими параметрами *Mg*1# для макрорасширения 1 и *Mg*2# для макрорасширения 2. Соответствие формальных и фактических параметров определяется последовательностью их задания в макроопределении и макрорасширении.

Директива %LOCAL объявляет перечисленные в ней имена переменных и констант действительными только внутри макроопределения, а генератор макрорасширений заменяет их уникальными именами ??××××, где ×××× = 0000 ÷ 9999 — 4-разрядные десятичные числа. Это исключает появление в программе одинаковых имен, имеющих разные численные значения.

Директива %EXITM (необязательная) прекращает генерацию транслятором макрорасширения, если требуется получить только его часть. При этом в первой части макрокоманды не должно быть ссылок на имена, определенные во второй ее части. Эта директива обычно используется совместно с условными директивами.

Директива %ENDM указывает транслятору конец макрокоманды и конец действия локальных имен.

Предупреждение:

1. Символические имена меток, переменных и констант должны быть расположены с первой позиции строки (перед именем не должно быть пробела).

2. Перед именем, вызывающим макрорасширение, должен быть хотя бы один пробел.

Пример 10:

```
Alpha    equ    39h                ; Alpha = 39h
Beta     db     2Ch, 'IBM PC'      ; M(Beta) = 2Ch, M(Beta+3) = 4Dh — ASCII-код буквы M
           ∴                        ; Основная программа
Add3     %macro Par1, Par2, Par3   ; Macro Definition Add3
         %local LJ1, LJ2, Beta     ; Definition Local Name LJ1, LJ2, Beta
; Register Pair BC = [Par1 – Par2 – Beta + M(Par3)]д — дополнительный код результата
         MVI    B, 0               ; B ← 0
         MVI    A, Par1            ; A ← Par1
         SUI    Par2 + Beta        ; A ← Par1 – Par2 – Beta
         JNC    LJ1
         DCR    B                  ; B ← B – 1, if CY = 1
LJ1:     LXI    H, Par3
         ADD    M                  ; A ← Par1 – Par2 – Beta + M(Par3)
         JNC    LJ2
         INR    B                  ; B ← B + 1, if CY = 1
LJ2:     MOV    C, A               ; C ← Par1 – Par2 – Beta + M(Par3)
Beta     equ    5                  ; Beta = 5
         %endm                     ; End Macros
Gamma    equ    9Eh                ; Gamma = 9Eh
; Macro Expansion 1 (Macros Add3): Par1 = Alpha + Ah, Par2 = Gamma, Par3 = Beta
         Add3   Alpha + Ah, Gamma, Beta
           ∴                       ; Основная программа
; Macro Expansion 2 (Macros Add3): Par1 = 55h, Par2 = Alpha, Par3 = Beta +3
         Add3   55h, Alpha, Beta + 3
           ∴                       ; Основная программа
         end                       ; End Program
```

Имя *Beta* используется в программе два раза: в основной программе оно определено как адрес числа 2C*h*, а в макрокоманде — как байт-константа, но конфликта между ними не возникает, так как имя *Beta* в макрокоманде объявлено локальным — ее действие не выходит за пределы макрокоманды.

После трансляции листинг с исключенными столбцами адресов и машинных кодов будет иметь вид:

```
Alpha    equ    39h
Beta     db     2Ch, 'IBM PC'
           ∴                       ; Основная программа
;Add3    %macro Par1, Par2, Par3   ; Macro Definition Add3
;        %local LJ1, LJ2, Beta     ; Definition Local Name LJ1, LJ2, Beta
; Register Pair BC = Two's Complement [Par1 – Par2 + M(Par3)]
;        MVI    B, 0               ; B ← 0
         MVI    A, Par1            ; A ← Par1
         SUI    Par2 + Beta        ; A ← Par1 – Par2 – Beta
         JNC    LJ1
,        DCR    B                  ; B ← B – 1, if CY = 1
;LJ1:    LXI    H, Par3
         ADD    M                  ; A ← Par1 – Par2 – Beta + M(Par3)
         JNC    LJ2
         INR    B                  ; B ← B + 1, if CY = 1
```

```
; LJ2:     MOV    C, A              ; C ← Par1 – Par2 – Beta + M(Par3)
; Beta     equ    5
;          %endm                    ; End Macros
Gamma      equ    9Eh               ; Gamma = 9Eh
; Macro Expansion 1 (Macros Add3): Par1 = Alpha + 0Ah, Par2 = Gamma, Par3 = Beta
          ; Add3   Alpha + Ah, Gamma, Beta
; Register Pair  BC = Two's Complement [Par1 – Par2 – Beta + M(Par3)]
          MVI    B, 0              ; B ← 0
          MVI    A, Alpha + 0Ah    ; A ← Par1
          SUI    Gamma + ??0002    ; A ← Par1 – Par2 – Beta
          JNC    ??0000
          DCR    B                 ; B ← B – 1, if CY = 1
??0000:   LXI    H, Beta
          ADD    M                 ; A ← Par1 – Par2 – Beta + M(Par3)
          JNC    ??0001
          INR    B                 ; B ← B + 1, if CY = 1
??0001:   MOV    C, A              ; C ← Par1 – Par2 – Beta + M(Par3) = 43h – 9Eh – 5 + 2Ch = FFCCh
??0002    equ    5
;         %endm                    ; End Macros (строка не выводится, если не задан параметр SC)
            ∴                      ; Основная программа
; Macro Expansion 2 (Macros Add3): Par1 = 55h, Par2 = Alpha, Par3 = Beta +3
          ; Add3   55h, Alpha, Beta + 3
; Register Pair  BC = Two's Complement [Par1 – Par2 – Beta + M(Par3)]
          MVI    B, 0              ; B ← 0
          MVI    A, 55h            ; A ← Par1
          SUI    Alpha + ??0005    ; A ← Par1 – Par2 – Beta
          JNC    ??0003
          DCR    B                 ; B ← B – 1, if CY = 1
??0003:   LXI    H, Beta + 3
          ADD    M                 ; A ← Par1 – Par2 – Beta + M(Par3)
          JNC    ??0004
          INR    B                 ; B ← B + 1, if CY = 1
??0004:   MOV    C, A              ; C ← Par1 – Par2 – Beta + M(Par3) = 55h – 39h – 5 + 4Dh = 0064h
??0005:   equ    5
;         %endm                    ; End Macros (строка не выводится, если не задан параметр SC)
            ∴                      ; Основная программа
          end
```

Комментарии на русском языке добавлены в листинг уже после трансляции. Для вывода в листинг *name*.prn макрорасширений транслятор из командной строки компьютера следует запускать командой

*avmac*85[.exe] *name*[.asm] [*xr*] *SM* [*SC*]

(параметр *SC* можно не задавать). Параметры *SM* (*ShowMacs*) и *SC* (*ShowComments*) управляют выводом в листинг макрорасширений, причем параметр *SC* управляет выводом в каждом макрорасширении только одной строки, содержащей закомментированную директиву %ENDM, идентифицирующую конец макрорасширения.

Исходный текст макрокоманды выводится в листинг в виде комментария — начинается с точки с запятой. Если параметр *SM* транслятору не задан, то текст макрорасширений 1 и 2

в листинг не выводится. Генератор макрорасширений производит сквозную нумерацию локальных имен ??×××× по всем макрорасширениям всех макрокоманд, имеющихся в программе. Локальным меткам LJ1, LJ2 и константе *Beta* присваиваются имена ??0000 ÷ ??0005, что исключает появление в программе одинаковых имен, имеющих разные численные значения.

Задача 7 (файл 1#08_07.asm). Для микроконтроллера, реализованного на МП 8085*A*, написать программу часов, производящих счет меток времени 1/100 секунды, поступающих от таймера. Микроконтроллер содержит ПЗУ 8К×8 бит по адресам 0000*h* ÷ 1FFF*h* и ОЗУ 2К×8 бит по адресам 2000*h* ÷ 27FF*h*. Использовать вход запроса прерывания *RST* 6.5. *Решение*:

```
; Адрес ячейки памяти ОЗУ:   2000h     2001h     2002h     2003h
;                            Часы      Минуты    Секунды   1/100 секунды
        seg     Code        ; ROM (начальный адрес равен 0000h)
        JMP     MAIN        ; Переход на выполнение основной программы
        defseg  S_rst65, start = 34h     ; 6,5 × 8 = 52d = 34h
        seg     S_rst65     ; ROM
; Подпрограмма обработки прерывания RST 6.5
        PUSH    PSW         ; Сохранение в стеке состояния прерванной основной программы
        PUSH    H           ;      (подпрограмма RST 6.5 использует регистры A, F и rp H)
        LXI     H, Ram + 3  ; HL = 2003h — адрес хранения сотых долей секунды
; Десятичный счет сотых долей секунды
;       DCX     H           ; Если использовать эту команду, то в предыдущей команде
        MOV     A, M        ;      операнд Ram + 3 следует заменить операндом Ram + 4
        ADI     1           ; Двоичный счет сотых долей секунды
        DAA                 ; Десятичная коррекция
        MOV     M, A
;       CPI     99h     ; Подпрограмма работает правильно и при отсутствии этой команды
        JNZ     LR1
;       MVI     M, 0    ; Подпрограмма работает правильно и при отсутствии этой команды
; Десятичный счет секунд
        DCX     H
        MOV     A, M
        ADI     1           ; Двоичный счет секунд
        DAA                 ; Десятичная коррекция
        MOV     M, A
        CPI     60h
        JNZ     LR1
        MVI     M, 0
; Десятичный счет минут
        DCX     H
        MOV     A, M
        ADI     1           ; Двоичный счет минут
        DAA                 ; Десятичная коррекция
        MOV     M, A
        CPI     60h
        JNZ     LR1
        MVI     M, 0
; Десятичный счет часов
        DCX     H
        MOV     A, M
```

```
        ADI   1          ; Двоичный счет часов
        DAA              ; Десятичная коррекция
        MOV   M, A
        CPI   24h
        JNZ   LR1
        MVI   M, 0
LR1:    POP   H          ; Восстановление состояния прерванной основной программы
        POP   PSW
        EI               ; Разрешение прерываний
        RET              ; Возврат из подпрограммы обработки прерывания RST 6.5
Ram     equ   2000h      ; Ram = 2000h — начальный адрес RAM
Rsz     equ   800h       ; Rsz = 0800h — объем RAM (RAM Size)
Ssz     equ   20h        ; Ssz = 20h — размер стека (Stack Size)
        defseg Clock_seg, start = Ram, class = Data                  ; Data Segment
        seg   Clock_seg
        ds    5          ; Резервирование в ОЗУ пяти ячеек памяти
        defseg Stack_seg, start = Ram + Rsz – Ssz, class = Data      ; Stack Segment
        seg   Stack_seg
        ds    Ssz
        defseg Main_seg, start = 100h, class = Code                  ; Code Segment
        seg   Main_seg   ; ROM
MAIN:   LXI   SP, Ram + Rsz
        MVI   A, 0Dh     ; A ← 0Bh = 0000 1101 (A3 = MCE = 1, A1 = M6.5 = 0)
        SIM              ; Разрешение прерывания по входу RST 6.5
        LXI   H, 5823h
        SHLD  Ram        ; Начальная установка часов и минут (23 часа 58 минут)
        LXI   H, 56h
        SHLD  Ram + 2    ; Начальная установка секунд и 1/100 сек (56,00 секунд)
        EI               ; Общее разрешение прерывания
LM1:    JMP   LM1        ; Свернутая в точку основная программа
        end              ; Конец программы
```

Данный вариант подпрограммы счета меток времени 1/100 секунды выполняется за минимально возможное время для рассмотренного типа часов.

Если использовать три закомментированные команды, то в подпрограмме можно выделить четыре группы по 8 команд, различающихся только операндом команды CPI. Понятно, что такие группы команд можно оформить или в виде макрокоманды, или в виде подпрограммы. На первый взгляд может показаться, что в виде подпрограммы оформить группы команд невозможно из-за команды условного перехода JNZ LR1, иногда досрочно заканчивающей выполнение подпрограммы — до выполнения команды возврата из подпрограммы RET. Но это не так, что и будет показано в *решении задачи* 9.

Задача 8 (файл 1#08_08.asm). Для *задачи* 7 написать подпрограмму обработки прерывания *RST* 6.5 с использованием макрокоманды *Count*. *Решение*:

```
; Подпрограмма обработки прерывания RST 6.5
        PUSH  PSW        ; Сохранение в стеке состояния прерванной основной программы
        PUSH  H          ;      (подпрограмма RST 6.5 использует регистры A, F и rp H)
        LXI   H, Ram + 4 ; HL = 2003h — адрес хранения сотых долей секунды
Count   %macro Par       ; Define Macros Count
        DCX   H          ;
```

```
        MOV     A, M        ;
        ADI     1           ; Двоичный счет
        DAA                 ; Десятичная коррекция
        MOV     M, A
        CPI     Par
        JNZ     LR1
        MVI     M, 0
        %endm               ; End Macros
        Count   99h         ; Счет 1/100 секунд
        Count   60h         ; Счет секунд
        Count   60h         ; Счет минут
        Count   24h         ; Счет часов
LR1:    POP     H           ; Восстановление состояния прерванной основной программы
        POP     PSW
        EI                  ; Разрешение прерываний
        RET                 ; Возврат из подпрограммы обработки прерывания RST 7.5
```

Текст подпрограммы уменьшился примерно в два раза, что, безусловно, полезно для улучшения ее восприятия. После трансляции этой подпрограммы будут получены четыре макрорасширения, полностью соответствующие тексту этой подпрограммы, приведенному в *решении задачи* 7. Полученный после трансляции программный код на 5 байт больше, чем в *решении задачи* 7 (появятся команды DCX H, CPI 99*h* и MVI M, 0 при счете 1/100 секунд). Время выполнения последней подпрограммы несколько больше времени выполнения предыдущей за счет лишних трех команд, введенных для счета сотых долей секунд.

Задача 9 (файл 1#08_09.asm). Написать программу по условиям *задачи* 7 с использованием подпрограммы *Count* в подпрограмме обработки прерывания *RST* 6.5. *Решение*:

```
        ∴                   ; Начало такое же, что и в решении задачи 7
;Подпрограмма обработки прерывания RST 6.5
        PUSH    PSW         ; Сохранение в стеке состояния прерванной основной программы
        PUSH    H           ;       (подпрограмма RST 7.5 использует регистры A, F, B и гр H)
        PUSH    B
        LXI     H, Ram + 4  ; HL = 2003h — адрес хранения сотых долей секунды
        MVI     B, 99h      ; Передача подпрограмме Count параметра 99
        CALL    Count       ; Счет сотых долей секунды (1/100)
        MVI     B, 60h      ; Передача подпрограмме Count параметра 60
        CALL    Count       ; Счет секунд
        MVI     B, 60h      ; Передача подпрограмме Count параметра 60
        CALL    Count       ; Счет минут
        MVI     B, 24h      ; Передача подпрограмме Count параметра 24
        CALL    Count       ; Счет часов
        JMP     LR2
LR1:    INX     SP          ; Компенсация указателя стека SP из-за досрочного выхода из
        INX     SP          ;       подпрограммы Count по команде условного перехода JNZ LR1
LR2:    POP     B           ; Восстановление состояния прерванной основной программы
        POP     H
        POP     PSW
        EI                  ; Разрешение прерываний
        RET                 ; Возврат из подпрограммы обработки прерывания RST 7.5
        ∴                   ; Продолжение такое же, что и в решении задачи 7
```

```
        defseg  Main_seg, start = 100h, class = Code          ; Code Segment
        seg     Main_seg    ; ROM
; Подпрограмма Count десятичного счета
Count   DCX     H
        MOV     A, M        ;
        ADI     1           ; Двоичный счет
        DAA                 ; Десятичная коррекция
        MOV     M, A
        CMP     B
        JNZ     LR1
        MVI     M, 0
        RET                 ; Конец подпрограммы Count
MAIN:   LXI     SP, Ram + Rsz
          ∴                 ; Окончание такое же, что и в решении задачи 7
```

Программный код этой подпрограммы обработки прерывания *RST* 7.5 только на два байта меньше, чем в *решении задачи* 7 (37 байт), но время выполнения значительно больше из-за увеличения числа стековых операций. Малый выигрыш в объеме программного кода объясняется тем, что команда CALL *Count* трехбайтовая, а заменяет она только 12 байт программного кода при необходимости введения дополнительных команд (PUSH B, POP B, MVI *d*8, JMP LR2 и INX SP). При больших же размерах подпрограмм CALL *addr* (десятки, сотни байт) будут получаться значительные выигрыши в объеме программного кода. Следует также иметь в виду, что чем большее число раз подпрограмма вызывается, тем больший будет получен выигрыш.

Несомненно, что логика построения программ при использовании подпрограмм становится более "прозрачной" — обеспечивается модульность программ, значительно облегчающая их проектирование и документирование.

Условные директивы. Эти директивы используются для управления условной трансляцией блоков операторов — трансляция блока зависит от проверки заданного условия.

Директивы %IF, %ELSEIF, %ELSE и %ENDIF имеют формат (директивы %ELSEIF и %ELSE являются необязательными):

```
%IF       Express1 ; Блок операторов ① транслируется, если выражение
  ∴①               ;     Express1 = true (тогда остальные блоки не транслируются)
[%ELSEIF  Express2 ; Блок операторов ② транслируется, если выражения
  ∴②]              ;     Express1 = false и Express2 = true
[%ELSEIF  Express3 ; Блок операторов ③ транслируется, если выражения
  ∴③]              ;     Express1 = false, Express2 = false и Express3 = true
[%ELSEIF  Express4 ; Блок операторов ④ транслируется, если выражения Express1 = false,
  ∴④]              ;     Express2 = false, Express3 = false и Express4 = true
[%ELSE             ; Блок операторов ⑤ транслируется, если выражения Express1 = false,
  ∴⑤]              ;     Express2 = false, Express3 = false и Express4 = false
%ENDIF             ; Конец условной директивы
```

Всегда транслируется только один блок операторов — первый блок от начала, у которого выражение *Express#* = *true*. Если все выражения *Express#* = *false*, то транслируется последний блок. Число директив %ELSEIF не ограничено.

Допустимо любое выражение *Express#*, которое может быть вычислено транслятором (результатом вычислений должно быть число). В выражениях можно использовать переменные, константы и операторы, описанные выше. Результат вычисления транслятором выражения *Ex-*

press# обозначим через ►*Express*#◄. Принятие транслятором решения основано на соотношениях:

►*Express*#◄ = 0 ⇒ *Express*# = *false*; например, ►3 shr 2◄ = 0 ⇒ 3 shr 2 = *false*,

►*Express*#◄ ≠ 0 ⇒ *Express*# = *true*; например, ►9 shr 2◄ = 2 ⇒ 9 shr 2 = *true*

(конечно, в выражениях для предоставления возможности принятия одного из двух взаимоисключающих решений должны использоваться и переменные, а не одни только константы).

Пример 11 (применение условных директив):

```
%if     0       ; Expression1 = false (начало условной директивы)
MVI     L, 77h
%elseif 0       ; Expression2 = false
RAR
%elseif 1       ; Expression3 = true
CMA             ; Транслируется только эта команда
%elseif 1       ; Expression4 = true
STC
%else
XCHG
%endif          ; Конец условной директивы
```

В этом примере блоки операторов представлены одной командой и для всех выражений использованы только константы 0 (*false*) и 1 (*true*). Далее для уменьшения программ директива %ELSEIF использоваться не будет, а в выражения будут входить символические имена и операторы.

Пример 12 (применение условных директив):

```
Alpha   equ     0FFABh
Beta    equ     55h
Gamma   equ     0ABh        ; ►Alpha + Beta◄ = 0000h ⇒ Alpha + Beta = false
        %if     Alpha + Beta ; Начало условной директивы 1
        MVI     A, Beta     ; Команда не транслируется
        %else
        LXI     D, Gamma    ; DE ← 00ABh
        MVI     L, 77h      ; L ← 77h
        %endif              ; Конец условной директивы 1
           ∴                ; ►Beta + Gamma◄ = 0100h ≠ 0 ⇒ Beta + Gamma = true
        %if     Beta + Gamma ; Начало условной директивы 2
        MVI     C, Beta     ; C ← 55h
        %else
        LXI     D, Alpha    ; Команда не транслируется
        LXI     H, Gamma    ; Команда не транслируется
        %endif              ; Конец условной директивы 2
           ∴                ; ►Beta shr 5◄ = 02h ≠ 0 ⇒ Beta shr 5 = true
        %if     Beta shr 5  ; Начало условной директивы 3
        MVI     A, Beta shr 5 ; A ← 02h
        %else
        LXI     D, Gamma/7  ; Команда не транслируется
        %endif              ; Конец условной директивы 3
```

В условных директивах не допускаются ссылки вперед, т. е. нельзя в них использовать символические имена, которые будут определены после применения условных директив.

Можно использовать вложенные условные директивы, которые разрешается помещать в любом месте блоков операторов как до директивы %ELSE, так и после нее. Допускается более 256 уровней вложенности.

Пример 13 (применение вложенных условных директив):

```
Alpha   equ   0FFABh
Beta    equ   55h          ; ▶Beta/Alpha◀ = 0 ⇒ Beta/Alpha = false
Gamma   equ   0ABh         ; ▶Beta mod 7◀ = 1 ⇒ Beta mod 7 = true
  ; Условные директивы с одним уровнем вложенности
        %if   Beta/Alpha   ; #1 — начало условной директивы
        MVI   A, Beta      ; Команда не транслируется
        %else
        LXI   D, Gamma     ; DE ← 00ABh
        %if   Beta mod 7   ; #2 — начало вложенной условной директивы
        MVI   A, Beta mod 7 ; A ← 01h
        %else
        LXI   D, Alpha     ; Команда не транслируется
        LXI   H, Gamma     ; Команда не транслируется
        %endif             ; #2 — конец вложенной условной директивы
        MVI   C, 77h       ; C ← 77h
        %endif             ; #1 — конец условной директивы
          ∴                ; ▶Alpha + Beta – 9◀ = FFF7h ≠ 0 ⇒ Alpha + Beta – 9 = true
          ∴                ; ▶Beta – 55h◀ = 0 ⇒ Beta – 55h = false
        %if   Alpha + Beta – 9 ; #3 — начало условной директивы
        %if   Beta – 55h   ; #4 — начало вложенной условной директивы
        MVI   A, Beta      ; Команда не транслируется
        %else
        LXI   D, Alpha     ; DE ← FFABh
        LXI   H, Gamma     ; HL ← 00ABh
        %endif             ; #4 — конец вложенной условной директивы
        MVI   A, Beta      ; A ← 55h
        %else
        LXI   D, Gamma     ; Команда не транслируется
        MVI   E, 77h       ; Команда не транслируется
        %endif             ; #3 — конец условной директивы
          ∴                ; ▶Alpha eq Beta◀ = 0 ⇒ Alpha eq Beta = false
  ; ▶Beta/36h◀ = 1 ⇒ Beta/36h = true, ▶Beta shr 7◀ = 0 ⇒ Beta shr 7 = false
  ; Условная директива с двумя уровнями вложенности
        %if   Alpha equ Beta ; #1 — начало условной директивы
        MVI   A, Beta      ; Команда не транслируется
        %else
        LXI   D, Gamma     ; DE ← 00ABh
        %if   Beta/36h     ; #2 — начало вложенной условной директивы первого уровня
        MVI   A, Beta      ; A ← 55h
        %if   Beta shr 7   ; #3 — начало вложенной условной директивы второго уровня
        MVI   A, Beta shr 5 ; Команда не транслируется
        %else
```

```
        LXI     H, Gamma/7      ; HL ← 0018h
        %endif                  ; #3 — конец условной директивы второго уровня
        %else
        LXI     D, Alpha        ; Команда не транслируется
        LXI     H, Gamma        ; Команда не транслируется
        %endif                  ; #2 — конец условной директивы первого уровня
        MVI     C, 77h          ; C ← 77h
        %endif                  ; #1 — конец условной директивы
```

Задача 10. С помощью условных директив оптимизировать подпрограмму обработки прерывания RST 6.5 в *задаче* 8 для получения программы, эквивалентной программе *задачи* 7.
Решение:

```
; Подпрограмма обработки прерывания RST 6.5
        PUSH    PSW             ; Сохранение в стеке состояния прерванной основной программы
        PUSH    H               ;       (подпрограмма RST 7.5 использует регистры A, F и rp H)
        LXI     H, Ram + 3      ; HL = 2003h — адрес хранения сотых долей секунды
Count   %macro  Par             ; Define Macros Count
        %if     Par ne 99h      ; Оператор отношений NE: если Par ≠ 99h, то Par ne 99h = true
        DCX     H               ; Эта команда не транслируется для макрорасширения Count 99h
        %endif
        MOV     A, M
        ADI     1               ; Двоичный счет
        DAA                     ; Десятичная коррекция
        MOV     M, A
        %if     Par eq 99h      ; Оператор отношений EQ: если Par = 99h, то Par eq 99h = true
        JNZ     LR1     ; Эта команда транслируется только для макрорасширения Count 99h
        %else
        CPI     Par             ; Эти три команды транслируются для остальных
        JNZ     LR1             ;   макрорасширений Count, когда Par ≠ 99h и Par eq 99h = false
        MVI     M, 0
        %endif
        %endm                   ; End Macros
        Count   99h             ; Счет 1/100 секунд
        Count   60h             ; Счет секунд
        Count   60h             ; Счет минут
        Count   24h             ; Счет часов
LR1:    POP     H               ; Восстановление состояния прерванной основной программы
        POP     PSW
        EI                      ; Разрешение прерываний
        RET                     ; Возврат из подпрограммы обработки прерывания RST 7.5
```

Директивы %SWITCH, %CASE, %DEFAULT и %ENDSW имеют формат:

```
% SWITCH    Express
%CASE       Express1  ; Блок операторов ① транслируется, если Express1 = Express
  ∴①                  ;       (тогда остальные блоки не транслируются)
[%CASE      Express2  ; Блок операторов ② транслируется, если Express1 ≠ Express
  ∴②]                 ;       и Express2 = Express
[%CASE      Express3  ; Блок операторов ③ транслируется, если Express1 ≠ Express,
  ∴③]                 ;       Express2 ≠ Express и Express3 = Express
```

```
[%DEFAULT          ; Блок операторов ④ транслируется, если Express1 ≠ Express,
  ∴④]              ;     Express2 ≠ Express и Express3 ≠ Express
%ENDSW             ; Конец условной директивы
```

Всегда транслируется только один блок операторов — первый блок от начала, для которого выражение *Express#* = *Express* (# = 1, 2 и 3). Если все выражения *Express#* ≠ *Express*, то транслируется последний блок. Число директив %CASE не ограничено. Выражения, используемые в директиве %SWITCH, имеют не два значения *false* и *true*, как в директиве %IF, а любые значения из диапазона 0000*h* ÷ FFFF*h*. Допустимо любое выражение, которое может быть вычислено транслятором (результатом вычислений должно быть число). *Пример* 14:

```
Alpha   equ     2A7Ch       ; Alpha = 2A7Ch
Beta    equ     55h         ; Beta = 0055h
Case    equ     0FEE8h      ; Case = FEE8h (можно использовать и другое имя)
        %switch Case        ; Express = FEE8h (начало условной директивы)
        %case   0           ; Express1 = 0
        LXI     D, Alpha    ; Команда не транслируется
        %case   2           ; Express2 = 2
        RAR                 ; Команда не транслируется
        %case   Alpha * 6   ; Express3 = Alpha * 6, ►Alpha * 6◄ = FEE8h
        MVI     A, Beta     ; A ← 55h
        %case   0FEE8h      ; Express4 = FEE8h
        MVI     L, 77h      ; Команда не транслируется
        %default
        MVI     A, Beta mod 7 ; Команда не транслируется
        %endsw              ; Конец условной директивы
```

Условная директива %SWITCH используется в основном для программирования альтернативных режимов работы интерфейсных БИС, для чего достаточно использовать выражения, представляющие собой константы 0, 1, 2, … . *Пример* 15:

```
; Программирование периферийного интерфейса 8255A (см. рис. 3.2, 3.3 и 3.7)
Cs4B    equ     4Bh     ; port = 4Bh — адрес регистра RGCW слова управления
Fetch   equ     2       ;                (Control Word) БИС 8255A
        %switch Fetch   ; Express = 2 (начало условной директивы)
        %case   0       ; Express1 = 0
Cw      equ     0BCh    ; Режимы: M1/In — порт PA, M1/Out — порт PB, In — порт PC
        %case   1       ; Express2 = 1
Cw      equ     0A6h    ; Режимы: M1/Out— порт PA, M1/In — порт PB, Out — порт PC
        %case   2       ; Express3 = 2
Cw      equ     0C6h    ; Режимы: M2 — порт PA, M1/In — порт PB
        %default
Cw      equ     80h     ; Режим M0/Out для всех портов
        %endsw          ; Конец условной директивы
          ∴             ; Результат трансляции: Cw = C6h
        MVI     A, Cw   ; A ← C6h — слово управления Cw
        OUT     Cs4B    ; RGCW ← C6h ⇒ режим M2 порта PA и режим M1/In порта PB
```

Директивы повторения %REPT, %ENDREPT и %FOR, %ENDFOR. Эти директивы (макрокоманды) облегчают заполнение данными (байтами и словами) областей памяти. Директивы %FOR могут быть вложены в директиву %REPT. Директива %REPT имеет один формат, а директива %FOR — четыре формата:

```
%REPT      Express      ;  Блок операторов ① транслируется число раз, равное
   ∴①                   ; вычисленному значению выражения ► Express ◄
%ENDREPT                ;  Конец директивы
   ∴
%FOR Var chars String   ;  Блок операторов ② транслируется столько раз, сколько
[db        Var]         ; символов (знаков) содержится в параметре String и переменной
[dw        Var]         ; Var на каждом шаге трансляции присваивается значение
  [∴②]                  ; очередного символа, которое должно быть определено
%ENDFOR                 ;  Конец директивы (пример: String = FgijKNOpqRStuvwz)
   ∴
%FOR Var in List        ;  Блок операторов ③ транслируется столько раз, сколько
[db        Var]         ; элементов перечислено в параметре List (но не более 16) и
[dw        Var]         ; переменной Var на каждом шаге трансляции присваивается
  [∴③]                  ; значение очередного элемента, которое должно быть определено
%ENDFOR                 ;  Конец директивы (пример: List = Beta, Tr, Source, Z, W)
   ∴
%FOR Var = Start to End ;  Блок операторов ④ транслируется Rept1 = End – Start + 1 раз
[db        Var]         ; и переменной Var на каждом шаге трансляции присваивается
[dw        Var]         ; значение Start, Start + 1, Start + 2 и т. д.
  [∴④]
%ENDFOR                 ;  Конец директивы
   ∴
%FOR Var = Start to End by Step  ; Блок операторов ⑤ транслируется
[db        Var]         ;          Rept2 = (End – Start + 1) : Step раз
[dw        Var]         ;  (деление с округлением в сторону большего целого) и
  [∴⑤]                  ;  переменной Var на каждом шаге трансляции присваивается
%ENDFOR                 ;  значение Start, Start + Step × 1, Start + Step × 2 и т. д.
   ∴
; Совместное использование директив повторений %REPT и %FOR
%REPT      Express
%FOR Var ××× ××× ×××    ; Любая из четырех директив %FOR. Число трансляций
   ∴⑥                   ;    блока операторов ⑥ задает директива %FOR, а число
%ENDFOR                 ;    повторений трансляций — выражение Express
%ENDREPT                ; Конец директивы
```

Если переменная *Var* в блоках операторов не используется, то символы в *String* и элементы в *List* могут быть не определены. Конечно, директивы %FOR предназначены в основном для использования с блоками операторов, содержащими одну из директив DB *Var* или DW *Var*.

Пример использования директивы %FOR ② приведен ниже в *задаче* 11. Директива %FOR ③ имеет подобное же назначение только вместо однобуквенных имен можно использовать как однобуквенные, так и полноценные символические имена переменных (не более 16 имен).

Директива %FOR ④ позволяет генерировать таблицы последовательных значений до 256 двоичных чисел, представляемых байтами или словами.

Пример 16:

```
Nbeg  equ     0FF80h  ; Nbeg = 0000 … FFFFh
Nvol  equ     0FFh    ; Nvol = 00 … FFh
      %for    Numb = Nbeg  to  Nbeg + Nvol
      dw      Numb    ; Запись в память двоичных чисел FF80 … FFFFh, 0000 … 007Fh
      %endfor
```

В *задаче* 4 для генерации таблицы 8-разрядных двоичных чисел в начале сегмента данных *Tabl_seg* можно было бы использовать директиву %FOR, исключив из программы 8 команд:

```
%for    Numb = 0 to 255 ; Запись в память, начиная с адреса Ram = 0800h, двоичных чисел
db      Numb            ;      00, 01, 02, ..., FFh
%endfor
```

Директива %FOR ⑤ позволяет генерировать таблицы значений с шагом *Step* до 32768 двоичных чисел, представляемых словами.

Пример 17:

```
Start   equ     300h        ; St = 0000 ... FFFFh
Vol     equ     400h        ; Vol = 0000 ... 7FFEh
Step    equ     100h        ; Sp = 0001 ... 7FFFh
        %for    Par = Start  to  Start + Vol  by  Step
        dw      Par         ; Запись в память пяти слов 0300 0400 0500 0600 0700h
        %endfor
```

Если в этой программе использовать директиву DB *Par*, то должно выполняться условие *Start* + *Vol* ≤ FF*h*. При значении шага *Step* = 1 директива %FOR ⑤ эквивалентна директиве %FOR ④, но можно генерировать более 256 чисел.

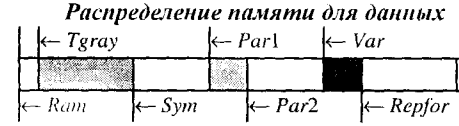
Совместное использование директив повторений %REPT и %FOR позволяет генерировать несколько периодов группы байт и слов. Блоки операторов могут содержать как директивы ассемблера, так и команды МП, которые будут транслированы в машинные коды.

Пример 18:

```
Ram     equ     1000h   ; Ram = 1000h — начальный адрес RAM
Rsz     equ     800h    ; Rsz = 800h — объем RAM
        defseg  Rep_seg, start = Ram, class = Data ;
        seg     Rep_seg ;
Abeg    equ     4       ; Abeg = 4
; 1. Директива повторения %REPT — генерация таблицы кода Грея
        org     Ram + Abeg  ; Ram + Abeg = 1004h
Tgray:                  ; Tgray — таблица кода Грея (см. табл. 1.14); метка Rept1 = 1004h
gray    teq     –1      ; gray = –1                идентифицирует начальный адрес
        %rept   256     ; 256 — число повторений трансляции блока операторов
gray    teq     gray + 1 ; gray = 00h ... FFh      Генерация кода Грея (см. рис. 1.26) ↓
        db      gray xor gray shr 1 ;              D7D6D5D4D3D2D1D0 ⊕ 0 D7D6D5D4D3D2D1
        %endrept        ; Конец директивы повторения %REPT
        ∴       ; Область памяти с адреса Ram по адрес Ram + Abeg – 1 автоматически
; 2. Директива повторений %FOR типа  %for Var chars String
P       equ     1       ; P = 01h       резервируется (не нужно использовать директиву DS)
S       equ     0FFh    ; S = FFh
N       equ     0       ; N = 00h
Sym:                    ; Метка Sym (можно использовать и другое имя) идентифицирует
; начальный адрес (1104h) массива данных, генерируемых директивой %FOR Sym
        %for    Sym chars PnPpppsssPpnPpppsssPpsPsP    ; В память, начиная с адреса 1104h,
        db      Sym     ; записываются 24 байта (числа 00h, 01h и FFh)
        %endfor         ; Конец директивы повторения %FOR
; 3. Директива повторений %FOR типа  %for Var in List
Par1:                   ; Метка Par1 идентифицирует начальный адрес (111Ch)
                        ;      массива данных, генерируемых директивой %FOR Par
```

Data Segment

Распределение памяти для данных



```
Beta    equ     55h         ; Beta = 55h
        %for    Par in P, p, Beta, S, s, Beta, N, n   ; Не более 16 параметров. В память, начиная
        db      Par         ;   с адреса 111Ch, записываются 8 байт (числа 01h, 00h, 55h и FFh)
        %endfor             ; Конец директивы повторения %FOR
; 4. Директива повторений %FOR типа  %for Var = Start to End
St      equ     3           ; St = 3      ; Rept1 = Ed - St + 1 = 7 — число повторений
Ed      equ     9           ; Ed = 9      ; Rept2 = (Ed - St + 1):Sp = 4 — число повторений
Sp      equ     2           ;  Sp = 2     ;   (деление с округлением в сторону большего целого)
Par2:                       ; Метка Par2 идентифицирует начальный адрес (1124h)
                            ;      массива данных, генерируемых директивой %FOR  Par
        %for    Par = St to Ed  ; Rept = Ed – St + 1 = 7 — число повторений
        db      Par         ; Запись в память, начиная с адреса 1124h, 7 групп из двух байт
        db      N           ;      m 00, где m = 3, 4, 5, 6, 7, 8, 9
        %endfor             ; Конец директивы повторения %FOR
; 5. Директива повторений %FOR типа  %for Var = Start to End by Step
Var:                        ; Метка Var идентифицирует начальный адрес (1132h)
                            ;      массива данных, генерируемых директивой %FOR  Var
        %for    Var = St to Ed by Sp ; Rept = (Ed – St + 1) : Sp = 4 — число повторений.
        db      Var         ; Запись в память, начиная с адреса 1132h, четырех байт 03 05 07 09
        %endfor             ; Конец директивы повторения %FOR
; 6. Совместное использование директив повторений %REPT и %FOR
Rept2:                      ; Метка Rept2 идентифицирует начальный адрес (1136h)
                            ;      массива данных, генерируемых директивами %REPT и %FOR
        %rept   2           ; 2 — число повторений трансляции блока операторов
        %for    Sym chars PnPpppsssPpnPppsssPpsPsPppppsPns   ; 32 символа в строке
        db      Sym         ; В память записываются, начиная с адреса 1136h, 64 байта
        %endfor             ; Конец директивы повторения %FOR
        %endrept            ; Конец директивы повторения %REPT
        defseg  Main_seg, start = 100h, class = Code ;                    Code Segment
        seg     Main_seg
        LXI     H, Tgray    ; HL ← 1004h
        LXI     H, Sym      ; HL ← 1104h
        LXI     H, Par1     ; HL ← 111Ch
        LXI     H, Par2     ; HL ← 1124h
        LXI     H, Var      ; HL ← 1132h
        LXI     H, Repfor   ; HL ← 1136h
        LDA     Tgray + 2   ; A ← M(1006) = 03h
        end
```

Эти команды использованы лишь для иллюстрации идентификации начальных адресов массивов данных с помощью меток.

Задача 11 (файл 1#08_11.asm). Написать программу вычисления в одной точке значения периодической автокорреляционной функции троичной последовательности длиною 57 символов (см. § 7.12 в книге [5])

+0++++−−−++0+++−−−++−+−++++++−+0−+−0−−−+−−0+−++−−++0−+−+00

(символы последовательности имеют три значения: +1, –1 и 0). *Решение*:

```
Ram     equ     1000h       ; Ram = 1000h — начальный адрес RAM                 Data Segment
Rsz     equ     800h        ; Rsz = 800h  — объем RAM (RAM Size)
        defseg  Corf_seg, start = Ram, class = Data
```

```
        seg     Corf_seg
        org     Ram + 8     ; Ram + 8 = 1008h
P       equ     1           ; P = +1
S       equ     0FFh        ; S = FFh = [–1]д
N       equ     0           ; N = 0
        %rept   2           ; 2 — число повторений трансляции блока операторов
        %for    Sym chars PnPpppsssPpnPppsssPpsPsPppppsPnsPsnsssPssnPsPpssPpnsPsPnn
        db      Sym         ; В память записывается 114 значений символов — два периода
        %endfor             ; Конец директивы повторения %FOR        троичной
        %endrept            ; Конец директивы повторения %REPT       последовательности
        defseg  Main_seg, start = 100h, class = Code ;                Code Segment
        seg     Main_seg
Length  equ     57          ; Length — длина троичной последовательности
Shift   equ     4           ; Shift = 0 ... Length – 1 (Shift = 0 — основной лепесток)
        LXI     D, Ram + 8 + Shift  ; rp D — адрес элементов X
        LXI     H, Ram + 8  ; rp H — адрес элементов Y
        MVI     C, Length – 1
        MVI     B, 0        ; B — значение автокорреляционной функции
LM5:    LDAX    D           ; A ← X
        CPI     1
        JC      LM1         ; Переход, если X = 0
        JZ      LM2         ; Переход, если X = +1
        MOV     A, M        ; A ← Y при X = –1 (X = FFh)
        CPI     1
        JC      LM1         ; Переход, если Y = 0
        MVI     A, 0FFh
        JZ      LM3         ; Переход, если Y = +1
        MVI     A, 1
LM3:    ADD     B           ; A ← A + B (+1 или –1)
        JMP     LM4
LM2:    MOV     A, M        ; A ← Y при X = +1
        ADD     B
LM4:    MOV     B, A
LM1:    INX     H
        INX     D
        DCR     C
        JNZ     LM5
        MOV     A, B
        STA     Ram         ; M(Ram) ← значение автокорреляционной функции
        end
```

Таблицу значений двух периодов символов троичной последовательности можно было бы задать и без использования директивы повторения %FOR:

```
%rept   2
db      1, 0, 1, 1, 1, 1, 0FFh, 0FFh, 0FFh, 1, 1, 0, 1, 1, 1, 0FFh, 0FFh, 0FFh, 1, 1, 0FFh
db      1, 0FFh, 1, 1, 1, 1, 1, 0FFh, 1, 0, 0FFh, 1, 0FFh, 0, 0FFh, 0FFh, 0FFh, 1, 0FFh
db      0FFh, 0, 1, 0FFh, 1, 1, 0FFh, 0FFh, 1, 1, 0, 0FFh, 1, 0FFh, 1, 0, 0
%endrept
```

Из этого примера видно, насколько директива %FOR облегчает ввод данных и увеличивает наглядность программы.

Директива управления файлами %INCLUDE. Эта директива используется для включения в исходный модуль файла другой программы. Директива %INCLUDE имеет форматы:

%INCLUDE *name_file* или %INCLUDE <*name_file*>.

При указании имени включаемого файла в виде *name_file* его поиск производится только в текущем каталоге, а при указании в виде <*name_file*> — только в каталоге, указанном в переменной окружения INCLUDE. Для этого в файл *autoexec*.bat следует включить строку

set INCLUDE=c:\LABORAT\AVSIM85\INCLUDE,

если включаемые файлы находятся в каталоге c:\LABORAT\AVSIM85\INCLUDE.

По умолчанию текст подключенного файла в листинг не выводится. Для вывода в листинг *name*.prn текста подключенного файла транслятор из командной строки компьютера следует запускать командой

*avmac*85[.exe] *name*[.asm] [*xr*] *SI*

(*SI* — *SnowIncs*). Если, например, 25 команд, отмеченных в *задаче* 11 вертикальной линией, выделить в файл 1#08_11*i*.inc, то их можно будет заменить одной директивой

%include <1#08_11*i*.inc>.

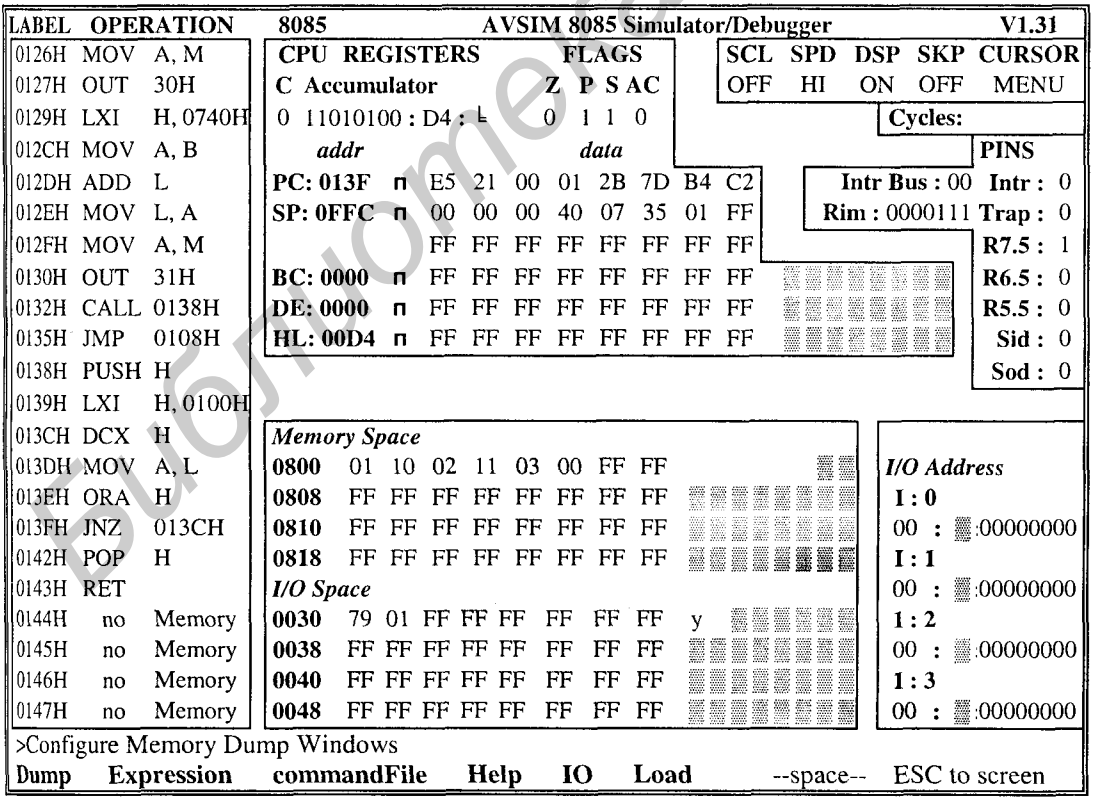


Рис. 1.27. Пример экрана отладчика

Отладчик фирмы *Avocet Systems, Inc.* Отладчик позволяет выполнить прогон всей программы (клавиша F1), выполнить программу в пошаговом режиме (клавиша F10), осуществить пошаговый возврат (клавиша F9), задать точки останова и др. Вид экрана отладчика изображен на рис. 1.27. Здесь для автоматической настройки экрана отладчика был использован командный файл 1#09_02.cmd (в самом файле может находиться только первый столбец):

```
LA1#09_02  ; LA — Load, A — Avocet, display — имя файла
D1A800h    ; D1 — дамп 1 (Memory Space), A — Address Memory,
           ; 800h — начальный адрес дампа памяти
D2AI:8     ; D2 — дамп 2 (I/O Space), AI: — Address I/O, 8 — начальный адрес дампа I/O
←0100      ; 0100 — адрес входа в программу
```

Дополнительную информацию можно найти в файле *debugger*.pdf на дискете.

1.9. Генератор и системный контроллер

Генератор тактовых сигналов 8224 и системный контроллер 8228/8238 фирмы *Intel* (отечественные ИС 580ГФ24 и 580ВК28/580ВК38) необходимы для построения законченного центрального процессорного устройства на МП 8080*A* (*CPU*, указанного на рис. 1.1) с тремя системными шинами — шинами данных, адреса и управления.

Генератор тактовых сигналов 8224. Структурная схема генератора изображена на рис. 1.28. Генератор состоит из усилителя (*Oscillator*) сигналов кварцевого резонатора, подключаемого к выводам $XTAL_1$ и $XTAL_2$ (*Crystal*), делителя частоты на 9 (*Clock Generator*), двух схем временно́й привязки сигналов $\overline{RESIN}$ и *RDYIN* к тактовому сигналу, выполненных на синхронных *D*-триггерах, и схемы формирования сигнала $\overline{STSTB}$ (*Status Strobe*), записывающего слово состояния *SW* во внешний регистр памяти. Для питания ИС используются два напряжения V_{CC} = +5 В и V_{DD} = +12 В ($I_{CC\,max}$ = 115 мА, $I_{DD\,max}$ = 12 мА).

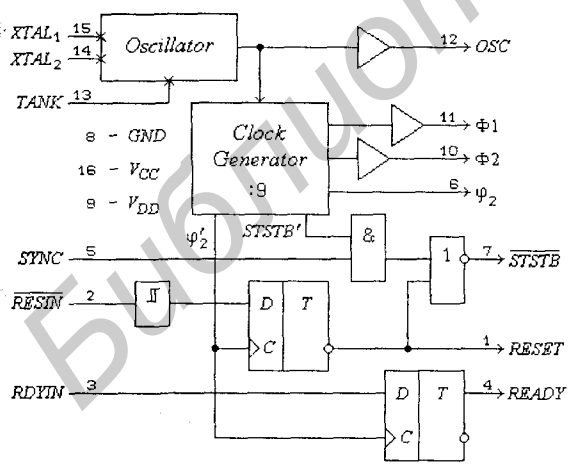


Рис. 1.28. Структурная схема генератора 8224

Временны́е диаграммы, поясняющие работу генератора, показаны на рис. 1.29. Частота выходного сигнала генератора *OSC* равна резонансной частоте кварцевого резонатора. Стандартное значение этой частоты равно $9 \cdot 2^{11}$ кГц = 18,432 МГц. Делением частоты сигнала *OSC* на 9 формируются сигналы Φ_1 и Φ_2 двухфазной синхронизации МП 8080*A*. С помощью двух драйверов (высоковольтных повторителей) высокий уровень сигналов Φ_1 и Φ_2 увеличивается с ТТЛ-уровня до $V_{OH} \geq 9{,}4$ В (остальные выходные сигналы имеют ТТЛ-уровень; сигнал φ_2 совпадает по форме с сигналом Φ_2). Вход *TANK* предназначен для подключения параллельного колебательного контура при работе на субгармониках кварцевого резонатора.

Длительность активного уровня сигнала *STSTB´* равна одному периоду сигнала *OSC* (50 нс при частоте f_{OSC} = 20 МГц). Выходной сигнал $\overline{STSTB}$ получается стробированием сигнала *STSTB´* сигналом *SYNC*, поступающим от МП, т. е. длительность активного уровня сигнала $\overline{STSTB} = 0$ тоже равна одному периоду сигнала *OSC*.

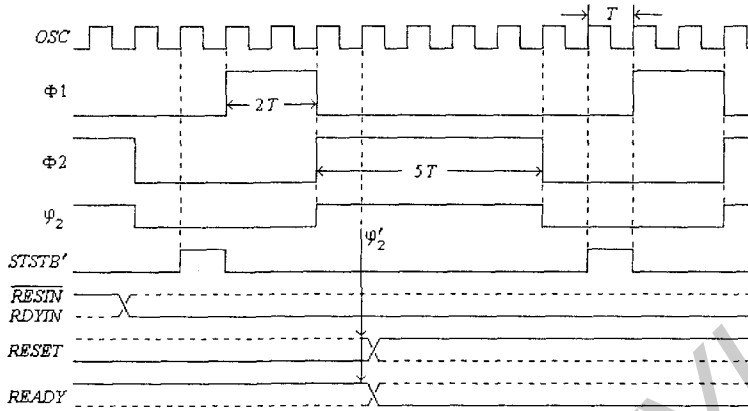


Рис. 1.29. Временны́е диаграммы работы генератора 8224

Сигнал $\overline{RESIN}$ подается на вход триггера Шмитта (см. § 5.3 в книге [5]), который способен формировать выходной сигнал с крутыми фронтами из медленно изменяющегося входного сигнала (ширина петли гистерезиса равна 0,25 В). Временна́я привязка входных сигналов сброса $\overline{RESIN}$ и готовности *RDYIN* производится сигналом φ_2, сдвинутым на один такт относительно сигнала φ_2.

Генератор выполнен по ТТЛШ-технологии и обеспечивает токи выходных сигналов:

I_{OL} = 15 мА (*OSC* и φ_2) и I_{OL} = 2,5 мА (остальные сигналы) при $V_{OL} \le 0{,}45$ В;
I_{OH} = –0,1 мА при $V_{OH} \ge 3{,}6$ В (*READY* и *RESET*);
I_{OH} = –0,1 мА (Φ_1 и Φ_2) при $V_{OH} \ge 9{,}4$ В;
I_{OH} = –1 мА при $V_{OH} \ge 2{,}4$ В (остальные сигналы).

Системный контроллер 8228/8238. Структурная схема системного контроллера изображена на рис. 1.30. Контроллер состоит из 8-разрядного приемопередатчика (*Transceiver*), 6-разрядного асинхронного потенциального регистра памяти (*Status Latch*) и дешифратора *DC*. Слово состояния *SW*, выдаваемое МП 8080*A* по шине данных D_{7-0} в первом такте каждого машинного цикла (при значении *SYNC* = 1), записывается в регистр сигналом $\overline{STSTB}$ = 0. Дешифратор *DC*, подключенный к регистру, вырабатывает по коду *SW* и сигналам *DBIN* и $\overline{WR}$ активный уровень одного из пяти системных сигналов управления $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}$, $\overline{INTA}$ и производит переключения направления передачи данных в приемопередатчике.

Ввод данных в МП (рис. 1.31) указывается значением сигнала *DBIN* = 1 и выполняется одним из сигналов управления: $\overline{MEMR}$ = 0 (чтение памяти — выборка команд программы и чтение данных), $\overline{I/OR}$ = 0 (чтение данных из внешних устройств) и $\overline{INTA}$ = 0 (выборка команд CALL *addr* или RST *n* из специального внешнего устройства, называемого контроллером прерываний). Электрические и временны́е параметры всех этих трех сигналов одинаковы, поэтому сигнал $\overline{MEMR}$ можно использовать и для чтения данных из внешних устройств при отображении их адресов на адресное простран-

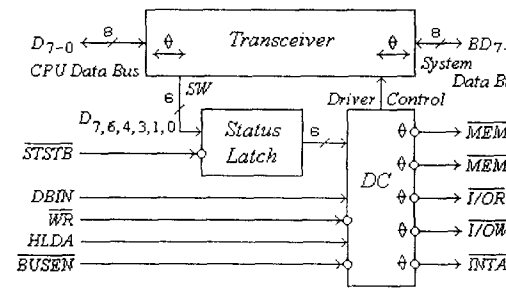


Рис. 1.30. Структурная схема системных контроллеров 8228/8238

ство памяти. Сигнал $\overline{INTA}$ предназначен для чтения данных из одного единственного внешнего устройства, поэтому эта операция не адресуется по шине адреса МП (безадресное чтение).

Если контакт $\overline{INTA}$ подключить через резистор 1 кОм к источнику питания +12 В, то на запрос прерывания $INT = 1$ от внешнего устройства из системного контроллера в МП поступит команда RST 7, машинный код которой равен FF*h*. Системная шина данных контроллера (*System Data Bus*) при этом блокируется. Этим обеспечивается одноуровневая система прерываний в простейших микроконтроллерах.

Вывод данных из МП указывается значением сигнала записи $\overline{WR} = 0$ и выполняется одним из сигналов управления $\overline{MEMW} = 0$ или $\overline{I/OW} = 0$. Все параметры этих сигналов одинаковы. Сигналы $\overline{MEMW}$ и $\overline{I/OW}$, выдаваемые системным контроллером 8238, показаны на рис. 1.31 штриховой линией (упреждающая запись), а контроллером 8228 — сплошной.

Коды слова состояния *SW* представлены в табл. 1.15. Каждый разряд *SW* содержит следующую информацию о состоянии (в скобках указаны названия разрядов):

$D_0 = 1$ — подтверждение прерывания (*INTA*);

$D_1 = 0$ — запись данных в память и вывод данных во внешнее устройство ($\overline{WO}$ — *Write/Output*);

$D_2 = 1$ — обращение к стеку (*STACK* — на шине адреса установлено содержимое *SP*);

$D_3 = 1$ — подтверждение останова (*HLTA*);

$D_4 = 1$ — вывод данных во внешнее устройство (*OUT* — на шине адреса установлено значение *port* второго байта команды OUT *port*);

$D_5 = 1$ — выполнение первого машинного цикла (*M*1 — выборка из памяти первого байта очередной команды);

$D_6 = 1$ — ввод данных из внешнего устройства (*INP* — на шине адреса установлено значение *port* второго байта команды IN *port*);

$D_7 = 1$ — чтение памяти (*MEMR*).

Из табл. 1.15 следует, что в качестве системных сигналов управления можно использовать сигналы:

$$\overline{MEMR} = \overline{D_7 \cdot DBIN},\ \overline{MEMW} = D_4 \vee \overline{WR},\ \overline{I/OR} = \overline{D_6 \cdot DBIN},\ \overline{I/OW} = \overline{D_4} \vee \overline{WR}.$$

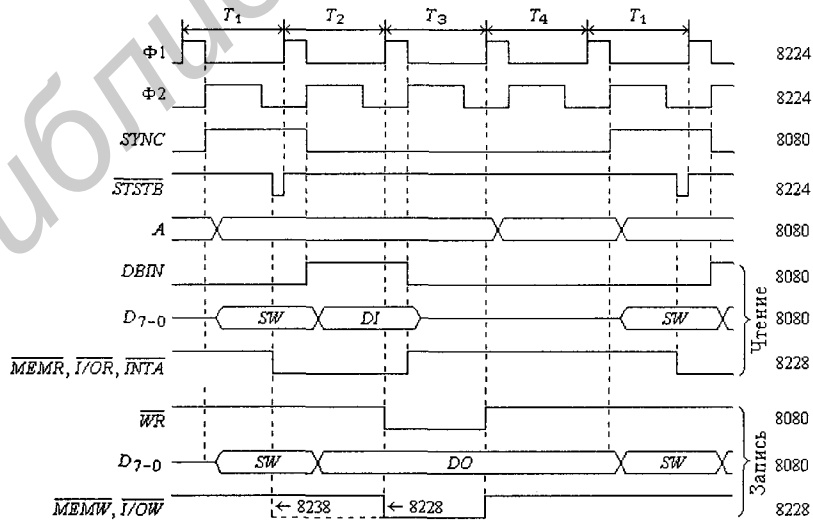


Рис. 1.31. Временны́е диаграммы сигналов процессора 8080*A*

Таблица 1.15. **Слово состояния МП 8080А**

№ *SW*	*SW*								Сигнал управления	Тип машинного цикла
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	1	0	1	0	0	0	1	0	$\overline{MEMR} = 0$	Выборка команды
2	1	0	0	0	0	0	1	0	$\overline{MEMR} = 0$	Чтение памяти
3	0	0	0	0	0	0	0	0	$\overline{MEMW} = 0$	Запись в память
4	1	0	0	0	0	1	1	0	$\overline{MEMR} = 0$	Чтение стека
5	0	0	0	0	0	1	0	0	$\overline{MEMW} = 0$	Запись в стек
6	0	1	0	0	0	0	1	0	$\overline{I/OR} = 0$	Чтение внешнего устройства
7	0	0	0	1	0	0	0	0	$\overline{I/OW} = 0$	Запись во внешнее устройство
8	0	0	1	0	0	0	1	1	$\overline{INTA} = 0$	Подтверждение прерывания
9	1	0	0	0	1	0	1	0	*NONE*	Подтверждение останова
10	0	0	1	0	1	0	1	1	$\overline{INTA} = 0$	Подтверждение прерывания при останове
	MEMR	*INP*	*M1*	*OUT*	*HLTA*	*STACK*	$\overline{WO}$	*INTA*	← Назначение разрядов слова состояния *SW*	

Сигналы *HLDA* и $\overline{BUSEN}$ (*Bus Enable*) используются для управления системным контроллером при передаче управления системными шинами контроллеру прямого доступа к памяти (*DMAC*). Сигнал *HLDA* = 1, подаваемый от МП, переводит все системные сигналы управления в неактивное состояние (высокий уровень). Поступающее затем от *DMAC* значение сигнала *AEN* = $\overline{BUSEN}$ = 1 переводит шину данных $BD_{7\,0}$ и системные сигналы управления в Z-состояние (*AEN — Address Enable*).

Системные контроллеры 8228/8238 изготовляются по ТТЛШ-технологии и обеспечивают токи выходных сигналов:

I_{OL} = 2 мА при $V_{OL} \le 0{,}45$ В и I_{OH} = –0,01 мА при $V_{OH} \ge 3{,}6$ В для шины данных МП $D_{7\text{–}0}$,
I_{OL} = 10 мА при $V_{OL} \le 0{,}45$ В и I_{OH} = –1 мА при $V_{OH} \ge 2{,}4$ В для остальных сигналов.

В одноплатных микроконтроллерах шины данных и адреса не буферируются, так как все БИС обеспечивают на своих выходах токи, достаточные для управления всеми подключенными устройствами ($I_{OL} \le 2$ мА для выходов БИС, изготовляемых по *n*-МОП-технологии). В многоплатных микроконтроллерах существенно увеличиваются паразитная емкость, подключенная к шинам, и статическая нагрузка. Паразитные емкости увеличивают длительности фронтов сигналов, а статическая нагрузка изменяет их уровни, поэтому на каждой плате требуется устанавливать шинные формирователи (драйверы) и приемопередатчики, выполняемые по ТТЛ-, МТТЛШ- или ТТЛШ-технологиям. На рис. 1.32 изображена принципиальная схема центрального процессорного устройства с буферированными шинами адреса и данных, выполненная на основе МП 8080А.

Для подачи сигнала системного сброса *RESET* = 1 при включении питания используется *RC*-цепь (51 кОм, 10 мкФ). Это обеспечивает старт микроконтроллера с адреса 0 и установку триггера *INTE* в 0 для запрета прерываний. Сброс может производиться и с пульта управления подачей значения сигнала $\overline{RESIN} = 0$. С адреса 0 обычно располагается программа тестирования ОЗУ, ПЗУ и всех подключенных интерфейсных БИС (ИС большой степени интеграции). После этого выполняется программа инициализации некоторых устройств, приводящая микроконтроллер в рабочее состояние.

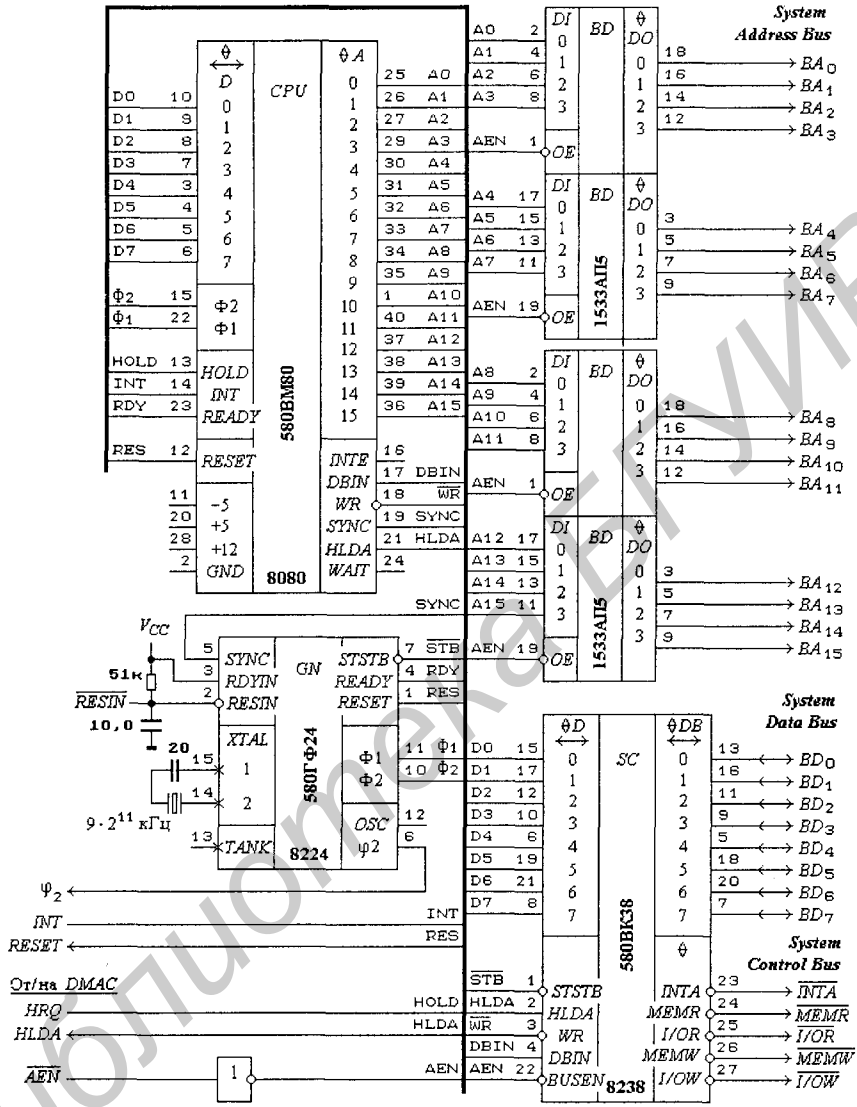


Рис. 1.32. Принципиальная схема центрального процессорного устройства

Предполагается, что все устройства, подключенные к центральному процессорному устройству, имеют высокое быстродействие, поэтому положили сигнал *RDYIN* ≡ 1. При использовании этого сигнала следует знать, что МП производит анализ его значения во втором такте каждого машинного цикла. При обнаружении значения *RDYIN* = 0 устанавливается значение сигнала *WAIT* = 1, и в машинный цикл вводятся холостые такты (такты ожидания) до тех пор, пока не будет обнаружено значение *RDYIN* = 1.

Сигналы *HOLD*, *HLDA* и *AEN* предназначены для подключения контроллера прямого доступа к памяти 8257 или 8237*A* фирмы *Intel* (отечественные БИС 580ВТ57 или 1810ВТ37А). Сигнал *HOLD* анализируется во втором такте каждого машинного цикла и при обнаружении значения *HOLD* = 1 при *RDYIN* = 1 после окончания выполнения текущего машинного цикла

выдается значение сигнала *HLDA* = 1 подтверждения *DMA* — МП переходит в состояние захвата (приостановка работы) с переводом своих шин в *Z*-состояние. Для перевода в *Z*-состояние системных шин используется сигнал *AEN* = $\overline{BUSEN}$ = 1, поступающий от контроллера прямого доступа к памяти.

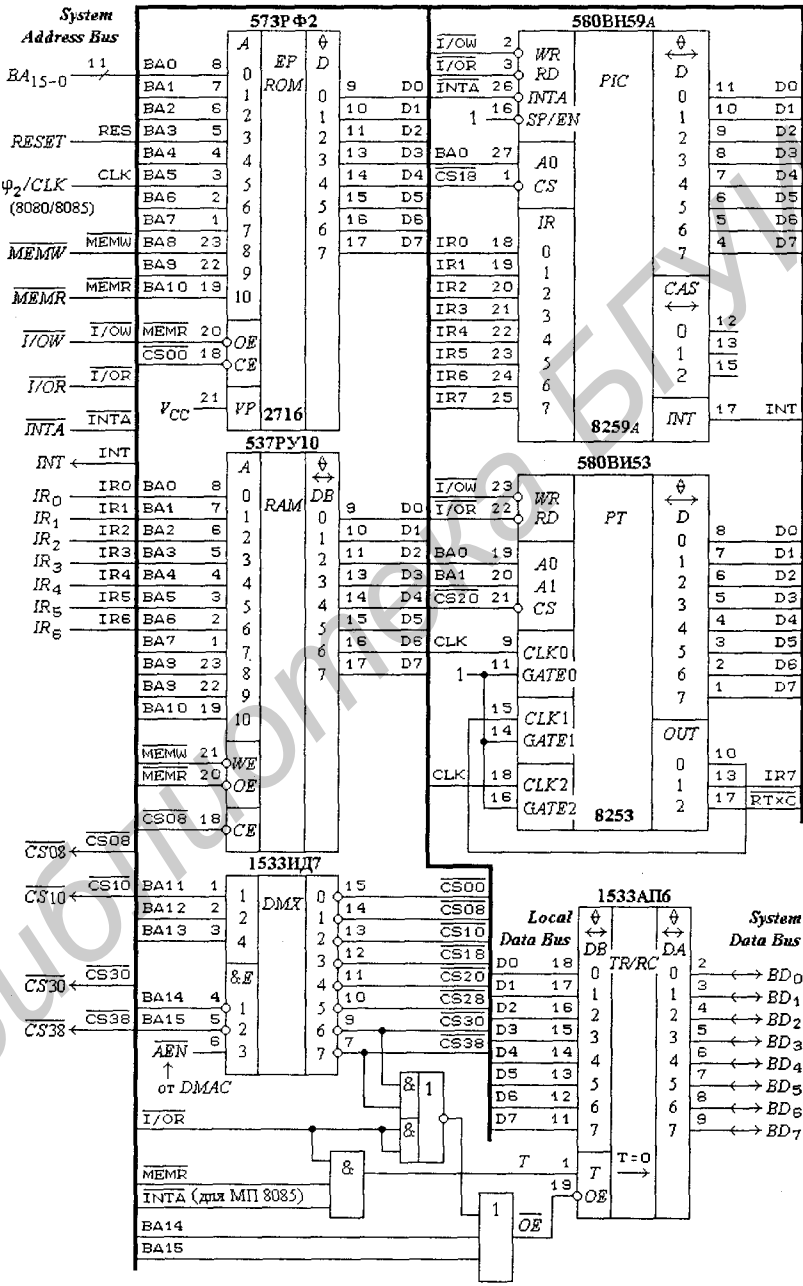


Рис. 1.33. Память, таймер и контроллер прерываний

Сигнал *INT* анализируется в последнем такте последнего машинного цикла и при обнаружении значения *INT* = 1 переходит к обслуживанию прерывания, если ранее флаг *INTE* был установлен в 1. Для этого МП выдает слово состояния SW_8 или SW_{10}, по которому системный контроллер вырабатывает активный уровень сигнала $\overline{INTA} = 0$.

Структурная схема микроконтроллера, построенного на основе МП 8080*A*, была приведена на рис. 1.5. Аналогичную структурную схему можно составить и для микроконтроллера, построенного на основе МП 8085*A*. Принципиальные схемы микроконтроллеров проектируются на базе таких структурных схем, предварительно определив требуемый объем памяти ОЗУ и ПЗУ и перечень необходимых интерфейсных БИС. Структурные схемы позволяют легче изучить принцип работы МП-системы и назначение всех сигналов управления.

Принципиальная схема микроконтроллера. На рис. 1.33 и 1.34 изображены принципиальные схемы памяти и интерфейсных устройств, которые могут быть подключены к системным шинам центральных процессорных устройств, изображенных на рис. 1.10 и 1.32. Все эти устройства расположены на одной печатной плате и обслуживаются одним приемопередатчиком 1533АП6. Селекция всех БИС производится адресным дешифратором, выполненным на ИС 1533ИД7 (рис. 1.33). На один из его входов управления подан сигнал $\overline{AEN}$ от контроллера прямого доступа к памяти 580ВТ57 (см. рис. 2.23), переводящий выходы дешифратора в неактивное состояние (1), запрещающее обращение к адресуемым БИС. Этот же дешифратор управляет селекцией контроллера клавиатуры и 7-сегментного дисплея (см. рис. 1.37 и 1.38).

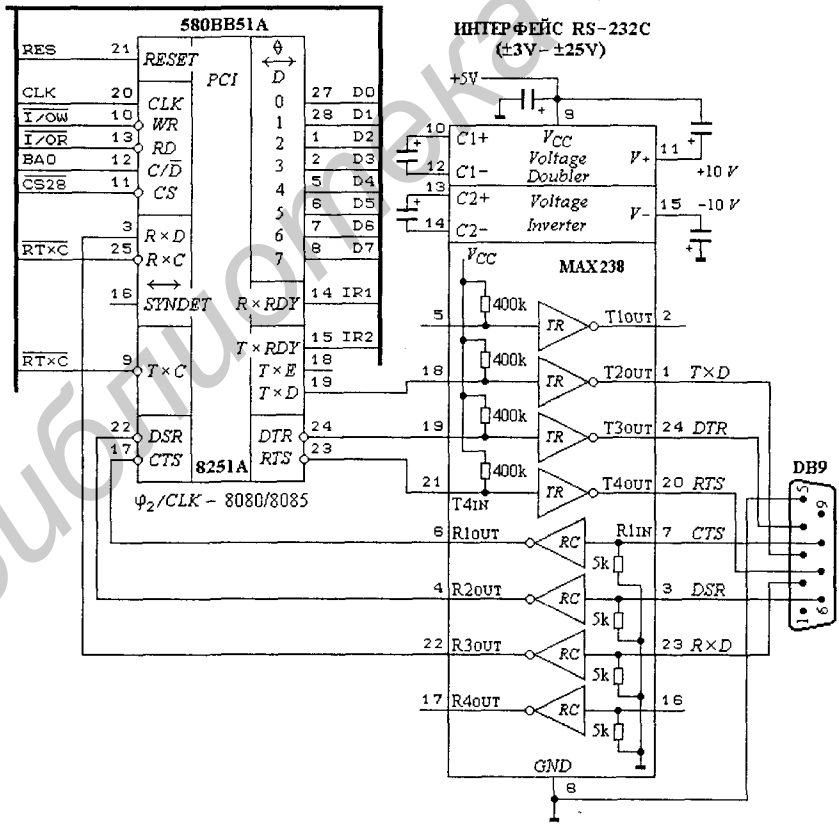


Рис. 1.34. Контроллер последовательного канала передачи данных

Селекция внешних устройств, не участвующих в операциях прямого доступа к памяти, всегда должна запрещаться, чтобы не было к ним ложного обращения при генерации адресных сигналов контроллером прямого доступа к памяти. БИС на рис. 1.33 и 1.34 имеют назначение:

573РФ2 (2716 фирмы *Intel*) — репрограммируемое ПЗУ 2К × 8 бит со стиранием ультрафиолетовыми лучами (*EPROM — Erasable Programmable ROM*; см. § 3.3);

537РУ10(*HM*6516-9 фирмы *Harris Semiconductor*) — ОЗУ 2К × 8 бит (*RAM — Random Access Memory*; см. § 1.10);

580ВН59 (8259 фирмы *Intel*) — программируемый контроллер прерываний (см. § 3.5);

580ВИ53 (8253 фирмы *Intel*) — программируемый интервальный таймер (см. § 3.4);

580ВВ51А (8251*A* фирмы *Intel*) — программируемый контроллер последовательного канала данных (см. § 3.7).

Адресация памяти и внешних устройств. В МП-системах применяется двухуровневая адресация ячеек памяти в ПЗУ и ОЗУ и регистров памяти в интерфейсных БИС: часть разрядов шины адреса подается на их внутренний дешифратор, а остальные разряды — на внешний (например, на ИС 1533ИД7). Этот внешний дешифратор обеспечивает выбор (включение) только одной БИС. Поскольку в рассматриваемом микроконтроллере используются БИС ПЗУ и ОЗУ по 2К × 8 бит (см. рис 1.33), то на их внутренние дешифраторы поступают 11 разрядов адреса A_{10-0} (2 К = 2^{11}). Эти дешифраторы *DC* 11 × 2048 в соответствии с кодом сигналов A_{10-0} выбирают только одну ячейку памяти. Если бы по каким-либо адресам ячейки памяти отсутствовали бы, то к ним нельзя было бы обращаться — некоторые адреса были бы потеряны. Точно так же адресуются и регистры памяти в интерфейсных БИС. Если некоторая интерфейсная БИС имеет по четыре 8-разрядных регистра ввода (для чтения данных) и вывода (для записи данных), то на ее внутренний дешифратор *DC* 2 × 4 должны быть поданы два разряда адреса A_1 и A_0 для обеспечения выбора только одной пары регистров ввода и вывода, имеющих один и тот же адрес порта. К какому из этих двух регистров производится обращение, распознается сигналами управления $\overline{I/OR}$ и $\overline{I/OW}$. Например, в схеме, изображенной на рис. 1.37, один и тот же адресный сигнал $\overline{CS00}$ с выхода внешнего дешифратора используется как для селекции порта ввода, так и порта вывода. Однако чтение данных из порта ввода и запись данных в порт вывода выполняются активными уровнями разных сигналов:

$$\overline{OE} = \overline{I/OR} \vee \overline{CS00} = \overline{I/OR \cdot CS00} \quad \text{и} \quad SR = \overline{\overline{I/OW} \vee \overline{CS00}} = I/OW \cdot CS00.$$

Пока значение сигнала $\overline{OE}$ = 1, выходы буфера данных 561ЛН3 находятся в *Z*-состоянии. Функционально точно так же выполнено чтение и запись данных внутри интерфейсных БИС. Некоторые интерфейсные БИС имеют неполный набор регистров памяти, например, три регистра ввода и четыре регистра вывода.

Дешифратор адреса 1533ИД7 на рис. 1.33 используется для селекции, как памяти, так и устройств ввода-вывода. В табл. 1.16, составленной на основании схемы его включения, представлены диапазоны адресов памяти и портов устройств ввода-вывода, селектируемых каждым выходом дешифратора (символ × = 0 и 1). Адресное пространство памяти, селектируемое каждым сигналом $\overline{CSm}$ (*m* = 00, 08, 10, 18, 20, 28, 30 и 38), составляет 2 К адресов, а селектируемое адресное пространство ввода-вывода — 8 портов.

Полное адресное пространство памяти, селектируемое дешифратором, составляет 16 К 8-разрядных ячеек памяти по адресам 0000*h* ÷ 3FFF*h*, и полное адресное пространство ввода-вывода — 64 порта по адресам 00*h* ÷ 3F*h* (четверть всего адресного пространства ввода-вывода). Селекция одним адресным дешифратором памяти и устройств ввода-вывода возможна потому, что адрес порта ввода-вывода содержится как в младшем, так и в старшем байте шины адреса: при выполнении команд IN *port* и OUT *port* на шине адреса МП устанавливает значения $A_{15-8} = A_{7-0}$.

Таблица 1.16. **Адресация памяти и внешних устройств**

Разряды адреса																Выход *DMX*	Адреса		ИС
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		памяти	портов	
0	0	0	0	0	×	×	×	×	×	×	×	×	×	×	×	0	0000 ÷ 07FF	00 ÷ 07	Клавиатура (рис. 1.37)
0	0	0	0	1	×	×	×	×	×	×	×	×	×	×	×	1	0800 ÷ 0FFF	08 ÷ 0F	1533ИР27 (рис. 1.38)
0	0	0	1	0	×	×	×	×	×	×	×	×	×	×	×	2	1000 ÷ 17FF	10 ÷ 17	1533ИР27 (рис. 1.38)
0	0	0	1	1	×	×	×	×	×	×	×	×	×	×	×	3	1800 ÷ 1FFF	18 ÷ 1F	580ВН59 (рис. 1.33)
0	0	1	0	0	×	×	×	×	×	×	×	×	×	×	×	4	2000 ÷ 27FF	20 ÷ 27	580ВИ53 (рис. 1.33)
0	0	1	0	1	×	×	×	×	×	×	×	×	×	×	×	5	2800 ÷ 2FFF	28 ÷ 2F	580ВВ51А (рис. 1.34)
0	0	1	1	0	×	×	×	×	×	×	×	×	×	×	×	6	3000 ÷ 37FF	30 ÷ 37	580ВВ55А (рис. 3.16)
0	0	1	1	1	×	×	×	×	×	×	×	×	×	×	×	7	3800 ÷ 3FFF	38 ÷ 3F	580ВВ79 (рис. 3.144)
0	0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	—	0000 ÷ 3FFF	00 ÷ 2F	1533АП6 (рис. 1.33)
← Адреса портов →																Используются только адреса портов 00*h* (клавиатура); 08*h* (ИР27); 10*h* (ИР27); 18*h* и 19*h* (ВН59); 20*h*, 21*h*, 22*h* и 23*h* (ВИ53); 28*h* и 29*h* (ВВ51); 30*h*, 31*h*, 32*h* и 33*h* (ВВ55); 38*h* и 39*h* (ВВ79).			
← Адреса памяти →																			

На рис. 1.33 показаны только 4 Кбайта памяти. Полная же схема содержит еще 12 Кбайт памяти (шесть БИС по 2К × 8 бит), селектируемые остальными шестью выходными сигналами ИС 1533ИД7 (всего, например, 12 Кбайт ПЗУ и 4 Кбайта ОЗУ). Такое ОЗУ было изображено на рис. 1.25.

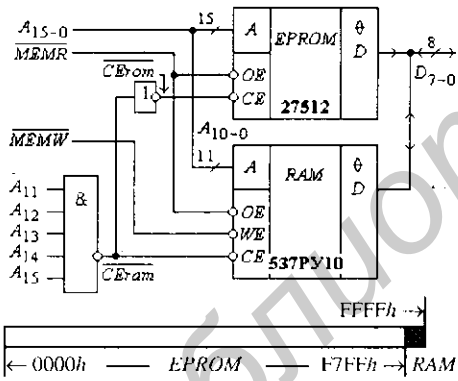


Рис. 1.35. Память 64К × 8 бит

На рис. 1.35 изображена структурная схема памяти объемом 64К × 8 бит, реализованная всего на двух БИС: *EPROM* 27512 64К × 8 бит фирмы *Intel* (см. § 3.3) и *RAM* 537РУ10 2К × 8 бит. Детектор состояния адресных сигналов A_{15-11} (логический элемент И–НЕ) разделяет адресное пространство памяти на две части: 62 Кбайта для ПЗУ и 2 Кбайта для ОЗУ. Действительно, если значение сигнала

$$\overline{CEram} = \overline{A_{15}A_{14}A_{13}A_{12}A_{11}} = 0 \text{ (выбрано ОЗУ)},$$

то значение сигнала $\overline{CErom} = 1$ — ПЗУ выключено. Из этого следует, что ПЗУ расположено по адресам 0000*h* ÷ F7FF*h*, а ОЗУ — по адресам F800*h* ÷ FFFF*h*, т. е. ОЗУ расположено в самом конце адресного пространства МП.

Изменяя некоторые из адресных сигналов $A_{15\ 11}$ на инверсные, можно перемещать положение ОЗУ в адресном пространстве МП. Так, если в схеме на рис. 1.35 сигналы A_{15} и A_{14} заменить инверсными сигналами $\overline{A}_{15}$ и $\overline{A}_{14}$, то ОЗУ будет занимать адреса 3800*h* ÷ 3FFF*h*, а ПЗУ — всю оставшуюся часть адресного пространства МП.

Сигналами $\overline{CSm}$ можно селектировать интерфейсные БИС, содержащие не более восьми регистров ввода и восьми регистров вывода. Если же интерфейсные БИС содержат меньшее число регистров, то при использовании для их адресации сигналов $\overline{CSm}$ часть портов будет не востребована, т. е. часть адресного пространства ввода-вывода будет потеряна. Все интерфейсные БИС, использованные в схемах на рис. 1.33 и 1.34, имеют внутренние дешифраторы 2 × 4 и 1 × 2 (у этих БИС имеются входы для подачи только одного A_0 или двух A_1 и A_0 разрядов адреса), поэтому многие адреса портов для микроконтроллера будут потеряны. Но это зачастую не имеет никакого значения, так как в небольших МП-системах во многих случаях нет необходи-

мости использовать не только все адресное пространство ввода-вывода, но и все адресное пространство памяти. Выгода же очевидна — уменьшаются аппаратные затраты на реализацию дешифрации адресов портов ввода-вывода.

Для устройств ввода-вывода можно использовать и независимый адресный дешифратор 1533ИД7. Так, каждый выход дешифратора, изображенного на рис. 1.36, будет селектировать по четыре порта ввода-вывода, адреса которых указаны в табл. 1.17.

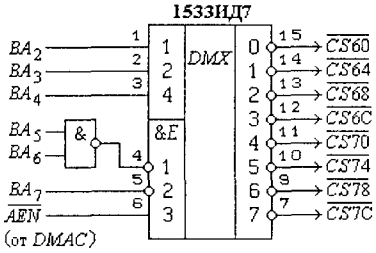


Рис. 1.36. Адресный дешифратор

Таблица 1.17. **Адресация внешних устройств**

Разряды адреса								Выход	Адреса
7	6	5	4	3	2	1	0	*DMX*	портов
0	1	1	0	0	0	×	×	0	60, 61, 62, 63
0	1	1	0	0	1	×	×	1	64, 65, 66, 67
0	1	1	0	1	0	×	×	2	68, 69, 6A, 6B
0	1	1	0	1	1	×	×	3	6C, 6D, 6E, 6F
0	1	1	1	0	0	×	×	4	70, 71, 72, 73
0	1	1	1	0	1	×	×	5	74, 75, 76, 77
0	1	1	1	1	0	×	×	6	78, 79, 7A, 7B
0	1	1	1	1	1	×	×	7	7C, 7D, 7E, 7F

Заметим, что в интерфейсных БИС применяются и более изощренные способы адресации внутренних регистров, чем их прямая адресация с помощью шины адреса. Например, одна часть байта, поступающего от МП в интерфейсную БИС, может содержать адрес регистра, а другая часть — данные. Такой способ адресации (по шине данных) используется с целью уменьшения числа адресов портов, занимаемых БИС в адресном пространстве ввода-вывода. Применяется также и способ адресации регистров БИС с помощью внутреннего цифрового автомата, изменяющего свои состояния в строго соблюдаемой последовательности при каждой записи в нее байта данных. Каждое же состояние автомата адресует определенный регистр. Таким образом, интерфейсные БИС могут иметь значительно большее число регистров памяти, доступных микропроцессору, чем непосредственно адресуемое командами IN *port* и OUT *port*.

Модульность построения МП-систем. Рассматриваемый микроконтроллер выполнен на трех небольших печатных платах, на каждой из которых имеется свой приемопередатчик для системной шины данных:

плата 1 — центральное процессорное устройство на МП 8085*A* (см. рис. 1.10) или МП 8080*A* (см. рис. 1.32), память и интерфейсные БИС (рис. 1.33 и 1.34), матричный шифратор 16-клавишной клавиатуры (см. рис. 1.37) и 5-разрядный 7-сегментный дисплей (см. рис. 1.38);

плата 2 — программатор *EPROM* 573РФ2/573РФ5, выполненный на программируемом параллельном интерфейсе 580ВВ55*A* (см. рис. 3.16);

плата 3 — контроллер 64-клавишной клавиатуры и 16-разрядного алфавитно-цифрового дисплея (см. рис. 3.144 и 3.145), который может использоваться параллельно или вместо матричного шифратора клавиатуры и 5-разрядного 7-сегментного дисплея.

Приведенное разбиение микроконтроллера на платы демонстрирует модульность построения МП-систем — плата 1 уже представляет собой законченный микроконтроллер, способный воспринимать команды, вводимые оператором с клавиатуры, и выводить результат выполнения программ на дисплей (функционирование платы 1 не зависит от наличия или отсутствия плат 2 и 3). С помощью таймера 580ВИ53 на его выходе OUT_1 можно получить метки времени 1 с из сигнала φ_2, частота которого равна 2^{11} кГц, если два канала таймера запрограммировать на коэффициенты деления 2^{11} и 10^3. Контроллер прерываний обеспечивает выполне-

ние операций ввода-вывода в реальном масштабе времени с обслуживанием 8 внешних устройств. В частности, по запросам прерываний сигналом $OUT_1 = IR_7$ программным способом можно организовать счет времени в минутах и часах с выводом текущего времени на дисплей.

При подключении плат расширения функциональных возможностей микроконтроллера следует обеспечить, чтобы при выполнении *операций ввода данных* в МП они могли поступать только *от одного* из приемопередатчиков, установленных на печатных платах (включая и плату 1). Сигналы выбора кристалла $\overline{CS30}$ и $\overline{CS38}$, формируемые на плате 1, управляют интерфейсными БИС 580ВВ55А, 580ВВ79 и соответствующими им приемопередатчиками, расположенными на платах 2 и 3, что экономит аппаратные затраты на дешифраторы адресов. В связи с этим приемопередатчиком 1533АП6 на плате 1 (рис. 1.33) должны управлять сигналы

$$\overline{OE} = BA_{15} \vee BA_{14} \vee (CS38 \vee CS30)\ I/\overline{OR} \quad \text{и} \quad T = \overline{MEMR \vee I/\overline{OR}}$$

включения приемопередатчика ($\overline{OE}$) и указания направления передачи данных (T). Значение сигнала $\overline{OE} = 1$ (приемопередатчик выключен) при обращении к внешним устройствам, селектируемым сигналами $\overline{CS38} = 0$ или $\overline{CS30} = 0$ при их чтении ($\overline{I/OR} = 0$). Таким образом, данный приемопередатчик обслуживает адресное пространство памяти 0000 ÷ 3FFF*h* и адресное пространство портов ввода-вывода 00 ÷ 2F*h*.

Матричный контроллер клавиатуры. На рис. 1.37, *а* приведен пример внешнего устройства ввода данных с помощью любого из двух методов: программного метода ввода с квитированием (используется флаг $\overline{IBF}$ — см. § 2.3) или метода ввода по прерыванию (используется сигнал *IRQ* — см. § 2.4). Данное внешнее устройство представляет собой матричный контроллер клавиатуры, содержащий 17 клавиш. Выход *IRQ* можно подключить к входу IR_0 контроллера прерываний 580ВН59А (рис. 1.33) или к входу *RST* 7.5 микропроцессора 8085.

Принцип работы контроллера клавиатуры основан на сканировании матрицы из 16 клавиш с помощью синхронного 4-разрядного двоичного счетчика 561ИЕ11 и 8-канального мультиплексора-демультиплексора 561КП2 (*MUX–DMX*; см. § 6.5 в книге [5]). Выходные сигналы трех младших разрядов счетчика $Q_{2\text{-}0}$ поданы на адресные входы каналов *MUX–DMX*. Эти сигналы поступают на его внутренний дешифратор *DC* 3 × 8, который обеспечивает периодическое сканирование (опрос) линий возврата $RL_{7\text{-}0}$ (*Return Line*) матрицы. Вертикальные сигнальные линии матрицы периодически переключаются внешним дешифратором *DC* 1 × 2, выдающим на них сигналы $\overline{Q}_3$ и Q_3. Значения выходных сигналов этих двух дешифраторов *однозначно* определяют текущее состояние счетчика. Поэтому при нажатии клавиши $k = 0 \ldots 15$ выходной сигнал *Scan* ИС 564КП2 примет значение 1 в момент времени, когда счетчик 561ИЕ11 находится в состоянии $k = Q_3Q_2Q_1Q_0$ (рис. 1.37, *б*). Это приведет к получению значения сигнала $\overline{P}_0 = 1$, которое блокирует дальнейшие изменения его состояний. Изменение сигнала $\overline{P}_0$ с 0 на 1 вызывает срабатывание триггера, выдающего сигналы $\overline{IBF}/IRQ$, информирующие МП о готовности кода нажатой клавиши. Пока клавиша нажата, счет будет заторможен, и на буфер данных 561ЛН3 счетчик будет выдавать число *k*. Далее будем считать, что микроконтроллер построен на основе МП 8085*A* и используется метод ввода данных по прерыванию.

В ответ на значение сигнала $IRQ = 1$ микропроцессор вызывает подпрограмму обслуживания прерывания, которая выполняет чтение состояния буфера данных

$$D_7D_6D_5D_4\ D_3D_2D_1D_0 = \overline{IBF}\ Shift \times\times\ Q_3Q_2Q_1Q_0,$$

где × — неопределенные значения разрядов; *Shift* — дополнительная 17 клавиша ("пассивная"), удваивающая число кодов основных 16 клавиш.

После ввода байта данных МП должен выполнить команду OUT *Cs*00 для сброса триггера, формирующего сигналы $\overline{IBF}/IRQ$, в исходное состояние. При отпускании клавиши сигнал $\overline{P}_0$ установится в 0, и счетчик продолжит счет — сканирование клавиатуры возобновится.

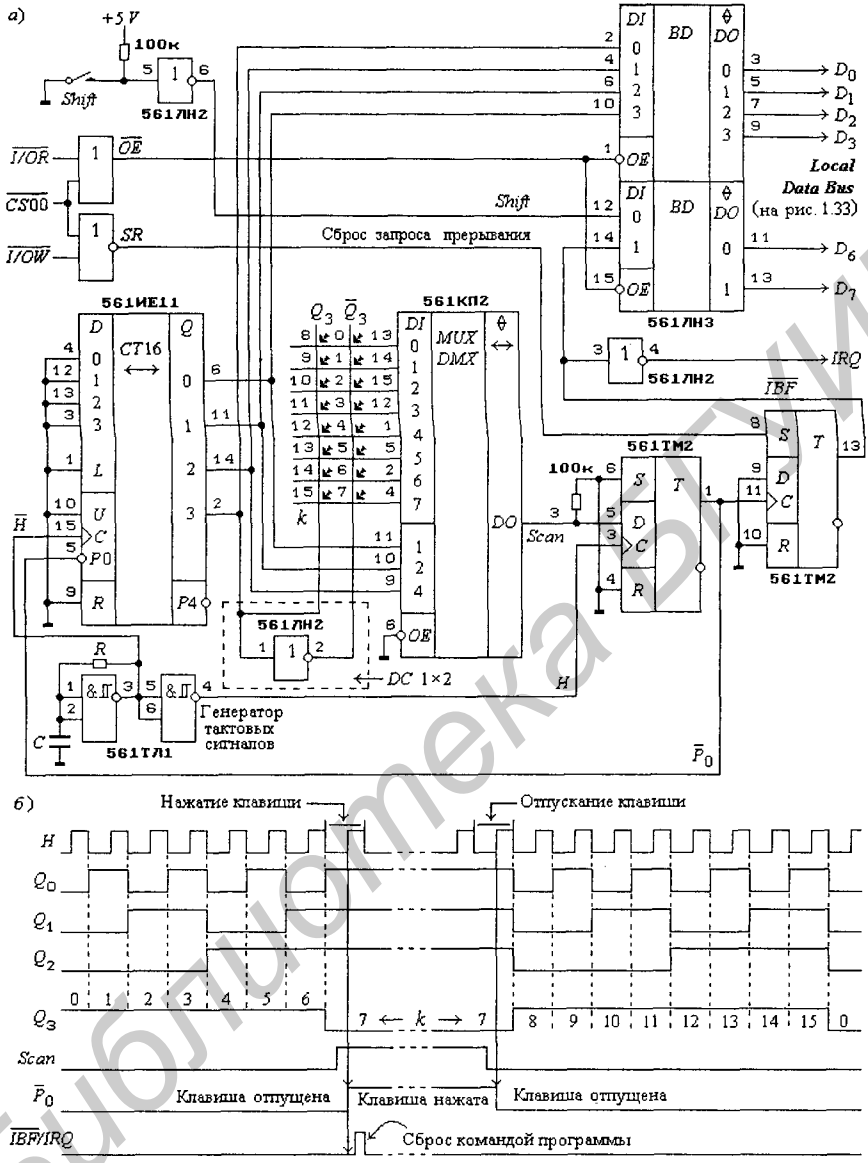


Рис. 1.37. Матричный контроллер клавиатуры

Задача 1. Написать программу ввода по прерыванию кода нажатой клавиши. Записать принятый код в ячейку памяти с символическим именем *Key*. *Решение*:

```
PUSH    PSW   ; Сохранение в стеке состояния прерванной программы
PUSH    H
PUSH    D
PUSH    B
IN      Cs00  ; A ← D7D6×× D3D2D1D0 = IBF Shift ×× Q3Q2Q1Q0 (Cs00 = 00h)
ANI     0CFh  ; A ← A & CFh (CFh = 1100 1111): "очистка" случайных значений D5D4
```

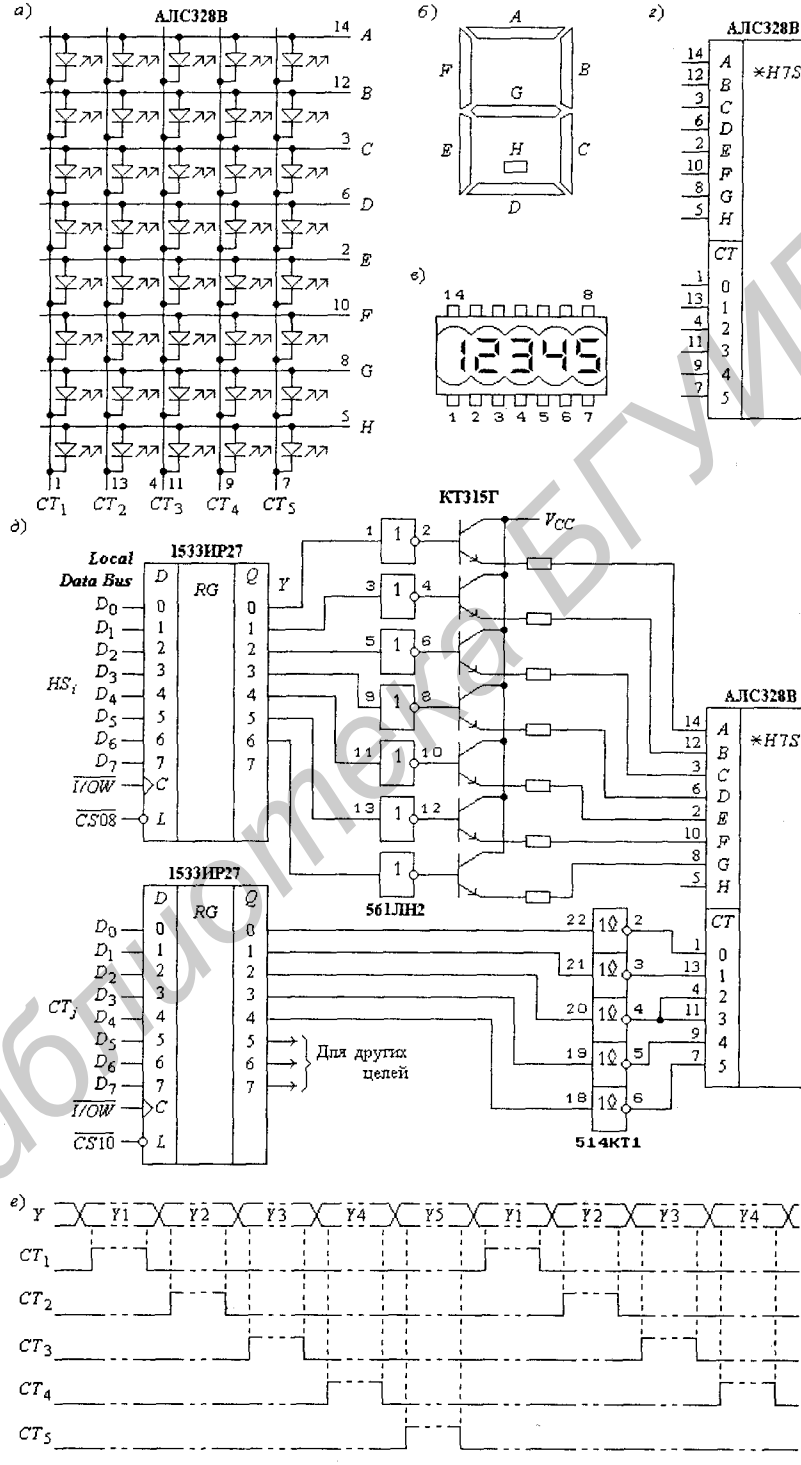


Рис. 1.38. 5-разрядный 7-сегментный дисплей

```
STA     Key     ; M(Key) ← A (запись "очищенного" кода в память)
OUT     Cs00    ; Вывод произвольного байта для сброса триггера IRQ запроса ввода
  ∴             ; Программа, выполняющая действия, по коду нажатой клавиши
POP     B       ; Восстановление состояния прерванной программы
POP     D
POP     H
POP     PSW
EI              ; Разрешение прерываний
RET             ; Возврат в прерванную программу
```

Для уменьшения потребляемой мощности контроллер клавиатуры выполнен на КМОП ИС. Если необходимо построить 64-клавишный контроллер клавиатуры, то вместо 4-разрядного счетчика и дешифратора *DC* 1 × 2 следует взять 6-разрядный счетчик и дешифратор *DC* 3 × 8 и на его адресные входы подать сигналы с трех старших разрядов счетчика (вместо двух вертикальных линий в сканируемой матрице будет восемь линий). Следует заменить также 6-разрядный буфер данных 561ЛН3 8-разрядным буфером. Остальная часть схемы контроллера останется без изменений.

Период сканирования 16-клавишной клавиатуры равен $16 \cdot T_H$, где T_H — период тактового сигнала *H*, вырабатываемого генератором, выполненном на триггере Шмитта 561ТЛ2. Чем ниже частота этого генератора, тем надежнее исключается "дребезг" контактов клавиш. При частоте генератора 7 кГц дребезг в 64-клавишной клавиатуре надежно устраняется даже при использовании не высококачественных клавиш.

5-разрядный 7-сегментный дисплей. В качестве дисплея в МП-системе можно использовать 7-сегментные светоизлучающие индикаторы [8]. На рис. 1.38, *а* показана структура 5-разрядного 7-сегментного индикатора АЛС328В, на рис. 1.38, *б* — расположение и обозначения сегментов, на рис. 1.38, *в* — внешний вид индикатора и на рис. 1.38, *г* — его условное графическое обозначение (*CT — Cathode*). Прямой постоянный ток через сегмент $I_{S\,max} = 5$ мА.

Принципиальная схема контроллера дисплея, содержащая два порта вывода для управления сегментами и катодами индикатора в мультиплексном режиме, изображена на рис. 1.38, *д*. Временны́е диаграммы, показанные на рис. 1.38, *е*, наглядно поясняют работу схемы. Для исключения мигания индикатора частота развертки выбирается не менее 50 Гц. Для мультиплексной развертки разрядов индикатора следует использовать унитарный код для представления значений катодов CT_{5-1} (табл. 1.18). Таблицу кодов управления катодами можно хранить в ПЗУ (*A* — адрес ПЗУ). Преобразование 16-ричных чисел *X* = 0, 1, ..., 9, A, B, ..., F и некоторых символов в 7-сегментный код $Y = Y_G Y_F Y_E Y_D Y_C Y_B Y_A$ проще всего выполняется табличным способом (табл. 1.19). Для этого все используемые символы должны быть пронумерованы числами *X*. Коды *Y* должны храниться в ПЗУ по адресам, возрастающим в порядке увеличения числа *X*. Это позволяет достаточно просто по числу *X* вычислить адрес кода *Y* и вывести его в регистр памяти значений сегментов в контроллере дисплея.

Информация, выводимая на дисплей, должна храниться в ОЗУ в пяти ячейках памяти в виде чисел *X*. Шестую ячейку памяти ОЗУ следует отвести для хранения текущего номера мультиплексируемого разряда дисплея. Например, для вывода на дисплей сообщения "–ПЗУ–" в ячейки памяти следует записать числа 17, 20, 3, 22 и 17, а для вывода сообщения "–ОЗУ–" — числа 17, 0, 3, 22 и 17. Эти сообщения можно использовать при тестировании для идентификации неисправных устройств.

Таблица 1.18. **Управление катодами**

Унитарный код	Катод	*Y*	*A*
0 0 0 0 0 0 0 0 1	CT_1	01	0740
0 0 0 0 0 0 0 1 0	CT_2	02	0741
0 0 0 0 0 0 1 0 0	CT_3	04	0742
0 0 0 0 0 1 0 0 0	CT_4	08	0743
0 0 0 0 1 0 0 0 0	CT_5	10	0744

Таблица 1.19. **Управление 7-сегментным дисплеем**

X		X_4	X_3	X_2	X_1	X_0	Y_G	Y_F	Y_E	Y_D	Y_C	Y_B	Y_A	Символ	Y	A
00	00	0	0	0	0	0	1	0	0	0	0	0	0	0	40	0720
01	01	0	0	0	0	1	1	1	1	1	0	0	1	1	79	0721
02	02	0	0	0	1	0	0	1	0	0	1	0	0	2	24	0722
03	03	0	0	0	1	1	0	1	1	0	0	0	0	3	30	0723
04	04	0	0	1	0	0	0	0	1	1	0	0	1	4	19	0724
05	05	0	0	1	0	1	0	0	1	0	0	1	0	5	12	0725
06	06	0	0	1	1	0	0	0	0	0	0	1	0	6	02	0726
07	07	0	0	1	1	1	1	1	1	1	0	0	0	7	78	0727
08	08	0	1	0	0	0	0	0	0	0	0	0	0	8	00	0728
09	09	0	1	0	0	1	0	0	1	0	0	0	0	9	10	0729
10	0A	0	1	0	1	0	0	0	0	1	0	0	0	A	08	072A
11	0B	0	1	0	1	1	0	0	0	0	0	1	1	b	03	072B
12	0C	0	1	1	0	0	1	0	0	0	1	1	0	C	46	072C
13	0D	0	1	1	0	1	0	1	0	0	0	0	1	d	21	072D
14	0E	0	1	1	1	0	0	0	0	0	1	1	0	E	06	072E
15	0F	0	1	1	1	1	0	0	0	1	1	1	0	F	0E	072F
16	10	1	0	0	0	0	1	1	1	1	1	1	1		7F	0730
17	11	1	0	0	0	1	0	1	1	1	1	1	1		3F	0731
18	12	1	0	0	1	0	1	0	0	1	1	1	0		4E	0732
19	13	1	0	0	1	1	0	0	0	1	0	0	1		09	0733
20	14	1	0	1	0	0	1	0	0	1	0	0	0		48	0734
21	15	1	0	1	0	1	0	0	0	1	1	0	0		0C	0735
22	16	1	0	1	1	0	0	0	1	0	0	0	1		11	0736
23	17	1	0	1	1	1	1	0	0	0	1	1	1		47	0737
24	18	1	1	0	0	0	1	0	0	0	0	0	1		41	0738
25	19	1	1	0	0	1	0	0	1	1	0	1	1		1B	0739
26	1A	1	1	0	1	0	1	1	1	0	0	0	1		71	073A

Задача 2. Написать программу управления 5-разрядным 7-сегментным дисплеем с выводом информации по прерыванию. *Решение*:

```
        defseg  S_rst75, start = 3Ch      ; 7,5 × 8 = 60d = 3Ch
        seg     S_rst75    ; ROM
; Подпрограмма обработки прерывания RST 7.5
        PUSH    PSW        ; Сохранение в стеке состояния прерванной основной программы
        PUSH    H          ;      (подпрограмма RST  7.5 использует РОНы H, L и B)
        PUSH    B
        LXI     H, Tn + 5  ; HL = 0805h — адрес хранения текущего активного разряда
        INR     M
        MVI     A, 5    ; Обеспечение счета по модулю 5 обслуживаемых разрядов индикатора
        CMP     M
        JNZ     LR1
        SUB     A
        MOV     M, A
LR1:    MOV     A, M
        ADD     L
        SUI     5
```

```
        MOV   L, A
        MOV   B, A      ; B — номер разряда индикатора
        MOV   A, M
        LXI   H, T7s    ; T7s = 0720h — начальный адрес таблицы 7-сегментных кодов
        ADD   L
        MOV   L, A
        SUB   A
        OUT   CSct      ; Гашение дисплея (Blanking Display)
        MOV   A, M
        OUT   CShs      ; Вывод значений сегментов (Output Segment)
        LXI   H, Tct    ; Tct = 0740h — начальный адрес таблицы унитарного кода катодов
        MOV   A, B
        ADD   L
        MOV   L, A
        MOV   A, M
        OUT   CSct      ; Включение разряда дисплея (Output Cathode)
        POP   B         ; Восстановление состояния прерванной основной программы
        POP   H
        POP   PSW
        EI
        RET             ; Возврат из подпрограммы обработки прерывания RST 7.5
Ram     equ   800h      ; Ram = 0800h — начальный адрес RAM          Data Segment
Rsz     equ   800h      ; Rsz = 800h — объем RAM (RAM Size)
Ssz     equ   20h       ; Ssz = 20h — размер стека (Stack Size)
        defseg D1_seg, start = Ram, class = Data
        seg   D1_seg    ; RAM
Tn      ds    6         ; Tn = Ram = 0800h, M(0805) — текущий номер разряда дисплея
        defseg D2_seg, start = 720h, class = Data ; T7s = 0720h — начальный адрес таблицы
        seg   D2_seg    ; ROM                                кодов сегментов T7s
T7s     db    40h, 79h, 24h, 30h, 19h, 12h, 2, 78h, 0, 10h, 8, 3, 46h, 21h, 6, 0Eh
        db    7Fh, 3Fh, 4Eh, 9, 48h, 0Ch, 11h, 47h, 41h, 1Bh, 71h
        defseg D3_seg, start = 740h, class = Data
        seg   D3_seg    ; ROM
Tct     db    1, 2, 4, 8, 10h, 0  ; Tct = 0740h — начальный адрес таблицы кодов катодов Tct
        defseg Stack_seg, start = Ram + Rsz – Ssz, class = Data      ; Stack Segment
        seg   Stack_seg
        ds    Ssz
        defseg IO_seg, start = 8, class = IOspace                    ; I/O Segment
        seg   IO_seg    ; Порты вывода
CShs    ds    1         ; CShs = 08h — адрес порта вывода кода сегментов (рис. 1.38)
        org   10h
CSct    ds    1         ; CSct = 10h — адрес порта вывода кода катодов
        defseg Main_seg, start = 100h, class = Code                  ; Code Segment
        seg   Main_seg ; ROM
        LXI   SP, Ram + Rsz  ; Инициализация указателя стека SP
        LXI   H, 0011h  ; Инициализация RAM для вывода на дисплей сообщения: –ОЗУ–
        SHLD  Tn        ; M(0800) ← 11h — символ "–", M(0801) ← 00h — символ "0"
        LXI   H, 1603h  ; H ← 16h, L ← 03h
        SHLD  Tn + 2    ; M(0802) ← 03h — символ "З", M(0803) ← 16h —символ "У"
```

```
        LXI     H, 0411h  ; H ← 04h, L ← 11h
        SHLD    Tn + 4    ; M(0804) ← 11h — символ "–"
                          ; M(0805) ← 04h — номер разряда дисплея
        MVI     A, 0Bh    ; A ← 0Bh = 0000 1011 (A3 = MCE = 1, A2 = M7.5 = 0)
        SIM               ; Разрешение прерывания по входу RST 7.5
        EI                ; Общее разрешение прерывания
LM1:    JMP     LM1       ; Свернутая в одну точку основная программа
        end               ; Конец программы
```

Частота запросов прерываний должна быть не менее 250 Гц, чтобы обеспечить частоту развертки не менее 50 Гц. Для генерации запросов прерываний обычно используется таймер 580ВИ53 (8253 фирмы *Intel*). Чем выше частота развертки, тем больше будут затраты процессорного времени на обслуживание индикатора.

В подпрограмму обслуживания дисплея, например перед командой POP B, можно добавить команды для программного ввода с квитированием байта данных с клавиатуры:

```
        IN      Cs00      ; A ← D7D6×× D3D2D1D0 = IBF Shift ×× Q3Q2Q1Q0 (Cs00 = 00h)
        ANI     0CFh      ; A ← A & CFh (CFh = 1100 1111): очистка случайных значений D5D4
        MOV     C, A      ; C ← очищенный код нажатой клавиши
        ANI     80h       ; Анализ значения флага IBF
        JNZ     LR2
        OUT     Cs00      ; Вывод произвольного байта для сброса триггера IRQ запроса ввода
        ∴                 ; Программа, выполняющая действия, по коду нажатой клавиши
LR2:    POP     B         ; Эта команда в подпрограмме уже есть
```

В этом случае вход запроса прерываний, обслуживающий клавиатуру, освобождается для других целей. Если время выполнения операций по коду нажатой клавиши будет больше 4 мс, то очередной запрос прерывания будет пропущен, что может привести к временному изменению яркости одного из разрядов индикатора. Частота опроса клавиатуры равна 250 Гц, что вполне приемлемо.

1.10. Статические запоминающие устройства

Для записи, хранения и чтения данных в процессе их обработки используются оперативные запоминающие устройства (ОЗУ) с произвольной выборкой (*RAM — Random Access Memory*). Они содержат 2^n ячеек памяти, однозначный выбор которых производится адресными сигналами A_{n-1}, …, A_0. В современных МП-системах находит применение как *статическая память* (*SRAM — Static RAM*), так и *динамическая память* (*DRAM — Dynamic RAM*). В *SRAM* запоминание информации производится в триггерах, а в *DRAM* — на конденсаторах. Длительность хранения информации в триггерах не ограничена, тогда как время хранения информации на конденсаторах определяется токами утечки через шунтирующие цепи, а, следовательно, с некоторой частотой необходимо производить регенерацию заряда на конденсаторах, что усложняет процесс управления памятью. В первых *DRAM* допустимое время хранения информации составляло 1 … 2 мс, что соответствует частоте регенерации ячеек памяти 1,0 … 0,5 кГц. В современных *DRAM* это время составляет 8 … 128 мс.

Площадь, занимаемая на кристалле одной ячейкой памяти в *DRAM*, значительно меньше, чем в *SRAM*, что обуславливает высокую плотность их упаковки и как результат — гораздо меньшую стоимость одного бита информации. Выпускаются и синхронные *SRAM* и *DRAM* (*SSRAM — Synchronous SRAM*, *SDRAM — Synchronous DRAM*).

Статические запоминающие устройства. По технологическим соображениям ячейки памяти располагаются на кристалле в виде прямоугольной матрицы, адресация которой производится двумя дешифраторами (рис. 1.39): *DCR* (*Decoder Row* — дешифратор строк) и *DCC* (*Decoder Column* — дешифратор столбцов). Дешифраторы реализуют минтермы от m и $n - m$ адресных переменных A_p ($e_p = 0$ и 1):

$$K_c = K_c(A_{m-1},\ldots,A_0) = \prod_{p=0}^{m-1} A_p^{ep},\ c = e_{m-1}\ldots e_0;\ K_r = K_r(A_{n-1},\ldots,A_m) = \prod_{p=m}^{n-1} A_p^{ep},\ r = e_{n-1}\ldots e_m$$

($c = 0 \ldots 2^m - 1$ — номер строки, $r = 0 \ldots 2^{n-m} - 1$ — номер столбца).

На пересечении линий минтермов K_c и K_r располагаются адресуемые ячейки памяти, т. е. их выбор осуществляется минтермом

$$K_i = K_c \cdot K_r = \prod_{p=0}^{n-1} A_p^{ep},\ i = e_{n-1}\ldots e_0 \text{ — адрес ячейки памяти.}$$

На рис. 1.40 показана эквивалентная схема одной одноразрядной ячейки памяти *SRAM*, построенной на асинхронном потенциальном *D–L*-триггере:

DI (*Data Input*) — входной информационный сигнал,

DO (*Data Output*) — выходной информационный сигнал,

$\overline{WE}$ (*Write Enable*) — входной сигнал разрешения записи,

$\overline{OE}$ (*Output Enable*) — разрешение выхода (в некоторых *SRAM* этот сигнал отсутствует),

$\overline{CE}$ или $\overline{CS}$ (*Chip Enable* или *Chip Select*) — входной сигнал выбора кристалла.

Сигнал данных *DI* подается на буфер, выходной сигнал которого *DB* поступает на 2^n входов *D* всех *D–L*-триггеров. Сигнал $WR = WE \cdot CE$ используется для управления записью данных в *D–L*-триггеры *SRAM*. Сигнал $OE' = \overline{WE} \cdot OE \cdot CE$ управляет *Z*-состоянием выходного буфера *SRAM*. Объем памяти данного *SRAM* равен $2^n \times 1$ бит. Выходной каскад каждой ячейки памяти выполнен на ЛЭ И–НЕ с открытым коллекторным выходом, что позволяет объединить выходы всех ячеек памяти (ЯП) с помощью функции "монтажное ИЛИ". На входы $\overline{WE}$ и $\overline{OE}$ подаются системные сигналы управления $\overline{MEMW}$ и $\overline{MEMR}$ соответственно.

Работа каждой ячейки памяти и выходной сигнал *SRAM* описываются функциями:

$$Q_i^+ = DI \cdot L_i \vee Q_i \overline{L_i},$$

$$L_i = K_c \cdot K_r \cdot WE \cdot CE = K_i \cdot WE \cdot CE,$$

$i = 0 \ldots 2^n - 1$ — адрес ячейки памяти,

$$DO = \begin{cases} \overline{\bigvee_{i=0}^{2^n-1} \overline{Q}_i \cdot K_i}, & \text{если } OE' = 1, \\ Z\text{-состояние}, & \text{если } OE' = 0. \end{cases}$$

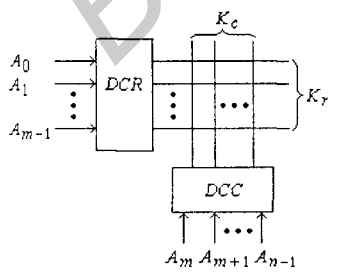


Рис. 1.39. Дешифратор адресов

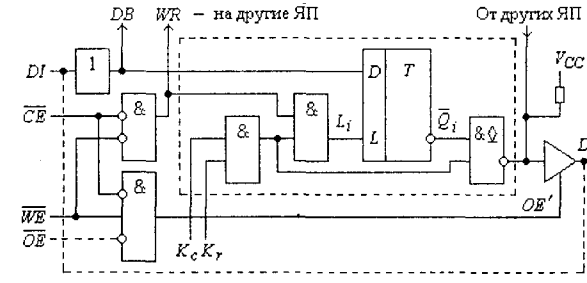


Рис. 1.40. Схема одной ячейки памяти

Таблица 1.20. **Режимы работы *SRAM***

Режим	$\overline{CE}$	$\overline{WE}$	$\overline{OE}$	*DO*
Запись	0	0	×	Z-состояние
Чтение	0	1	0	Выход
Нет операций	0	1	1	Z-состояние
Хранение	1	×	×	Z-состояние

В табл. 1.20 наглядно представлены режимы работы *SRAM* (нет операций — режим чтения без выдачи информации на выход *DO*). Для получения двунаправленной линии данных *DB* (*Data Bidirectional*) вход *DI* следует соединить с выходом *DO* (штриховая линия на рис. 1.40). Выпускаемые *SRAM* различаются как числом входов выбора кристалла $\overline{CE}$ (один или два, объединенных операцией &) и наличием или отсутствием входа $\overline{OE}$, так и числом информационных входов-выходов.

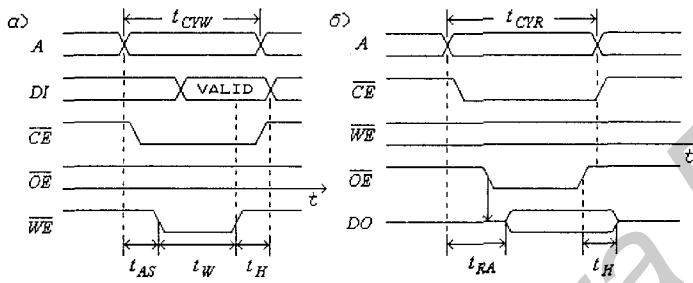


Рис. 1.41. Временны́е диаграммы записи и чтения данных

На рис. 1.41, *а* приведены временны́е диаграммы для режима записи информации, а на рис. 1.41, *б* — для режима чтения. На дешифрацию адреса требуется время t_{AS} (*Address Setup Time* — время установки адреса), только после истечения которого можно подавать активный уровень сигнала разрешения записи $\overline{WE}$ (иначе данные могут быть записаны по неправильному адресу).

Минимальная длительность t_W активного уровня сигнала $\overline{WE}$ определяется быстродействием ячеек памяти. Часто требуется сохранение значений входных данных *DI* некоторое время t_H (*Data Hold Time* — время удержания данных) после окончания активного уровня сигнала записи $\overline{WE} = 0$. В современных *SRAM* обеспечиваются минимальные значения $t_{AS} = 0$ и $t_H = 0$. Длительность *цикла записи* t_{CYW} (*Write Cycle Time*) примерно равна сумме значений t_{AS}, t_W и t_H.

При чтении время t_{RA} (*Read Access Time* — время доступа при чтении) характеризует задержку выходных данных *DO* относительно изменения адресных сигналов при отсутствии сигнала управления $\overline{OE}$ или $\overline{OE} \equiv 0$. Время t_H характеризует удержание значений выходных данных *DO* по окончании активного уровня сигнала управления $\overline{OE} = 0$ (или адресного сигнала $\overline{CE}$ при отсутствии сигнала $\overline{OE}$). Если сигнал управления $\overline{OE}$ имеется, то выходные данные *DO* могут появиться только при значении $\overline{OE} = 0$. Время задержки выходных данных *DO* относительно изменения сигнала управления $\overline{OE}$ с 1 на 0 значительно меньше значения $t_{RA\ min}$. Время *цикла чтения* t_{CYR} примерно равно времени цикла записи t_{CYW}. При равенстве этих циклов основной динамической характеристикой *SRAM* является время цикла t_{CY}.

На ранней стадии развития микроэлектроники выпускались *SRAM*, имеющие небольшой объем памяти, например, 155РУ2 (*SN*7489) 4 × 4 бит, 155РУ5 (*F*93410) 256 × 1 бит, 155РУ7 (*F*93415) 1024 × 1 бит, 531РУ8/531РУ9/589РУ01 (*SN*74189/*SN*74289/*I*3101*A*) 4 × 4 бит, 561РУ2 (*CD*4061A) 256 × 1 бит [8].

На рис. 1.42 показаны *SRAM*, находившие применение в микроконтроллерах с небольшим объемом системной памяти:

132РУ2, 565РУ2 — *SRAM* 1024 × 1 бит. Нет *Z*-состояния при записи данных — выход *DO* переводится в *Z*-состояние только значением сигнала $\overline{CE} = 1$, а при записи $\overline{CE} = 0$, $\overline{WE} = 0 \Rightarrow DO = DI$. Отсутствие *Z*-состояния выхода при записи информации позволяет реализовать на данном *SRAM* 1024-разрядный сдвигающий регистр с последовательным вводом и выводом

данных. В этих *SRAM* нельзя соединять вход и выход для организации двунаправленной передачи данных по одной линии;

132РУ4 — *SRAM* 1024 × 1 бит. Значения адресных сигналов, управляющего сигнала $\overline{WE}$ и данные *DI* фиксируются в регистрах *SRAM* сигналом $d\overline{CE} = 1$, где *d* оператор переходов. В соответствии с зафиксированным значением сигнала $\overline{WE}$ производится запись или чтение информации по выбранному адресу. Выход *DO* находится в *Z*-состоянии при $\overline{CE} = 1$, которое сохраняется и при изменении сигнала $\overline{CE}$ с 1 на 0, если сигнал $\overline{WE} = 0$;

537РУ2 — *SRAM* 4K × 1 бит. Реализованы синхронная запись и чтение информации. Запись выполняется подобно тому, как это было описано для ИС 132РУ4;

537РУ3, 132РУ5 — *SRAM* 4K × 1 бит. Запись и чтение асинхронные ($\overline{CE} = 0$, $\overline{WE} = 0$ — запись с переводом выхода *DO* в *Z*-состояние, что позволяет объединить вход *DI* и выход *DO* для организации двунаправленной шины данных; $\overline{CE} = 0$, $\overline{WE} = 1$ — чтение данных; $\overline{CE} = 1$ — *Z*-состояние выхода *DO*);

537РУ13 — *SRAM* 1K × 4 бит с двунаправленной шиной данных *DB*;

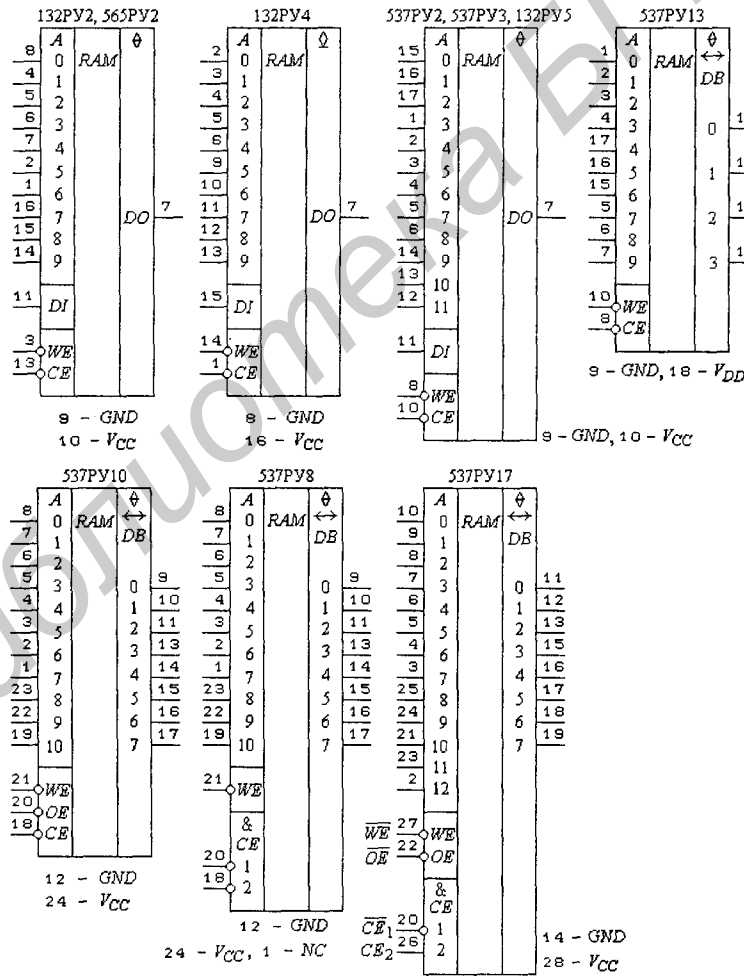


Рис. 1.42. Примеры условных графических обозначений *SRAM*

537РУ10, 537РУ8 — *SRAM* 2К × 8 бит с двунаправленной шиной данных *DB*. ИС 537РУ8 отличается от ИС 537РУ10 только тем, что имеет два равноправных входа выбора кристалла $\overline{CE_1}$ и $\overline{CE_2}$, изменение сигналов на которых с 1 на 0 дает значение управляющего сигнала $CE_2 d\overline{CE_1} \vee CE_1 d\overline{CE_2} = 1$, фиксирующего адресные сигналы A_p во внутреннем 11-разрядном регистре памяти. Режимы работы определяются значениями управляющих сигналов: $\overline{CE_1} = 0$, $\overline{CE_2} = 0$, $\overline{WE} = 0$ — запись по фиксированному в регистре адресу; $\overline{CE_1} \vee \overline{CE_2} \vee WE = 1$ — *Z*-состояние выходов;

537РУ17 — *SRAM* 8К × 8 бит с двунаправленной шиной данных *DB* ($\overline{CE_1} = 0$ и $CE_2 = 1$ — выбор БИС).

В табл. 1.21 приведены основные параметры рассмотренных *SRAM*. Ток потребления КМОП *SRAM* в режиме хранения ($I_{\text{п. хр.}}$) значительно меньше тока потребления в динамическом режиме ($I_{\text{п. дин.}}$), что позволяет использовать резервное питание от аккумулятора для сохранения информации при отказах основного источника питания. Напряжение аккумулятора без разрушения информации в *SRAM* может изменяться в широких пределах (2,7 … 4,5 В).

Таблица 1.21. **Основные параметры *SRAM***

ИС	Зарубежный аналог	Технология	Объем Памяти	t_{CY}, нс	$I_{\text{п. дин.}}$, мА	$I_{\text{п. хр.}}$, мкА
132РУ2А/Б	*M*2102*A*-4	*n*-МОП	1К × 1	400/550	76	—
565РУ2А/Б	2102*A*-4/6	*n*-МОП	1К × 1	450/850	70	—
132РУ4А/Б	2125*AL*	*n*-МОП	1К × 1	33/45	60	—
132РУ5А/Б	2147	*n*-МОП	4К × 1	75/100	160	—
537РУ2А/Б	*HM*6504-5	КМОП	4К × 1	300/430	5	500/1000
537РУ3А/Б	*HM*6504-5	КМОП	4К × 1	230/150	20	5/0,5
537РУ8А/Б	*MSM*5128	КМОП	2К × 8	190/320	30	1000/2000
537РУ10А/Б	*HM*6516-9	КМОП	2К × 8	220/450	70	400
537РУ13А/Б	*TC*5514*AD*	КМОП	1К × 4	120/200	70	10
537РУ17	*MB*8464-15	КМОП	8К × 8	200	85	2

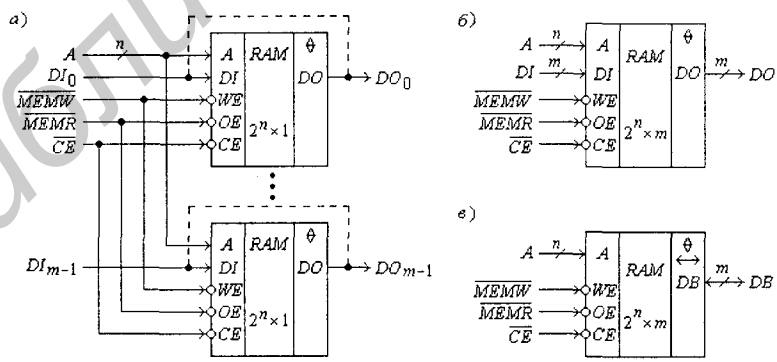


Рис. 1.43. Построение *m*-разрядных ОЗУ на 1-разрядных ОЗУ

На рис. 1.43, *а* приведен пример увеличения разрядности шины данных при использовании *m* одноразрядных *SRAM* $2^n \times 1$ бит (штриховой линией показана организация двунаправленной шины данных). Объем этой памяти равен $2^n \times m$ бит. На рис. 1.43, *б* изображено условное графическое обозначение этой памяти при использовании раздельных входов и выходов

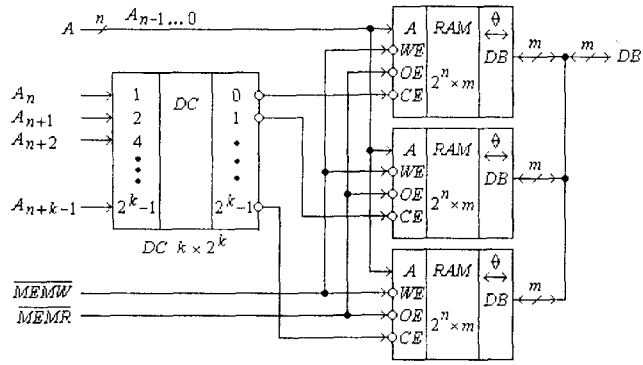


Рис. 1.44. Увеличение объема памяти *m*-разрядных *SRAM*

данных, а на рис. 1.43, *в* — для двунаправленной шины данных. Для увеличения объема памяти в 2^k раз (рис. 1.44) следует использовать дешифратор *DC* $k \times 2^k$, выходные сигналы которого производят выбор одного из *SRAM* $2^n \times m$ бит (объем этой памяти равен $2^{n+k} \times m$ бит).

Статические ОЗУ фирмы *Micron Semiconductor, Inc.* На рис. 1.45 приведены 1-, 4-, 8- и 16-разрядные *SRAM*, выпускаемые фирмой *Micron Semiconductor, Inc.* (*LC* — низковольтные *SRAM* с напряжением питания +3,3 В). Одноразрядные *SRAM* имеют раздельные вход и выход, а остальные — двунаправленную шину данных.

На рис. 1.45 фирменные обозначения сигналов имеют смысл: *D* = *DI* — входные данные памяти, *Q* = *DO* — выходные данные памяти, *DQ* = *DB* — двунаправленные данные (вход/выход), V_{SS} = *GND* (*Ground*) — земля, *NC* (*No Connection*) — нет соединения, $\overline{OE}$ (*Output Enable*) — разрешение выхода, $\overline{CE}$ (*Chip Enable*) — разрешение кристалла. В табл. 1.22 приведены параметры *SRAM* со стандартным напряжением питания (V_{CC} = +5 В), а в табл. 1.23 — параметры низковольтных *SRAM* (V_{CC} = +3,3 В).

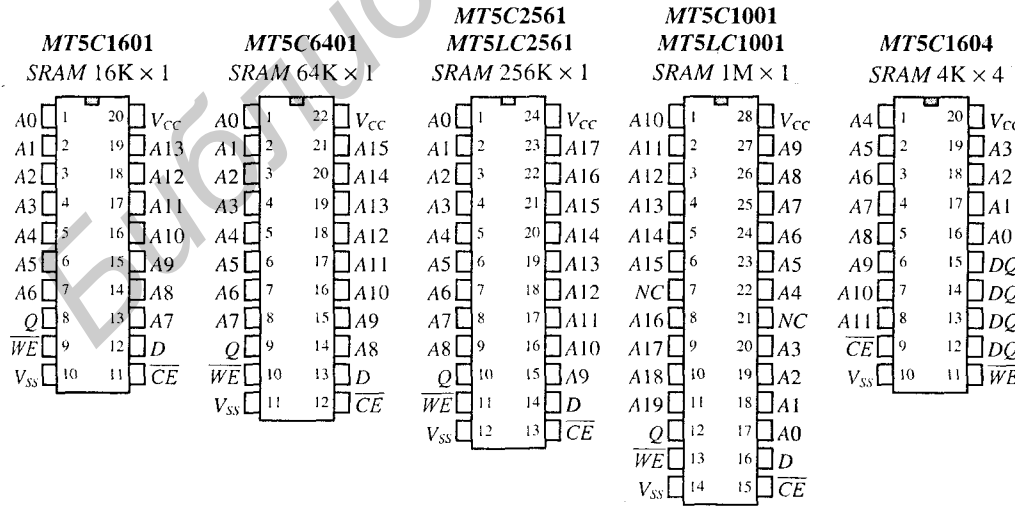
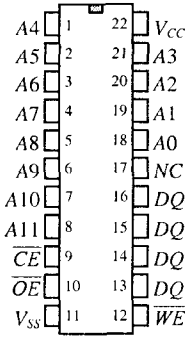
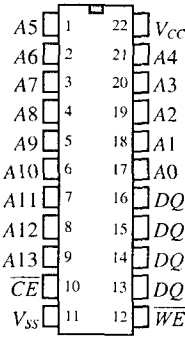


Рис. 1.45. Статические ОЗУ фирмы *Micron Semiconductor, Inc.*
(см. также с. 114 – 115)

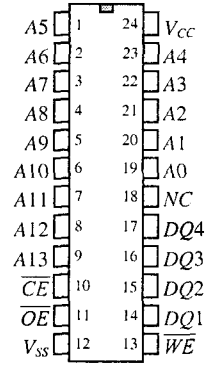
MT5C1605
SRAM 4K × 4



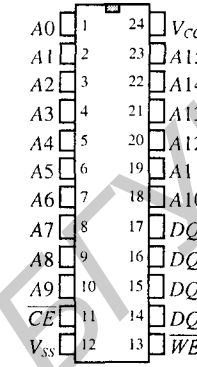
MT5C6404
SRAM 16K × 4



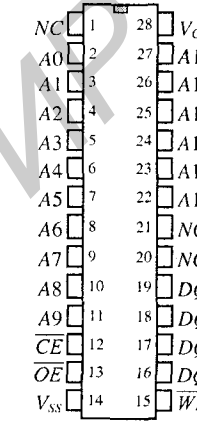
MT5C6405
SRAM 16K × 4



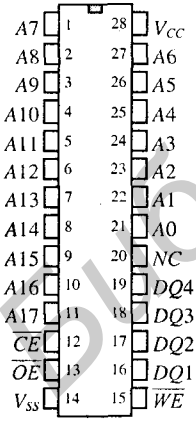
MT5C2564
MT5LC2564
SRAM 64K × 4



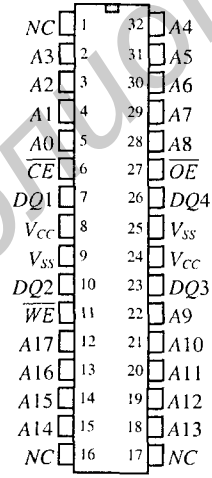
MT5C2565
MT5LC2565
SRAM 64K × 4



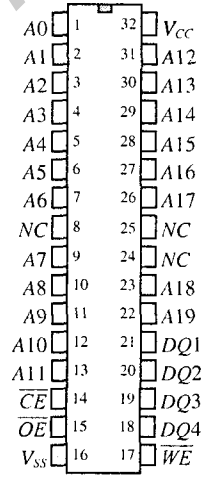
MT5C1005
MT5LC1005
SRAM 256K × 4



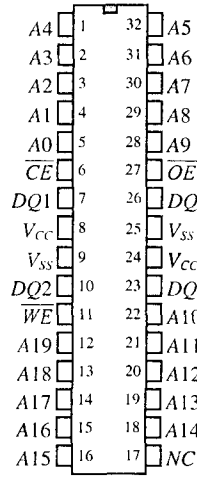
MT5C256K4A1
MT5LC256K4D4
SRAM 256K × 4



MT5C1M4A1
MT5LC1M4C3
SRAM 1M × 4



MT5C1M4B2
MT5LC1M4C4
SRAM 1M × 4



MT5C1608
SRAM 2K × 8

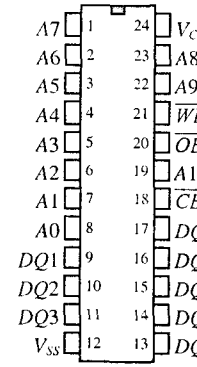
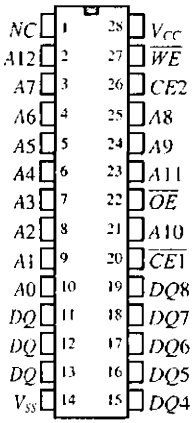
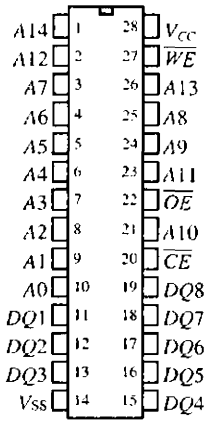


Рис. 1.45 (продолжение)

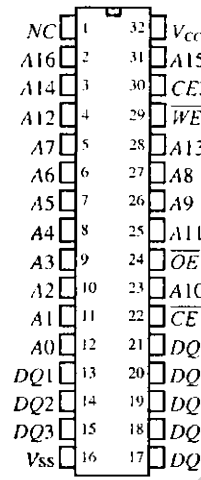
MT5C6408
SRAM 8K × 8



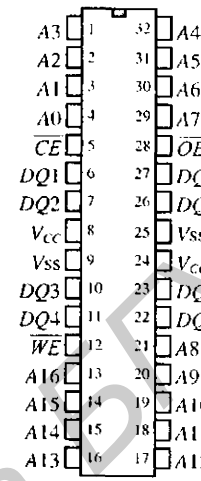
MT5C2568
MT5LC2568
SRAM 32K × 8



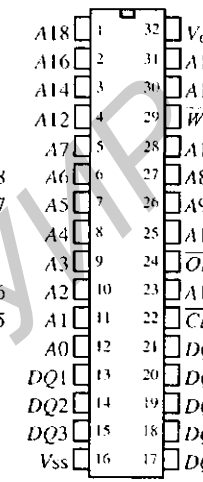
MT5C1008
MT5LC1008
SRAM 128K × 8



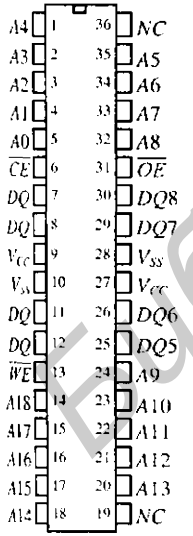
MT5C128K8A1
MT5LC128K8D4
SRAM 128K × 8



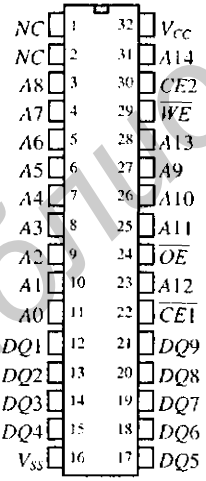
MT5C512K8A1
MT5LC512K8C3
SRAM 512K × 8



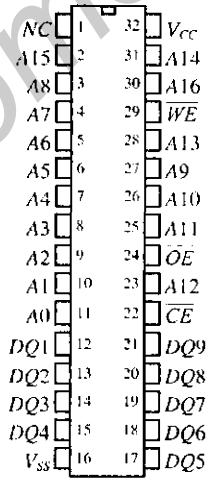
MT5C512K8B2
MT5LC512K8D4
SRAM 512K × 8



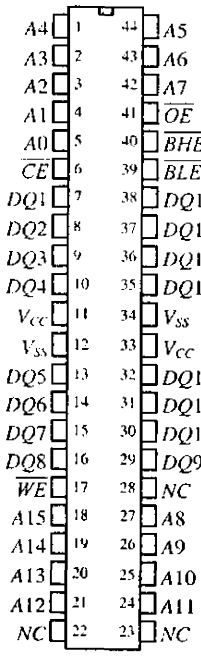
MT5C2889
SRAM 32K × 9



MT5C1189
SRAM 128K × 9



MT5C64K16A1
MT5LC64K16D4
SRAM 64K × 16



MT5C256K16B2
MT5LC256K16D4
SRAM 256K × 16

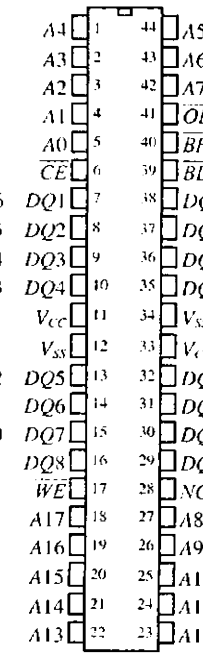


Рис. 1.45 (окончание)

Таблица 1.22. **Параметры *SRAM* фирмы *Micron Semiconductor, Inc.* (+5 В)**

Тип БИС ОЗУ	Объем памяти	Сигналы управления	Циклы, нс	Тип корпуса и число выводов				
				PDIP	*SOJ*	*ZIP*	*SOIC*	*TSOP*
MT5C1601	16K × 1	$\overline{CE}$	8 – 25	20	24	–	–	–
MT5C6401	64K × 1	$\overline{CE}$	8 – 25	22	24	–	–	–
MT5C2561	256K × 1	$\overline{CE}$	10 –35	24	24	–	–	–
MT5C1001	1M × 1	$\overline{CE}$	12 – 45	28	28	–	–	–
MT5C1604	4K × 4	$\overline{CE}$	8 – 25	20	24	–	–	–
MT5C1605	4K × 4	$\overline{CE}$, $\overline{OE}$	8 – 25	22	24	–	–	–
MT5C6404	16K × 4	$\overline{CE}$	8 – 25	22	24	–	–	–
MT5C6405	16K × 4	$\overline{CE}$, $\overline{OE}$	8 – 25	24	24	–	–	–
MT5C2564	64K × 4	$\overline{CE}$	10 – 35	24	24	–	24	–
MT5C2565	64K × 4	$\overline{CE}$, $\overline{OE}$	10 – 35	28	28	–	–	–
MT5C1005	256K × 4	$\overline{CE}$, $\overline{OE}$	12 – 45	28	28	–	–	–
*MT5C256K4A1**	256K × 4	$\overline{CE}$, $\overline{OE}$	12 – 25	–	32	–	–	–
MT5C1M4A1	1M × 4	$\overline{CE}$, $\overline{OE}$	20 – 55	–	32	–	–	–
*MT5C1M4B2**	1M × 4	$\overline{CE}$, $\overline{OE}$	20 – 35	–	32	–	–	32
MT5C1608	2K × 8	$\overline{CE}$, $\overline{OE}$	8 – 25	24	24	–	–	–
MT5C6408	8K × 8	$\overline{CE1}$, *CE2*, $\overline{OE}$	8 – 25	28	28	–	–	–
MT5C2568	32K × 8	$\overline{CE}$, $\overline{OE}$	10 – 35	28	28	28	–	–
MT5C1008	128K × 8	$\overline{CE1}$, *CE2*, $\overline{OE}$	12 – 45	32	32	–	–	–
*MT5C128K8A1**	128K × 8	$\overline{CE}$, $\overline{OE}$	12 – 25	–	32	–	–	–
MT5C512K8A1	512K × 8	$\overline{CE}$, $\overline{OE}$	20 – 55	–	32	–	–	–
*MT5C512K8B2**	512K × 8	$\overline{CE}$, $\overline{OE}$	20 – 35	–	36	–	–	36
MT5C2889	32K × 9	$\overline{CE1}$, *CE2*, $\overline{OE}$	15 – 25	–	32	–	–	–
MT5C1189	128K × 9	$\overline{CE}$, $\overline{OE}$	15 – 35	–	32	–	–	–
*MT5C64K16A1**	64 × 16	$\overline{CE}$, $\overline{OE}$, $\overline{BHE}$, $\overline{BLE}$	12 – 25	–	44	–	–	44
*MT5C256K16B2**	256K × 16	$\overline{CE}$, $\overline{OE}$, $\overline{BHE}$, $\overline{BLE}$	20 – 35	–	44	–	–	–

Примечание: * с центральным расположением выводов питания.

Таблица 1.23. **Параметры *SRAM* фирмы *Micron Semiconductor, Inc.* (+3,3 В)**

БИС ОЗУ типа *MT*× … ×	Объем памяти	Сигналы управления	Циклы, нс	Тип корпуса и число выводов				
				PDIP	*SOJ*	*ZIP*	*SOIC*	*TSOP*
5LC2561	256K × 1	$\overline{CE}$	15, 20, 25, 35	24	24	–	–	–
5LC1001	1M × 1	$\overline{CE}$	20, 25, 35, 45	28	28	–	–	–
5LC2564	64K × 4	$\overline{CE}$	15, 20, 25, 35	24	24	–	24	–
5LC2565	64K × 4	$\overline{CE}$, $\overline{OE}$	15, 20, 25, 35	28	28	–	–	–
5LC1005	256K × 4	$\overline{CE}$, $\overline{OE}$	20, 25, 35, 45	28	28	–	–	–
*5LC256K4D4**	256K × 4	$\overline{CE}$	20, 25	–	32		–	32
5LC1M4C3	1M × 4	$\overline{CE}$, $\overline{OE}$	20, 25, 35, 55	–	32	–	–	–
*5LC1M4C4**	1M × 4	$\overline{CE}$, $\overline{OE}$	20, 25, 35	–	32	–	–	32

Продолжение табл. 1.23

БИС ОЗУ типа *MT*× … ×	Объем памяти	Сигналы управления	Циклы, нс	Тип корпуса и число выводов				
				PDIP	*SOJ*	*ZIP*	*SOIC*	*TSOP*
5*LC*2568	32K × 8	$\overline{CE}$, $\overline{OE}$	15, 20, 25, 35	28	28	28	–	–
5*LC*1008	128K × 8	$\overline{CE}$1, *CE*2, $\overline{OE}$	20, 25, 35, 45	32	32	–	–	–
5*LC*128*K*8*D*4*	128K × 8	$\overline{CE}$, $\overline{OE}$	20, 25	–	32	–	–	32
5*LC*512*K*8*C*3	512K × 8	$\overline{CE}$, $\overline{OE}$	20, 25, 35, 55	–	32	–	–	–
5*LC*512*K*8*D*4*	512K × 8	$\overline{CE}$, $\overline{OE}$	20, 25, 35	–	36	–	–	36
5*LC*64*K*16*D*4*	64K × 16	$\overline{CE}$, $\overline{OE}$	20, 25	–	44	–	–	44
5*LC*256*K*16*D*4*	256K × 16	$\overline{BHE}$, $\overline{BLE}$	20, 25, 35	–	44	–	–	–

Примечание: * оперативные запоминающие устройства с центральным расположением выводов питания.

Статические ОЗУ фирмы *SGS-Thomson Microelectronics*. На рис. 1.46 изображены пять БИС сверхбыстродействующих (*Very Fast*) *SRAM* с фирменными обозначениями сигналов и выводов питания: A_k — разряды адреса ячеек памяти, DQ_m — разряды двунаправленных данных, $\overline{E}$ (*Chip Enable*) — выбор кристалла, $\overline{G}$ (*Output Enable*) — разрешение выхода, $\overline{W}$ (*Write Enable*) — разрешение записи, V_{CC} (*Supply Voltage*) — напряжение питания, V_{SS} (*Ground*) — земля. Используя ранее введенные обозначения сигналов, следует положить, что $\overline{E} = \overline{CE}$, $\overline{G} = \overline{OE}$, $\overline{W} = \overline{WE}$, $DQ_m = DB_m$ ($V_{SS} = GND$).

Изготовляются БИС по усовершенствованной фирменной *CMOS* технологии, характеризующейся малой мощностью потребления, высоким быстродействием и совместимостью всех входов и выходов с ТТЛ ИС. В каждой БИС обеспечены одинаковые времена выборки адреса и циклов записи и чтения. У БИС *SRAM M*62× … × напряжение питания V_{CC} = +5 В, а у БИС *SRAM M*63× … × — V_{CC} = +3,3 В.

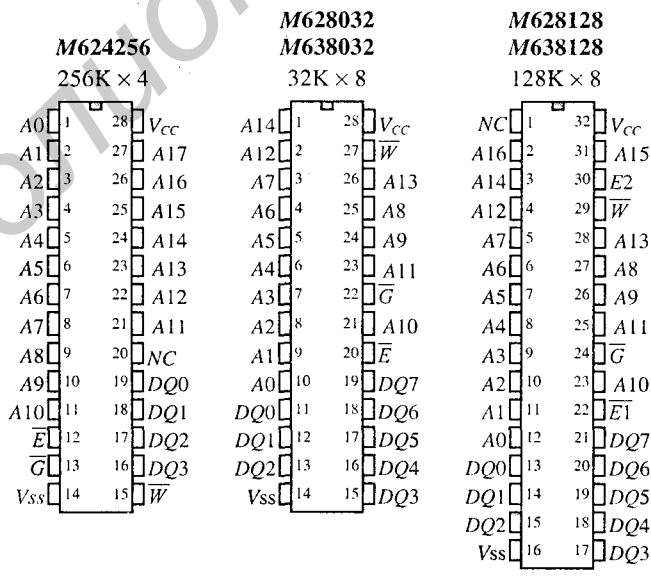


Рис. 1.46. *SRAM* фирмы *SGS-Thomson Microelectronics*

В табл. 1.24 представлены режимы работы *SRAM*, а на рис. 1.47 и 1.48 — временны́е диаграммы, поясняющие режимы записи и чтения данных на примере *SRAM M*628128-15. Это статическое оперативное запоминающее устройство имеет объем памяти 128К × 8 бит, время циклов записи и чтения t_{CY} = 15 нс и выпускается в корпусе *JEDEC Plastic SOJ*32, 400 *mil* (32 *lead Plastic Small Outline J-lead Package*). Селекция *SRAM* при записи и чтении данных DQ_m (m = 0, 1, …, 7) производится значением сигнала $\overline{E} = \overline{E_1 \cdot E_2} = 0$, т. е. при значениях физических сигналов выбора кристалла $\overline{E}_1 = 0$ и $E_2 = 1$. Селекция одной из $2^{17} = 2^7 \cdot 2^{10}$ = 128К 8-разрядных ячеек памяти производится значениями адресных сигналов A_{16-0}.

В пассивном состоянии ток потребления *SRAM* значительно меньше, чем в активном состоянии (табл. 1.25). Хранение данных без их разрушения обеспечивается при снижении напряжения питания V_{CC} до 2 В, что позволяет использовать аварийное батарейное питание. Ток потребления I_{CCDR} (*DR — Data Retention* — хранение данных) в этом режиме значительно меньше, чем при нормальной работе *SRAM*.

Таблица 1.24. **Режимы работы *SRAM***

Режим	$\overline{E}$	$\overline{W}$	$\overline{G}$	DQ_m	Состояние
Хранение	0	1	1	*Z*	Активное
Чтение	0	1	0	Выходы	Активное
Запись	0	0	×	Входы	Активное
Хранение	1	×	×	*Z*	Пассивное

Примечание: *Z* — *Z*-состояние; сигнал $E = E_1 \& E_2$ для *SRAM M*628128 и *M*638128.

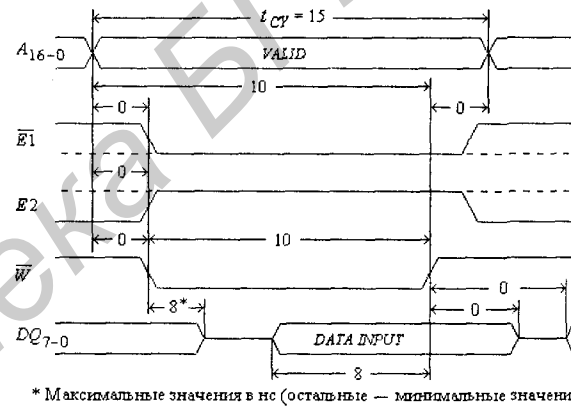


Рис. 1.47. Временны́е диаграммы режима записи *SRAM M*628128-15 ($\overline{G}$ = 0)

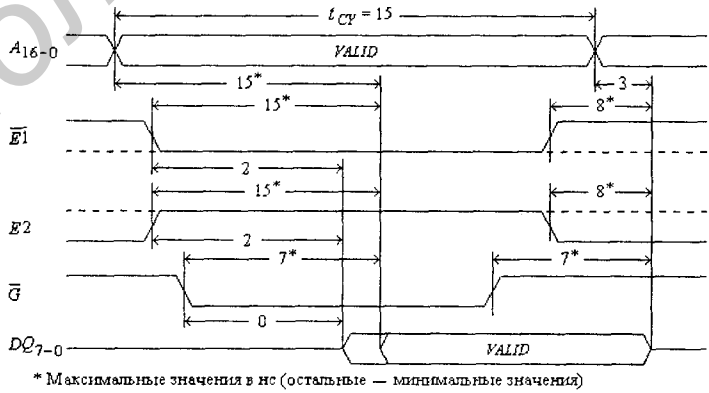


Рис. 1.48. Временны́е диаграммы режима чтения *SRAM M*628128-15 ($\overline{W}$ = 1)

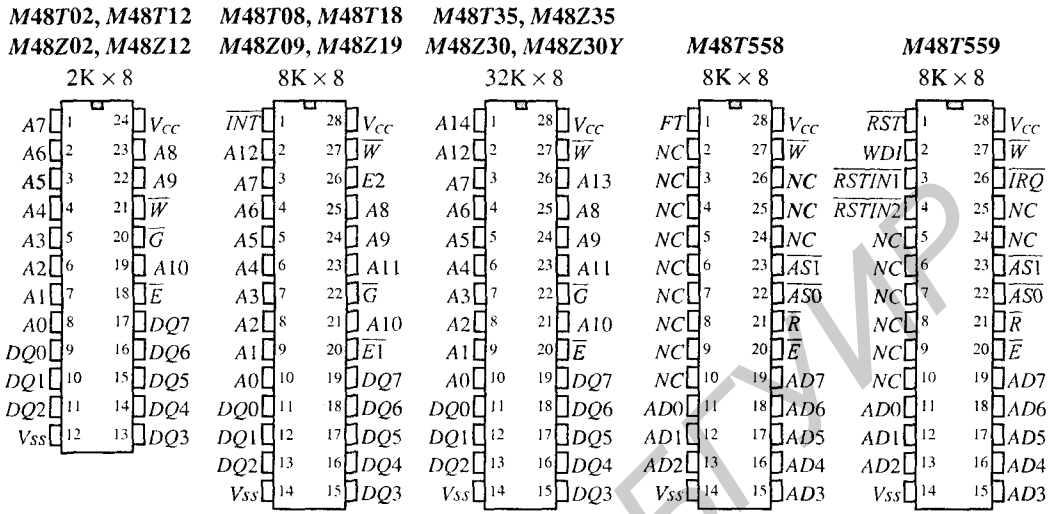
Таблица 1.25. **Параметры *SRAM* фирмы *SGS-Thomson Microelectronics***

Параметр	*M*624256 *M*628128	*M*628032	*M*638032	*M*638128
Длительность циклов записи/чтения, нс (*min*)	15/20	12/15/20	12/15/20	12/15/20
Ток потребления в активном состоянии ($\overline{E} = 0$):				
I_{CC1}[1] при V_{CC} = 5,5 В, мА (*max*)	175/140	160	150/130/115	—
Ток потребления в пассивном состоянии:				
I_{CC2}[2] при $\overline{E} = V_{IH}$ (ТТЛ), мА (*max*)	30	25	25	—
I_{CC3}[3] при $\overline{E} = V_{CC} - 0{,}2$ В (КМОП), мА (*max*)	4	1	1	—
Входные напряжения:				
низкого уровня V_{IL}, В (*max*)	0,8	0,8	0,8	—
высокого уровня V_{IH}, В (*min*)	2,2	2,2	2,2	—
Выходные напряжения:				
низкого уровня V_{OL} при I_{OL} = 8 мА, В (*max*)	0,4	0,4	0,4	—
высокого уровня V_{OH} при I_{OH} = –4 мА, В (*min*)	2,4	2,4	2,4	—
Токи утечки:				
входов при $0 \le V_{IN} \le V_{CC}$, мкА (*max*)	±1	±1	±1	—
выходов при $0 \le V_{OUT} \le V_{CC}$, мкА (*max*)	±5	±5	±1	—
Ток потребления в режиме хранения:				
I_{CCDR}[4] при V_{CC} = 3 В и $\overline{E} \ge V_{CC} - 0{,}2$ В, мкА (*max*)	500	200	150	—

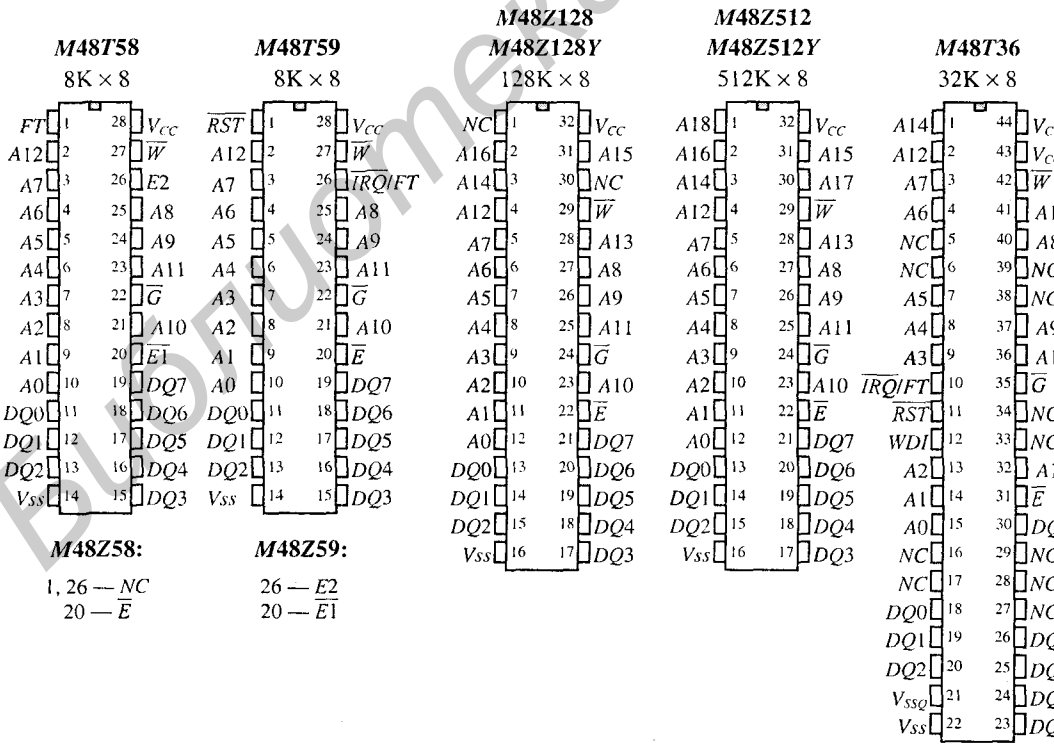
Примечание: [1] среднее значение переменного тока (открытые выходы, минимальная длительность циклов записи/чтения для *M*63××× V_{CC} = 3,6 В); [2] для всех других входов $V_{IL} \le 0{,}8$ В, $V_{IH} \ge 2{,}2$ В (V_{CC} = 5,5 В); для *SRAM M*628128 $\overline{E}_1 = V_{IH}$, $E_2 = V_{IL}$; [3] для всех других входов $V_{IL} \le 0{,}2$ В, $V_{IH} \ge V_{CC} - 0{,}2$ В (V_{CC} = 5,5 В); для *SRAM M*628128 $\overline{E}_1 = V_{CC} - 0{,}2$ В, $E_2 \le 0{,}2$ В; [4] для других входов $V_{IL} \le 0{,}2$ В, $V_{IH} \ge V_{CC} - 0{,}2$ В; для *SRAM M*628128 $\overline{E}_1 = V_{CC} - 0{,}2$ В, $E_2 \le 0{,}2$ В; для *SRAM M*638032 V_{CC} = 2 В.

Статические ОЗУ фирмы *SGS-Thomson Microelectronics* со встроенными часами. На рис. 1.49 представлены *SRAM*, имеющие встроенные часы (*Timekeeper*), и *SRAM* с ультранизким потреблением мощности (*Ultra Low Power*, *Zero Power*). Маркируются эти *SRAM* буквами *T* (*Timer*) и Z (*Zero*).

Все эти БИС имеют встроенный в корпус литиевый элемент (*Lithium Cell* — рис. 1.50), обеспечивающий работу часов и (или) сохранение записанных данных при выключении или выходе из строя внешнего источника питания. Для перехода в режим хранения с запретом записи данных используются датчик напряжения и переключающая схема (*Voltage Sense and Switching Circuitry*), контролирующие напряжения питания V_{CC} на предмет выхода его значения из допустимого диапазона. При этом автоматически блокируется сигнал выбора кристалла (*Chip Deselect*) и включается защита записи (*Write Protection*), что обеспечивает высокую степень защиты данных от непредсказуемой системной операции, вызванной низким значением напряжения питания V_{CC}. Когда напряжение питания V_{CC} падает ниже приблизительно 3 В, схема управления подключает батарею, которая поддерживает хранение данных и операции часов до восстановления допустимого значения напряжения питания. В отсутствие питания литиевый элемент обеспечивает хранение данных и операции часов в течение 10 лет.



M48Z08, M48Z18: 1, 26 — *NC*, 20 — $\overline{E}$



M48Z58:
1, 26 — *NC*
20 — $\overline{E}$

M48Z59:
26 — *E2*
20 — $\overline{E1}$

Рис. 1.49. *SRAM* фирмы *SGS-Thomson Microelectronics*

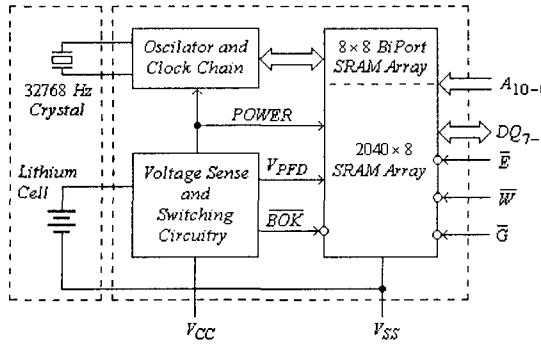


Рис. 1.50. Структурная схема *SRAM M48T02/M48T12*

Таблица 1.26. **Режимы работы *SRAM M48T02/M48T12***

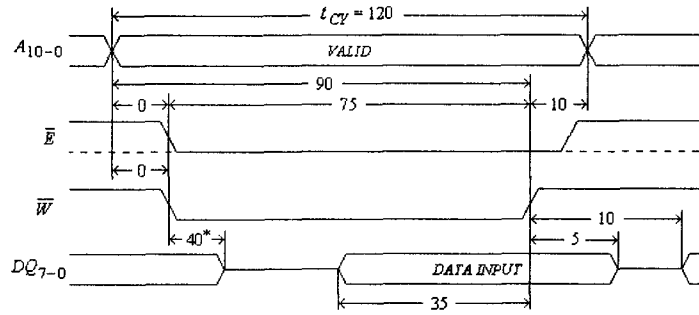
Режим	V_{CC}	$\overline{E}$	$\overline{G}$	$\overline{W}$	DQ_m	Состояние
Хранение	От 4,75 до 5,5 В	1	×	×	*Z*-состояние	Пассивное
Запись	для *M48T02*	0	×	0	Входы	Активное
Чтение	От 4,5 до 5,5 В	0	0	1	Выходы	Активное
Чтение	для *M48T12*	0	1	1	*Z*-состояние	Активное
Хранение	От V_{SO} до $V_{PFD\,min}$	×	×	×	*Z*-состояние	*CMOS*-пассивное
Хранение	$\leq V_{SO}$	×	×	×	*Z*-состояние	Батарейное питание

Примечание: V_{PFD} — *Power-fail Deselect Voltage* (напряжение отключения кристалла при повреждении питания); $V_{SO\,typ} = 3$ В — *Battery Back-up Switchover Voltage* (напряжение переключения на батарейное питание).

По расположению выводов и их функциональному назначению эти БИС выполнены в соответствии со стандартом *JEDEC* (*Joint Electron Device Engineering Council* — Объединенный Совет по электронным приборам), как и БИС *ROM*, *PROM*, *EPROM* и *EEPROM* других зарубежных фирм. В табл. 1.26 показаны режимы работы *SRAM M48T02/M48T012* (напряжения защиты записи: 4,5 В $\leq V_{PFD} \leq$ 4,75 В для *M48T02* и 4,2 В $\leq V_{PFD} \leq$ 4,5 В для *M48T12*).

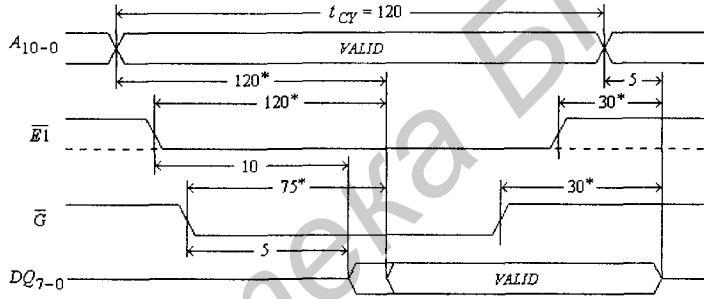
Поскольку *SRAM* типа *T* (*Timer*) и *Z* (*Zero*) предназначены для применения с батарейным питанием, то для уменьшения мощности потребления от источника питания они спроектированы с большими значениями длительностей циклов записи и чтения (рис. 1.51 и 1.52), чем описанные выше БИС *SRAM*, предназначенные для использования только в качестве статических оперативных запоминающих устройств (см. рис. 1.47 и 1.48).

Встроенный в корпус кварцевый резонатор на 32,768 кГц обеспечивает стабильную частоту тактового сигнала счетчиков часов (*Oscillator and Clock Chain* — см. рис. 1.50), выдающих год (*Year*), месяц (*Month*), день (*Day*), дату (*Date*), часы (*Hours*), минуты (*Minutes*) и секунды (*Seconds*) в двоично-десятичном коде (табл. 1.27). Корректировка числа дней в месяце 28, 29 (високосного года), 30 и 31 производится автоматически. Точность хода часов составляет ±1 мин/месяц при температуре 25 °С. Для приложений высокой точности предусмотрено программное управление калибровкой часов (рис. 1.53). Данные переписываются из счетчиков часов в 8-байтовую двухпортовую область *SRAM* (8 × 8 *BiPort SRAM Array*) один раз в секунду. Чтение данных из этой области памяти производится так же, как и чтение остальных данных.



* Максимальные значения в нс (остальные – минимальные значения)

Рис. 1.51. Временны́е диаграммы режима записи *SRAM M48T02/M48T12*-120 ($\overline{G}$ = 0)



* Максимальные значения в нс (остальные – минимальные значения)

Рис. 1.52. Временны́е диаграммы режима чтения *SRAM M48T02/M48T12*-120 ($\overline{W}$ = 1)

Таблица 1.27. **Адресация регистров часов *SRAM M48T02/M48T12***

Адрес	Данные								Функция	Значение
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
7FF*h*	10 *Year*				*Year*				*Year*	00 ÷ 99
7FE*h*	0	0	0	10 *M*	*Month*				*Month*	01 ÷ 12
7FD*h*	0	0	10 *Date*		*Date*				*Date*	01 ÷ 31
7FC*h*	0	*FT*	0	0	0	*Day*			*Day*	01 ÷ 07
7FB*h*	*KS*	0	10 *Hours*		*Hours*				*Hours*	00 ÷ 23
7FA*h*	0	10 *Minutes*			*Minutes*				*Minutes*	00 ÷ 59
7F9*h*	*ST*	10 *Seconds*			*Seconds*				*Seconds*	00 ÷ 59
7F8*h*	*W*	*R*	*S*	*Calibration*					*Control*	—

Примечание: *FT* (*frequency test*) — тестирование частоты (0 — для нормальной работы часов), *KS* (*kick-start*) — запуск часов, *ST* (*stop*) — останов часов, *W* (*write*) — запись, *R* (*read*) — чтение, *S* (*sign*) — знак калибровки *Calibration*, 0 — разряд следует установить в 0.

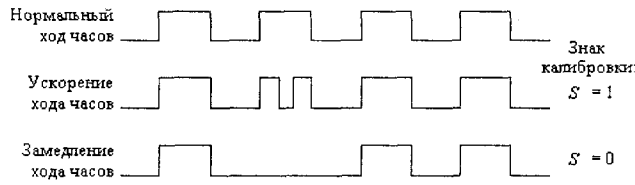


Рис. 1.53. Калибровка часов

Байт, расположенный по адресу 7F8*h*, используется как регистр управления часами. Этот байт предназначен для управления доступом к информации часов и хранения установки калибровки часов. Установка значений полей *S* (знак калибровки 0 или 1) и *Calibration* (0 ÷ 31) позволяют замедлить (*S* = 0) или ускорить (*S* = 1) ход часов.

Точность хода часов без калибровки составляет ±1 мин/месяц при температуре окружающей среды 25 °С. Изменение значения поля калибровки *Calibration* на ±1 изменяет скорость хода часов на +10,7 с/месяц и –5,35 с/месяц. Диапазон изменения скорости хода часов составляет +5,5 ÷ –2,75 минуты в месяц.

1.11. Обнаружение и исправление ошибок в оперативных запоминающих устройствах

Если при тестировании операций записи и чтения данных из оперативного запоминающего устройства (ОЗУ) получаются неверные данные, то такое ОЗУ должно быть заменено на исправное. Однако ошибки в работе ОЗУ могут быть вызваны и редкими случайными помехами, хотя тестирование ОЗУ проходит успешно. В МП-системах, имеющих особо важное назначение, используются аппаратные средства обнаружения и исправления ошибок, возможно и ни разу не возникающих в одном сеансе их работы.

Принцип обнаружения однократных ошибок. В системах связи наиболее вероятной ошибкой передачи *n*-разрядных данных являются искажения одного их разряда (однократная ошибка). Для обнаружения таких ошибок используются 9-разрядные ИС контроля четности/нечетности — *Parity Generators/Checker* (рис. 1.54; *NC* — *No Connection* — контакт не используется). Основные параметры этих ИС приведены в табл. 1.28.

Принципиальная схема ИС ‘280 изображена на рис. 1.55 — реализуются прямая *PO* и инверсная *PE* функции по модулю два от всех девяти входных сигналов I_p (p = 0, 1, …, 8).

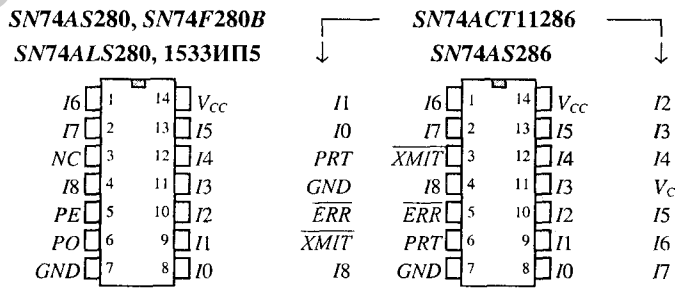


Рис. 1.54. Схемы контроля четности/нечетности

Таблица 1.28. **Параметры ИС контроля четности/нечетности**

ИС	$I_{CC\ typ}$, мА	$I_{CC\ max}$, мА	$V_{OH\ min}/I_{OH}$, В/мА	$V_{OH\ typ}/I_{OH}$, В/мА	$V_{OL\ typ}/I_{OL}$, В/мА	$V_{OL\ max}/I_{OL}$, В/мА	$t_{p\ min}$, нс	$t_{p\ typ}$, нс	$t_{p\ max}$, нс
'*ALS*280	10	16	2,4/–2,6	3,3/–2,6	0,35/24	0,5/24	3	—	20
'*AS*280	25	35	$(V_{CC}-2)$/–2		0,35/20	0,5/20	3	—	12
'*F*280*B*	26	35	2,5/–1	3,4/–1	0,3/20	0,5/20	2,7	—	11
'*AS*286	35	50	2,4/–15	—	0,35/20	0,5/20	3	—	16
'*ACT*11286	—	0,008	3,94/–24	—	—	0,36/24	3,6	7,3	10,8

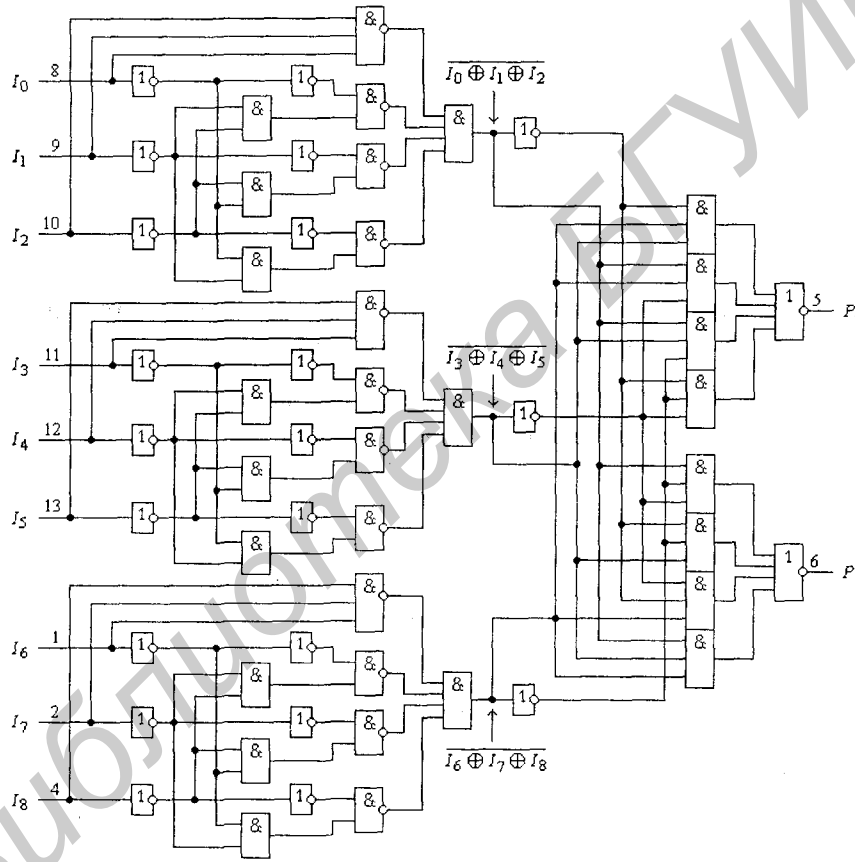


Рис. 1.55. Принципиальная схема ИС '280

Все ИС типа '280 описываются функциями:

$$PE = \overline{\sum_{p=0}^{8} I_p} = \overline{I_8} \oplus \sum_{p=0}^{7} I_p \quad \text{и} \quad PO = \sum_{p=0}^{8} I_p = I_8 \oplus \sum_{p=0}^{7} I_p,$$

где I_p — входные сигналы (p = 0, 1, …, 8), PE (*Parity Even*) — четный паритет (выходной сигнал) и PO (*Parity Odd*) — нечетный паритет (выходной сигнал), причем $PO = \overline{PE}$, Σ — операция сумма по модулю два.

В передатчиках 8-разрядных данных (рис. 1.56) ИС ‘280 используются в качестве генератора паритета (*Parity Generators*), формирующего проверочный разряд *PE* (или *PO*). С помощью сигнала I_8 (или любого другого) задается контроль четности (следует задать $I_8 \equiv 1$) или нечетности ($I_8 \equiv 0$). Например, если входной сигнал $I_8 \equiv 0$, то выходной сигнал $PE = 1$ только при четном числе информационных сигналов I_{7-0}, равных единице, т. е. по линии связи передаются 9-разрядные коды

$$PE_1 D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0,$$

содержащие нечетное число единиц. В приемниках 8-разрядных данных ИС ‘280 используются в качестве устройств проверки паритета (*Parity Checker*) — на вход I_8 подается проверочный разряд *PE* (или *PO*), сформированный генератором паритета передатчика, а на выходах *PE* и *PO* приемника генерируются сигналы ошибки.

Канал передачи данных с контролем нечетности, приведенный на рис. 1.56, описывается функциями

$$PE_1 = \overline{\sum_{p=0}^{7} D_p}, \quad PE_2 = \overline{PE'}_1 \oplus \sum_{p=0}^{7} D'_p.$$

Воздействующие на передаваемые коды помехи могут исказить некоторые разряды — на выходе линии связи будет получен код $PE'_1 D'_7 D'_6 D'_5 D'_4 D'_3 D'_2 D'_1 D'_0$, отличающийся от передаваемого кода. Если при передаче кода ошибки отсутствуют, то сигнал ошибки

$$PE_2 = \overline{PE'}_1 \oplus \sum_{p=0}^{7} D'_p = \sum_{p=0}^{7} D_p \oplus \sum_{p=0}^{7} D_p = 0 \text{ — нечетный паритет (ошибки нет).}$$

Если в линии связи произойдет искажение нечетного числа разрядов, то в приемнике будет получено значение сигнала $PE_2 = 1$ — четный паритет (ошибка в передаче данных). Искажение четного числа разрядов не обнаруживается, поэтому считается, что устройства контроля четности/нечетности используются для обнаружения однократных ошибок. Но и появление ошибки в одном разряде данных наиболее вероятно.

Из девятиразрядных ИС ‘280 легко построить схемы контроля четности/нечетности большей разрядности — на рис. 1.57 представлена 81-разрядная схема контроля четности/нечетности.

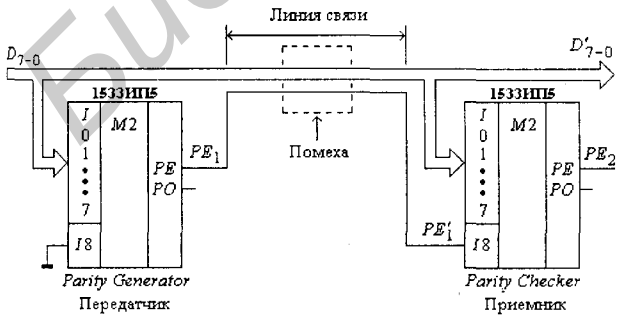


Рис. 1.56. Канал передачи данных с контролем нечетности

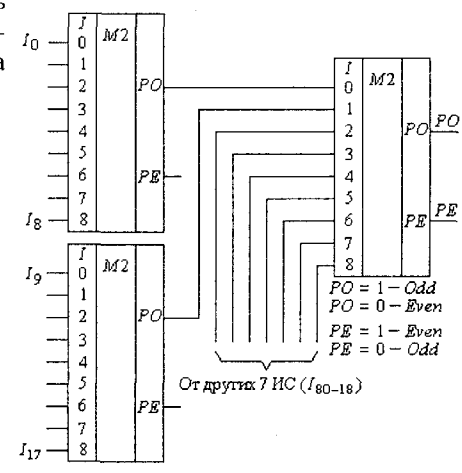


Рис. 1.57. Каскадирование ИС ‘280

ОЗУ с обнаружением однократных ошибок. Ошибки в работе ОЗУ происходят не в линиях связи, а при записи и чтении данных, например, из-за импульсных помех источника питания. Для построения ОЗУ с обнаружением однократных ошибок необходимо в каждую ячейку памяти добавить один разряд для записи и чтения проверочного разряда (разряда паритета). Следовательно такое ОЗУ можно построить на основе БИС *SRAM MT5C*2889 (32 К × 9 бит) или *MT5C*1189 (128 К × 9 бит; см. рис. 1.45). Так как шина данных у ОЗУ двунаправленная, то и ИС контроля четности/нечетности должна иметь двунаправленный проверочный разряд паритета. Специально для этих целей выпускаются ИС *SN*74*AS*286 и *SN*74*ACT*11286 — *9-bit Parity Generators/Checker with Bus Driver Parity I/O Port* (см. рис. 1.54).

Принципиальная схема ИС ‘286 изображена на рис. 1.58, *а*:

$\overline{XMIT}$ (*Transmit Control*) — входной сигнал управления направлением передачи проверочного разряда *PRT*,

PRT (*Parity I/O*) — двунаправленный проверочный разряд (*PRT* — выход при $\overline{XMIT} = 0$ и *PRT* — вход при $\overline{XMIT} = 1$),

$\overline{ERR}$ (*Parity Error*) — выходной сигнал ошибки паритета.

Данная ИС описывается функциями:

$$PRT = \overline{\alpha} = \overline{\sum_{p=0}^{8} I_p} \text{ при } \overline{XMIT} = 0, \quad \overline{ERR} = \overline{\beta \cdot \overline{XMIT}} = \overline{(\overline{PRT} \oplus \sum_{p=0}^{8} I_p) \cdot \overline{XMIT}},$$

где *PRT* — выходной и входной сигнал. При значении сигнала $\overline{XMIT} = 0$ ИС ‘280 выполняют функцию генератора паритета, формирующего проверочный разряд *PRT*, а при значении $\overline{XMIT} = 1$ — функцию устройства проверки паритета, генерирующего сигнал ошибки

$$\overline{ERR} = PRT \oplus \sum_{p=0}^{8} I_p,$$

где *PRT* — входной сигнал. В табл. 1.29 приведено наглядное описание этих функций.

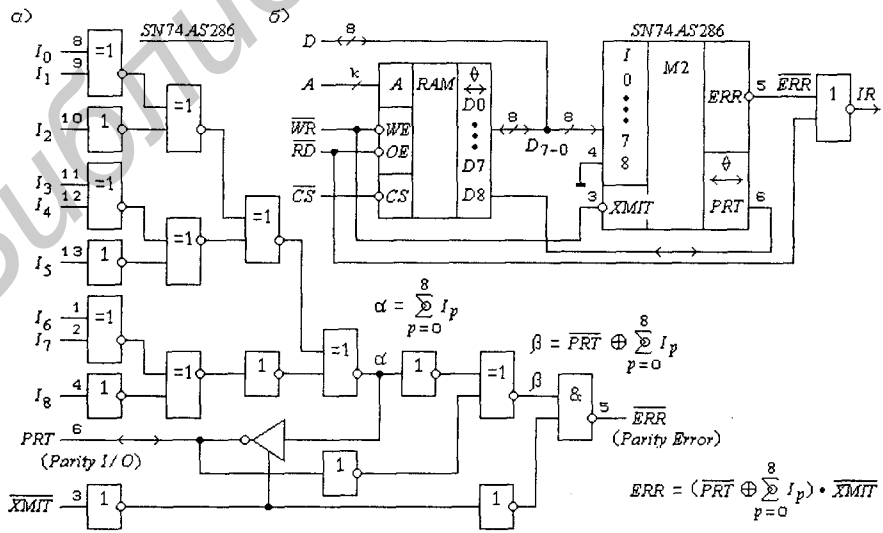


Рис. 1.58. ИС контроля четности/нечетности ‘286

Таблица 1.29. **Описание работы ИС ‘286**

Число единиц на входах I_p	$\overline{XMIT}$	PRT	$\overline{ERR}$	Примечание
Четное ($\alpha = 0$)	0	1 (выход)	1	Передача (нет ошибки)
Нечетное ($\alpha = 1$)	0	0 (выход)	1	Передача (нет ошибки)
Четное ($\alpha = 0$)	1	1 (вход)	1	Прием (нет ошибки)
	1	0 (вход)	0	Прием (есть ошибка)
Нечетное ($\alpha = 1$)	1	1 (вход)	0	Прием (есть ошибка)
	1	0 (вход)	1	Прием (нет ошибки)

На рис. 1.58, *б* показана структурная схема ОЗУ с обнаружением однократных ошибок. При записи данных в ОЗУ значения сигналов $\overline{XMIT} = \overline{WR} = 0$, $\overline{RD} = 1$, $\overline{ERR} = 1$ (неактивный уровень сигнала ошибки) и ИС ‘286 генерирует проверочный разряд

$$PRT = \overline{\sum_{p=0}^{7} D_p},$$

являющийся ее выходным сигналом. Значение проверочного разряда PRT вместе с данными D_{7-0} записывается в ОЗУ. При чтении данных из ОЗУ значения сигналов $\overline{XMIT} = \overline{WR} = 1$, $\overline{RD} = 0$ и разряд PRT' поступает из ОЗУ (возможно искаженный), а сигнал ошибки паритета

$$\overline{ERR} = PRT' \oplus \sum_{p=0}^{7} D'_p.$$

При отсутствии ошибок

$$PRT' = PRT = \overline{\sum_{p=0}^{7} D_p} \quad \text{и} \quad \overline{ERR} = \overline{\sum_{p=0}^{7} D_p} \oplus \sum_{p=0}^{7} D_p = 1.$$

На рис. 1.59 показана структурная схема МП-системы с обнаружением однократных ошибок в ОЗУ. При отсутствии ошибок сигнал запроса прерывания $IR \equiv 1$. При однократной ошибке сигнал $\overline{ERR} = 0$ и запрос прерывания $IR = 0$. Так как ошибки в работе ОЗУ имеют катастрофические последствия, то нормальная работа МП-системы должна быть приостановлена. Поэтому сигнал ошибки используется для вызова подпрограммы обработки прерывания (находящейся в постоянном запоминающем устройстве) по входу запроса прерывания *TRAP*, реагирующего на положительный фронт сигнала *IR* с последующим удержанием его высокого уровня.

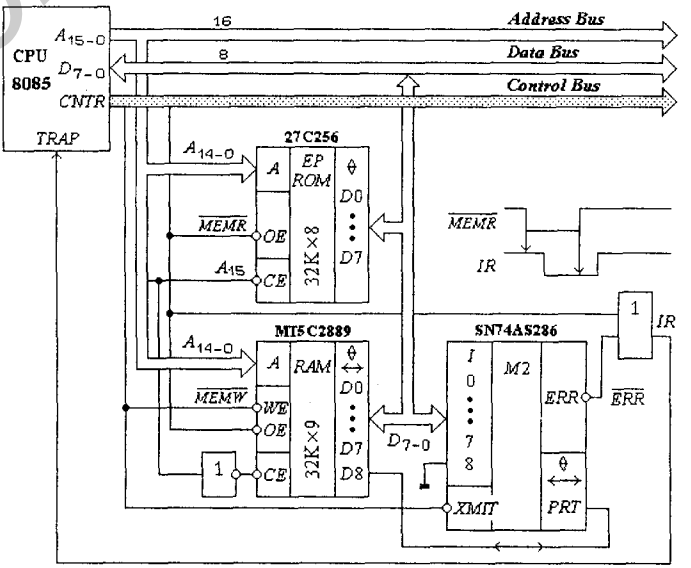


Рис. 1.59. ОЗУ с обнаружением однократных ошибок

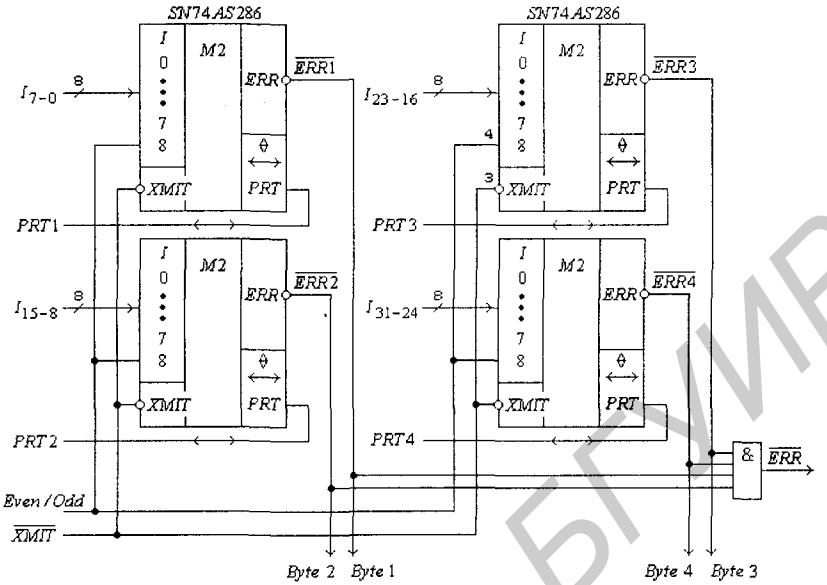


Рис. 1.60. Каскадирование ИС ‘286

В МП-системах с 16-разрядной и 32-разрядной шинами данных также можно использовать ИС ‘286 для обнаружения однократных ошибок. В таких МП-системах производится независимый контроль паритета каждого байта данных. На рис. 1.60 показана схема для побайтного контроля паритета в 32-разрядных МП-системах, генерирующая сигнал ошибки паритета $\overline{ERR} = \overline{ERR1} \vee \overline{ERR2} \vee \overline{ERR3} \vee \overline{ERR4}$. Этот сигнал следует использовать для формирования сигнала запроса прерывания *IR*.

Обнаружение и исправление ошибок в ОЗУ. Некоторые линейные коды, называемые кодами Хэмминга [9], позволяют обнаруживать двукратные и исправлять однократные ошибки в каналах передачи данных. В кодах Хэмминга используется не один, а несколько проверочных разрядов, представляющих собой некоторые функции по модулю два от информационных разрядов. Для обнаружения двукратных и исправления однократных ошибок в ОЗУ фирмой *Texas Instruments* на основе модифицированных кодов Хэмминга были разработаны устройства обнаружения и исправления ошибок (*EDAC — Parallel Error Detection and Correction Circuits*), которые могут использоваться в МП-системах с 8-, 16- и 32-разрядными шинами данных (рис. 1.61):

*SN*74*LS*636 — 8-разрядный *EDAC* с *Z*-состоянием шин данных и 5-разрядного проверочного слова;

*SN*74*LS*637 — 8-разрядный *EDAC* с открытым коллекторным выходом шин данных и 5-разрядного проверочного слова;

*SN*74*LS*630 (555ВЖ1) — 16-разрядный *EDAC* с *Z*-состоянием шин данных и 6-разрядного проверочного слова без побайтного управления записью данных;

*SN*74*LS*631 — 16-разрядный *EDAC* с открытым коллекторным выходом шин данных и 6-разрядного проверочного слова без побайтного управления записью данных;

*SN*74*ALS*616 — 16-разрядный *EDAC* с *Z*-состоянием шин данных и 6-разрядного проверочного слова с побайтным управлением записью данных;

*SN*74*ALS*617 — 16-разрядный *EDAC* с открытым коллекторным выходом шин данных и 6-разрядного проверочного слова с побайтным управлением записью данных;

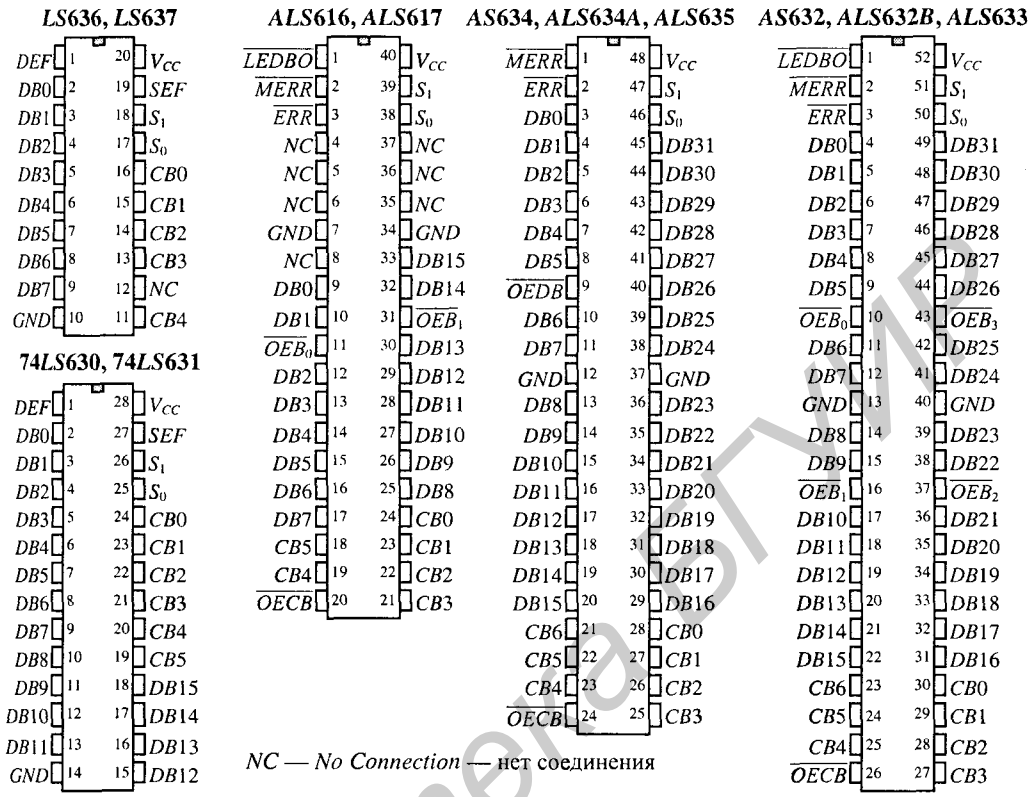


Рис. 1.61. ИС для обнаружения и исправления ошибок (*EDAC*)

*SN*74*ALS*632*B*, *SN*74*AS*632 — 32-разрядный *EDAC* с *Z*-состоянием шины данных и 7-разрядного проверочного слова с побайтным управлением записью данных;

*SN*74*ALS*633 — 32-разрядный *EDAC* с открытым коллекторным выходом шины данных и 7-разрядного проверочного слова с побайтным управлением записью данных;

*SN*74*ALS*634*A*, *SN*74*AS*634 — 32-разрядный *EDAC* с *Z*-состоянием шины данных и 7-разрядного проверочного слова без побайтного управления записью данных;

*SN*74*ALS*635 — 32-разрядный *EDAC* с открытым коллекторным выходом шины данных и 7-разрядного проверочного слова без побайтного управления записью данных.

Основные параметры этих ИС приведены в табл. 1.30 (время распространения сигналов t_p приведено только от входов данных DB_i до выходов проверочных разрядов CB_j). Устройства обнаружения и исправления ошибок *EDAC* с побайтным управлением записью данных '636/'637 можно использовать в МП-системах с 8-разрядной шиной данных (МП 8080 и 8085), '616/'617 — с 16-разрядной шиной данных (МП 8086, 80186 и 80286) и '632/'633 — с 32-разрядной шиной данных (МП 80386 и 80486).

Структурная схема *EDAC* '636 показана на рис. 1.62 (*EDAC* '630 описывается такой же структурной схемой, но разрядность шин данных и проверочного слова другие — 16 и 6 бит):

DB_{7-0} (*Data Bite*) — разряды шины данных МП;

CB_{4-0} (*Check Bite*) — разряды проверочного слова;

S_1, S_0 — сигналы управления внутренними операциями *EDAC* (табл. 1.31);

SEF (*Single-bit Errors Flag*) — флаг обнаружения и исправления однократных ошибок;

DEF (*Dual-bit Errors Flag*) — флаг обнаружения двукратных ошибок.

Таблица 1.30. **Параметры** *EDAC*

ИС	$I_{CC\ typ}$, мА	$I_{CC\ max}$, мА	$V_{OH\ min}/I_{OH}$, В/мА	$V_{OH\ typ}/I_{OH}$, В/мА	$V_{OL\ typ}/I_{OL}$, В/мА	$V_{OL\ max}/I_{OL}$, В/мА	$t_{p\ min}$, нс	$t_{p\ typ}$, нс	$t_{p\ max}$, нс
‘616	110	170	2,4/–2,6	3,2/–2,6	0,25/12	0,4/12	10	—	49
‘617	110	—	V_{CC} – 2/–0,4	—	0,25/12	0,4/12	—	40	—
‘630 *	143	230	2,7/–0,4	3,4/–0,4	0,35/8	0,5/8	—	31	65
‘631 *	113	180	2,7/–0,4	3,4/–0,4	0,35/8	0,5/8	—	31	65
‘632*B*, ‘634	150	250	2,4/–2,6	3,2/–2,6	0,35/12	0,4/12	10	—	48
‘633*A*, ‘635	150	250	V_{CC} – 2/–0,4	—	0,25/12	0,4/12	10	40	60
‘636 *	100	160	2,7/–0,4	3,4/–0,4	0,35/8	0,5/8	—	31	45
‘637 *	90	144	2,7/–0,4	3,4/–0,4	0,25/12	0,4/12	—	38	55

Примечание: * — *LS*-технология, остальные — *ALS*-технология.

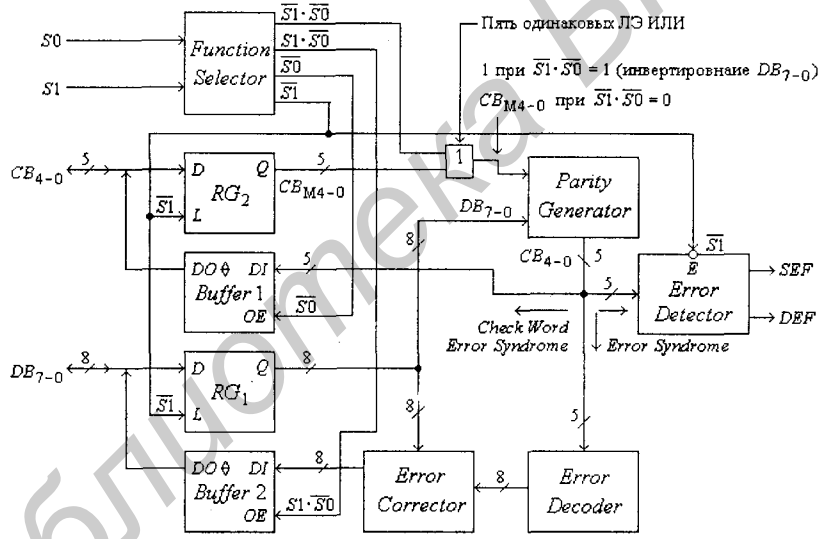


Рис. 1.62. Структурная схема *EDAC* ‘636

Таблица 1.31. **Управление работой ИС ‘636, ‘637, ‘630 и ‘631**

Цикл *RAM*	S_1	S_0	Выполняемая *EDAC* функция	DB_i	CB_j	*SEF*	*DEF*
Запись	0	0	Генерация проверочного слова CB_{4-0}	Вход	Выход *CW*	0	0
Чтение	0	1	Чтение данных DB_{7-0} и проверочного слова CB_{4-0} из памяти	Вход	Вход *CW*	0	0
Чтение	1	1	Фиксация DB_{7-0} и CB_{4-0} в регистрах и установка флагов ошибок *SEF* и *DEF*	Фиксация	Фиксация *CW*	0/1	0/1
Чтение	1	0	Коррекция слова данных DB_{7-0} и генерация синдрома ошибки CB_{4-0}	Выход	Выход синдрома	0/1	0/1

Разряды CB_j (j = 0, 1, ..., 4) 5-разрядного проверочного слова *CW* (*Check Word*) и синдрома ошибки CB_j (*Error Syndrome*), используемого для обнаружения ошибок и исправления однократных ошибок в байте данных DB_{7-0}, вычисляет узел *Parity Generator* (рис. 1.62) на основании соотношений:

$$CB_0 = DB_4 \oplus DB_3 \oplus DB_1 \oplus DB_0 \oplus (CB_{M0} \vee \overline{S}_1\overline{S}_0),$$
$$CB_1 = DB_6 \oplus DB_5 \oplus DB_3 \oplus DB_2 \oplus DB_0 \oplus (CB_{M1} \vee \overline{S}_1\overline{S}_0),$$
$$CB_2 = DB_7 \oplus DB_5 \oplus DB_4 \oplus DB_2 \oplus DB_1 \oplus (CB_{M2} \vee \overline{S}_1\overline{S}_0),$$
$$CB_3 = DB_7 \oplus DB_6 \oplus DB_2 \oplus DB_1 \oplus DB_0 \oplus (CB_{M3} \vee \overline{S}_1\overline{S}_0),$$
$$CB_4 = DB_7 \oplus DB_6 \oplus DB_5 \oplus DB_4 \oplus DB_3 \oplus (CB_{M4} \vee \overline{S}_1\overline{S}_0),$$

где $\overline{S}_1\overline{S}_0$ — сигнал управления, задающий вычисление проверочного слова ($\overline{S}_1\overline{S}_0 = 1$, DB_i — записываемые в память разряды слова данных) и синдрома ошибки ($\overline{S}_1\overline{S}_0 = 0$, $DB_i = DB_{Mi}$ и CB_{Mj} — разряды байта данных и проверочного слова, прочитанных из памяти), а индекс М означает, что разряд может быть прочитан из памяти с ошибкой (0 вместо 1 или 1 вместо 0).

Таблица 1.32. **Вычисление проверочных разрядов в ИС ‘636 и ‘637**

CB_j	Разряды байта данных DB_{7-0}							
	7	6	5	4	3	2	1	0
CB_0	–	–	–	+	+	–	+	+
CB_1	–	+	+	–	+	+	–	+
CB_2	+	–	+	+	–	+	+	–
CB_3	+	+	–	–	–	+	+	+
CB_4	+	+	+	+	+	–	–	–

Таким образом, проверочные разряды вычисляются по формулам (в табл. 1.32 знаком “+” отмечены разряды байта данных DB_{7-0}, используемые при вычислении проверочных разрядов):

$$CB_0 = DB_4 \oplus DB_3 \oplus DB_1 \oplus DB_0 \oplus 1,$$
$$CB_1 = DB_6 \oplus DB_5 \oplus DB_3 \oplus DB_2 \oplus DB_0 \oplus 1,$$
$$CB_2 = DB_7 \oplus DB_5 \oplus DB_4 \oplus DB_2 \oplus DB_1 \oplus 1,$$
$$CB_3 = DB_7 \oplus DB_6 \oplus DB_2 \oplus DB_1 \oplus DB_0 \oplus 1,$$
$$CB_4 = DB_7 \oplus DB_6 \oplus DB_5 \oplus DB_4 \oplus DB_3 \oplus 1,$$

а разряды синдрома ошибки — по формулам:

$$CB_0 = DB_{M4} \oplus DB_{M3} \oplus DB_{M1} \oplus DB_{M0} \oplus CB_{M0},$$
$$CB_1 = DB_{M6} \oplus DB_{M5} \oplus DB_{M3} \oplus DB_{M2} \oplus DB_0 \oplus CB_{M1},$$
$$CB_2 = DB_{M7} \oplus DB_{M5} \oplus DB_{M4} \oplus DB_{M2} \oplus DB_1 \oplus CB_{M2},$$
$$CB_3 = DB_{M7} \oplus DB_{M6} \oplus DB_{M2} \oplus DB_{M1} \oplus DB_0 \oplus CB_{M3},$$
$$CB_4 = DB_{M7} \oplus DB_{M6} \oplus DB_{M5} \oplus DB_{M4} \oplus DB_3 \oplus CB_{M4}.$$

Таблица 1.33. **Синдром ошибки**

Ошибка в разряде	Код синдрома ошибки				
	CB_4	CB_3	CB_2	CB_1	CB_0
DB_0	1	0	1	0	0
DB_1	1	0	0	1	0
DB_2	1	0	0	0	1
DB_3	0	1	1	0	0
DB_4	0	1	0	1	0
DB_5	0	1	0	0	1
DB_6	0	0	1	0	1
DB_7	0	0	0	1	1
CB_0	1	1	1	1	0
CB_1	1	1	1	0	1
CB_2	1	1	0	1	1
CB_3	1	0	1	1	1
CB_4	0	1	1	1	1
—	1	1	1	1	1

Разряды синдрома ошибки можно представить в виде:

$$CB_0 = DB_{M4} \oplus DB_{M3} \oplus DB_{M1} \oplus DB_{M0} \oplus (DB_4 \oplus DB_3 \oplus DB_1 \oplus DB_0)_M \oplus 1,$$
$$CB_1 = DB_{M6} \oplus DB_{M5} \oplus DB_{M3} \oplus DB_{M2} \oplus DB_0 \oplus (DB_6 \oplus DB_5 \oplus DB_3 \oplus DB_2 \oplus DB_0)_M \oplus 1,$$
$$CB_2 = DB_{M7} \oplus DB_{M5} \oplus DB_{M4} \oplus DB_{M2} \oplus DB_1 \oplus (DB_7 \oplus DB_5 \oplus DB_4 \oplus DB_2 \oplus DB_1)_M \oplus 1,$$
$$CB_3 = DB_{M7} \oplus DB_{M6} \oplus DB_{M2} \oplus DB_{M1} \oplus DB_0 \oplus (DB_7 \oplus DB_6 \oplus DB_2 \oplus DB_1 \oplus DB_0)_M \oplus 1,$$
$$CB_4 = DB_{M7} \oplus DB_{M6} \oplus DB_{M5} \oplus DB_{M4} \oplus DB_3 \oplus (DB_7 \oplus DB_6 \oplus DB_5 \oplus DB_4 \oplus DB_3)_M \oplus 1.$$

Легко убедиться, что если при чтении памяти ошибки отсутствуют, т. е. все разряды $DB_{Mi} = DB_i$ и $CB_{Mj} = CB_j$, то все разряды синдрома ошибки $CB_j = 1$ (для этого из последних формул следует просто исключить индекс M). Свойства синдрома ошибки представлены в табл. 1.33.

Могут использоваться и инверсные значения некоторых разрядов CB_j, например:

$$CB_0 = DB_4 \oplus DB_3 \oplus DB_1 \oplus DB_0 \oplus (CB_{M0} \vee \overline{S}_1\overline{S}_0) \oplus 1,$$

$$CB_1 = DB_6 \oplus DB_5 \oplus DB_3 \oplus DB_2 \oplus DB_0 \oplus (CB_{M1} \vee \overline{S}_1\overline{S}_0) \oplus 1,$$

что не влияет на значения разрядов синдрома ошибки, хотя и будут записываться в память инверсные значения проверочных разрядов $\overline{CB}_0$ и $\overline{CB}_1$.

Устанавливаемые в ИС ‘636 и ‘637 значения флагов ошибок представлены в табл. 1.34 — при обнаружении двукратной ошибки (катастрофической) флаг *DEF* используется для подачи на МП сигнала запроса прерывания, вызывающего подпрограмму обработки катастрофического отказа ОЗУ. Однократные же ошибки исправляются без каких-либо последствий для МП-системы. В табл. 1.35 приведена связь между кодом синдрома ошибки и типом ошибки (DB_i — однократная ошибка, 2 — двукратная ошибка, *ML* — многократная ошибка).

На рис. 1.63, *а* изображена структурная схема МП-системы с 8-разрядной шиной данных, в которой используется ОЗУ с обнаружением двукратных и исправлением однократных ошибок. Узел *Control Logic* формирует сигналы управления S_0 и S_1 для *EDAC* ‘636 из системных сигналов управления чтением $\overline{MEMR}$ и записью $\overline{MEMW}$ данных *D*7–0 в ОЗУ (рис. 1.63, *б*).

При записи значением сигнала $\overline{MEMW} = 0$ байта данных DB_{7-0} в память значения разрядов DB_{7-0} сразу же появляются на выходах “прозрачного” регистра памяти RG_1 (см. рис. 1.62). Генератор паритета (*Parity Generator*) при записи данных в память ($\overline{S}_1 \cdot \overline{S}_0 = 1$) выполняет функцию генератора проверочного слова CB_{4-0}, которое проходит через открытый буфер (*Buffer* 1) и записывается значением сигнала $\overline{MEMW} = 0$ в дополнительное 5-разрядное ОЗУ (см. табл. 1.31).

Таблица 1.34. **Флаги ошибок ИС ‘636 и ‘637**

Число ошибок		Флаг ошибки		Примечание
DB_i	CB_j	*SEF*	*DEF*	
0	0	0	0	Ошибок нет
1	0	1	0	Коррекция данных
0	1	1	0	Коррекция данных
1	1	1	1	Запрос прерывания
2	0	1	1	Запрос прерывания
0	2	1	1	Запрос прерывания

Таблица 1.35. **Типы ошибок ИС ‘636 и ‘637**

CW		$CB_4\ CB_3\ CB_2$							
CB_1	CB_0	000	001	010	011	100	101	110	111
0	0	*ML*	2	2	DB_3	2	DB_0	2	2
0	1	2	DB_6	DB_5	2	DB_2	2	2	CB_1
1	0	2	2	DB_4	2	DB_1	2	2	CB_0
1	1	DB_7	2	2	CB_4	2	CB_3	CB_2	0

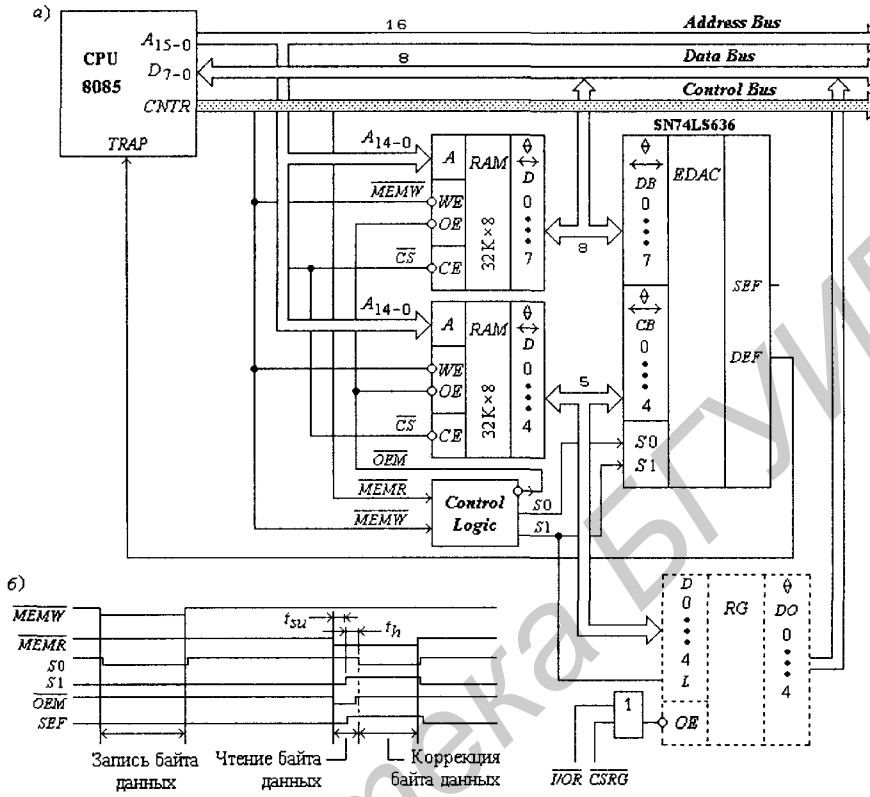


Рис. 1.63. Структурная схема МП-системы с *EDAC*

При чтении значением сигнала $\overline{MEMR} = 0$ байта данных из памяти в регистры памяти RG_1 и RG_2 поступают и фиксируются в них значения DB_{7-0} и CB_{4-0} соответственно. Эти значения поступают на входы генератора паритета (*Parity Generator*), выполняющего при чтении данных из памяти функцию генератора синдрома ошибки CB_{4-0} ($\overline{S}_1 \cdot \overline{S}_0 = 0$). Синдром ошибки используется для установки флагов ошибок *SEF* и *DEF* (узел *Error Detector*), а также для исправления байта данных DB_{7-0}, принимаемого МП, при однократной ошибке (узлы *Error Decoder*, *Error Corrector* и *Buffer* 2). При этом буфер *Buffer* 1 выдает на выходы CB_{4-0} значение синдрома ошибки для указания разрядов, вызвавших ошибку (см. табл. 1.31).

Для исследования синдрома ошибки его можно зафиксировать сигналом $L = S_1 = 1$ в регистре памяти *RG* (рис. 1.63, *а*) и затем командой IN *CSRG* ввести в аккумулятор МП (обычно этого делать не требуется).

Моделирование ИС '636. Для более детального ознакомлением с принципом построения *EDAC* было произведено моделирование *EDAC* '636 с помощью модифицированного программного пакета *Micro-Logic II* (этот пакет в упакованном виде был размещен на одной дискете и вместе с книгой [5] поступил в продажу в 1997 г.).

На рис. 1.64 изображена разработанная принципиальная схема *EDAC* '636, которая может несколько отличаться от фирменной схемы (неизвестной). Лабораторная работа для исследования *EDAC* с помощью пакета *Micro-Logic II* (файл *u_edac*.dwg) приведена на рис. 1.65 — ОЗУ моделируется двумя регистрами типа *SN*74*ALS*992 [5], а ошибки вводятся с помощью приемопередатчиков *TRN*2 (*Noise* — помехи).

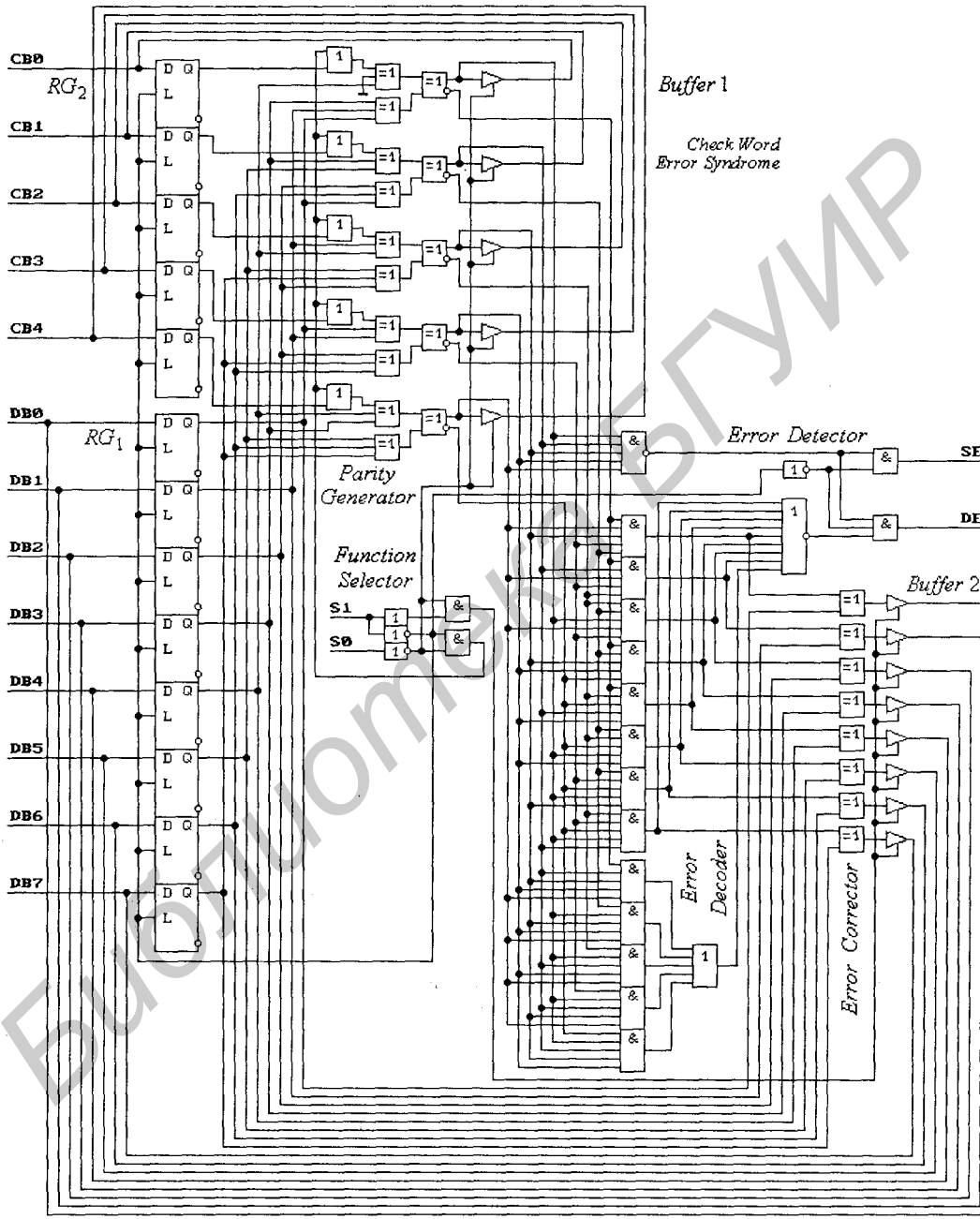


Рис. 1.64. Принципиальная схема *EDAC* '636

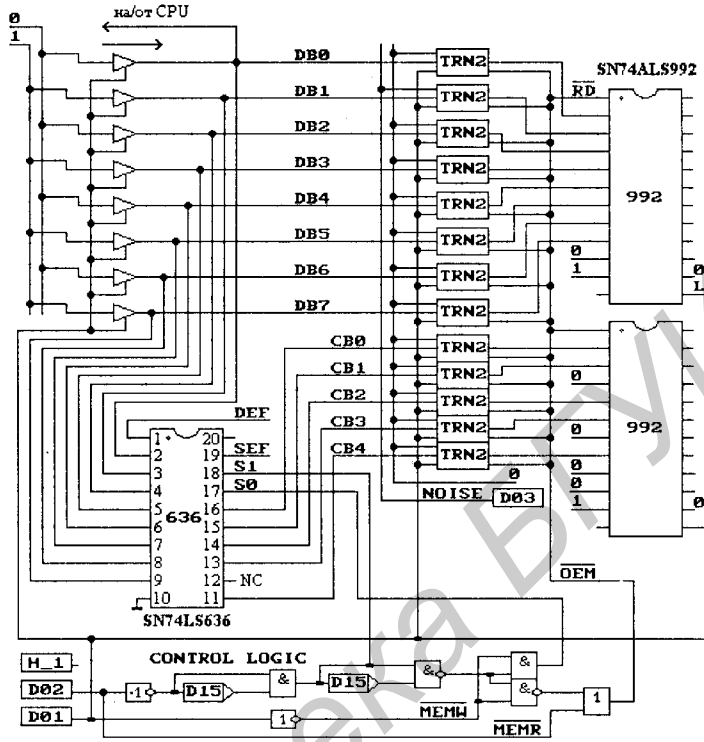


Рис. 1.65. Лабораторная работа из пакета *Micro-Logic II*

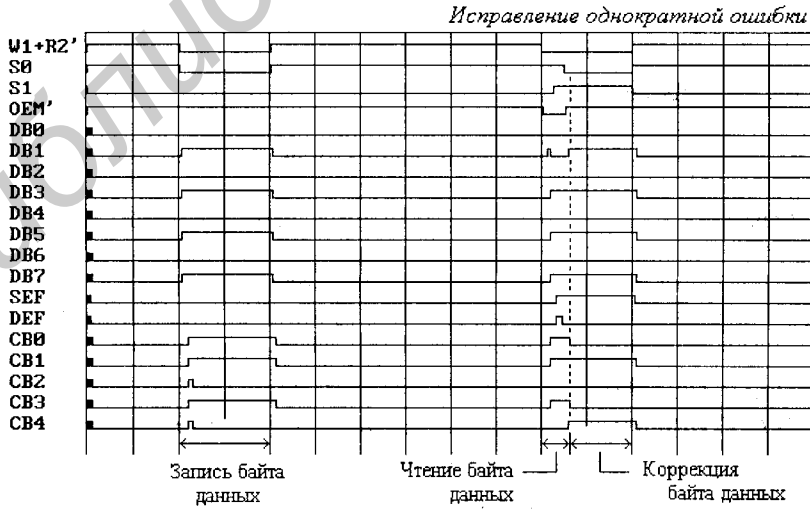


Рис. 1.66. Временны́е диаграммы работы *EDAC* для ошибок в одном и двух разрядах данных (см. также с. 136)

Рис. 1.66 (продолжение)

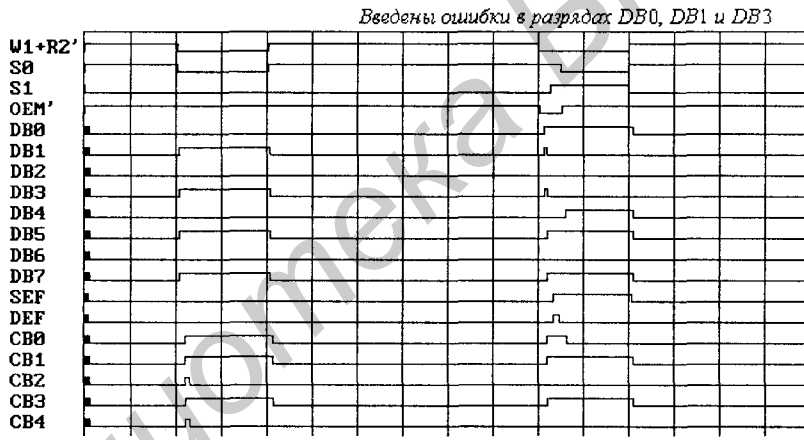


Рис. 1.67. Временны́е диаграммы работы *EDAC* (ошибки в 3 разрядах)

Результаты исследований при введении однократной и двукратной ошибок показаны на рис. 1.66, а при введении трехкратной ошибки — на рис. 1.67.

***EDAC* 74*LS*630 (555ВЖ1) и 74*LS*631.** Структурная схема этих *EDAC* такая же, что и показанная на рис. 1.62 — только разрядность шин данных и проверочного слова равны 16 и 6 бит соответственно. Для слова данных DB_i ($i = 0, 1, \ldots, 15$) разряды CB_j ($j = 0, 1, \ldots, 5$) проверочного слова и синдрома ошибки, используемого для обнаружения двукратных и исправления однократных ошибок в слове данных, вычисляются на основании соотношений:

$$CB_0 = DB_{13} \oplus DB_{10} \oplus DB_9 \oplus DB_8 \oplus DB_4 \oplus DB_3 \oplus DB_1 \oplus DB_0 \oplus (CB_{M0} \vee \overline{S}_1\overline{S}_0) \oplus 1,$$

$$CB_1 = DB_{14} \oplus DB_{11} \oplus DB_8 \oplus DB_6 \oplus DB_5 \oplus DB_3 \oplus DB_2 \oplus DB_0 \oplus (CB_{M1} \vee \overline{S}_1\overline{S}_0) \oplus 1,$$

$$CB_2 = DB_{15} \oplus DB_{12} \oplus DB_9 \oplus DB_7 \oplus DB_5 \oplus DB_4 \oplus DB_2 \oplus DB_1 \oplus (CB_{M2} \vee \overline{S}_1\overline{S}_0),$$

$$CB_3 = DB_{12} \oplus DB_{11} \oplus DB_{10} \oplus DB_7 \oplus DB_6 \oplus DB_2 \oplus DB_1 \oplus DB_0 \oplus (CB_{M3} \vee \overline{S}_1\overline{S}_0),$$

$$CB_4 = DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_7 \oplus DB_6 \oplus DB_5 \oplus DB_4 \oplus DB_3 \oplus (CB_{M4} \vee \overline{S}_1\overline{S}_0),$$

$$CB_5 = DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_{12} \oplus DB_{11} \oplus DB_{10} \oplus DB_9 \oplus DB_8 \oplus (CB_{M5} \vee \overline{S}_1\overline{S}_0),$$

где $\overline{S}_1\overline{S}_0$ — сигнал управления, задающий вычисление проверочного слова ($\overline{S}_1\overline{S}_0 = 1$, DB_i — записываемые в память разряды слова данных) и синдрома ошибки ($\overline{S}_1\overline{S}_0 = 0$, $DB_i = DB_{Mi}$ и CB_{Mj} — разряды слова данных и проверочного слова, прочитанных из памяти), а индекс М означает, что разряд может быть прочитан из памяти с ошибкой (0 вместо 1 или 1 вместо 0).

Таблица 1.36. **Вычисление проверочных разрядов в ИС ‘630 и ‘631**

Разряды проверочного слова	Разряды слова данных															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB_0	–	–	+	–	–	+	+	+	–	–	–	+	+	–	+	+
CB_1	–	+	–	–	+	–	–	+	–	+	+	–	+	+	–	+
CB_2	+	–	–	+	–	–	+	–	+	–	+	+	–	+	+	–
CB_3	–	–	–	+	+	+	–	–	+	+	–	–	–	+	+	+
CB_4	+	+	+	–	–	–	–	–	+	+	+	+	+	–	–	–
CB_5	+	+	+	+	+	+	+	+	–	–	–	–	–	–	–	–

Таким образом, проверочные разряды вычисляются по формулам (табл. 1.36):

$$CB_0 = DB_{13} \oplus DB_{10} \oplus DB_9 \oplus DB_8 \oplus DB_4 \oplus DB_3 \oplus DB_1 \oplus DB_0,$$
$$CB_1 = DB_{14} \oplus DB_{11} \oplus DB_8 \oplus DB_6 \oplus DB_5 \oplus DB_3 \oplus DB_2 \oplus DB_0,$$
$$CB_2 = DB_{15} \oplus DB_{12} \oplus DB_9 \oplus DB_7 \oplus DB_5 \oplus DB_4 \oplus DB_2 \oplus DB_1 \oplus 1,$$
$$CB_3 = DB_{12} \oplus DB_{11} \oplus DB_{10} \oplus DB_7 \oplus DB_6 \oplus DB_2 \oplus DB_1 \oplus DB_0 \oplus 1,$$
$$CB_4 = DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_7 \oplus DB_6 \oplus DB_5 \oplus DB_4 \oplus DB_3 \oplus 1,$$
$$CB_5 = DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_{12} \oplus DB_{11} \oplus DB_{10} \oplus DB_9 \oplus DB_8 \oplus 1,$$

а разряды синдрома ошибки — по формулам:

$$CB_0 = DB_{M13} \oplus DB_{M10} \oplus DB_{M9} \oplus DB_{M8} \oplus DB_{M4} \oplus DB_{M3} \oplus DB_{M1} \oplus DB_{M0} \oplus CB_{M0} \oplus 1,$$
$$CB_1 = DB_{M14} \oplus DB_{M11} \oplus DB_{M8} \oplus DB_{M6} \oplus DB_{M5} \oplus DB_{M3} \oplus DB_{M2} \oplus DB_{M0} \oplus CB_{M1} \oplus 1,$$
$$CB_2 = DB_{M15} \oplus DB_{M12} \oplus DB_{M9} \oplus DB_{M7} \oplus DB_{M5} \oplus DB_{M4} \oplus DB_{M2} \oplus DB_{M1} \oplus CB_{M2},$$
$$CB_3 = DB_{M12} \oplus DB_{M11} \oplus DB_{M10} \oplus DB_{M7} \oplus DB_{M6} \oplus DB_{M2} \oplus DB_{M1} \oplus DB_{M0} \oplus CB_{M3},$$
$$CB_4 = DB_{M15} \oplus DB_{M14} \oplus DB_{M13} \oplus DB_{M7} \oplus DB_{M6} \oplus DB_{M5} \oplus DB_{M4} \oplus DB_{M3} \oplus CB_{M4},$$
$$CB_5 = DB_{M15} \oplus DB_{M14} \oplus DB_{M13} \oplus DB_{M12} \oplus DB_{M11} \oplus DB_{M10} \oplus DB_{M9} \oplus DB_{M8} \oplus CB_{M5}.$$

Разряды синдрома ошибки можно представить в виде:

$$CB_0 = DB_{M13} \oplus DB_{M10} \oplus DB_{M9} \oplus DB_{M8} \oplus DB_{M4} \oplus DB_{M3} \oplus DB_{M1} \oplus DB_{M0} \oplus CB_{M0} \oplus$$
$$\oplus (DB_{13} \oplus DB_{10} \oplus DB_9 \oplus DB_8 \oplus DB_4 \oplus DB_3 \oplus DB_1 \oplus DB_0)_M \oplus 1,$$
$$CB_1 = DB_{M14} \oplus DB_{M11} \oplus DB_{M8} \oplus DB_{M6} \oplus DB_{M5} \oplus DB_{M3} \oplus DB_{M2} \oplus DB_{M0} \oplus CB_{M1} \oplus$$
$$\oplus (DB_{14} \oplus DB_{11} \oplus DB_8 \oplus DB_6 \oplus DB_5 \oplus DB_3 \oplus DB_2 \oplus DB_0)_M \oplus 1,$$
$$CB_2 = DB_{M15} \oplus DB_{M12} \oplus DB_{M9} \oplus DB_{M7} \oplus DB_{M5} \oplus DB_{M4} \oplus DB_{M2} \oplus DB_{M1} \oplus CB_{M2} \oplus$$
$$\oplus (DB_{15} \oplus DB_{12} \oplus DB_9 \oplus DB_7 \oplus DB_5 \oplus DB_4 \oplus DB_2 \oplus DB_1)_M \oplus 1,$$
$$CB_3 = DB_{M12} \oplus DB_{M11} \oplus DB_{M10} \oplus DB_{M7} \oplus DB_{M6} \oplus DB_{M2} \oplus DB_{M1} \oplus DB_{M0} \oplus CB_{M3} \oplus$$
$$\oplus (DB_{12} \oplus DB_{11} \oplus DB_{10} \oplus DB_7 \oplus DB_6 \oplus DB_2 \oplus DB_1 \oplus DB_0)_M \oplus 1,$$

Таблица 1.37. **Синдром ошибки (ИС 74*LS*630 и 74*LS*631)**

Ошибка в разряде	Код синдрома ошибки						Ошибка в разряде	Код синдрома ошибки					
	CB_5	CB_4	CB_3	CB_2	CB_1	CB_0		CB_5	CB_4	CB_3	CB_2	CB_1	CB_0
DB_0	1	1	0	1	0	0	DB_{12}	0	1	0	0	1	1
DB_1	1	1	0	0	1	0	DB_{13}	0	0	1	1	1	0
DB_2	1	1	0	0	0	1	DB_{14}	0	0	1	1	0	1
DB_3	1	0	1	1	0	0	DB_{15}	0	0	1	0	1	1
DB_4	1	0	1	0	1	0							
DB_5	1	0	1	0	0	1	CB_0	1	1	1	1	1	0
DB_6	1	0	0	1	0	1	CB_1	1	1	1	1	0	1
DB_7	1	0	0	0	1	1	CB_2	1	1	1	0	1	1
DB_8	0	1	1	1	0	0	CB_3	1	1	0	1	1	1
DB_9	0	1	1	0	1	0	CB_4	1	0	1	1	1	1
DB_{10}	0	1	0	1	1	0	CB_5	0	1	1	1	1	1
DB_{11}	0	1	0	1	0	1	—	1	1	1	1	1	1

Таблица 1.38. **Типы ошибок (ИС 74*LS*630 и 74*LS*631)**

CW			$CB_5\ CB_4\ CB_3$							
CB_2	CB_1	CB_0	000	001	010	011	100	101	110	111
0	0	0	2	*ML*	*ML*	2	*ML*	2	2	*ML*
0	0	1	*ML*	2	2	*ML*	2	DB_5	DB_2	2
0	1	0	*ML*	2	2	DB_9	2	DB_4	DB_1	2
0	1	1	2	DB_{15}	DB_{12}	2	DB_7	2	2	CB_2
1	0	0	*ML*	2	2	DB_8	2	DB_3	DB_0	2
1	0	1	2	DB_{14}	DB_{11}	2	DB_6	2	2	CB_1
1	1	0	2	DB_{13}	DB_{10}	2	*ML*	2	2	CB_0
1	1	1	*ML*	2	2	CB_5	2	CB_4	CB_3	0

$$CB_4 = DB_{M15} \oplus DB_{M14} \oplus DB_{M13} \oplus DB_{M7} \oplus DB_{M6} \oplus DB_{M5} \oplus DB_{M4} \oplus DB_{M3} \oplus CB_{M4} \oplus$$
$$\oplus (DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_7 \oplus DB_6 \oplus DB_5 \oplus DB_4 \oplus DB_3)_M \oplus 1,$$

$$CB_5 = DB_{M15} \oplus DB_{M14} \oplus DB_{M13} \oplus DB_{M12} \oplus DB_{M11} \oplus DB_{M10} \oplus DB_{M9} \oplus DB_{M8} \oplus CB_{M5} \oplus$$
$$\oplus (DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_{12} \oplus DB_{11} \oplus DB_{10} \oplus DB_9 \oplus DB_8)_M \oplus 1.$$

Свойства синдрома ошибки представлены в табл. 1.37. Устанавливаемые в ИС ‘630 и ‘631 значения флагов ошибок такие же, что и для ИС ‘636 и ‘637 (табл. 1.34). В табл. 1.38 приведена связь между кодом синдрома ошибки и типом ошибки (DB_i — однократная ошибка, 2 — двукратная ошибка, *ML* — многократная ошибка). Однократные ошибки исправляются без каких-либо последствий.

***EDAC* 74*LS*616 и 74*LS*617.** Структурная схема *EDAC* ‘616 изображена на рис. 1.68 (*EDAC* ‘617 описывается такой же структурной схемой, но буферы *Buffer* 1, 2 и 3 имеют открытые коллекторные выходы):

DB_{15-0} (*Data Bite*) — разряды шины данных МП;

CB_{5-0} (*Check Bite*) — разряды проверочного слова;

$\overline{S_1}$, $\overline{S_0}$ — сигналы управления внутренними операциями *EDAC* (табл. 1.39);

$\overline{ERR}$ (*Single-bit Errors Flag*) — флаг обнаружения и исправления однократных ошибок;

$\overline{MERR}$ (*Dual-bit Errors Flag*) — флаг обнаружения двукратных ошибок (*Multi-Error*);

$\overline{OECB}$ (*Output Enable Check Bite*) — сигнал разрешения выхода проверочного слова;

$\overline{LEDBO}$ (*Data Bite Output Latch Enable*) — сигнал разрешения фиксации в регистре исправленного слова;

$\overline{OEB}_0$ — (*Output Enable Byte* 0) — сигнал разрешения выхода младшего байта данных;

$\overline{OEB}_1$ — (*Output Enable Byte* 1) — сигнал разрешения выхода старшего байта данных.

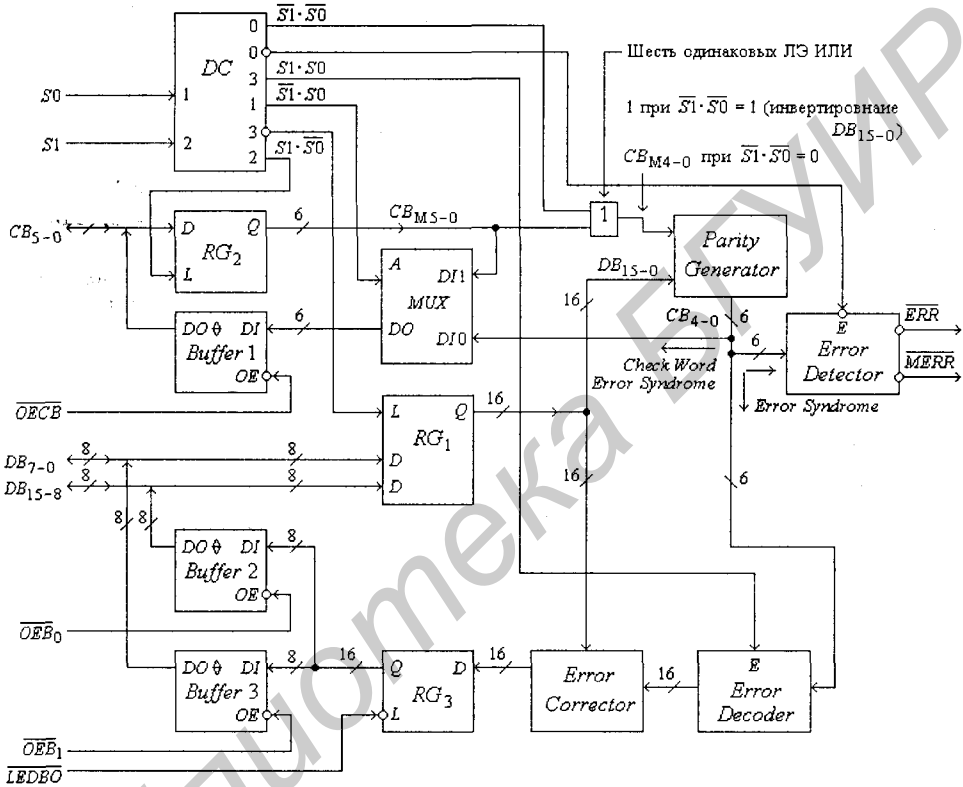


Рис. 1.68. Структурная схема *EDAC* ‘616

Таблица 1.39. **Управление работой ИС ‘616 и ‘617**

Цикл *RAM*	S_1S_0	Функция *EDAC*	DB_i	$\overline{OEB}_0$	$\overline{OEB}_1$	$\overline{LEDBO}$	CB_j	$\overline{OECB}$	$\overline{ERR}$	$\overline{MERR}$
Запись	0 0	Генерация CB_{5-0}	Вход	1	1	×	Выход	0	1	1
Чтение	1 0	Чтение DB_{15-0} и CB_{5-0} из памяти и установка флагов ошибок	Вход	1	1	×	Вход	1	1/0	1/0
Чтение	1 1	Фиксация в регистрах DB_{15-0} и CB_{5-0}	Фиксация DB_i	1	1	0	Фиксация CB_j	1	1/0	1/0
Чтение	1 1	Коррекция DB_{15-0} и генерация синдрома ошибки CB_{5-0}	Выход	0	0	×	Выход синдрома	0	1/0	1/0

Для слова данных DB_i ($i = 0 \ldots 15$) разряды CB_j ($j = 0 \ldots 5$) проверочного слова и синдрома ошибки, используемого для обнаружения двукратных и исправления однократных ошибок в слове данных, вычисляются на основании соотношений:

$$CB_0 = DB_{13} \oplus DB_{11} \oplus DB_{10} \oplus DB_9 \oplus DB_6 \oplus DB_3 \oplus DB_1 \oplus DB_0 \oplus (CB_{M0} \vee \overline{S}_1\overline{S}_0) \oplus 1,$$
$$CB_1 = DB_{14} \oplus DB_{12} \oplus DB_{10} \oplus DB_9 \oplus DB_8 \oplus DB_4 \oplus DB_2 \oplus DB_0 \oplus (CB_{M1} \vee \overline{S}_1\overline{S}_0) \oplus 1,$$
$$CB_2 = DB_{15} \oplus DB_{12} \oplus DB_{11} \oplus DB_8 \oplus DB_7 \oplus DB_5 \oplus DB_2 \oplus DB_1 \oplus (CB_{M2} \vee \overline{S}_1\overline{S}_0),$$
$$CB_3 = DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_9 \oplus DB_8 \oplus DB_5 \oplus DB_4 \oplus DB_3 \oplus (CB_{M3} \vee \overline{S}_1\overline{S}_0),$$
$$CB_4 = DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_{12} \oplus DB_{11} \oplus DB_{10} \oplus DB_7 \oplus DB_6 \oplus (CB_{M4} \vee \overline{S}_1\overline{S}_0),$$
$$CB_5 = DB_7 \oplus DB_6 \oplus DB_5 \oplus DB_4 \oplus DB_3 \oplus DB_2 \oplus DB_1 \oplus DB_0 \oplus (CB_{M5} \vee \overline{S}_1\overline{S}_0),$$

где $\overline{S}_1\overline{S}_0$ — сигнал управления, задающий вычисление проверочного слова ($\overline{S}_1\overline{S}_0 = 1$, DB_i — записываемые в память разряды слова данных) и синдрома ошибки ($\overline{S}_1\overline{S}_0 = 0$, $DB_i = DB_{Mi}$ и CB_{Mj} — разряды слова данных и проверочного слова, прочитанных из памяти), а индекс M означает, что разряд может быть прочитан из памяти с ошибкой (0 вместо 1 или 1 вместо 0).

Таким образом, проверочные разряды вычисляются по формулам (табл. 1.40):

$$CB_0 = DB_{13} \oplus DB_{11} \oplus DB_{10} \oplus DB_9 \oplus DB_6 \oplus DB_3 \oplus DB_1 \oplus DB_0,$$
$$CB_1 = DB_{14} \oplus DB_{12} \oplus DB_{10} \oplus DB_9 \oplus DB_8 \oplus DB_4 \oplus DB_2 \oplus DB_0,$$
$$CB_2 = DB_{15} \oplus DB_{12} \oplus DB_{11} \oplus DB_8 \oplus DB_7 \oplus DB_5 \oplus DB_2 \oplus DB_1 \oplus 1,$$
$$CB_3 = DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_9 \oplus DB_8 \oplus DB_5 \oplus DB_4 \oplus DB_3 \oplus 1,$$
$$CB_4 = DB_{15} \oplus DB_{14} \oplus DB_{13} \oplus DB_{12} \oplus DB_{11} \oplus DB_{10} \oplus DB_7 \oplus DB_6 \oplus 1,$$
$$CB_5 = DB_7 \oplus DB_6 \oplus DB_5 \oplus DB_4 \oplus DB_3 \oplus DB_2 \oplus DB_1 \oplus DB_0 \oplus 1,$$

а разряды синдрома ошибки — по формулам:

$$CB_0 = DB_{M13} \oplus DB_{M11} \oplus DB_{M10} \oplus DB_{M9} \oplus DB_{M6} \oplus DB_{M3} \oplus DB_{M1} \oplus DB_{M0} \oplus CB_{M0} \oplus 1,$$
$$CB_1 = DB_{M14} \oplus DB_{M12} \oplus DB_{M10} \oplus DB_{M9} \oplus DB_{M8} \oplus DB_{M4} \oplus DB_{M2} \oplus DB_{M0} \oplus CB_{M1} \oplus 1,$$
$$CB_2 = DB_{M15} \oplus DB_{M12} \oplus DB_{M11} \oplus DB_{M8} \oplus DB_{M7} \oplus DB_{M5} \oplus DB_{M2} \oplus DB_{M1} \oplus CB_{M2},$$
$$CB_3 = DB_{M15} \oplus DB_{M14} \oplus DB_{M13} \oplus DB_{M9} \oplus DB_{M8} \oplus DB_{M5} \oplus DB_{M4} \oplus DB_{M3} \oplus CB_{M3},$$
$$CB_4 = DB_{M15} \oplus DB_{M14} \oplus DB_{M13} \oplus DB_{M12} \oplus DB_{M11} \oplus DB_{M10} \oplus DB_{M7} \oplus DB_{M6} \oplus CB_{M4},$$
$$CB_5 = DB_{M7} \oplus DB_{M6} \oplus DB_{M5} \oplus DB_4 \oplus DB_{M3} \oplus DB_{M2} \oplus DB_{M1} \oplus DB_{M0} \oplus CB_{M5}.$$

Таблица 1.40. **Вычисление проверочных разрядов в ИС ‘616 и ‘617**

Разряды проверочного слова	Разряды слова данных															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB_0	–	–	+	–	+	+	+	–	–	+	–	–	+	–	+	+
CB_1	–	+	–	+	–	+	+	+	–	–	–	+	–	+	–	+
CB_2	+	–	–	+	+	–	–	+	+	–	+	–	–	+	+	–
CB_3	+	+	+	–	–	–	+	+	–	–	+	+	+	–	–	–
CB_4	+	+	+	+	+	+	–	–	+	+	–	–	–	–	–	–
CB_5	–	–	–	–	–	–	–	–	+	+	+	+	+	+	+	+

Устанавливаемые значения флагов ошибок представлены в табл. 1.41. Свойства синдрома ошибки приведены в табл. 1.42, а связь между кодом синдрома ошибки и типом ошибки (DB_i — однократная ошибка, 2 — двукратная ошибка, *ML* — многократная ошибка) — в табл. 1.43. Однократные ошибки исправляются без каких-либо последствий.

Таблица 1.41. **Флаги ошибок ИС ‘616 и ‘617**

Число ошибок		Флаг ошибки		Примечание
DB_i	CB_j	$\overline{ERR}$	$\overline{MERR}$	
0	0	1	1	Ошибок нет
1	0	0	1	Коррекция данных
0	1	0	1	Коррекция данных
1	1	0	0	Запрос прерывания
2	0	0	0	Запрос прерывания
0	2	0	0	Запрос прерывания

Таблица 1.42. **Синдром ошибки ИС ‘616 и ‘617**

Ошибка в разряде	Код синдрома ошибки						Ошибка в разряде	Код синдрома ошибки					
	CB_5	CB_4	CB_3	CB_2	CB_1	CB_0		CB_5	CB_4	CB_3	CB_2	CB_1	CB_0
DB_0	0	1	1	1	0	0	DB_{12}	1	0	1	0	0	1
DB_1	0	1	1	0	1	0	DB_{13}	1	0	0	1	1	0
DB_2	0	1	1	0	0	1	DB_{14}	1	0	0	1	0	1
DB_3	0	1	0	1	1	0	DB_{15}	1	0	0	0	1	1
DB_4	0	1	0	1	0	1							
DB_5	0	1	0	0	1	1	CB_0	1	1	1	1	1	0
DB_6	0	0	1	1	1	0	CB_1	1	1	1	1	0	1
DB_7	0	0	1	0	1	1	CB_2	1	1	1	0	1	1
DB_8	1	1	0	0	0	1	CB_3	1	1	0	1	1	1
DB_9	1	1	0	1	0	0	CB_4	1	0	1	1	1	1
DB_{10}	1	0	1	1	0	0	CB_5	0	1	1	1	1	1
DB_{11}	1	0	1	0	1	0	—	1	1	1	1	1	1

Таблица 1.43. **Типы ошибок ИС ‘616 и ‘617**

CW			$CB_5\ CB_4\ CB_3$							
CB_2	CB_1	CB_0	000	001	010	011	100	101	110	111
0	0	0	2	*ML*	*ML*	2	*ML*	2	2	*ML*
0	0	1	*ML*	2	2	DB_2	2	DB_{12}	DB_8	2
0	1	0	*ML*	2	2	DB_1	2	DB_{11}	*ML*	2
0	1	1	2	DB_7	DB_5	2	DB_{15}	2	2	CB_2
1	0	0	*ML*	2	2	DB_0	2	DB_{10}	DB_9	2
1	0	1	2	*ML*	DB_4	2	DB_{14}	2	2	CB_1
1	1	0	2	DB_6	DB_3	2	DB_{13}	2	2	CB_0
1	1	1	*ML*	2	2	CB_5	2	CB_4	CB_3	0

Таблица 1.44. **Управление побайтной работой ИС ‘616 и ‘617**

Цикл *RAM*	S_1S_0	Функция *EDAC*	$Byte_n$	$\overline{OEB_n}$	$\overline{LEDBO}$	CB_j	$\overline{OECB}$	$\overline{ERR}$	$\overline{MERR}$
Чтение	1 0	Чтение DB_{15-0} и CB_{5-0} из памяти и установка флагов ошибок	Вход	1	×	Вход	1	1/0	1/0
Чтение	1 1	Фиксация в регистрах DB_{15-0} и CB_{5-0}	Фиксация входных данных	1	0	Фиксация CB_j	1	1/0	1/0
Чтение	1 1	Коррекция DB_{15-0} и генерация синдрома ошибки CB_{5-0}	Фиксация выходного слова	1	1	Выход синдрома	0	1/0	1/0
Запись	0 0	Модификация одного или двух байт и генерация нового *CW*	Вход Выход	1 0	1	Выход *CW*	0	1	1

Если МП записывает в память только один байт слова, то сначала нужно прочитать недостающий байт из памяти, затем вычислить проверочное слово *CW* и записать его в память вместе с байтом, поступившим из МП (табл. 1.44).

Обнаружение и исправление ошибок в ОЗУ требует дополнительных аппаратных затрат и может уменьшить (незначительно) быстродействие МП-системы. В МП-системах, предназначенных для решения особо важных задач, связанных с управлением объектами военного назначения, необходимо идти на такие затраты для обеспечения повышенной надежности их работы. Наименьших аппаратных затрат требует контроль паритета (обнаружение однократных ошибок) в связи с чем серийно выпускаются 9-разрядные статические (см. § 1.10) и динамические оперативные запоминающие устройства, а также 9-разрядные приемопередатчики (см. § 1.12) и 9-разрядная память типа *FIFO* (см. § 2.6).

1.12. Шинные драйверы, приемопередатчики и регистры памяти

Логические элементы, шинные драйверы, приемопередатчики, триггеры и регистры памяти, используемые при проектировании МП-систем, подробно описаны в книге [5]. В настоящее время фирмой *Texas Instruments* разработано много новых высококачественных технологий производства ИС, по которым изготовляются (повторяются) как старые функциональные устройства *SN74TTT*×××(*TTT* — аббревиатура технологии), имеющие универсальное назначение, так и новые, большинство которых построено на основе функциональных устройств *SN74TTT*×××. Так, спроектировано два новых семейства ИС:

SN74TTT1G××× (1*G* — ИС, содержащие один ЛЭ, шинный драйвер или триггер в корпусе), позволяющие уменьшить размер и вес проектируемых устройств (печатных плат);

*SN74TTT*16××× (технологии *TTT Widebus*™), где число 16 означает ИС с удвоенной разрядностью шины данных (8-, 9- и 10-разрядные ИС *SN74TTT*×××, предназначенные для использования в 16-, 32- и 64-разрядных МП-системах).

Номера ××× этих ИС равны номерам функциональных аналогов ИС *SN74TTT*×××.

Семейство ИС *SN74TTT1G*×××. Эти ИС приведены в табл. 1.45 — все они выполняются в миниатюрных корпусах, имеющих 5 выводов (рис. 1.69; *A*, *B*, D — информационные входы; $\overline{OE}$, *OE*, *CLK* — управляющие входы; *Y*, *Q* — выходы).

Таблица 1.45. **Интегральные схемы типа *SN74TTT1Gxxx***

Тип ИС xxx	Описание	Технология						Выводы корпуса
		AHC	*AHCT*	*LVC*	*ALVC*	*CBT*	*CBTLV*	
00	*2NAND*	#	#	+	–	–	–	P1
02	*2NOR*	#	#	+	–	–	–	P1
04	*NOT*	#	#	+	+	–	–	P2
*U*04	*Unbuffered NOT*	#	–	–	–	–	–	P2
05	*NOT* (*ODO*)	–	–	+	–	–	–	P2
08	*2AND*	#	#	+	+	–	–	P1
14	*ST–NOT* ($Y = \overline{Q}$)	#	#	+	+	–	–	P2
32	*2OR*	#	#	+	+	–	–	P1
79	*D*-триггер ($CLK = \overline{H}$, Q)	–	–	+	+	–	–	P5
86	*XOR*	#	#	+	–	–	–	P1
125	*BD* ($\overline{OE}$, *TS*)	#	#	+	+	#	+	P3
126	*BD* (*OE*, *TS*)	#	#	+	+	–	–	P4
132	*2AND–ST–NOT* ($Y = \overline{Q}$)	#	#	+	–	–	–	P1

Примечание: # — выпускается, + — намечено к выпуску (1998 г.); *TS — Three-state Output*, *ODO — Open-Drain Output*, *ST* — триггер Шмитта, *BD — Bus Driver*.

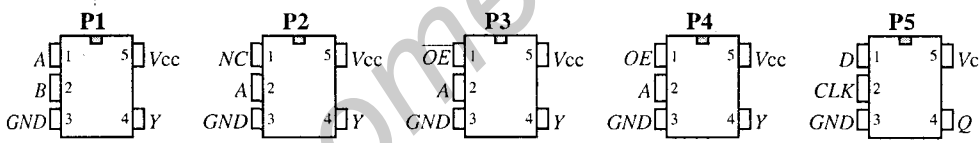


Рис. 1.69. Интегральные схемы типа *SN74TTT1Gxxx*

Размеры корпусов показаны на рис. 1.70 (*SOT — Small-Outline Transistor Package*). Чрезвычайно малые размеры ИС, названных логикой *Microgate* и *PicoGate*, упрощают их размещение на печатной плате (*Printed Circuit Board — PCB*) и предоставляют удобства при использовании их для изменения функциональных возможностей специализированных ИС — *ASIC* (*Application Specific Integrated Circuits* — интегральные схемы прикладной ориентации). Логика *Microgate* и *PicoGate* позволяет проектировщику существенно уменьшить длину соединительных проводников на печатной плате и тем самым снизить в системе *EMI*-помехи (*Electromagnetic Interference*). Возможность при ничтожно малых затратах изменять функции выходов *ASIC* позволяет использовать их для новых приложений, а также исправлять малые дефекты в проектах *ASIC*. Параметры ИС типа *SN74TTT1Gxxx* приведены в табл. 1.46.

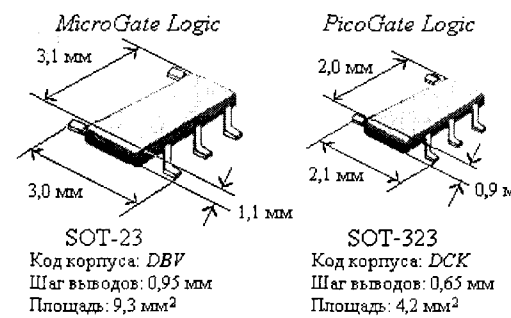


Рис. 1.70. Корпусы ИС *SN74TTT1Gxxx*

Таблица 1.46. **Параметры ИС типа *SN74TTT1Gxxx***

Технология	V_{CC}, В	t_{pd}, нс (*max*)	I_{OL}/V_{CC}, мА/В (*max*)	I_{OH}/V_{CC}, мА/В (*max*)	$V_{OL}/V_{CC}/I_{OL}$, В/В/мА (*max*)	V_{OH}/V_{CC}, В/В (*min*)	I_{CC}, мкА
AHC	5 3,3	5,5 7,9	8/5±0,5 В 4/3,3±0,3 В	–8/5±0,5 В –4/3,3±0,3 В	0,36/4,5/8 0,36/3/4	3,94/4,5/–8 2,58/3/–4	— 10
AHCT	5,0	6,5	8/5	–8/5	0,36/4,5/8	3,94/4,5/–8	10
LVC	3,3 2,5	— 4,3	24/3; 12/2,7 8/2,3	–24/3; –12/2,7 –8/2,3	0,55/3/24 0,7/2,3/8	2,2/3/–24 1,7/2,3/–8	— 10
ALVC	3,3 2,5	— 3,0	24/3 12/2,3	–24/3 –12/2,3	0,55/3/24 0,7/2,3/12	2,0/3/–24 1,7/2,3/–12	— 10
CBT	5,0	0,25	—	—	—	—	1,0
CBTVL	2,5/3,3	0,25	—	—	—	—	1,0

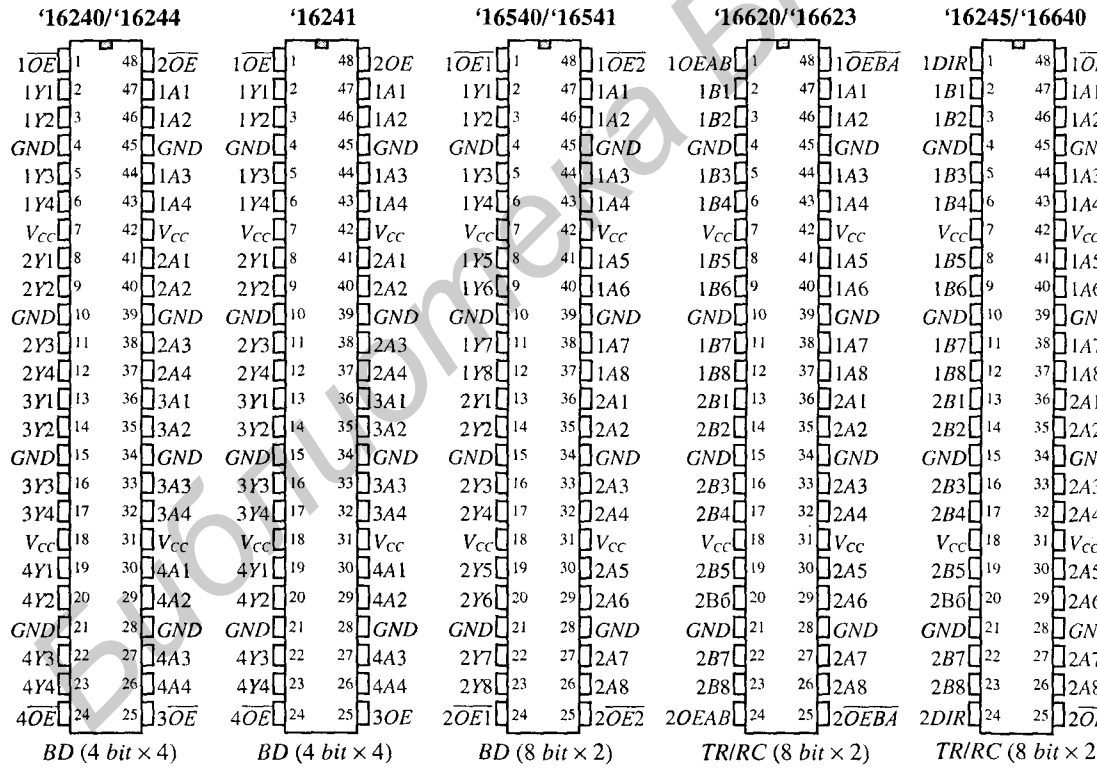


Рис. 1.71. Шинные драйверы и приемопередатчики *Widebus*™ фирмы *Texas Instruments*

Шинные драйверы и приемопередатчики *Widebus*™ фирмы *Texas Instruments*. Шинные драйверы (*Buffers/Drivers*) и приемопередатчики (*Bus Transceiver*) *Widebus*™ изображены на рис. 1.71, а в табл. 1.47 и 1.48 дано описание их работы. На рис. 1.72 приведен пример приемопередатчика *Widebus+*™.

Таблица 1.47. **Описание работы шинных драйверов**

'16240/'16540*			**'16241**						**'16244/'16541***		
$\overline{OE}$	A	Выход Y	$\overline{OE}$	A	Выход Y	OE	A	Выход Y	$\overline{OE}$	A	Выход Y
0	0	1	0	0	0	1	0	0	0	0	0
0	1	0	0	1	1	1	1	1	0	1	1
1	×	Z	1	×	Z	0	×	Z	1	×	Z

Примечание: * для ИС'16540/'16541 сигнал $OE = OE_1 \& OE_2$.

Таблица 1.48. **Описание работы приемопередатчиков**

'16245			**'16640**			**'16620**			**'16623**		
$\overline{OE}$	DIR	Операция	$\overline{OE}$	DIR	Операция	$\overline{OE}\overline{BA}$	$OEAB$	Операция	$\overline{OE}\overline{BA}$	$OEAB$	Операция
0	0	$B \to A$	0	0	$\overline{B} \to A$	0	0	$\overline{B} \to A$	0	0	$B \to A$
0	1	$A \to B$	0	1	$\overline{A} \to B$	1	1	$\overline{A} \to B$	1	1	$A \to B$
1	×	*Isolation*	1	×	*Isolation*	1	0	*Isolation*	1	0	*Isolation*
						0	1	$\overline{B} \to A$, $\overline{A} \to B$	0	1	$B \to A$, $A \to B$

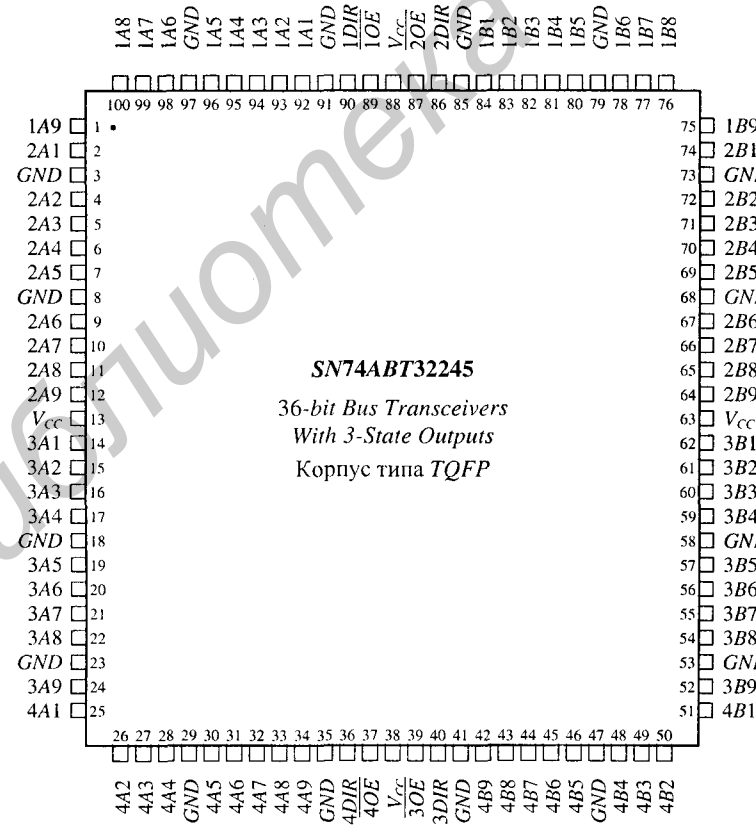


Рис. 1.72. Четыре 9-разрядных приемопередатчика (*Widebus+*™)

Таблица 1.49. **Интегральные схемы типа *SN74TTT16xxx Widebus™***

Тип ИС xxx	Технология								
	AHC	*AHCT*	*AC*	*ACT*	*ABT*	*ALB*	*ALVC*	*LVT*	*LVC*
240	+	+	+	+	+	–	+	+	+
241	–	–	–	+	+	–	–	+	–
244	+	+	+	+	+	+	+	+	+
245	+	+	+	+	+	+	+	+	+
373	+	+	+	+	+	–	+	+	+
374	+	+	+	+	+	–	+	+	+
540	+	+	–	+	+	–	–	–	+
541	+	+	–	+	+	–	–	+	+
623	–	–	+	+	+	–	–	–	–
640	–	–	+	+	+	–	–	–	–
821	–	–	–	+	+	–	+	–	–
823	–	–	+	+	+	–	+	–	–
841	–	–	–	+	+	–	+	–	–

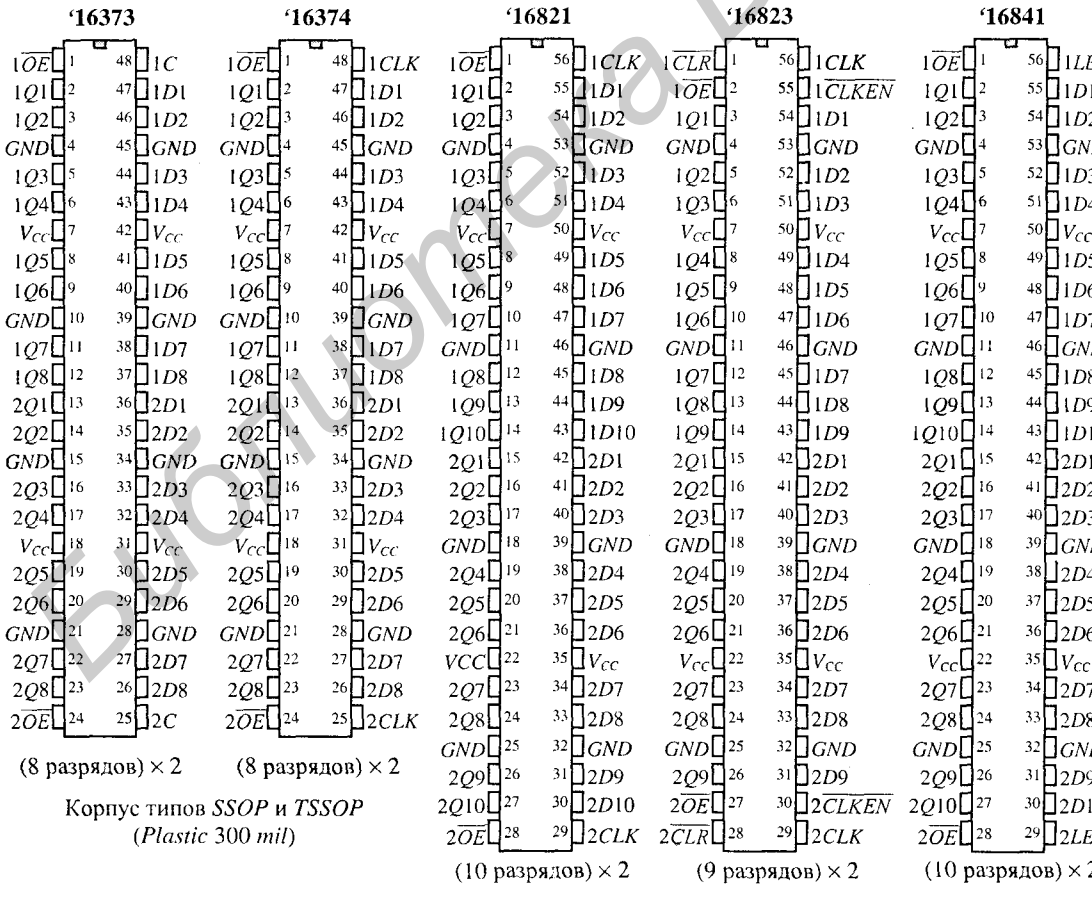


Рис. 1.73. Регистры памяти *Widebus*™ фирмы *Texas Instruments*

В табл. 1.49 перечислены технологии фирмы *Texas Instruments*, по которым изготовляются ИС типа *SN74TTT*16××× *Widebus*™. Номера ××× этих ИС совпадают с соответствующими номерами 8-разрядных шинных драйверов, приемопередатчиков и регистров памяти [5].

Регистры памяти *Widebus*™ фирмы *Texas Instruments*. На рис. 1.73 приведены регистры памяти *Widebus*™, наиболее часто используемые при проектировании МП-систем.

Типы корпусов ИС. К настоящему времени разработано много типов корпусов, различающихся габаритными размерами, шагом контактов (выводов), материалом корпуса и другими параметрами (длина корпуса определяется числом выводов). Размеры и коды некоторых типов корпусов приведены в табл. 1.50. Для описания линейных размеров корпусов и их выводов принято использовать единицы длины: *mm* (миллиметр), *inch* (дюйм) и *mil* (мил) — одна тысячная дюйма. Так как 1 *inch* = 25,4 мм, то 1 *mil* = 0,0254 мм. Первые ИС, имеющие 14 и 16 выводов, выпускались в корпусе *PDIP*, ширина которого (без учета размеров выводов) равна 300 *mil* (7,62 мм — калибр 3-линейной винтовки С. И. Мосина).

Фирма *Texas Instruments* для производства ИС использует следующие типы корпусов (звездочкой отмечены корпусы для ИС военного назначения):

CDIP * — *Ceramic Dual-In-Line Package* (керамический корпус с двухрядным расположением выводов);

CFP * — *Ceramic Flat Package* (керамический корпус с планарными выводами);

CPGA * — *Ceramic Pin Grid Array* (керамический корпус с многорядными штырьками);

CQFP * — *Ceramic Quad Flat Package* (керамический корпус с четырехсторонними планарными выводами);

LCCC * — *Leadless Ceramic Chip Carrier* (керамический кристаллоноситель без выводов; контакты напылены на керамику);

LFBGA —*Low-Profile Fine-Pitch Ball Grid Array* (*MicroStar BGA*™; корпус с многорядными круглыми штырьками);

PDIP — *Plastic Dual-In-Line Package* (пластмассовый корпус с двухрядным расположением выводов);

PLCC — *Plastic Leaded Chip Carrier* (пластмассовый корпус с выводами на кристаллоносителе — J-образные контакты, заходящие под нижнюю поверхность корпуса; см. рис. 2.25);

QFP — *Plastic Quad Flat Package* (пластмассовый корпус с четырехсторонними планарными выводами);

Таблица 1.50. **Типы корпусов интегральных схем**

Тип корпуса	Число выводов	Ширина корпуса, мм	Шаг выводов, мм	Фирменные коды корпусов				
				TI	*Philips*	*Fairchild*	*Toshiba*	*Motorola*
PDIP	14, 16, 18, 20	7,87	2,54	*N*	*N*	*N*	*P*	*P, N*
	24	7,87	2,54	*NT*	*N*	—	*P*	*N*
SOIC	8, 14, 16	4,0	1,27	*D*	*D*	*M/S*	*F*	*D*
	16, 20, 24	7,59	1,27	*DW*	*D*	*WM*	*FW*	*DW*
SSOP	14, 16, 20, 24	5,6	0,65	*DB*	*DB*	*MSA*	*FN*	*SD*
	48, 56	7,59	0,635	*DL*	*DL*	*MEA*	—	—
TSSOP	8, 14, 16, 20, 24	4,5	0,65	*PW*	*PW*	*MTC*	*FS*	*DT*
	48, 56, 64	6,4	0,5	*DGG*	*DGG*	*MTD*	*FT*	—
TVSOP	14, 16, 20, 24, 48, 56	4,5	0,4	*DGV*	—	—	—	—
	80	8,4	0,4	*DBB*	—	—	—	—
1*G*	5	1,8	0,95	*DBV*	—	—	*F*	—

QSOP — Quarter-Size Outline Package;

SOIC — Small-Outline Integrated Circuit (ИС с выводами малого размера);

SOP — Small-Outline Package (корпус с выводами малого размера);

SOT — Small-Outline Transistor Package (корпус типа транзисторного с пятью выводами малого размера);

SOT-23 — *MicroGate* (см. рис. 1.69 и 1.70);

SOT-323 — *PicoGate* (см. рис. 1.69 и 1.70);

SSOP — Shrink Small-Outline Package (корпус с выводами уменьшенного размера);

TQFP — Plastic Thin Quad Flat Package (пластмассовый корпус с тонкими четырехсторонними планарными выводами — см. рис. 1.72);

TSSOP — Thin Shrink Small-Outline Package (корпус с тонкими выводами уменьшенного размера — см. рис. 1.73);

TVSOP — Thin Very Small-Outline Package (корпус с тонкими сверхмалыми выводами).

Новые технологии интегральных схем. В табл. 1.45 и 1.49 приведены некоторые новые технологии изготовления ИС, разработанные фирмой *Texas Instruments*. По этим технологиям изготавливается широкий набор функциональных узлов, наиболее часто используемых при проектировании МП-систем: шинные драйверы, шинные приемопередатчики (без регистров памяти, с регистрами памяти, со схемами контроля четности), синхронные и асинхронные регистры памяти. Ниже приведено краткое описание этих технологий.

AC/ACT (*Advanced CMOS Logic*) — надежные маломощные семейства ИС, изготавливаемые по 1-мкм *CMOS*-технологии и содержащие более 160 ИС (ЛЭ, триггеры, регистры, драйверы, счетчики, приемопередатчики и др.), обеспечивающие выходные токи 24 мА. Выпускаются ИС со стандартным и центральным расположением выводов питания, позволяющим уменьшить помехи в быстродействующей логике при одновременном переключении нескольких сигналов. Входы ИС семейства *AC* совместимы с выходами КМОП ИС, а входы ИС семейства *ACT* — с выходами ТТЛ ИС.

AHC/AHCT (*Advanced High-Speed CMOS Logic*) — логические семейства ИС, представляющие собой естественное усовершенствование семейств *HC/HCT*. Эти семейства предназначены для пользователей, которые нуждаются в большем быстродействии для маломощных с низким уровнем помех приложений. Изготавливаются ИС семейства *AHC* на основе процесса *EPIC*1-*S* (*Enhanced-Performance Implanted CMOS*), который характеризуется высокой эффективностью при низкой стоимости. ИС семейства *AHC* имеют рабочие характеристики:

быстродействие — типовая задержка распространения сигналов составляет 5,2 нс (для 8-разрядных функциональных узлов), т. е. приблизительно в три раза меньше, чем в ИС семейства *HC*;

низкий уровень помех без проблем положительных и отрицательных выбросов напряжения, типичных для ИС с большим быстродействием и большими выходными токами, как в семействе *AHC*;

выходной ток драйверов равен ±8 мА и ±4 мА при напряжениях питания +5 В и +3,3 В соответственно;

малая мощность потребления — максимальное значение тока в статическом режиме равно 40 мкА (вдвое меньше, чем в ИС семейства HC).

LVC (*Low-Voltage CMOS Technology*) — семейство ИС, специально спроектированное для использования при напряжениях питания +3,3 В, +2,5 В и +1,8 В (1993 г.). ИС этого семейства изготовляются по высокоэффективной версии 0,8-мкм *CMOS*-технологии и имеют максимальное значение задержки распространения сигналов 6,5 нс и токи выходных сигналов 24 мА. Все ИС семейства *LVC* имеют 5-В допуск напряжения по входам и выходам.

ALVC (*Advanced Low-Voltage CMOS Technology*) — одно из самых высокоэффективных семейств ИС шинного интерфейса с напряжением питания +3,3 В и +2,5 В (1994 г.). Изготов-

ляются ИС по 0,6-мкм *CMOS*-технологии (субмикронный *EPIC*™-процесс) и имеют типовое значение задержки распространения сигналов меньше чем 3 нс, токи выходных сигналов 24 мА и ток потребления в статическом режиме 40 мкА.

BCT (*BiCMOS Bus-Interface Technology*) — семейство ИС 8-, 9- и 10-разрядных шинных драйверов, синхронных и асинхронных потенциальных регистров памяти (защелок) и приемопередатчиков (без регистров памяти и с регистрами памяти). Семейство ИС разработано специально для применения в шинных интерфейсах и обеспечивает высокую скорость ввода-вывода для ТТЛ-интерфейсов, выходной ток драйверов 64 мА и очень малую мощность потребления в пассивном режиме. В настоящее время выпускается более 50 типов ИС. В семейство *BCT* включен также ряд драйверов памяти, которые имеют последовательно включенные демпфирующие резисторы для подавления положительных и отрицательных выбросов напряжения, которые могут возникать при использовании драйверов памяти.

ABT (*Advanced BiCMOS Technology*) — семейство второго поколения *BiCMOS* ИС шинного интерфейса (усовершенствованное семейство *BCT* с 0,8-микронными нормами). Это семейство обеспечивает выходной ток драйверов 64 мА и задержки распространения сигналов ниже 5 нс при сверхнизком потреблении мощности. Для уменьшения помех в линиях передачи в семействе *ABT* используются последовательно включенные демпфирующие резисторы. Кроме того, выпускаются специальные ИС, которые обеспечивают чрезвычайно большой ток (180 мА) для линий передачи с волновым сопротивлением до 25 Ом. Семейство содержит 8-разрядные, *Widebus*™ и *Widebus+*™ функциональные устройства. Расположение выводов корпусов ИС *Widebus*™ и *Widebus+*™ выбрано с учетом требований уменьшения помех и упрощения размещения ИС на печатной плате.

LVT (*Low-Voltage BiCMOS Technology Logic*) — семейство ИС с напряжением питания +3,3 В, изготавливаемое по 0,72-мкм *BiCMOS* технологии и предназначенное для функций шинного интерфейса (1992 г.). Подобно его 5-В аналогу *ABT*, семейство ИС *LVT* характеризуется выходными токами 64 мА, значением задержки распространения сигналов, равным 3,5 нс, и, кроме того, в пассивном режиме потребляет ток меньше чем 100 мкА. На выходах, находящихся в *Z*-состоянии, задается высокий уровень напряжения (*Power-Up Tri-state*).

ALVT (*Advanced Low-Voltage BiCMOS Technology Logic*) — семейство ИС с напряжением питания +3,3 В и +2,5 В, изготавливаемое по 0,6-мкм *BiCMOS* технологии и предназначенное для замены ИС семейств *ABT* и *LVT*. ИС семейства *ALVT* имеют быстродействие на 28% больше, чем у ИС семейства *LVT*. Обеспечиваются выходные токи до 64 мА при напряжении питания +3,3 В и до 24 мА при напряжении питания +2,5 В. Входы автоматически удерживаются в последнем правильном логическом состоянии, устраняя неопределенность плавающего значения входных сигналов (*Auto Tri-state*).

ALB (*Advanced Low-Voltage BiCMOS*) — специально разработанное семейство ИС с напряжением питания +3,3 В, изготавливаемое по последней 0,6-мкм *BiCMOS* технологии и предназначенное для функций шинного интерфейса. Семейство ИС характеризуется выходными токами 25 мА при напряжении питания 3,3 В и максимальным значением задержки распространения сигналов, равным 2,2 нс. Входы имеют фиксирующие диоды, устраняющие положительные и отрицательные выбросы. ИС выпускаются в корпусах с улучшенными параметрами, таких как *SSOP* (1989 г.), *TSSOP* (1991 г.) и *TVSOP* (1996 г.).

AVC (*Advanced Very-Low-Voltage CMOS Logic*) — семейство ИС с напряжением питания +3,3 В, +2,5 В, +1,8 В, +1,5 В и +1,2 В. Это семейство обеспечивает проектировщиков инструментальными средствами для создания расширенных высокоскоростных систем с задержками распространения меньшими чем 2 нс.

CBT (*Crossbar Technology*) — семейство ИС с напряжением питания +5 В, обеспечивающее выполнение двух основных требований на сегодняшнем компьютерном рынке: малую мощность потребления и высокое быстродействие. Семейство *CBT* предназначено для приме-

нения в быстродействующих шинных интерфейсах, связующих компоненты компьютерной системы. ИС этого семейства могут использоваться как преобразователи (трансляторы) уровней сигналов ИС с напряжением питания +5 В в уровни сигналов ИС с напряжением питания +3,3 В, позволяя проектировщикам в одной системе использовать компоненты с разными напряжениями питания (+5 В и +3,3 В).

CBTLV (*Low-Voltage Crossbar Technology Logic*) — семейство ИС с напряжением питания +3,3 В, +2,5 В, дополняющее семейство *CBT* для низковольтных приложений.

Маркировка ИС фирмы *Texas Instruments Incorporated*. Маркировка ИС производится буквенно-цифровым кодом, содержащим 10 полей, некоторые из которых могут отсутствовать:

SN	74	ALVC	H	16	2	244	A	DGG	R
1	2	3	4	5	6	7	8	9	10

1 — *Standard Prefix*

2 — *Temperature Range*
- 54 — *Military*
- 74 — *Commercial*

3 — *Family*
- Blank — *Transistor-Transistor Logic*
- ABT — *Advanced BiCMOS Technology*
- ABTE/ETL — *Advanced BiCMOS Technology/ Enchanced Transceiver Logic*
- AC/ACT — *Advanced CMOS Logic*
- AHC/AHCT — *Advanced High-Speed CMOS Logic*
- ALB — *Advanced Low-Voltage BiCMOS*
- ALS — *Advanced Low-Power Schottky Logic*
- ALVC — *Advanced Low-Voltage CMOS Technology*
- AS — *Advanced Schottky Logic*
- AVC — *Advanced Very Low-Voltage CMOS Logic*
- BCT — *BiCMOS Bus-Interface Technology*
- CBT — *Crossbar Technology*
- CBTLV — *Low-Voltage Crossbar Technology*
- CD4000 — *CMOS B-Series Integrated Circuits*
- F — *F Logic*
- FB — *Backplane Transceiver Logic/Futurebus+*
- FCT — *Fast CMOS TTL Logic*
- GTL — *Gunning Transceiver Logic*
- HC/HCT — *High-Speed CMOS Logic*
- HSTL — *High-Speed Transceiver Logic*
- LS — *Low-Power Schottky Logic*
- LV — *Low-Voltage CMOS Technology Logic*
- LVC — *Low-Voltage CMOS Technology Logic*
- LVT — *Low-Voltage BiCMOS Technology*
- PCA — I^2C Inter-*Integrated Circuit Applications*
- S — *Schottky Logic*
- SSTL/SSTV — *Stub Series Terminated Logic*
- TVC — *Translation Voltage Clamp Logic*

4 — *Special Features* (*Examples*)
- Blank = *No Special Features*
- C — *Configurable V_{CC}* (*LVCC*)
- D — *Level-Shifting Diode* (*CBTD*)

4 — *Special Features* (*continued*)
- H — *Bus Hold* (*ALVCH*)
- K — *Undershoot-Protection Circuitry* (CBTK)
- R — *Damping Resistor on Inputs/Outputs* (*LVCR*)
- S — *Schottky Clamping Diode* (*CBTS*)
- Z — *Power-Up 3-State* (*LVCZ*)

5 — *Bit Width*
- Blank = *Gates, MSI and Octals*
- 1G — *Single Gate*
- 8 — *Octal IEEE* 1149.1 (*JTAG*)
- 16 — *Widebus*™ (16, 18 *and* 20 *bit*)
- 18 — *Widebus IEEE* 1149.1 (*JTAG*)
- 32 — *Widebus+*™ (32 *and* 36 *Bit*)

6 — *Options*
- Blank = *No Options*
- 2 — *Series Damping Resistor on Outputs*
- 4 — *Level Shifter*
- 25 — 25-Ω *Line Driver*

7 — *Function* (*Examples*)
- 244 — *Noninverting Octal Buffer/Driver*
- 374 — *Octal D-Type Flip-Flop*
- 573 — *D-Type Transparent Latch*
- 640 — *Inverting Octal Transceiver*

8 — *Device Revision*
- Blank = *No Revision*
- *Letter Designator A–Z*

9 — *Packages*
- D, DW — *SOIC*; DB, DL — *SSOP*; DBB, DGV — *TVSOP*
- DBV — *SOT*; DGG, PW — *TSSOP*; FK — *LCCC*
- FN — *PLCC*; GB — *CPGA*; HFP, HS, HT, HV — *CQFP*
- J, JT — *CDIP*; N, NP, NT — *PDIP*; NS, PS — *SOP*
- PAG, PAH, PCA, PCB, PM, PN, PZ — *TQFP*
- PH, PQ, RC — *QFP*; W, WA, WD — *CFP*
- DBQ — *QSOP*; GKE, GKF, GQL — *LFBGA*

10 — *Tape and Reel*
- LE — *Left Embossed* (*valid for DB and PW packages only*)
- R — *Standard* (*valid for surface-mount packages*)

Источник: ***Logic Selection Guide*** (*Second Half* 2000)

JTAG — стандарт последовательного интерфейса тестирования цифровых устройств (*IEEE Standard* 1149.1–1990 *Test Access Port and Boundary-Scan Architecture*), *IEEE* (*Institute of Electrical and Electronic Engineers*) — Институт инженеров по электротехнике и радиоэлектронике (ИИЭР).

Глава 2

МЕТОДЫ ВВОДА-ВЫВОДА

2.1. Классификация регистров памяти и методов ввода-вывода

Большое разнообразие типов регистров памяти и триггеров (одноразрядных регистров), выпускаемых в виде отдельных ИС и используемых в интерфейсных БИС [5], может затруднить изучение их применения в МП-системах. Все регистры памяти необходимо разделить на группы, исходя из их свойств как элементов МП-системы, а не из свойств элементов цифровых автоматов.

Классификация регистров памяти. Регистры памяти, как внешние устройства, по назначению подразделяются на *регистры данных* (устройства ввода, вывода и ввода-вывода), которые служат для выдачи и приема данных, *регистры состояния* (только устройства ввода), используемые для выдачи в МП информации о состоянии внешнего устройства, и *регистры управления* (только устройства вывода), позволяющие МП изменять функции, выполняемые внешним устройством. Регистры памяти могут быть как синхронными, так и асинхронными потенциальными. В интерфейсных БИС часто используются *регистры вывода с обратным чтением* (*Read-Back*), позволяющие как выводить, так и контролировать записанную в них информацию (рис. 2.1). Полное описание таких регистров (например, *SN*74*ALS*990 и *SN*74*AS*996) приведено в [5]. Регистры ввода-вывода данных состоят из двух регистров — регистра ввода и регистра вывода, подключенных к общей двунаправленной шине данных и имеющих один и тот же адрес порта, определяемый сигналом $\overline{CS3}$ (рис. 2.1).

Изображенный на рис. 2.1 регистр ввода-вывода данных предназначен для подключения внешнего устройства, имеющего двунаправленную шину данных. Этот регистр предназначен для программного ввода-вывода с квитированием (сигналы $\overline{STB}$ и $\overline{ACK}$, управляют триггерами флагов квитирования, которые на рис. 2.1 не показаны — см. § 2.3). Регистры данных называются также *буферными регистрами*, так как они часто используются для согласования шин данных внешних устройств с системной шиной данных МП-системы.

Пример 1 (операции ввода и вывода для устройств, изображенных на рис. 2.1):

```
    MVI  B, 55h  ; Тестирование порта вывода с обратным чтением (B ← 55h = 0101 0101)
    MOV  A, B
; Cs1, Cs2 и Cs3 — символические имена адресов портов, определяемые директивами
;   ассемблера, CS1, CS2 и CS3 — физические адресные сигналы на рис. 2.1,
;   соответствующие этим портам
    OUT  Cs1     ; ⇒ α = I/OW ∨ CS1 = I/OW·CS1 = ⊔⊓ — запись числа 55h в регистр вывода
    IN   Cs1     ; ⇒ OE = I/OR ∨ CS1 = I/OR·CS1 = ⊔⊓ — обратное чтение регистра вывода
    CMP  B       ; Сравнение содержимого аккумулятора с числом 55h
    JNZ  Err     ; Переход, если при выводе или вводе произошла ошибка
    CMA          ; A ← Ā = AAh — инвертирование содержимого аккумулятора
    MOV  B, A
    OUT  Cs1     ; Запись в регистр вывода числа AAh = 1010 1010
```

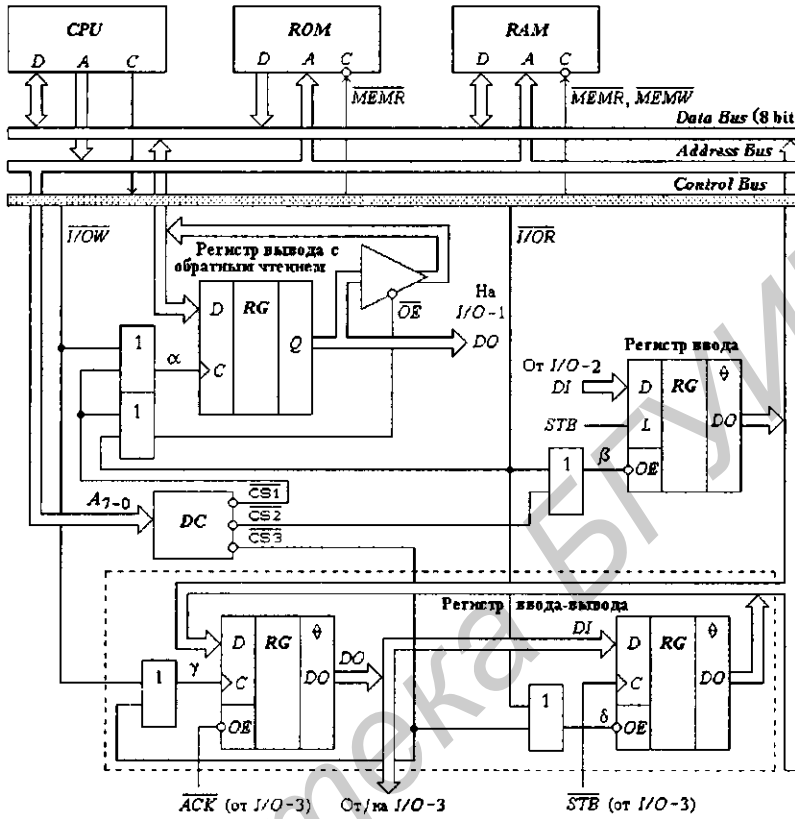


Рис. 2.1. Регистры ввода, вывода и ввода-вывода

```
        IN    Cs1   ; Чтение (обратное) регистра вывода (A ← I/O–1)
        CMP   B     ; Сравнение содержимого аккумулятора с числом AAh
        JNZ   Err   ; Переход, если при выводе или вводе произошла ошибка
         ∴          ; Чтение регистра ввода
        IN    Cs2   ; ⇒ β = I/OR ∨ CS2 = I/OR · CS2 = ⊔⊓ — чтение регистра ввода (A ← I/O–2)
         ∴          ; Чтение регистра ввода-вывода
        IN    Cs3   ; ⇒ δ = I/OR ∨ CS3 = I/OR · CS3 = ⊔⊓ — ввод (A ← I/O–3)
         ∴
        MVI   A, d8 ; Запись в регистр вывода
        OUT   Cs3   ; ⇒ γ = I/OW ∨ CS3 = I/OW · CS3 = ⊔⊓ — вывод (I/O–3 ← d8)
Err:     ∴          ; Вывод на дисплей сообщения об ошибке записи данных в порт вывода
```

На рис. 2.2 показан 8-разрядный регистр ввода-вывода, работа которого на ввод или вывод задается программным способом: *MUX*–1 и *MUX*–2 — двухканальные 8-разрядный и одноразрядный мультиплексоры. Мультиплексор *MUX*–1 подключает к входам данных регистра *RG* системную шину данных (адресный вход мультиплексора $A = Q = 0$) или шину данных внешнего устройства ($A = Q = 1$). Мультиплексор *MUX*–2 соответственно этому переключает сигнал записи данных в регистр:

$$\overline{H} = \alpha \cdot \overline{Q} \vee \overline{STB} \cdot Q = \overline{I/OW \cdot CS2} \cdot \overline{Q} \vee \overline{STB} \cdot Q.$$

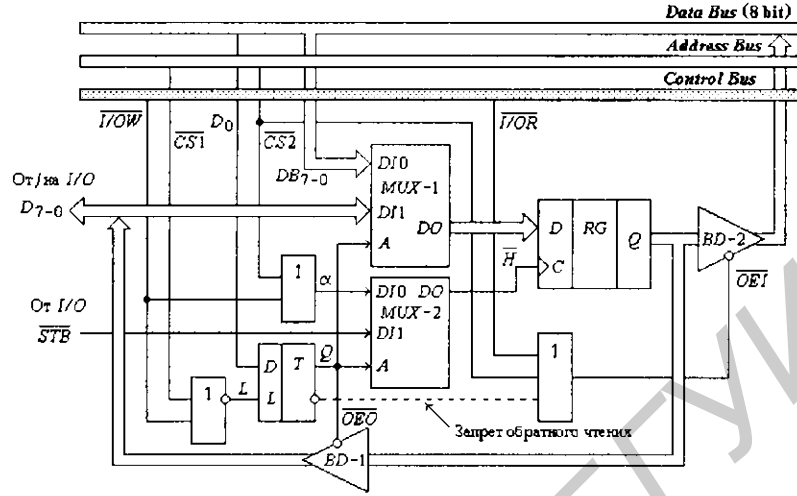


Рис. 2.2. Программно переключаемый регистр ввода-вывода

Шинные драйверы *BD*–1 и *BD*–2 с *Z*-состоянием выходов подключают выходы регистра к шине внешнего устройства ($\overline{OEO} = Q = 0$) или к системной шине данных ($\overline{OEI} = \overline{I/OR \cdot CS2} = 0$).

Перед использованием переключаемого регистра следует произвести настройку его режима работы на ввод или вывод заданием состояния триггера *Q*:

```
SUB  A     ; A ← 0 — программирование работы регистра на вывод
OUT  Cs1   ; ⇒ D0 = 0, L = I/OW·CS1 = ⎍ — установка значения
           ;      выходного сигнала триггера Q = 0 (задание режима вывода)
MVI  A, d8 ; A ← d8 = 00 … FFh
OUT  Cs2   ; ⇒ H = α = I/OW ∨ CS2 = I/OW·CS2 = ⎎ — запись байта d8 в регистр вывода
  ∴
MVI  A, 1  ; A ← 1 — программирование работы регистра на ввод
OUT  Cs1   ; ⇒ D0 = 1, L = I/OW·CS1 = ⎍ — установка значения
           ;      выходного сигнала триггера Q = 1 (задание режима ввода)
; H = STB = ⎎ — запись данных в регистр ввода, Cs1 — порт управления, Cs2 — порт данных
IN   Cs2   ; ⇒ OEI = I/OR·CS2 = ⎎ — чтение регистра ввода
```

Если $Q = 0$, то включен канал 0 мультиплексоров, значение сигнала $\overline{OEO} \equiv 0$ и на выходы вентилей *BD*–1 поступают сигналы Q_r ($r = 0 \ldots 7$) с выходов регистра памяти — включен режим вывода. Обратное чтение данных из регистра вывода с помощью вентилей *BD*–2 возможно только при отсутствии связи, показанной на рис. 2.2 штриховой линией.

Если $Q = 1$, то включен канал 1 мультиплексоров, значение сигнала $\overline{OEO} \equiv 1$ и выходы вентилей *BD*–1 находятся в *Z*-состоянии, а сигнал $\overline{OEI} = \overline{I/OR \cdot CS2 \cdot Q} = \overline{I/OR \cdot CS2}$ управляет вводом данных из регистра памяти в аккумулятор.

Классификация методов ввода-вывода. Во всех МП-системах могут использоваться методы ввода-вывода: программный ввод-вывод, ввод-вывод по прерыванию и ввод-вывод по прямому доступу к памяти. Программный ввод-вывод подразделяется на два типа: программный ввод-вывод без квитирования и программный ввод-вывод с квитированием, в котором используется программное чтение флагов квитирования — один флаг (триггер) для ввода и один флаг для вывода. Остальные методы ввода-вывода используют аппаратные средства (сигналы) квитирования.

Отображение ввода-вывода на память. МП 8080*A* и 8085*A* имеют независимые адресные пространства для памяти (64 Кбайта) и внешних устройств (256 устройств ввода и 256 устройств вывода). Обращения к этим адресным пространствам сопровождается выдачей системных сигналов управления $\overline{MEWR}$, $\overline{MEMW}$ и $\overline{I/OR}$, $\overline{I/OW}$ соответственно. Адресное пространство внешних устройств можно и не использовать, если для чтения и записи в них данных сигналы $\overline{I/OR}$ и $\overline{I/OW}$ заменить сигналами $\overline{MEWR}$ и $\overline{MEMW}$. В этом случае часть адресного пространства памяти следует зарезервировать для внешних устройств и обращение к ним производить теми же командами, что и при обращении к памяти. Для управления адресным пространством ввода-вывода МП 8080*A* и 8085*A* имеют только две команды: IN *port* и OUT *port*, предназначенные только для передачи данных с участим аккумулятора *A*. Для управления же адресным пространством памяти имеется значительно больше команд, выполняющих в том числе и преобразования операндов (см. § 1.6). Из этого следует, что к достоинствам отображения ввода-вывода на память следует отнести:

1) упрощение аппаратной части МП-системы (не нужно использовать системные сигналы управления $\overline{I/OR}$ и $\overline{I/OW}$);

2) упрощение программной части МП-системы за счет использования более сложных команд для ввода-вывода, чем команды IN *port* и OUT *port*;

3) возможность построения МП-системы с большим числом внешних устройств, чем непосредственно адресуемых командами IN *port* и OUT *port*.

В МП-системе адресное пространство памяти можно использовать для расширения стандартного адресного пространства ввода-вывода, отобразив на память внешние устройства, для которых не хватило места в стандартном адресном пространстве ввода-вывода. При отображении всех или части устройств ввода-вывода на память несколько усложняется адресация памяти и внешних устройств, так как адресное пространство памяти необходимо разделить на две части. Например, если на память требуется отобразить 512 портов ввода-вывода, то разделение адресного пространства памяти на две части можно осуществить выходным сигналом детектора состояния адресных сигналов A_{15-10}, реализующего функцию

$$EM = \overline{A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9}$$

(*EM* = 1 — обращение к памяти, *EM* = 0 — обращение к внешним устройствам). Для этого сигнал *EM* должен быть использован для включения/выключения раздельных адресных дешифраторов памяти и внешних устройств. Адреса портов ввода-вывода будут определяться значениями 1111 111× ×××× ××××, где × = 0 и 1. Следовательно, для внешних устройств и памяти можно использовать непересекающиеся адресные пространства FE00 ÷ FFFF*h* и 0000 ÷ FDFF*h* соответственно.

Пример МП-системы с отображением устройств ввода-вывода на память приведен в § 3.9 (см. рис. 3.127 и соответствующую ему программу на с. 303).

2.2. Программный ввод-вывод без квитирования

Программный ввод-вывод без квитирования осуществляется непосредственно командами IN *port* и OUT *port*. Для выполнения этих команд затрачивается 10 тактов (5 мкс при частоте тактового сигнала 2 МГц, используемого в МП-системах, построенных на основе МП 8080*A*). Этот тип ввода-вывода требует меньших затрат времени по сравнению с программным вводом-выводом с квитированием. Внешнее устройство в этом случае должно быть немедленно готовым выдать байт данных по значению сигнала $\overline{I/OR} = 0$, генерируемому при выполнении команды IN *port*, и принять байт данных по значению сигнала $\overline{I/OW} = 0$, генерируемому при выполнении команды OUT *port*.

Такими внешними устройствами обычно являются регистры памяти, выполненные как в виде отдельных ИС средней степени интеграции, так и содержащиеся внутри интерфейсных БИС, если они используются в качестве регистров состояния и регистров управления --- ввод слова состояния (*Status Word*) в МП и вывод слова управления (*Control Word*) из МП. Именно для этих целей программный ввод и вывод без квитирования используется наиболее часто (передача между МП и внешним устройством по определенному адресу одиночных байт).

При последовательной передаче блоков данных (более одного байта) дело обстоит сложнее, так как необходимо согласовывать скорость обработки байт данных микропроцессором и внешним устройством. Но и в этом случае иногда можно использовать программный ввод-вывод без квитирования.

Программный ввод-вывод без квитирования. Обычно для согласования системной шины данных с внешним устройством между ними включаются буферные регистры (регистры данных). Пример буферных регистров вывода был приведен на рис. 1.38, *д*. На рис. 2.3 показана структурная схема внешних устройств, предназначенных для программного ввода и вывода без квитирования и чтения состояния переключателей $SW_3 \div SW_0$ (переключатели по их использованию аналогичны регистру состояния). Вместо дешифратора адреса в этой схеме использован демультиплексор (ИС 555ИД7) для коммутации сигнала управления $\overline{G} = \overline{I/OR} \vee \overline{I/OW}$ по адресам внешних устройств, приведенным в табл. 2.1. Здесь порты ввода и вывода не могут иметь одинаковые адреса, так как демультиплексируется дизъюнкция сигналов $\overline{I/OR}$ и $\overline{I/OW}$ — операции ввода и вывода неразличимы (часть адресного пространства ввода-вывода теряется).

Адреса портов очень легко вычисляются по схеме включения демультиплексора [5], например, выход 1 демультиплексора описывается функцией

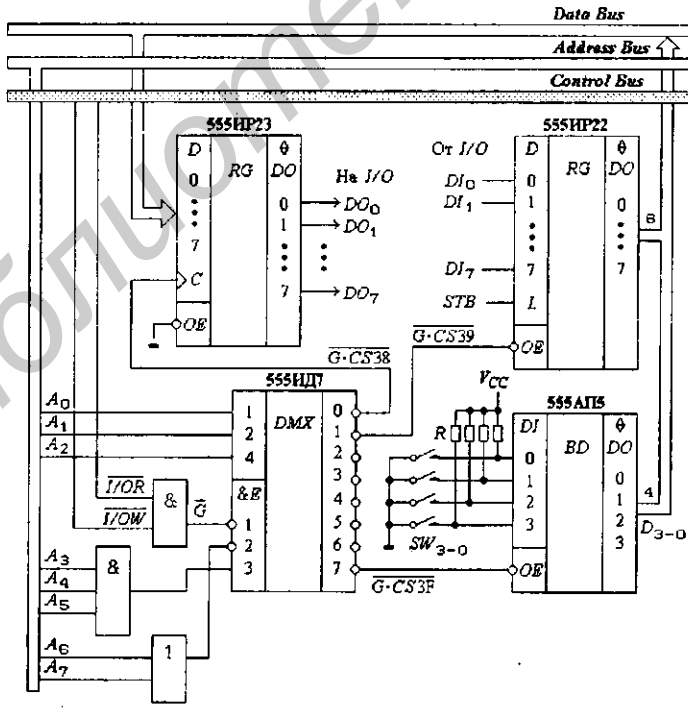


Рис. 2.3. Схема устройства ввода-вывода без квитирования

$$\overline{G \cdot CS39} = \overline{(I/OR \vee I/OW) \cdot \overline{A_2} \cdot \overline{A_1} \cdot A_0} \text{ при } A_7A_6A_5A_4A_3 = 00111,$$

т. е. *port* = $A_7A_6A_5A_4\ A_3A_2A_1A_0$ = 0011 1001 = 39*h*. При обращении к этому порту значение сигнала $\overline{G \cdot CS39} = 0$ (⊔), в остальных же случаях значение сигнала $\overline{G \cdot CS39} = 1$.

Таблица 2.1. **Адресация *I/O***

Разряды адреса								Выход *DMX*	*port*
7	6	5	4	3	2	1	0		
0	0	1	1	1	0	0	0	0	38*h*
0	0	1	1	1	0	0	1	1	39*h*
0	0	1	1	1	0	1	0	2	3A*h*
0	0	1	1	1	0	1	1	3	3B*h*
0	0	1	1	1	1	0	0	4	3C*h*
0	0	1	1	1	1	0	1	5	3D*h*
0	0	1	1	1	1	1	0	6	3E*h*
0	0	1	1	1	1	1	1	7	3F*h*

По адресу 3F*h* производится чтение регистра состояния, выполненного на механических переключателях $SW_{3\text{-}0}$ и драйвере 555АП5. Вывод данных осуществляется в регистр памяти 555ИР23 по адресу 38*h*. Для ввода данных использован регистр памяти 555ИР22, имеющий адрес 39*h*, информация в который записывается сигналом *STB* = 1 (*STB* — *Strobe*), вырабатываемым внешним устройством *I/O*.

Задача 1. Записать в ячейку памяти *M*(22AC) содержимое регистра памяти 555ИР22; прочитать состояние переключателей $N = SW_3SW_2SW_1SW_0$; вывести в регистр 555ИР23 содержимое ячейки памяти *M*(22AD) при $N < 6$ или содержимое ячейки памяти *M*(22AE) при $N \geq 6$. *Решение*:

```
      LXI   H, 22ACh  ; rp H ← 22ACh — адрес ячейки памяти
      IN    39h       ; A ← I/O(39) — ввод из регистра 555ИР22
      MOV   M, A      ; M(22AC) ← A — запись в память по адресу 22ACh
      INX   H         ; Содержимое rp H увеличить на 1
      IN    3Fh       ; A ← I/O(3F) = ××××SW3SW2SW1SW0 — чтение регистра состояния
      ANI   0Fh       ; 0Fh = 0000 1111 — маска для выделения младшей тетрады
      CPI   6         ; Сравнение N = SW3SW2SW1SW0 с числом 6
      JC    L1        ; Если N < 6, то переход на метку L1
      INX   H         ; Если N ≥ 6, то увеличить на 1 адрес памяти
L1:   MOV   A, M      ; A ← M(rp H) — чтение памяти по адресу 22ADh или 22AEh
      OUT   38h       ; I/O(38) ← A — вывод в регистр 555ИР23
```

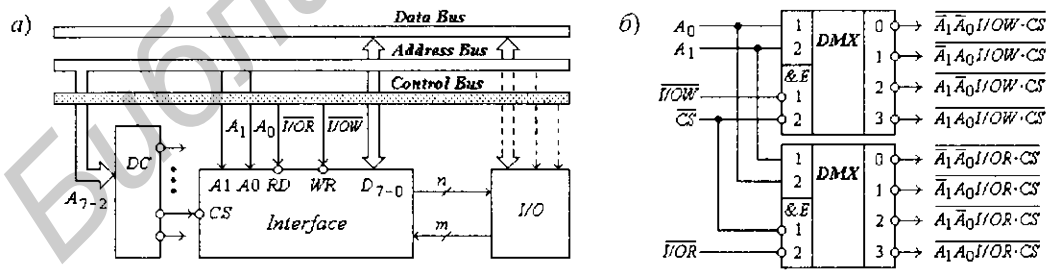


Рис. 2.4. Структурная схема интерфейсного устройства

На рис. 2.4, *а* показана структурная схема, поясняющая подключение интерфейсных БИС (*Interface*) к системным шинам адреса, данных и управления, обслуживающих внешние устройства (*I/O*). Эти БИС содержат, как правило, несколько регистров данных, регистр состояния и регистр управления. Число адресных разрядов, подаваемых непосредственно на БИС, определяет число адресуемых регистров ввода и вывода (разряды A_1 и A_0 подаются на внутренний

сдвоенный 4-канальный демультиплексор, что позволяет адресовать четыре регистра ввода и четыре регистра вывода — рис. 2.4, *б*). Адресация всей БИС выполняется с помощью внешнего дешифратора (*DC*), на который подаются остальные адресные разряды — $A_{7\text{-}2}$.

Интерфейсные БИС могут иметь самые различные значения n (число выходов) и m (число входов), связывающих БИС с внешним устройством. Связи внешнего устройства с системными шинами, показанные штриховыми линиями, могут отсутствовать. Одним из назначений интерфейса является согласование входных и выходных сигналов *I/O* с системными шинами микроконтроллера.

Управление программным вводом-выводом сигналом готовности *READY*. Сигнал готовности используется для увеличения длительности активных уровней системных сигналов управления внешними устройствами ($\overline{I/OR}$, $\overline{I/OW}$) и памятью ($\overline{MEMR}$, $\overline{MEMW}$) при их недостаточном быстродействии:

READY ⌞ ⇒ МП переходит в состояние ожидания, *READY* ⌠ ⇒ МП продолжает работу.

Продолжительность состояния ожидания МП всегда составляет целое число тактов, причем в этом состоянии сигналы системных шин не изменяются, а значит, сигнал готовности предоставляет разработчику МП-систем возможность управления длительностью активных уровней (0) системных сигналов управления в широких пределах.

Управление вводом-выводом сигналом готовности *READY* используется в тех случаях, когда быстродействия внешних устройств недостаточно для их реагирования на значения сигналов $\overline{I/OR} = 0$ и (или) $\overline{I/OW} = 0$ стандартной длительности. Для обеспечения надежной работы интерфейса в этом случае требуется увеличить длительность активных уровней этих сигналов на один или несколько тактов. Для этого используется схема формирования сигнала готовности *READY* для МП 8085 или *RDYIN* (*Ready Input*) для МП 8080, подаваемого на генератор 8224, который формирует привязанный к тактовому сигналу φ_2 сигнал готовности *READY*. Для краткости управление вводом-выводом сигналом готовности *READY* будем называть *вводом-выводом по готовности*.

Структурная схема устройства ввода-вывода по готовности с адресным управлением показана на рис. 2.5, *а*. Выходной сигнал $\overline{RDYIN_1}$ триггера устанавливается в состояние 1 перепадом ⌠ сигнала

$$\overline{H} = (I/OR \vee I/OW) \cdot CS_1.$$

Элемент задержки на время τ сигнала

$$\overline{I/OR} \;\&\; \overline{I/OW} = \overline{I/OR \vee I/OW}$$

позволяет задать значение сигнала $RDYIN = 0$ необходимой длительности.

Элементы задержки можно использовать как асинхронные, так и синхронные, выполненные в виде цифрового автомата. Временные диаграммы, изображенные на рис. 2.5, *б*, поясняют формирование сигнала $\overline{RDYIN_1}$. Для объединения сигналов $\overline{RDYIN_j}$ ($j = 1 \ldots m$) от m независимых внешних устройств применены ЛЭ НЕ с открытым коллекторным выходом [5].

Для введения одного такта ожидания элемент задержки всегда можно реализовать с помощью простейшего цифрового автомата с использованием имеющихся сигналов МП. На рис. 2.6, *а* изображена принципиальная схема адресного управления сигналом готовности в МП-системах, построенных на основе МП 8080. Синхронный *D*-триггер по значению сигнала МП $SYNC = 1$ (см. рис. 1.6) в начале каждого машинного цикла генерирует значение сигнала $Q = 1$ длительностью, равной одному периоду тактового сигнала φ_2. Сигнал готовности *RDYIN* описывается функцией

$$RDYIN = \overline{Q \cdot AC_1 \vee \ldots \vee Q \cdot AC_m}, \; AC_i \;\&\; AC_j \equiv 0 \text{ при } i \neq j.$$

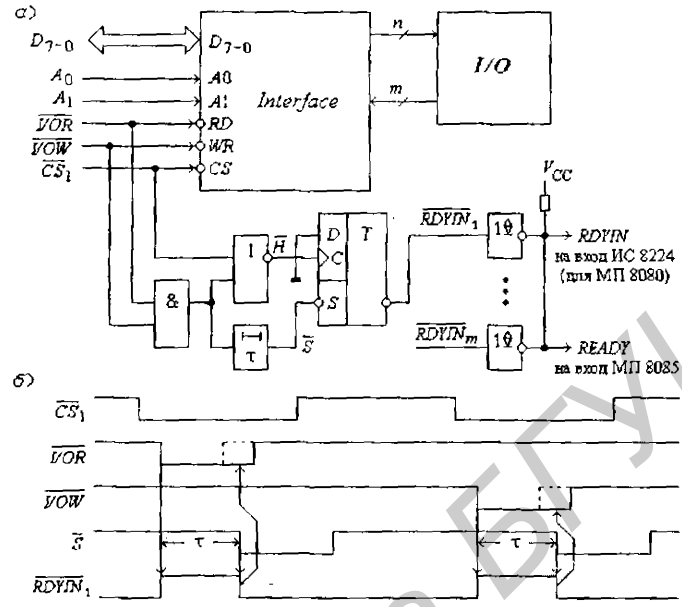


Рис. 2.5. Устройство ввода-вывода по готовности

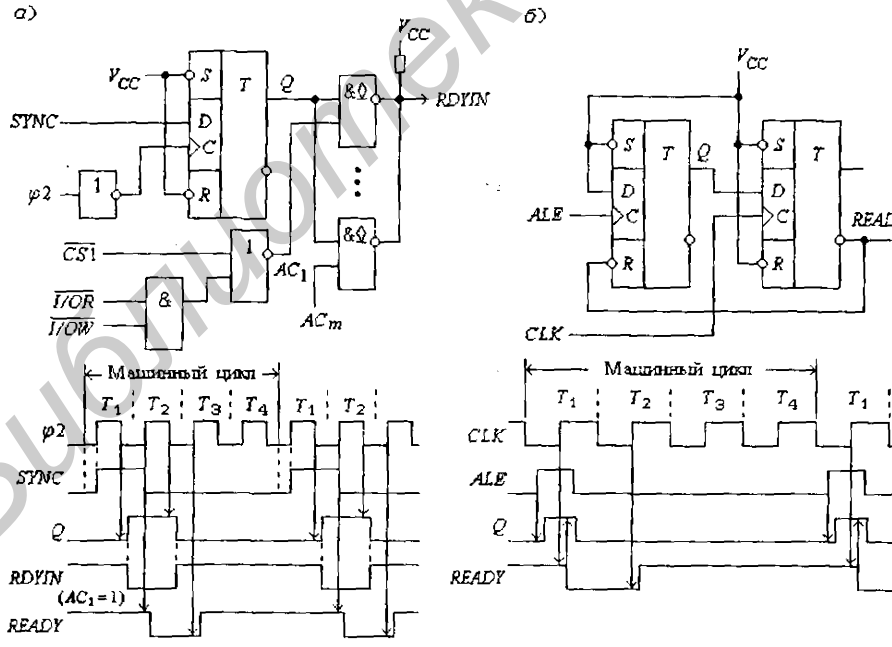


Рис. 2.6. Устройство ввода-вывода по готовности

Сигнал адреса-управления $AC_1 = (I/OR \vee I/OW) \cdot \overline{CS_1} = 1$ только при селекции внешнего устройства ($\overline{CS_1} = 0$) и выполнении операции ввода ($\overline{I/OR} = 0$) или вывода ($\overline{I/OW} = 0$). Тогда сигнал готовности $RDYIN = \overline{Q} = 0$ заставит МП ввести один такт ожидания.

На рис. 2.6, *б* изображена принципиальная схема генератора сигнала готовности, используемого в МП-системах, построенных на основе МП 8085. Сигнал готовности *READY* вводит один такт ожидания, а адресное управление сигналом готовности выполняется так же, как и в предыдущей схеме.

На рис. 2.7 показана принципиальная схема цифрового формирователя сигнала *RDYIN* с переключаемой длительностью значения *RDYIN* = 0. Сигнал

$$\overline{H} = \overline{I/OR} \oplus \overline{I/OW} = I/OR \vee I/OW,$$

так как $I/OR \cdot I/OW \equiv 0$. Любое изменение этого сигнала с 0 на 1 при значении $A_7A_6A_5 = 1$ устанавливает значение сигнала $\overline{RDYIN}_1 = 1$, которое переводит МП в состояние ожидания и включает счетчик 555ИЕ10.

Значение сигнала $\overline{R} = 0$ сбрасывает триггер, формирующий сигнал готовности $\overline{RDYIN}_1$. Полученное при этом значение сигнала $RDYIN_1 = 0$ блокирует счет, устанавливая нулевое состояние счетчика. Схема может обслуживать восемь внешних устройств — семь выходов у адресного дешифратора свободны.

Длительности значений сигналов $\overline{I/OR} = 0$ и $\overline{I/OW} = 0$ определяются выражениями

$$t_{IOR} + 2^r \cdot T_{\varphi 2} \quad \text{и} \quad t_{IOW} + 2^r \cdot T_{\varphi 2}$$

соответственно, где t_{IOR} и t_{IOW} — стандартные длительности сигналов управления (при значении сигнала *READY* ≡ 1 — переключатель в положении *OFF*), r — номер триггера Q_r, выходной сигнал которого выбран переключателем, $T_{\varphi 2}$ — период тактового сигнала φ_2.

Механический переключатель можно заменить электронным коммутатором. Для этого схему следует дополнить пятиканальным мультиплексором и трехразрядным регистром памяти, управляющим адресами каналов мультиплексора. Такой схемой можно будет управлять программным способом: выполнив команду OUT *port* для записи в регистр одного из чисел 0…4, можно задать число тактов ожидания 0, 1, 2, 4 или 8. Описанная схема полезна на стадии отладки МП-систем.

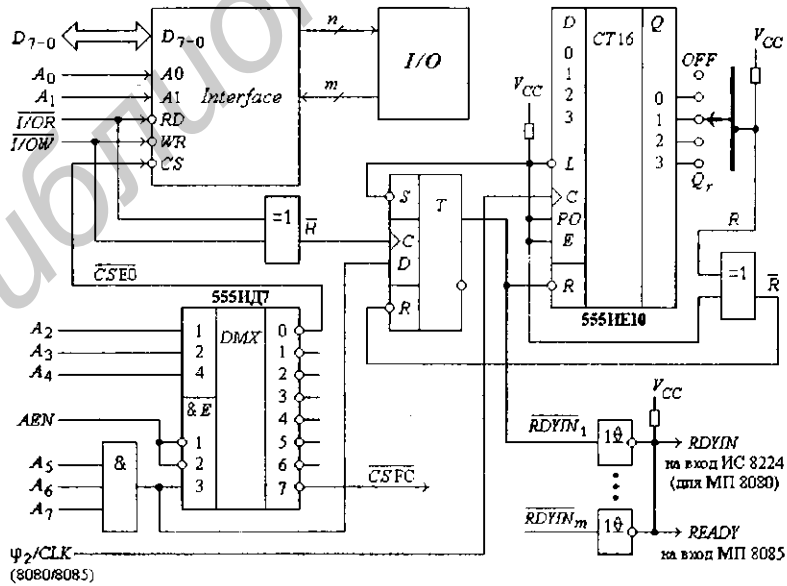


Рис. 2.7. Схема управляемой задержки сигналов управления

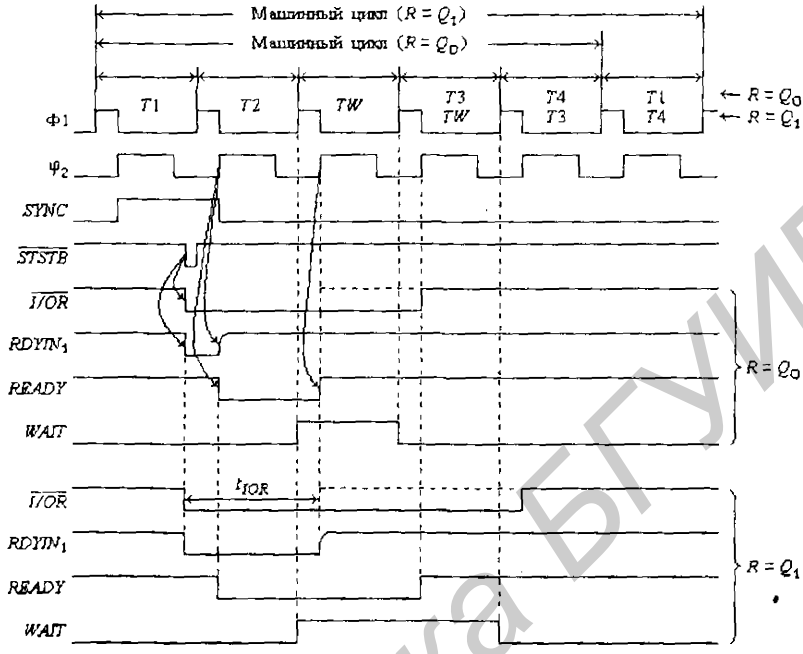


Рис. 2.8. Временны́е диаграммы режима ввода по готовности

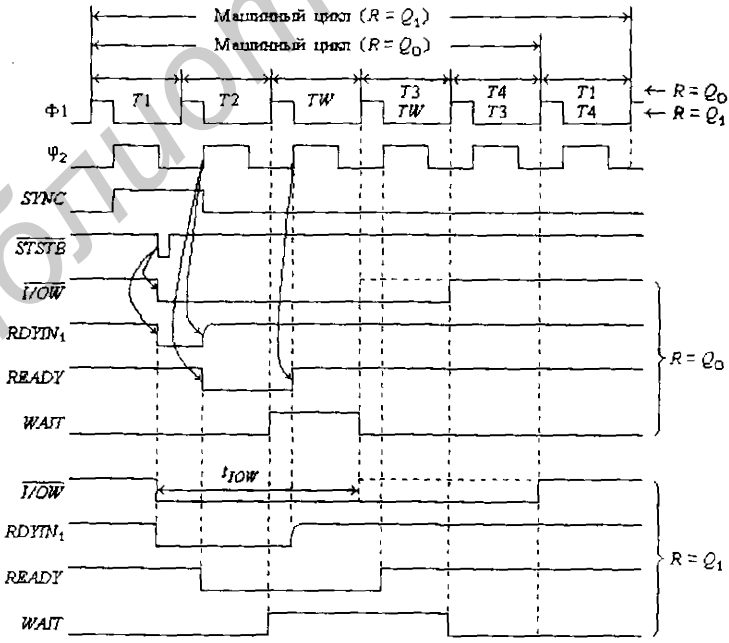


Рис. 2.9. Временны́е диаграммы режима вывода по готовности

Временные диаграммы, изображенные на рис. 2.8, поясняют работу схемы в режиме ввода данных по готовности при использовании МП 8080. Аналогичным образом схема работает и в режиме вывода данных по готовности (рис. 2.9).

Анализ значения сигнала *READY* в МП 8080 производится в такте T_2 каждого машинного цикла. Если МП обнаружит значение сигнала *READY* = 0, то он перейдет в состояние ожидания выдаст значение сигнала *WAIT* = 1. При выходе МП из состояния ожидания сигнал *WAIT* автоматически сбрасывается в 0. Это же справедливо и для МП 8085, только сигнал *WAIT* у него отсутствует.

Если в схеме, изображенной на рис. 2.7, вместо сигналов $\overline{I/OR}$ и $\overline{I/OW}$ подать сигналы $\overline{MEMR}$ и $\overline{MEMW}$, то ее можно будет использовать для введения тактов ожидания при чтении и записи данных в память. При этом, конечно, на адресный дешифратор 555ИЕ7 нужно подать другие разряды адреса.

2.3. Программный ввод-вывод с квитированием

Ввод данных без квитирования часто неприемлем, так как микропроцессору не известны моменты времени, в которые внешние устройства записывают сигналом *STB* = 1 данные в буферный регистр ввода (см. рис. 2.3). Осложнения возникают и при выводе данных, если внешнее устройство, подключенное к буферному регистру вывода, не в состоянии принимать данные с такой скоростью, с которой их может выдавать МП. В таких случаях используется *квитирование* (*acknowledgement*, *handshaking* — подтверждение приема) — передача устройством управляющих сигналов в ответ на принятое сообщение. Для синхронизации передачи данных между внешними устройствами и МП используются сигналы квитирования (флаги) *IBF* (*Input Buffer Full* — входной буфер заполнен) и $\overline{OBF}$ (*Output Buffer Full* — выходной буфер заполнен) — флаги могут быть как прямыми, так и инверсными.

На рис. 2.10, *а* показана структурная схема внешнего устройства, предназначенного для программного ввода и вывода с квитированием: 1533ИР22 — буферный регистр ввода данных, 1533ИР23 — буферный регистр вывода данных, 1533ТМ2 — два *D*-триггера регистра состояния флагов IBF_1 и $\overline{OBF}_1$, чтение которого МП производит через драйвер 1533АП5. Регистр состояния может содержать восемь триггеров для обслуживания программного ввода-вывода с квитированием четырех внешних устройств.

Значение сигнала $\overline{STB} = 0$ (*STB* — *Strobe* — строб) загружает байт данных в регистр ввода и устанавливает значение флага $IBF_1 = 1$, а значение сигнала $\overline{ACK} = 0$ (*ACK* — *Acknowledge* — квитировать) подтверждает прием байта данных внешним устройством из регистра вывода и устанавливает значение флага $\overline{OBF}_1 = 1$. Флаги IBF_1 и $\overline{OBF}_1$ поступают в МП программным способом (читаются по шине данных), а на внешнее устройство — аппаратно (по проводникам). Чтение флагов производится значением сигнала $\overline{OE}_2 = \overline{I/OR \cdot CS94} = 0$, чтение регистра ввода и сброс флага IBF_1 — значением сигнала $\overline{OE}_1 = \overline{I/OR \cdot CS80} = 0$, запись байта данных в регистр вывода и установка значения $\overline{OBF}_1 = 0$ — значением сигнала $\overline{H} = \overline{I/OW \cdot CS94} = \int$.

На рис. 2.10, *б*, *в* изображены блок-схемы алгоритмов программного ввода и вывода с квитированием: *port_F* — имя порта регистра состояния (регистра флагов *IBF* и $\overline{OBF}$), *port_1* — имя порта ввода, *port_2* — имя порта вывода. Флаги IBF_1 и $\overline{OBF}_1$ фиксируются в двух разрядах регистра состояния и перед каждой операцией ввода или вывода МП производит чтение этого регистра (команда IN *port_F*) для проверки сигналов квитирования *IBF* (при вводе — рис. 2.10, *б*) или $\overline{OBF}$ (при выводе — рис. 2.10, *в*).

При получении значения *IBF* = 0 при вводе МП будет читать регистр состояния до тех пор, пока внешнее устройство не выдаст данные в буферный регистр ввода и не установит флаг

IBF = 1 (рис. 2.10, *б*). Только обнаружив, что флаг IBF установлен в 1, МП производит ввод байта данных из буферного регистра ввода и сброс флага IBF. При получении значения разряда $\overline{OBF}$ = 0 при выводе МП будет читать регистр состояния до тех пор, пока внешнее устройство не примет байт данных из буферного регистра вывода и не установит значение флага $\overline{OBF}$ = 1 (рис. 2.10, *в*). Только обнаружив, что флаг $\overline{OBF}$ = 1, МП производит вывод следующего байта данных в буферный регистр вывода и установку флага $\overline{OBF}$ в 0. Из этого следует назначение флагов:

$$IBF_i = \begin{cases} 0 - \text{указание } I/O \text{ загрузить буферный регистр,} \\ 1 - \text{указание МП принять байт данных;} \end{cases}$$

$$\overline{OBF_i} = \begin{cases} 0 - \text{указание } I/O \text{ принять байт данных,} \\ 1 - \text{указание МП загрузить буферный регистр.} \end{cases}$$

Адресация регистров ввода, вывода и состояния приведена в табл. 2.2. Здесь произведена неполная дешифрация адресных сигналов — в дешифрации не участвуют разряды адреса A_1 и A_0, а значит, часть адресов теряется. Из табл. 2.2 следует, что при использованном способе адресации одним дешифратором можно адресовать как отдельные регистры, так и интерфейсные БИС типа приведенной на рис. 2.4, содержащие до четырех устройств ввода и вывода.

Временные диаграммы, изображенные на рис. 2.11, *а*, поясняют процесс ввода с квитированием, а на рис. 2.11, *б* — процесс вывода с квитированием (значения сигналов $\overline{I/OR}$ = 0 и $\overline{I/OW}$ = 0 помечены командами IN *port* и OUT *port*, которыми они порождаются).

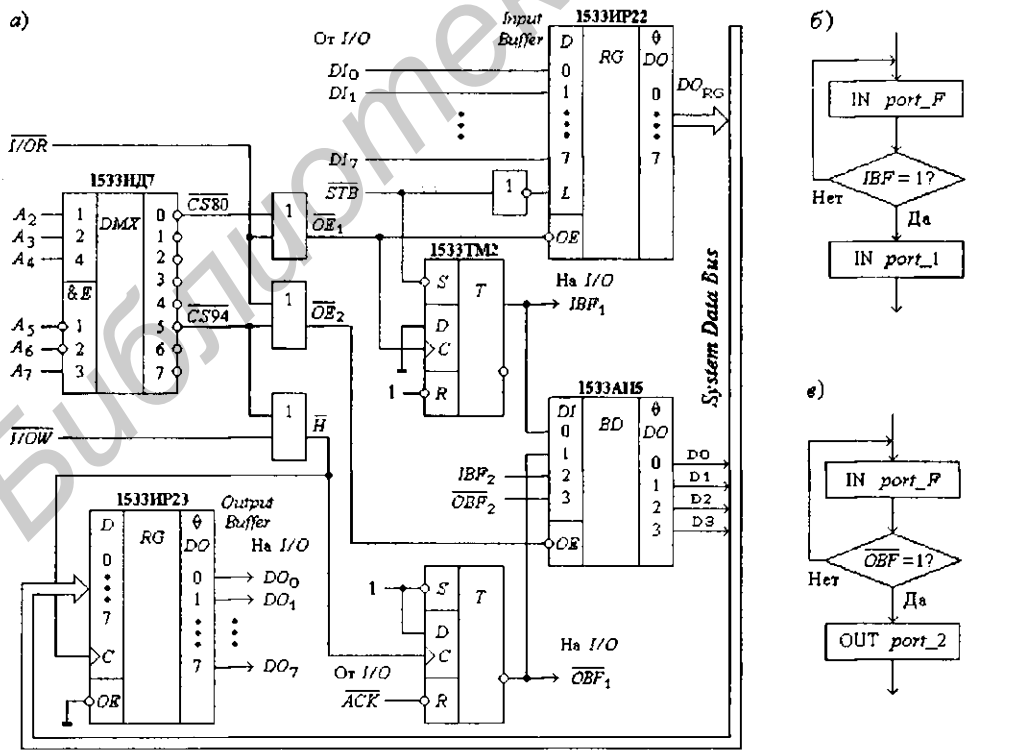


Рис. 2.10. Программный ввод-вывод с квитированием

Таблица 2.2. **Адресация *I/O***

Разряды адреса								Выход *DC*	*port*
7	6	5	4	3	2	1	0		
1	0	0	0	0	0	×	×	0	80 ÷ 83
1	0	0	0	0	1	×	×	1	84 ÷ 87
1	0	0	0	1	0	×	×	2	88 ÷ 8B
1	0	0	0	1	1	×	×	3	8C ÷ 8F
1	0	0	1	0	0	×	×	4	90 ÷ 93
1	0	0	1	0	1	×	×	5	94 ÷ 97
1	0	0	1	1	0	×	×	6	98 ÷ 9B
1	0	0	1	1	1	×	×	7	9C ÷ 9F

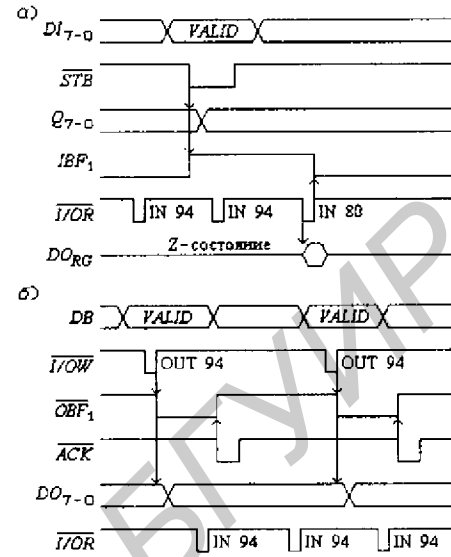


Рис. 2.11. Временные диаграммы ввода-вывода с квитированием

В программах ввод-вывод с квитированием обычно используется несколько раз. Поэтому его целесообразно оформлять в виде подпрограмм.

Задача 1. Из регистра ввода 1533ИР22 переслать байт данных в ячейку памяти *M*(23A7) и из ячейки памяти *M*(23A8) байт данных переслать в регистр вывода 1533ИР23. *Решение*:

```
        LXI   H, 23A7h  ; rp H ← 23A7 — адрес памяти
        CALL  Cin       ; Подпрограмма ввода Cin
        INX   H         ; rp H ← 23A8 — адрес памяти
        CALL  Cout      ; Подпрограмма вывода Cout
         ∴
Cin:    IN    94h       ; A ← SW = I/O(94) = ××××OBF2IBF2OBF1IBF1 — чтение флагов
        ANI   1         ; 1 = 0000 0001 — маска для выделения флага IBF1
        JZ    Cin       ; Переход, если IBF1 = 0
        IN    80h       ; A ← I/O(80) — ввод из регистра 1533ИР22
        MOV   M, A      ; M(rp H) ← A — запись в память
        RET
Cout:   IN    94h       ; A ← SW = I/O(94) = ××××OBF2IBF2OBF1IBF1 — чтение флагов
        ANI   2         ; 2 = 0000 0010 — маска для выделения флага OBF1
        JZ    Cout      ; Переход, если OBF1 = 0
        MOV   A, M      ; A ← M(rp H) — чтение памяти
        OUT   94h       ; I/O(94) ← M(rp H) — вывод в регистр 1533ИР23
        RET
```

Квитирование позволяет согласовать во времени работу асинхронно работающих МП и внешних устройств — быстродействие МП может быть и больше, и меньше быстродействия внешних устройств.

2.4. Ввод-вывод по прерыванию

При вводе-выводе по прерыванию внешние устройства подают на вход *INT* микропроцессора 8080 сигнал запроса прерывания *IRQ* (*Interrupt Request*). Значение сигнала *INT* МП анализирует в последнем такте последнего машинного цикла каждой выполняемой команды. При обнаружении значения *INT* = 1 при состоянии внутреннего триггера *INTE* = 1 микропроцессор приостанавливает выполнение текущей программы и переходит к выполнению подпрограммы обработки прерывания (ППОП), закончив которую возвращается к выполнению прерванной программы. Для реализации ввода-вывода по прерыванию требуются дополнительные аппаратные средства. Типичным примером ввода по прерыванию является ввод кода нажатой клавиши клавиатуры (частота нажатия клавиш невелика и потери времени при программном вводе с квитированием были бы необоснованно большими). Выполняются ППОП на фоне основной программы, что обеспечивает наиболее рациональное использование процессорного времени.

Одновекторные системы прерываний. На рис. 2.12 изображены структурные схемы МП-систем 8080 с обслуживанием одновекторных прерываний. Вектор прерывания — это адрес подпрограммы обработки прерывания или число, по которому этот адрес автоматически вычисляется микропроцессором. Подключение выхода $\overline{INTA}$ системного контроллера 8238 к источнику напряжения питания +12 В переводит его в режим выдачи в МП машинного кода FF*h* команды RST 7 в ответ на запрос прерывания *IRQ* = 1 по входу *INT* [10]. При приеме запроса прерывания (при условии, что командой EI предварительно было установлено значение сигнала *INTE* = 1) МП 8080 выдает в системный контроллер слово состояния $SW_{8,10} = 0010{\times}011$ (см. табл. 1.15), младший разряд которого $D_0 = 1$ блокирует системную шину данных, т. е. выборка команды RST 7 производится не из памяти команд программы, а из самого системного контроллера 8238. Формат команд RST *n* показан на рис. 2.13 — адрес вызова ППОП вычисляется автоматически и определяется числом $8 \times n = (8 \times 7)_{10} = 56_{10} = 38_{16}$.

На рис. 2.12, *а* изображена структурная схема МП-системы с обслуживанием по прерыванию одного внешнего устройства *I/O*–0. Прием запросов прерывания *IRQ* = 1 возможен только после установки командой EI значения сигнала *INTE* = 1. Сразу после приема запроса прерывания МП сбрасывает сигнал *INTE* в 0, что используется для сброса в 0 сигнала *IRQ*. Таким образом, ввод-вывод по прерыванию определяется схемой:

$$IRQ\,\lrcorner \Rightarrow INTE\,\urcorner \Rightarrow IRQ\,\urcorner \quad \text{(аппаратное квитирование ввода-вывода)},$$
$$\Rightarrow SW_{8,10} \Rightarrow \text{RST 7} \Rightarrow \text{ППОП по адресу } addr = 0038h$$

(команда RST 7 поступает в МП по локальной шине данных из системного контроллера 8238).

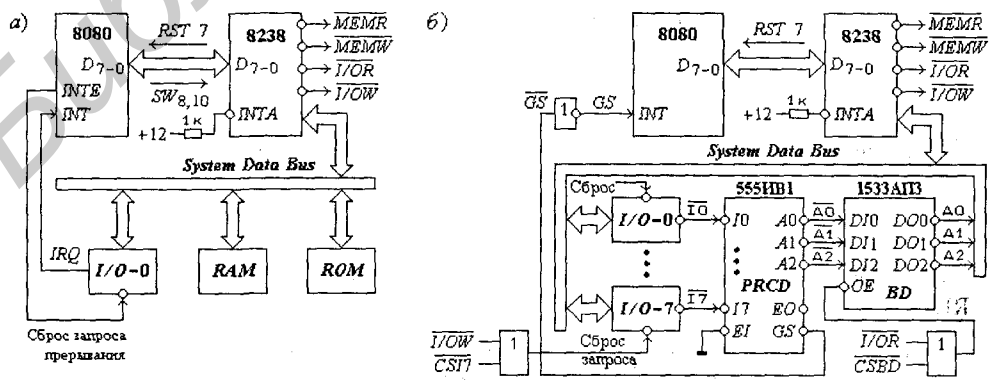


Рис. 2.12. Одновекторные системы прерываний

D7	D6	D5	D4	D3	D2	D1	D0
1	1	*n*	*n*	*n*	1	1	1

Рис. 2.13. Машинные коды команд RST *n*

Содержимое всех внутренних регистров МП, которые будут использоваться подпрограммой обработки прерывания, необходимо сохранить в оперативной памяти, а в самом конце ППОП следует восстановить содержимое этих регистров. Для этих целей наиболее целесообразно использовать команды PUSH *rp* и POP *rp*, адресующие стек. Так как ввод и вывод данных при обработке прерываний от внешних устройств производится командами IN *port* и OUT *port* с использованием аккумулятора, то всегда следует сохранять и восстанавливать, по крайней мере, регистр *PSW* — аккумулятор и регистр признаков. Типовое оформление подпрограммы обработки прерывания при сохранении и восстановлении содержимого всех внутренних регистров МП приведено ниже:

```
; Начало подпрограммы обработки прерывания RST 7 с адресом вызова 7 × 8 = 56d = 0038h
      PUSH   PSW   ; Сохранение в стеке состояния прерванной (основной) программы
      PUSH   H
      PUSH   D
      PUSH   B
        ∴          ; Собственно подпрограмма обработки прерывания
      POP    B     ; Восстановление состояния прерванной программы
      POP    D
      POP    H
      POP    PSW
      EI           ; Разрешение прерываний (INTE ← 1)
      RET          ; Возврат в прерванную программу
```

На рис. 2.12, *б* показана структурная схема МП-системы с одновекторным обслуживанием по прерыванию восьми внешних устройств *I/O–m* ($m = 0 \ldots 7$) с восемью уровнями приоритета. При выдаче активного уровня сигнала запроса прерывания $\overline{I_m} = 0$ хотя бы одним внешним устройством приоритетный шифратор *PRCD* вырабатывает активный уровень группового сигнала запроса прерывания $\overline{GS} = 0$ и выдает двоичный код номера $m = A_2A_1A_0$, внешнего устройства, имеющего наибольший приоритет из всех внешних устройств, одновременно запросивших обслуживание. Чем больше номер m входа I_m у приоритетного шифратора 555ИВ1, тем выше приоритет этого входа. Детальное описание ИС 555ИВ1, а также других приоритетных шифраторов, можно найти в § 6.6 книги [5].

Пример 1 (подпрограмма обслуживания прерывания, соответствующая рис. 2.12, *б*):

```
; Начало подпрограммы обработки прерывания RST 7 с адресом вызова 7 × 8 = 56d = 0038h
      PUSH   PSW   ; Сохранение в стеке состояния прерванной программы
      PUSH   H
      PUSH   D
      PUSH   B
      IN     CSbd  ; A ← ×××× ×A2A1A0 — чтение номера I/O, запросившего обслуживание
      ANI    7     ; A ← 0000 0A2A1A0 — выделение кода A2A1A0 = m
      CPI    1     ; Сравнение кода A2A1A0 с числом 1
      JZ     IO_1  ; Переход на обслуживание I/O–1
      JC     IO_0  ; Переход на обслуживание I/O–0
      CPI    3     ; Сравнение кода A2A1A0 с числом 3
```

```
        JZ    IO_3        ; Переход на обслуживание I/O–3
        JC    IO_2        ; Переход на обслуживание I/O–2
        CPI   5           ; Сравнение кода A2A1A0 с числом 5
        JZ    IO_5        ; Переход на обслуживание I/O–5
        JC    IO_4        ; Переход на обслуживание I/O–4
        CPI   6           ; Сравнение кода A2A1A0 с числом 6
        JZ    IO_6        ; Переход на обслуживание I/O–6
          ∴               ; Начало подпрограммы обслуживания I/O–7
        OUT   Reset_IR7   ; Сброс запроса прерывания от I/O–7 (сигнал I/OW & CSI7 = 0)
        JMP   ENDRST      ; Конец подпрограммы обслуживания I/O–7
IO_0:   ∴                 ; Начало подпрограммы обслуживания I/O–0
        OUT   Reset_IR0   ; Сброс запроса прерывания от I/O–0
        JMP   ENDRST      ; Конец подпрограммы обслуживания I/O–0
IO_1:     ∴               ; и т. д.
          ∴
IO_6      ∴
ENDRST: POP   B           ; Восстановление состояния прерванной программы
        POP   D
        POP   H
        POP   PSW
        EI                ; Разрешение прерываний (INTE ← 1)
        RET               ; Возврат в прерванную программу
```

Прервать эту подпрограмму до ее завершения другие внешние устройства не в состоянии, так как команда разрешения прерываний EI находится в самом конце подпрограммы.

Для определения адресов переходов на программы обслуживания *I/O–m* удобно использовать косвенно-регистровый метод адресации переходов (см. § 1.7). *Пример* 2:

```
; Начало подпрограммы обработки прерывания RST 7 с адресом вызова 7 × 8 = 56d = 38h
        PUSH  PSW         ; Сохранение в стеке состояния прерванной программы
        PUSH  H
        PUSH  D
        PUSH  B
        IN    CSbd        ; A ← ××××A2A1A0 — чтение номера I/O–m, пославшего запрос
        ANI   7           ; A ← 0000 0A2A1A0 — выделение кода A2A1A0 = m
        LXI   H, Addr_t   ; HL ← Addr_t (начальный адрес таблицы адресов
        MVI   D, 0        ;                передачи управления)
        ADD   A           ; A ← m × 2
        MOV   E, A        ; DE = m × 2
        DAD   D           ; HL ← Addr_t + (m × 2)
        MOV   A, M
        INX   H
        MOV   H, M
        MOV   L, A        ; HL = Addr_m
        PCHL              ; PC ← Addr_m (переход на обслуживание I/O–m, где m = A2A1A0)
          ∴
; Addr_t       Addr_0     ; Таблица адресов переходов: Addr_m — двухбайтовый адрес
; Addr_t + 2   Addr_1     ;     передачи управления подпрограмме обслуживания
; Addr_t + 4   Addr_2     ;     внешнего устройства I/O–m (m = 0 … 7)
```

```
; Addr_t + 6      Addr_3
; Addr_t + 8      Addr_4   ; Передача управления подпрограмме обслуживания
; Addr_t + A      Addr_5   ;      внешнего устройства I/O–m осуществляется с помощью
; Addr_t + C      Addr_6   ;      косвенно-регистрового метода адресации переходов
; Addr_t + E      Addr_7
```

В этой подпрограмме используется таблица для восьми адресов переходов *Addr_m* и вычисление конкретного адреса таблицы для загрузки из нее адреса перехода в программный счетчик *PC*. Если при отладке программы адреса переходов изменятся или потребуется перенести подпрограммы обслуживания прерываний в другое место памяти, то достаточно будет изменить только таблицу адресов.

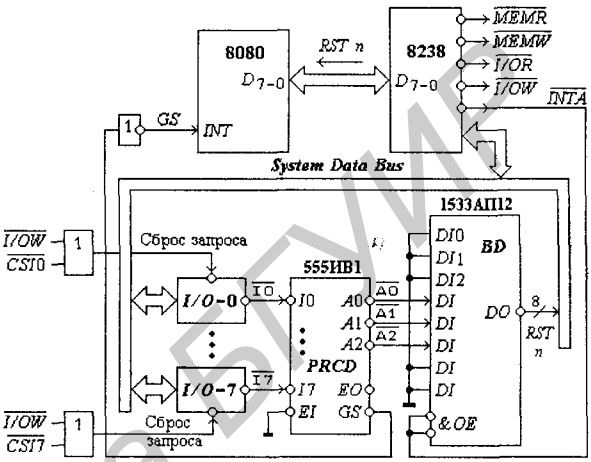


Рис. 2.14. Восьмивекторная система прерываний

Восьмивекторные системы прерываний. На рис. 2.14 показана структурная схема МП-системы с восьмивекторным обслуживанием по прерыванию внешних устройств *I/O–m* ($m = 0 \dots 7$). Отличие этой системы прерываний от предыдущей заключается в том, что здесь производится чтение команд RST *n* из элементарного контроллера прерываний, выполненного на ИС 555ИВ1 и 1533АП12. Машинный код 11×××111 команд RST *n* формируется шинным драйвером 1533АП12. В ответ на групповой запрос прерывания *GS* = 1 (при *INTE* = 1) МП выдает в системный контроллер 8238 слово состояния $SW_{8,10} = 0010{\times}011$ (см. табл. 1.15), по которому вырабатывается сигнал $\overline{INTA} = \sqcup\!\!\sqcap$, производящий чтение шинного драйвера *BD* (безадресное чтение команд из внешнего устройства — контроллера прерываний).

Таким образом, ввод-вывод по прерыванию определяется схемой:

$$\overline{I}_m \sqcup \Rightarrow GS \sqcap \Rightarrow INTE \sqcup \text{ (аппаратное квитирование ввода-вывода)}$$
$$\Rightarrow SW_{8,10} \Rightarrow \overline{INTA} = \sqcup\!\!\sqcap \Rightarrow \text{RST } n \Rightarrow \text{ППОП по адресу } addr = 8 \times n$$

(команда RST *n* поступает в МП по системной шине данных через приемопередатчик системного контроллера 8238). В МП-системах с восьмивекторным обслуживанием внешних устройств по прерыванию не нужно программным способом определять внешнее устройство, запросившее обслуживание, а значит, уменьшается размер ППОП и увеличивается производительность системы.

Пример 3 (подпрограммы обслуживания прерываний, соответствующие рис. 2.14):

```
0000   JMP   Main   ; Точка входа по запросу прерывания RST 0 и по аппаратному сбросу
         ∴          ;      RESET = 1 (при включении питания или нажатии кнопки сброса)
0008   PUSH  PSW    ; Начало подпрограммы обработки прерывания RST 1
       PUSH  H      ; Сохранение в стеке состояния прерванной (основной) программы
       PUSH  D      ;      (абсолютные адреса 0000, 0008, 0010 и 0018 добавлены только
       PUSH  B      ;      для удобства анализа программы — в транслируемой программе
       JMP   IO_1   ;      они должны отсутствовать)
       NOP
0010   PUSH  PSW    ; Начало подпрограммы обработки прерывания RST 2
```

```
        PUSH  H
        PUSH  D
        OUT   Reset_2  ; Сброс запроса прерывания от I/O–2
        JMP   IO_2
0018    PUSH  PSW      ; Начало подпрограммы обработки прерывания RST 3
          ∴
        JMP   IO_3
          ∴            ; и т. д. (для внешних устройств I/O–4 ... I/O–7)
IO_1:     ∴            ; Продолжение подпрограммы обработки прерывания RST 1
        OUT   Reset_1  ; Сброс запроса прерывания от I/O–1
        POP   B        ; Восстановление состояния прерванной программы
        POP   D
        POP   H
        POP   PSW
        EI             ; Разрешение прерываний (INTE ← 1)
        RET            ; Возврат в прерванную программу
IO_2:   PUSH  B        ; Продолжение подпрограммы обработки прерывания RST 2
          ∴            ;
        POP   B        ; Восстановление состояния прерванной программы
        POP   D
        POP   H
        POP   PSW
        EI             ; Разрешение прерываний (INTE ← 1)
        RET            ; Возврат в прерванную программу
IO_3:     ∴            ; Продолжение подпрограммы обработки прерывания RST 3
        RET
          ∴            ; и т. д. (для внешних устройств I/O–4 ... I/O–7)
Main:     ∴            ; Начало основной программы: тестирование и инициализация
          ∴            ;                          устройств МП-системы
```

Команды сброса запросов прерывания от внешних устройств OUT *Reset_m* могут располагаться в любом месте подпрограмм, но не после команды EI, а команда разрешения прерываний EI — только в конце ППОП. В противном случае выполнение ППОП с большим приоритетом будут прерывать запросы прерываний, имеющие меньший приоритет. Катастрофически же опасен в этом случае повторный запрос прерывания обслуживаемого внешнего устройства до выполнения команды RET ППОП, вызванной предыдущим запросом — будет нарушена работа стека. Если есть гарантия, что во время выполнения ППОП не будет повторных запросов (частота запросов невелика), то прерывания можно разрешать и до завершения ППОП. Если команда EI расположена в конце ППОП, то приоритетность внешних устройств срабатывает только лишь при выборе их для обслуживания. Все это справедливо и для МП 8085, имеющего пять входов запросов прерываний с пятью уровнями приоритетов.

Рассмотренные выше системы прерываний являются одноуровневыми — выполнение текущей ППОП не могут прервать другие внешние устройства. В многоуровневых системах прерываний любое внешнее устройство, имеющее больший приоритет, может прервать выполнение ППОП внешнего устройства, имеющего меньший приоритет, а после выполнения новой ППОП будет автоматически продолжено выполнение прерванной подпрограммы. Понятно, что в этом случае команда EI в допускающей прерывание подпрограмме должна находиться не в самом ее конце. Если система прерываний имеет m уровней прерываний, то одновременно в обслуживании могут находиться до m ППОП в разной степени завершенности. Последова-

тельность выполнения ППОП аналогична последовательности выполнения обычных команд CALL *addr*, вызывающих подпрограммы, имеющие *m* уровней вложенности. Для построения многоуровневой системы прерываний требуются дополнительные аппаратные затраты.

Система прерываний МП 8085. Этот МП имеет 5 входов запросов прерываний с пятью уровнями приоритетов (табл. 2.3) и четырехвекторную систему прерываний. Вход *TRAP* немаскируемого запроса прерываний имеет наибольший приоритет, а вход *INTR* (аналог входа *INT* в МП 8080) — наименьший приоритет. На рис. 2.15 изображена внутренняя схема МП, обеспечивающая прием запросов прерываний *TRAP* по положительному фронту с последующим удержанием высокого уровня сигнала *TRAP* до приема запроса микропроцессором.

На рис. 2.16 изображена МП-система с восьмиуровневой системой приоритетных прерываний, обеспечиваемой программируемым контроллером прерываний 8259 (580ВН59). Вход IR_0 запроса прерываний имеет наибольший приоритет, а вход IR_7 — наименьший приоритет.

Ввод-вывод по прерыванию определяется схемой:

$$IR_m \uparrow \Rightarrow INT \uparrow \begin{array}{l} \Rightarrow INTE \downarrow (8080) \,/\, IE \downarrow (8085) \quad \text{(аппаратное квитирование ввода-вывода)}, \\ \Rightarrow \overline{INTA} = \sqcup\!\sqcap\sqcup\!\sqcap\sqcup\!\sqcap \Rightarrow \text{CALL } Addr\_m \Rightarrow \text{ППОП по адресу } Addr\_m \end{array}$$

(команда CALL *Addr_m* поступает в МП по системной шине данных из контроллера прерываний 8259). Чтение трехбайтовых команд CALL *Addr_m* из контроллера прерываний 8259 безадресное — выполняется сигналом $\overline{INTA}$, который больше нигде не используется. В контроллере прерываний запросы прерываний с меньшим уровнем приоритета и повторные запросы блокируются, поэтому команду EI можно помещать в самом начале ППОП. Прервать выполнение ППОП могут лишь запросы прерываний с большим уровнем приоритета. После выполнения новой ППОП будет автоматически продолжено выполнение прерванной подпрограммы. Одновременно в обслуживании могут находиться до восьми ППОП в разной степени завершенности.

Таблица 2.3. **Система прерываний МП 8085**

Сигнал	Приоритет	Адрес	Тип запроса (чувствительность входа)
TRAP	1	24*h*	Положительный фронт и высокий уровень до приема запроса
RST 7.5	2	3C*h*	Положительный фронт (запрос запоминается в триггере)
RST 6.5	3	34*h*	Высокий уровень до приема запроса
RST 5.5	4	2C*h*	Высокий уровень до приема запроса
INTR	5	—	Высокий уровень до приема запроса

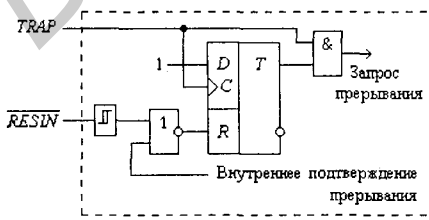


Рис. 2.15. Потенциально-импульсная схема запроса прерываний

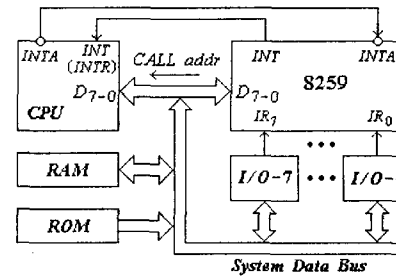


Рис. 2.16. Восьмиуровневая система прерываний

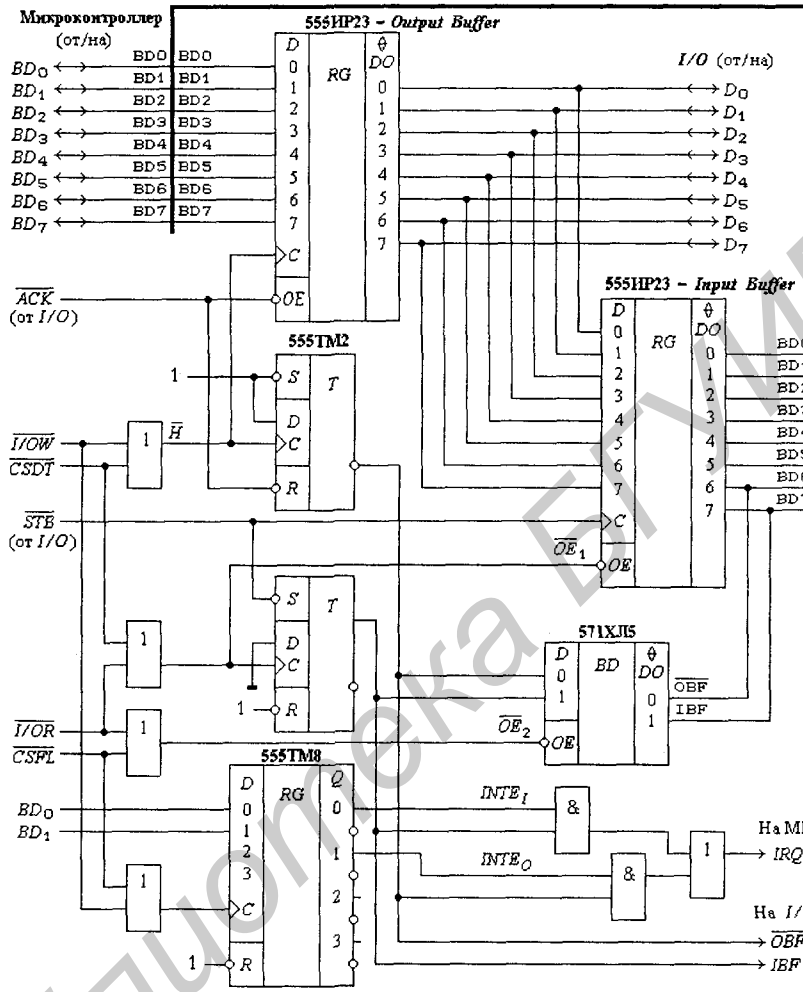


Рис. 2.17. Принципиальная схема приемопередатчика

Программный ввод-вывод с квитированием по прерыванию. Для ввода и вывода *N*-байтовых блоков данных можно использовать запрос прерывания, указывающий МП о готовности внешнего устройства передать или принять блок данных, и программный ввод-вывод с квитированием для согласования разной скорости работы МП и внешних устройств. Принципиальная схема приемопередатчика для обмена данными по такому методу между МП и внешним устройством, имеющим двунаправленную шину данных, изображена на рис. 2.17. Основная часть этого приемопередатчика соответствует схеме, изображенной на рис. 2.10. Раздельное маскирование запросов прерывания *IRQ* для ввода и вывода выполнено на регистре памяти 555ТМ8 и трех логических элементах. Сигнал запроса прерывания

$$IRQ = INTE_I \cdot IBF \vee INTE_O \cdot \overline{OBF} = 1,$$

если входной буфер заполнен ($IBF = 1$) или выходной буфер пустой ($\overline{OBF} = 1$) при условии, что прерывания разрешены ($INTE_I = 1$, $INTE_O = 1$). Разрешением и запретом прерываний ($INTE_I = 0$,

$INTE_O$ = 0) управляет МП. Запрос прерывания *IRQ* можно подать, например, на вход RST 7.5 МП 8085.

Задача 1. Написать подпрограмму ввода и вывода 8-байтовых блоков данных из микроконтроллера во внешнее устройство по его запросу прерывания. *Решение*:

```
        PUSH  PSW      ; Начало подпрограммы обслуживания прерываний
        PUSH  H        ; Сохранение в стеке состояния прерванной (основной) программы
        PUSH  B
        IN    CSFL     ; A ← I/O(CSFL) = IBFOBF×× ×××× — чтение флагов (CSFL = 0 на
        ANI   80h      ; 80h = 1000 0000 — маска для выделения флага IBF      рис. 2.17)
        JNZ   LIN1     ; Переход на выполнение ввода
        LXI   H, Aout  ; Aout — адрес первой ячейки памяти данных для вывода
        MVI   C, 8     ; C ← 8 — число байт в блоке данных
LOUT:   IN    CSFL     ; A ← I/O(CSFL) — чтение флагов (CSFL = 0 на рис. 2.17)
        ANI   40h      ; 40h = 0100 0000 — маска для выделения флага OBF
        JZ    LOUT     ; Переход, если OBF = 0 — выходной буфер занят
        MOV   A, M
        OUT   CSDT     ; I/O(CSDT) ← A — вывод байта данных в буферный регистр
        INX   H        ;                    Output Buffer  (CSDT = 0 на рис. 2.17)
        DCR   C
        JNZ   LOUT
        JZ    LEND
LIN1:   LXI   H, Ain   ; Ain — адрес первой ячейки памяти данных для ввода
        MVI   C, 8     ; C ← 8 — число байт в блоке данных
LIN2:   IN    CSFL     ; A ← I/O(CSFL) — чтение флагов (CSFL = 0 на рис. 2.17)
        ANI   80h      ; 80h = 1000 0000 — маска для флага IBF
        JZ    LIN2     ; Переход, если IBF = 0 — входной буфер пустой
        IN    CSDT     ; A ← I/O(CSDT) — ввод байта данных из буферного регистра
        MOV   M, A     ;                     Input Buffer  (CSDT = 0 на рис. 2.17)
        INX   H
        DCR   C
        JNZ   LIN2
LEND:   POP   B        ; Восстановление состояния прерванной программы
        POP   H
        POP   PSW
        EI             ; Разрешение прерываний
        RET            ; Конец подпрограммы обработки прерывания
```

В этой подпрограмме символические имена *CSFL* и *CSDT* адресов портов соответствуют сигналам $\overline{CSFL}$ и $\overline{CSDT}$ на рис. 2.17. Рассмотренный приемопередатчик может быть использован и для обмена данными между двумя микроконтроллерами, или микроконтроллером и персональным компьютером. В этом случае флаги *IBF* и $\overline{OBF}$ на второй микроконтроллер или персональный компьютер подаются по шине данных.

Многорежимный буферный регистр 8212. Для увеличения функциональных возможностей и гибкости использования регистров памяти в них вводится дополнительная управляющая логика. Регистры памяти 8212 фирмы *Intel* (589ИР12) и 74*F*412 (рис. 2.18, *а*), построенные на основе *D–L–R*-триггеров с приоритетом входа *L* [5] называются *многорежимными буферными регистрами* (*Multi Mode Buffered Latches*). Данные ИС предназначены для использования в МП-системах для выполнения обмена данными между МП и периферийными устройствами.

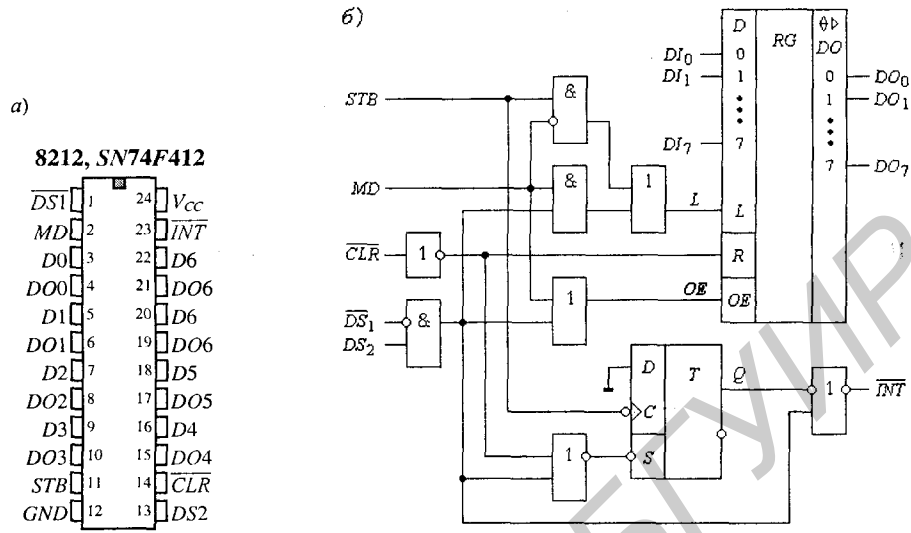


Рис. 2.18. Многорежимный буферный регистр 8212

Структурная схема ИС 8212 изображена на рис. 2.18, *б* и состоит из 8-разрядного регистра памяти *RG*, триггера запроса прерывания *T* и схемы управления режимами работы. Для управления регистром памяти используются сигналы: $\overline{DS_1}$, DS_2 (*Device Select*) — селекция регистра, *STB* — строб, *MD* (*Mode*) — режим, $\overline{CLR}$ (*Clear*) — сброс, $\overline{INT}$ (*Interrupt Request*) — запрос прерывания.

Функционирование регистров памяти определяется функцией переходов *D–L–R*-триггера с приоритетом входа *L* [5]:

$$Q_r^+ = D_r \cdot L \vee Q_r \cdot \overline{L \vee CLR},\ L = STB \cdot \overline{MD} \vee DS_1DS_2 \cdot MD,\ r = 0, 1, \ldots, 7,$$

где Q_r — выходные сигналы триггеров, *L* — мультиплексная функция с адресным сигналом *MD*, позволяющая организовать загрузку данных от внешнего устройства сигналом *STB* при $MD = 0$ (ввод данных) или от микропроцессора сигналом $DS_1DS_2 = I/OW \cdot CS$ при $MD = 1$ (вывод данных). Управление выходами регистра DO_r осуществляется сигналом $OE = MD \vee DS_1DS_2$ в соответствии с соотношением

$$DO_r = \begin{cases} Q_r \text{ при } OE = 1, \\ Z \text{ - состояние при } OE = 0. \end{cases}$$

Сигнал запроса прерывания описывается функциями:

$$\overline{INT} = Q \cdot \overline{DS_1DS_2},\ Q^+ = S \vee Q \cdot \overline{dSTB} = DS_1DS_2 \vee CLR \vee Q \cdot \overline{dSTB}.$$

При использовании ИС 8212 в качестве буферного регистра ввода данных по прерыванию следует положить $MD \equiv 0$, $D_r = DI_r$ — разряды байта данных от внешнего устройства, $DO_r = DB_r$ — разряды байта данных, поступающие в МП по системной шине данных, $\overline{DS_1} = \overline{I/OR}$ — сигнал чтения буферного регистра ввода, $DS_2 = CS$ — сигнал с дешифратора адреса буферного регистра ввода (рис. 2.19, *а*). Сигнал сброса $\overline{CLR}$ можно не использовать, т. е. можно положить $\overline{CLR} \equiv 1$. На основании вышеприведенных соотношений буферный регистр ввода описывается функциями:

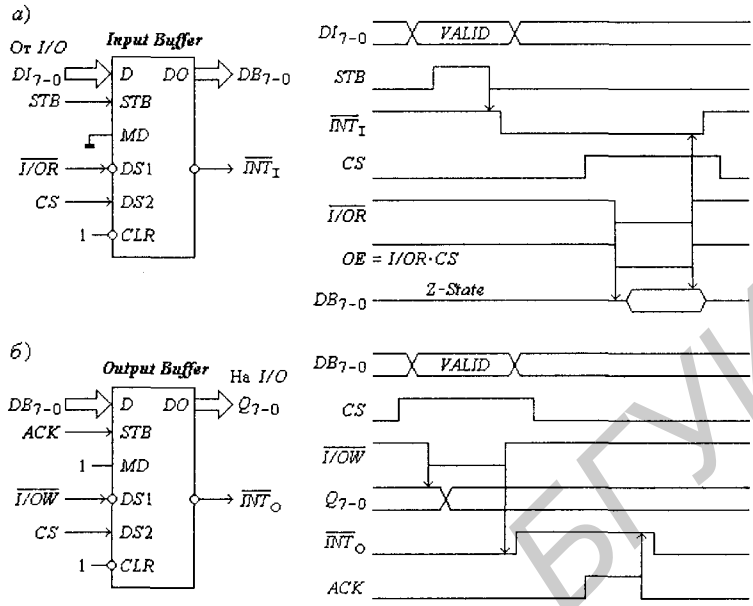


Рис. 2.19. Буферные регистры ввода и вывода

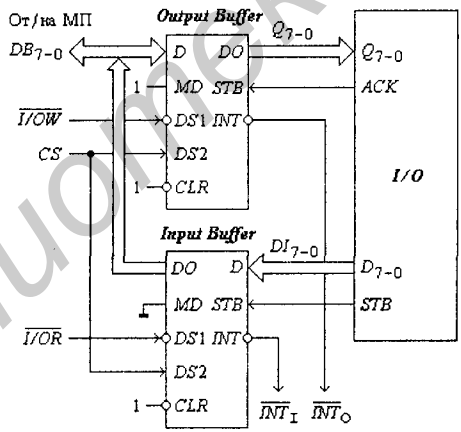


Рис. 2.20. Буферный регистр ввода-вывода

$$Q_r^+ = DI_r \cdot STB \vee Q_r \cdot \overline{STB},\ r = 0, 1, \ldots, 7;\ \overline{INT_I} = Q \cdot \overline{I/OR \cdot CS},\ Q^+ = I/OR \cdot CS \vee Q \cdot \overline{dSTB},$$

$$DB_r = \begin{cases} Q_r \text{ при } I/OR \cdot CS = 1, \\ Z\text{ - состояние при } I/OR \cdot CS = 0, \end{cases}$$

где $\overline{INT_I}$ — сигнал запроса ввода данных.

При использовании ИС 8212 в качестве буферного регистра вывода данных по прерыванию следует положить $MD \equiv 1$, $D_r = DB_r$ — разряды байта данных МП, поступающие на сис-

темную шину данных, DO_r — разряды байта данных для внешнего устройства, $\overline{DS_1} = \overline{I/OW}$ — сигнал записи в буферный регистр вывода, $DS_2 = CS$ — сигнал с дешифратора адреса буферного регистра вывода (рис. 2.19, *б*). Сигнал сброса $\overline{CLR}$ можно не использовать, т. е. можно положить $\overline{CLR} \equiv 1$. На этом основании буферный регистр вывода описывается функциями:

$$Q_r^+ = DB_r \cdot I/OW \cdot CS \vee Q_r \cdot \overline{I/OW \cdot CS},\ r = 0, 1, \ldots, 7;$$

$$\overline{INT_O} = Q \cdot \overline{I/OW \cdot CS},\ Q^+ = I/OW \cdot CS \vee Q \cdot \overline{dACK},\ DO_r = Q_r,$$

где $\overline{INT_O}$ — сигнал запроса вывода данных.

На рис. 2.20 изображена структурная схема устройства ввода-вывода (*I/O*) с двумя буферными регистрами 8212. Неактивный уровень (1) сигналов запроса прерывания $\overline{INT_I}$ и $\overline{INT_O}$ устанавливается при чтении и записи данных командами IN *port* и OUT *port*, генерирующими значения сигналов $\overline{I/OR} = 0$, $\overline{I/OW} = 0$ и $CS = 1$.

2.5. Ввод-вывод по прямому доступу к памяти

При использовании программного ввода-вывода без квитирования затрачивается десять тактов на выполнение команд IN *port* и OUT *port*, причем данные либо принимаются, либо выдаются аккумулятором МП. Ввод-вывод с квитированием требует еще больших затрат времени. Ввод-вывод по прерыванию осуществляется с помощью вызываемой подпрограммы обработки прерывания, в которой собственно ввод и вывод реализуются командами IN *port* и OUT *port*, т. е. программным способом. При использовании микроЭВМ в радиотехнических системах требуется, как правило, работа в реальном масштабе времени (скорость поступления вводимой информации определяется радиосигналами, поступающими на приемное устройство). В этих условиях может оказаться, что программный ввод-вывод даже без квитирования не способен обеспечить ввод последовательно поступающих байтов данных, тем более что требуется еще выполнение дополнительных команд для пересылки данных из аккумулятора в память.

Значительно ускорить пересылку данных позволяет метод ввода-вывода по прямому доступу к памяти (*Direct Memory Access — DMA*), реализуемый с помощью специальных контроллеров *DMA* (*DMAC*), например, 8257 (580ВТ57). На один цикл *DMA* (пересылка одного байта данных) затрачивается всего четыре такта, причем данные записываются непосредственно в память или читаются из памяти с автоматическим инкрементом адреса. Ввод-вывод по прямому доступу к памяти целесообразно использовать также при пересылках больших массивов данных между системной и внешней памятью, например, при использовании в микроЭВМ в качестве внешней памяти накопителей на гибких магнитных дисках (НГМД) или НМД типа винчестер (на жестких дисках).

Принцип работы *DMAC*. Структурная схема МП-системы с вводом-выводом по прямому доступу к памяти, изображенная на рис. 2.21, поясняет прямые пересылки между системной и внешней памятью (*I/O*), реализованной на таких же БИС, что и системная память. Здесь предполагается, что все *DMA*-операции производятся по командам, поступающим от МП (без запроса *DMA* от внешней памяти).

Любой контроллер *DMA* работает в двух режимах — пассивном и активном:

в *пассивном режиме DMAC* является для МП обычным внешним устройством (системными шинами управляет МП, а шины контроллера *DMA* находятся в *Z*-состоянии);

в *активном режиме DMAC* является специализированным процессором ввода-вывода (системными шинами управляет *DMAC*, а шины МП на это время переводятся в *Z*-состояние).

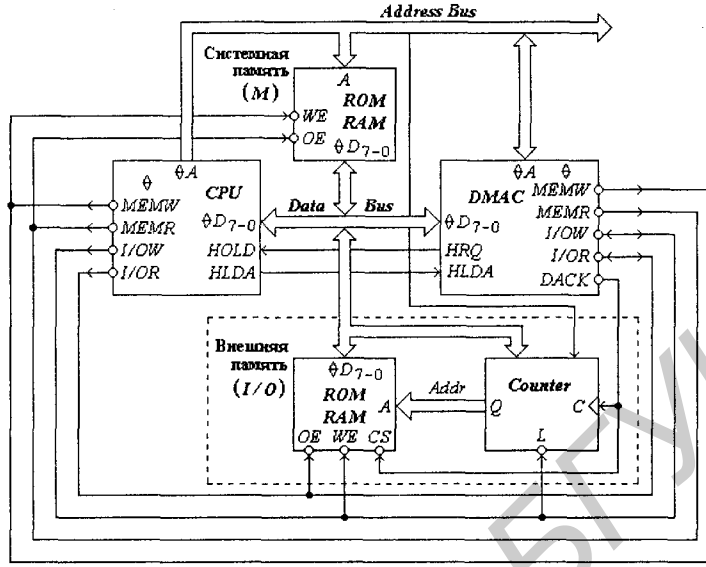


Рис. 2.21. Структурная схема МП-системы с вводом-выводом по прямому доступу к памяти

В активном режиме *DMAC* выдает активные уровни пар сигналов $\overline{MEMR} = 0$ и $\overline{I/OW} = 0$ при пересылке каждого байта данных типа $I/O \leftarrow M$ и $\overline{MEMW} = 0$ и $\overline{I/OR} = 0$ при пересылке типа $M \leftarrow I/O$.

В пассивном режиме *DMAC* производится его инициализация — запись во внутренние регистры начального адреса системной памяти *Abeg* и объема ΔA пересылаемого массива данных с указанием типа *DMA*-пересылок (направления передачи данных). В адресный счетчик *Counter* внешней памяти МП также загружает некоторый начальный адрес. Далее МП подает в *DMAC* специальную команду, которая вызывает последовательность действий:

$$HRQ \uparrow \Rightarrow HLDA \uparrow \text{ (МП перевел свои шины в Z-состояние)} \Rightarrow$$

$$\Rightarrow \overline{DACK} \sqcap\!\sqcup\sqcup\sqcup \ldots \sqcup\sqcup \text{ (до конца пересылок)} \Rightarrow HRQ \downarrow \Rightarrow HLDA \downarrow .$$

Если использовать 40-разрядный адресный счетчик *Counter*, то объем соответствующей внешней памяти (электронного диска) будет равен 1 Тбайт (1 099 511 627 776 × 8 бит).

Принципиальная схема электронного диска. На рис. 2.22 изображены временны́е диаграммы для случая пересылки двух байт данных, поясняющие работу электронного диска, принципиальная схема которого приведена на рис. 2.23. Контроллер *DMA* выполнен на основе БИС 8257. Длительность одного цикла *DMA* при значении сигнала $READY \equiv 1$ равна $4 \cdot T_{CLK}$, скорость пересылки данных — 500 Кбайт/с при частоте тактового сигнала $CLK = \varphi_2$, равной 2 МГц.

Для управления электронным диском использован канал 0 при значении сигнала $DRQ_0 \equiv 1$ (циклы *DMA* выполняются по инициативе МП, выдающего в *DMAC* определенное слово управления). В пассивном режиме *DMAC* микропроцессор командами OUT 90*h* и OUT 0A0*h* записывает в счетчики 555ИЕ7 начальный 16-разрядный адрес внешней памяти, которая используется для обмена данными с системной памятью — для этого специально следует использовать синхронные счетчики с асинхронной потенциальной загрузкой данных [5].

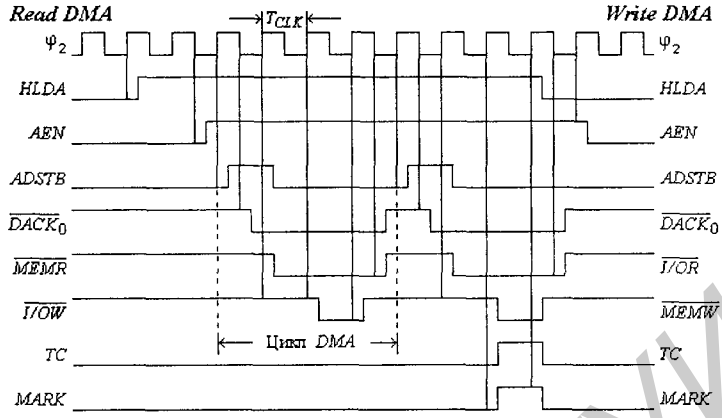


Рис. 2.22. Временны́е диаграммы циклов *DMAC*

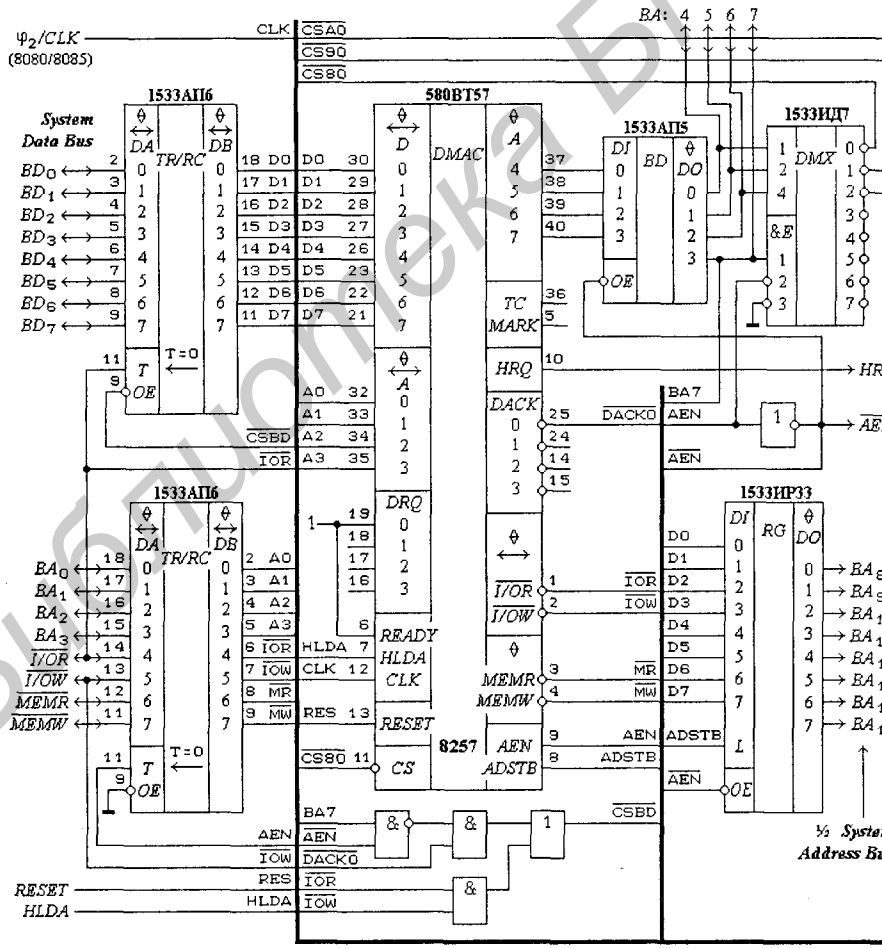


Рис. 2.23. Принципиальная схема электронного диска

Таблица 2.4. **Адресация внешних устройств**

Разряды адреса								Выход *DMX*	Адреса портов	ИС
7	6	5	4	3	2	1	0			
1	0	0	0	×	×	×	×	0	80 ÷ 8F	580ВТ57
1	0	0	1	×	×	×	×	1	90 ÷ 9F	1533ИЕ7
1	0	1	0	×	×	×	×	2	A0 ÷ AF	1533ИЕ7
1	×	×	×	×	×	×	×	0 ÷ 7	80 ÷ FF	1533АП6

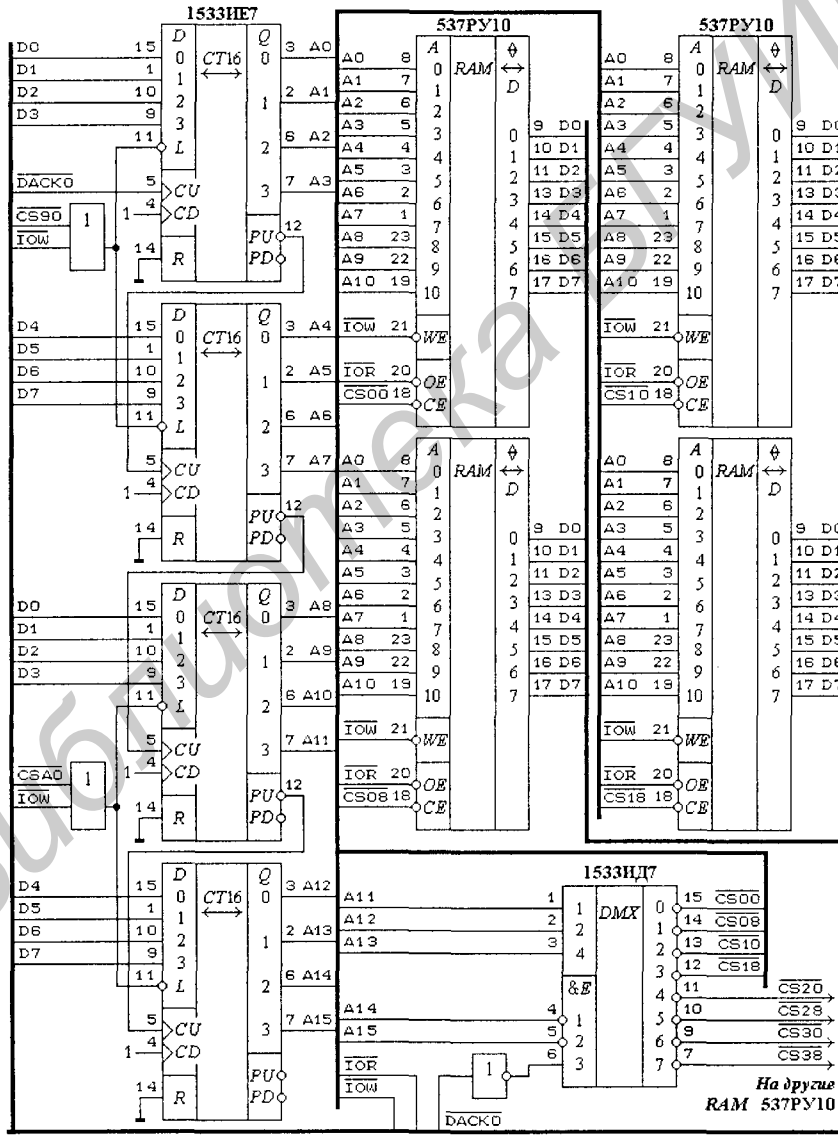


Рис. 2.23 (продолжение)

Адресация внешних устройств МП приведена в табл. 2.4. Сигнал $\overline{CSBD}$ включения приемопередатчика 1533АП6 описывается выражением

$$\overline{CSBD} = \overline{BA_7 \cdot \overline{AEN} \cdot \overline{DACK_0} \vee \overline{IOR} \cdot \overline{IOW}} = \overline{(BA_7 \cdot \overline{AEN} \vee DACK_0) \cdot (IOR \vee IOW)},$$

из которого следует, что в пассивном режиме работы *DMAC* (сигнал *AEN* = 0) приемопередатчик включается значением разряда адреса $BA_7 = 1$, а в активном режиме — значением сигнала $\overline{DACK_0} = 0$. Контроллеры прямого доступа к памяти 8257 (580ВТ57) и 8237 (1810ВТ37) описаны в § 3.6.

2.6. Память типа *FIFO*

Память типа *FIFO* (*First-In, First-Out* — первый вошел, первый вышел) позволяет выполнять запись и чтение данных независимо друг от друга с независимыми скоростями передачи данных при сохранении последовательности загрузки и выгрузки слов данных (запись и чтение данных можно производить и одновременно). Эти свойства обусловливают широкое использование *FIFO* в качестве буферной памяти при обмене данными между МП-системами.

Принципы организации *FIFO*. Большинство выпускаемых *FIFO* $n \times m$ бит, где n — число слов, m — число разрядов в слове, имеют, по крайней мере, два флага: $\overline{EMPTY}$ и $\overline{FULL}$. Эти флаги позволяют идентифицировать два крайних состояния *FIFO*: $\overline{EMPTY} = 0$ — *FIFO* пустое и $\overline{FULL} = 0$ — *FIFO* полное (рис. 2.24). Попытка записи слова данных в полностью загруженное *FIFO* ($\overline{FULL} = 0$) называется *переполнением FIFO*, а чтение слова данных из пустого *FIFO* ($\overline{EMPTY} = 0$) — *переопустошением* (антипереполнением) *FIFO*. Естественно, флаги используются для квитирования вывода и ввода данных с целью исключения потери передаваемых данных (флаг $\overline{FULL}$) и приема ложных данных (флаг $\overline{EMPTY}$).

Если *FIFO* имеет только флаги $\overline{EMPTY}$ и $\overline{FULL}$, то для исключения ошибок при выводе и вводе данных квитирование необходимо производить для каждого слова данных, что значительно снижает скорость их передачи. Поэтому *FIFO* с большой глубиной (большим числом n хранимых слов данных) снабжаются еще флагом *HF* (*Half-Full Flag*) — половина *FIFO* заполнена (*HF* = 1, когда *FIFO* загружено не менее чем наполовину). Обнаружение значения флага *HF* = 0 позволяет передающей МП-системе без опасности переполнения передать без побайтного квитирования блок данных размером в половину глубины *FIFO* ($n/2$), а обнаружение значения флага *HF* = 1 позволяет принимающей МП-системе без опасности переопустошения принять без побайтного квитирования блок данных такого же размера. Флаг *HF* целесообразно использовать в качестве сигнала запроса прерывания для вызова подпрограмм вывода и ввода без квитирования блока данных размером не более половины глубины *FIFO*. Например, следующий фрагмент подпрограммы выполняет ввод $n/2$ байт данных при $n = 2048$:

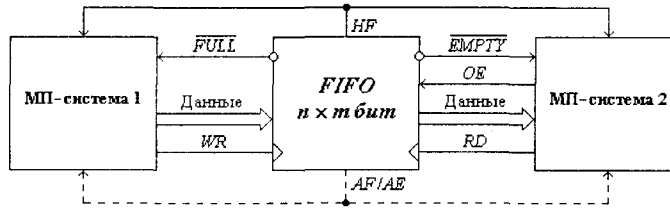


Рис. 2.24. Использование *FIFO* в качестве буферной памяти

```
        LXI     H, a_beg    ; rp H ← a_beg — начальный адрес системной памяти
        LXI     B, 1024     ; rp B ← 1024 = n/2 — половина глубины FIFO
L1:     IN      port_F      ; A ← I/O(port_F) — ввод байта данных из FIFO
        MOV     M, A        ; M(rp H) ← A
        INX     H
        DCX     B
        MOV     A, C
        ORA     B
        JNZ     L1          ; Переход, если BC ≠ 0
```

Некоторые *FIFO* снабжаются дополнительными флагами, идентифицирующими другие состояния *FIFO* (например, флагом *AF/AE — Almost-Full/Almost-Empty Flag* — почти полное/почти пустое *FIFO*). Такие флаги позволяют МП-системам при противоположных значениях флага *HF* без опасности переполнения и переопустошения передавать и принимать без побайтного квитирования блоки данных меньшего размера, чем половина глубины *FIFO*.

Разработано несколько методов построения *FIFO* $n \times m$ бит. Лучшими динамическими характеристиками обладает метод, основанный на использовании двухпортового оперативного запоминающего устройства *SRAM* (*Static Random Access Memory*) с *Z*-состоянием выходов, двух кольцевых *n*-разрядных адресных счетчиков (один для адресации записи, а другой для адресации чтения) и цифровых компараторов адресных сигналов, вырабатывающих флаги. Состояния адресных счетчиков изменяются при записи (*WR* = ⎍) и чтении (*RD* = ⎍) каждого слова данных. При чтении следует подать значение сигнала *OE* = 1 (при *OE* = 0 выходы данных находятся в *Z*-состоянии). Таким образом, *FIFO* — это безадресная память, так как извне адресные сигналы не подаются, а вырабатываются внутри самого *FIFO*, т. е. *FIFO* для МП является обычным внешним устройством.

В табл. 2.5 представлены *FIFO*, выпускаемые фирмой *Texas Instruments Inc.*

Таблица 2.5. ***FIFO* фирмы *Texas Instruments***

ИС	Объем памяти	*F*, МГц	Тип	$\overline{FULL}$ / $\overline{EMPTY}$	*HF*	*AF/AE*	Технология	Число выводов
'222*A*	16 × 4	10	*U*	–	–	–	*LS*	20
'224*A*		10		+	–	–	*LS*	16
'232*B*		40		+	–	–	*ALS*	16, 20
'225	16 × 5	10	*U*	+	–	–	*S*	20
'233*B*		40		+	–	–	*ALS*	16, 20
'236	64 × 4	30	*U*	+	–	–	*ALS*	16
'2226	Два *FIFO* 64 × 1	22	*C*	+	+	–	*ACT*	24
'2227		60		+	+	–		28
'2228	Два *FIFO* 256 × 1	22	*C*	+	+	–	*ACT*	24
'2229		60		+	+	–		28
'2232*A*	64 × 8	40	*U*				*ALS*	24, 28
'2233*A*	64 × 9	40	*U*				*ALS*	28
'7200*L*	256 × 9	–	*U*				*ACT*	28, 32
'7201*LA*	512 × 9	–	*U*				*ACT*	28, 32
'72211*L*		–						32
'2235	1K × 9	50	*B*	+	+	+	*ACT*	44, 64

Продолжение табл. 2.5

ИС	Объем памяти	*F*, МГц	Тип	$\overline{FULL}$ / $\overline{EMPTY}$	*HF*	*AF/AE*	Технология	Число выводов
‘7202*LA* ‘72221*L*		– –	*U*				*ACT*	28, 32 32
‘2236	1K × 9	–	*B*				*ACT*	44
‘7807	2K × 9	67	*U, C*	–	+	+	*ACT*	44, 64
‘7203*L* ‘7808 ‘72231*L*		– 50 –	*U*	+	+	+	*ACT*	28, 32 44, 64 32
‘7204*L* ‘72241*L*	4K × 9	– –	*U*				*ACT*	28, 32 32
‘7205*L* ‘7206*L*	8K × 9 16K × 9	– –	*U* *U*				*ACT* *ACT*	28, 32 32, 32
‘7813 ‘7814	64 × 18	67/50 50/40	*U*	– +	+ +	+ +	*ACT/ALVC*	56
‘7805 ‘7806	256 × 18	67/50 40	*U, C* *U*	– +	+ +	+ +	*ACT/ALVC* *ALVC*	56
‘7803 ‘7804	512 × 18	67/50 50/40	*U, C* *U*	– +	+ +	+ +	*ACT/ALVC*	56
‘7819 ‘7820		80 67	*B, C* *B*	– +	+ +	+ +	*ABT*	80
‘7811 ‘7881	1K × 18	40 67	*U, C*	– –	+ +	+ +	*ACT*	68, 80
‘7802		40	*U*	+	+	+	*ACT*	68, 80
‘7882	2K × 18	–	*C*				*ACT*	68, 80
‘3612 ‘3614	64 × 36	67	*B, C*	+ +	– –	+ +	*ABT*	132, 120
‘3611 ‘3613		67	*U, C*	+ +	– –	+ +	*ABT*	132, 120
‘3622	256 × 36	67	*B, C*	+	–	+	*ACT*	132, 120
‘3638	512 × 32	67	*B, C*	+	–	+	*ACT*	132, 120
‘3631 ‘3632	512 × 36	67/100 67	*U, C* *B, C*	+ +	– –	+ +	*ACT/ALVC* *ACT*	132, 120
‘3641	1K × 36	67/100	*U, C*	+	–	+	*ACT/ALVC*	132, 120
‘3651	2K × 36	67/100	*U, C*	+	–	+	*ACT/ALVC*	132, 120

Примечание: *C* — *Clocked*, *B* — *Bidirectional* (двунаправленная передача данных), *U* — *Unidirectional*.

***FIFO SN74ALS232B* (16 × 4 бит).** Чтение и запись в этом *FIFO* (рис. 2.25) асинхронные: запись данных производится положительным фронтом сигнала *LDCK* (*load-clock*), а чтение — положительным фронтом сигнала *UNCK* (*unload-clock*), причем эти сигналы могут совпадать по времени. Передача данных однонаправленная со скоростью от 0 до 40 МГц. На рис. 2.26 показана структурная схема *FIFO*:

Ring Counter — кольцевые адресные счетчики;

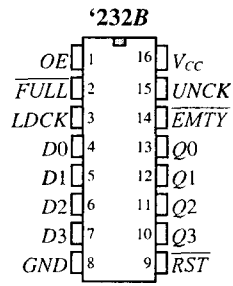
Comparator — цифровой компаратор адресных сигналов кольцевых счетчиков;

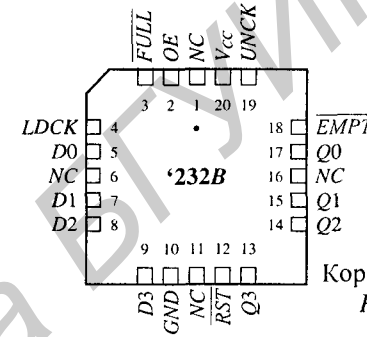
RAM 16 × 4 бит — синхронная статическая память объемом 16 × 4 (16 четырехразрядных слов), независимо адресуемая двумя кольцевыми счетчиками;

Control Logic — схема управления, формирующая с помощью разностных элементов [5] из входных потенциальных сигналов *LDCK* и *UNCK* импульсные тактовые сигналы *CWR* (*Clock Write*) и *CRD* (*Clock Read*) для кольцевых адресных счетчиков; сигнал *CWR* используется и для записи входных данных D_{3-0} в память.

Кольцевые счетчики содержат по 16 триггеров — дешифрация адресов не требуется, так как у кольцевых счетчиков выходной сигнал только одного триггера равен 1 (кодирование адресов производится 16-разрядными унитарными кодами *P* и *Q*) [5].

Корпус типов *SOIC* и *PDIP* 300 *mil*





Корпус типа *PLCC*

Рис. 2.25. *FIFO* '232*B*

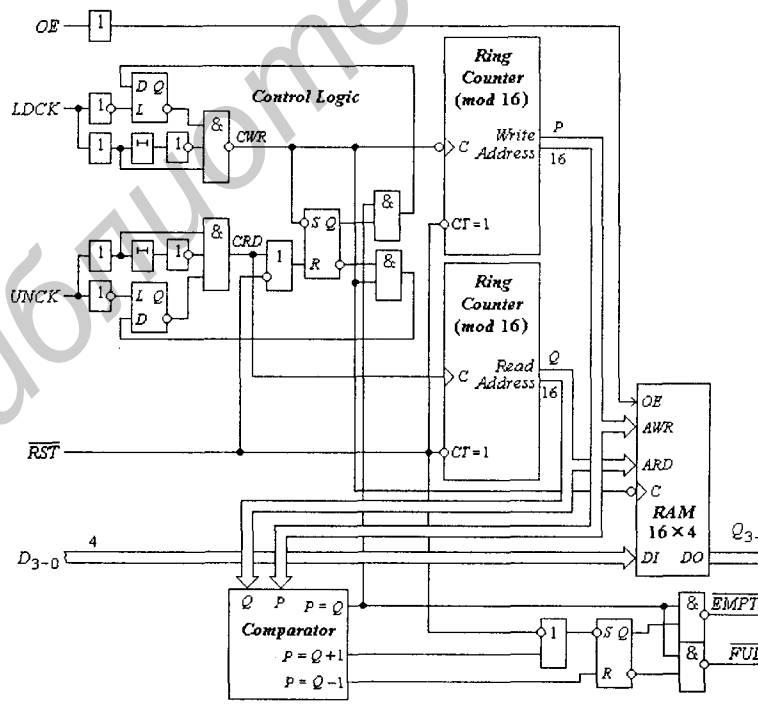


Рис. 2.26. Структурная схема *FIFO* '232*B*

Все ячейки памяти можно пронумеровать числами от 1 до 16 в порядке их выбора при поступлении тактовых сигналов на счетчики. Адресам записи P и чтения Q присваиваются эти же числа, хотя и используется не двоичное их кодирование: $AWR = P = 1 \div 16$ — адреса записи данных D_{3-0}, $ARD = Q = 1 \div 16$ — адреса чтения данных Q_{3-0}. После выбора последней (16) ячейки памяти адресуется снова первая (1) ячейка памяти, т. е. память можно представить в виде кольца (рис. 2.27). При загрузке и выгрузке слова данных из *FIFO* производится инкремент адреса. Память с такой адресацией представляет собой кольцевой стек.

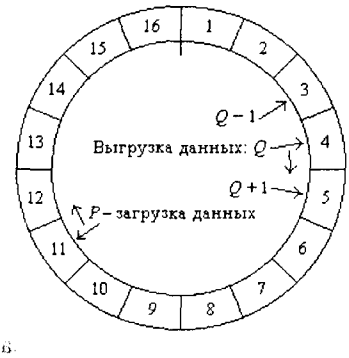


Рис. 2.27. Кольцевой стек

Для квитирования ввода и вывода *FIFO* снабжено двумя флагами: $\overline{EMPTY}$ и $\overline{FULL}$ ($\overline{EMPTY} = 0$ — *FIFO* пустое, $\overline{FULL} = 0$ — *FIFO* полностью загружено). Активное значение сигнала сброса $\overline{RST} = 0$ инициализирует счетчики (в состояние 1 устанавливается только один триггер, адресующий первую ячейку памяти) и устанавливает значения флагов $\overline{EMPTY} = 0$ и $\overline{FULL} = 1$ (рис. 2.28). Данные загружаются в *FIFO* положительным фронтом сигнала *LDCK*, когда флаг $\overline{FULL} = 1$, а чтение — положительным фронтом сигнала *UNCK*, когда флаг $\overline{EMPTY} = 1$. Полная загрузка *FIFO* возникает в непредсказуемый момент времени, когда количество загруженных слов превысит на 16 число выгруженных слов. Выход $P = Q$ компаратора равен 1 при совпадении состояний счетчиков, выход $P = Q - 1$ равен 1 при опережении на 1 состояния счетчика адресов записи состояния счетчика адресов чтения, а выход $P = Q + 1$ равен 1 при отставании на 1 состояния счетчика адресов записи от состояния счетчика адресов чтения. С помощью этих сигналов формируются флаги $\overline{EMPTY}$ и $\overline{FULL}$, и производится запрет счета адресов записи после полной загрузки *FIFO*.

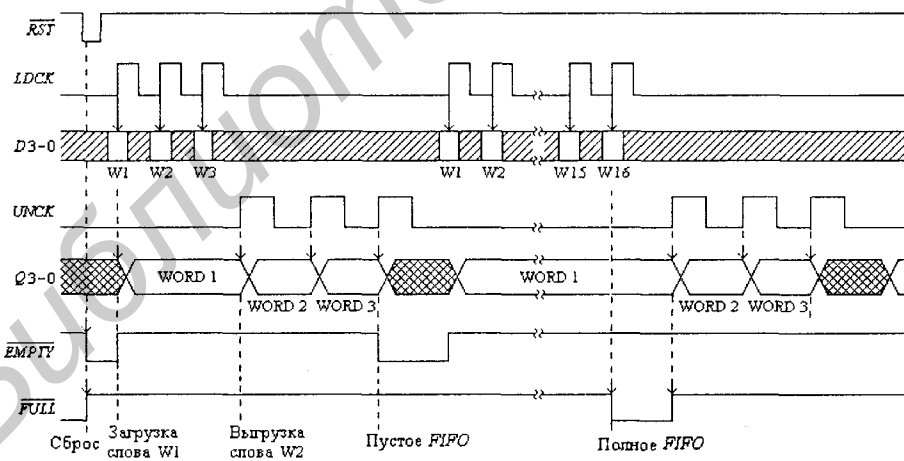


Рис. 2.28. Временны́е диаграммы для *FIFO* ‘232*B* ($OE = 1$)

FIFO имеет два порта данных: D_{3-0} — 4-разрядный порт ввода данных (для МП-системы это порт вывода данных) и Q_{3-0} — 4-разрядный порт вывода данных (для МП-системы это порт ввода данных). Выходы данных Q_{3-0} имеют Z-состояние при значении сигнала $OE = 0$. Выходные данные не инвертированы относительно входных данных. Запись слова данных производится по заранее установленному адресу, а затем производится инкремент адреса. При

полной загрузке *FIFO* сигнал *LDCK* не оказывает никакого влияния на находящиеся в нем данные. Выгрузка слова данных из *FIFO* — это, по существу, инкремент адреса чтения. Обычно выгрузка слова данных сопровождается его чтением из *FIFO* по заранее установленному адресу значением сигнала $OE = 1$, а затем производится инкремент адреса (выгрузка). Когда *FIFO* пусто, сигнал *UNCK* не оказывает на него воздействия. В МП-системах сигналы *LDCK* и *UNCK* должны формироваться при выполнении команд вывода OUT *port* и ввода IN *port* соответственно: $LDCK = I/OW \cdot CS$ и $UNCK = I/OR \cdot CS = OE$.

По описанному выше методу построено и *FIFO SN74ALS2232A* 64 × 8 бит. Каскадирование этих *FIFO* легко выполняется в направлении увеличения разрядности слов, но невозможно в направлении увеличения их глубины.

Структурная схема *FIFO*, изображенная на рис. 2.26, столь подробна, что не составляет труда нарисовать его принципиальную схему, что и было сделано для программного пакета *Micro-Logic II*, предназначенного для моделирования цифровых устройств. На рис. 2.29 изображены временны́е диаграммы, полученные с помощью этого программного пакета. Сигналы *CWR* и *CRD* формируются разностными элементами (см. рис. 2.26) и используются в качестве тактовых сигналов адресных счетчиков и синхронной памяти, в качестве которой был использован синхронный регистровый файл 16 × 4 бит.

Рис. 2.29. Временны́е диаграммы модели *FIFO* '232*B*

***FIFO SN74ACT7808* (2048×9 бит).** Чтение и запись в данном *FIFO* (рис. 2.30) асинхронные: запись данных производится положительным фронтом сигнала *LDCK* (*load-clock*), а чтение — положительным фронтом сигнала *UNCK* (*unload-clock*), причем эти сигналы могут совпадать по времени. Передача данных однонаправленная со скоростью от 0 до 50 МГц (время доступа равно 15 нс при емкостной нагрузке 50 пФ). *FIFO* имеет два порта данных (рис. 2.31): D_{8-0} — 9-разрядный порт ввода данных (для МП-системы это порт вывода данных) и Q_{8-0} — 9-разрядный порт вывода данных (для МП-системы это порт ввода данных). Выходы данных Q_{8-0} имеют *Z*-состояние при значении сигнала $OE = 0$. Для организации двунаправленной передачи данных между МП-системами нужно использовать два *FIFO*.

Для квитирования ввода и вывода *FIFO* имеет четыре флага, информирующих МП-системы о его состоянии: $\overline{EMPTY}$ — *FIFO* пустое ($\overline{EMPTY} = 0$), $\overline{FULL}$ — *FIFO* полностью загружено ($\overline{FULL} = 0$), *HF* (*Half-Full Flag*) — *FIFO* загружено наполовину ($HF = 1$, когда *FIFO* содержит 1024 или большее количество слов) и программируемый флаг *AF*/*AE* (*Programmable*

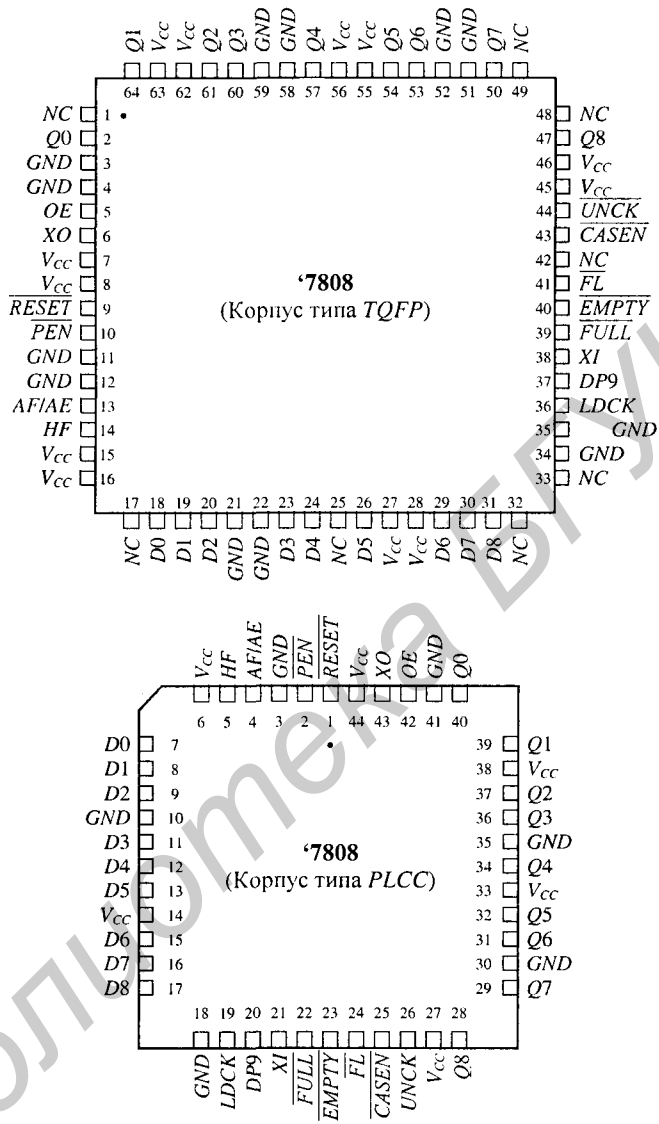


Рис. 2.30. *FIFO* '7808 фирмы *Texas Instruments*

Almost-Full/Almost-Empty Flag) — почти полное/почти пустое *FIFO* (*AF/AE* = 1). Активное значение сигнала сброса $\overline{RESET}$ = 0 устанавливает значения флагов: $\overline{EMPTY}$ = 0, $\overline{FULL}$ = 1, *HF* = 0 и *AF/AE* = 1 (сброс должен производиться при каждом включении питания системы).

Данные загружаются в *FIFO* положительным фронтом сигнала *LDCK*, когда значение флага $\overline{FULL}$ = 1, а чтение — положительным фронтом сигнала *UNCK*, когда значение флага $\overline{EMPTY}$ = 1. Полное заполнение *FIFO* возникает при превышении на 2048 числа загруженных слов количества выгруженных слов. При полной загрузке *FIFO* сигнал *LDCK* не оказывает никакого влияния на находящиеся в нем данные. Когда *FIFO* пусто, сигнал *UNCK* не оказывает на него воздействия. В МП-системах сигналы *LDCK* и *UNCK* должны формироваться при вы-

полнении команд вывода OUT *port* и ввода IN *port* соответственно. При передаче блоков слов данных можно использовать вывод и ввод по прерыванию или по прямому доступу к памяти, используя изменения, например, флага *HF* для формирования сигнала запроса прерывания или прямого доступа к памяти.

Флаг *AF/AE* характеризуется двумя программируемыми значениями глубин смещения (*Depth-offset*): *X* и *Y*. По умолчанию (если значения *X* и *Y* не программируются) глубина смещения для флага *AF/AE* устанавливается равной 256 словам: *X* = 256 для флага *AE* (почти пустое *FIFO*) и *Y* = 256 для флага *AF* (почти полное *FIFO*). Значение флага *AF/AE* = 1, когда *FIFO* содержит *X* или меньшее количество слов или 2048 – *Y* или большее количество слов.

Смещения *X* и *Y* можно программировать только после сброса *FIFO* (до загрузки в *FIFO* первого слова данных) и установленном значении сигнала *LDCK* = 0 (рис. 2.32). Разрешение программирования производится подачей значения сигнала $\overline{PEN}$ = 0 (*Program Enable* — разрешение программирования). По первому положительному фронту сигнала *LDCK* в регистре памяти фиксируется 10-разрядное значение DP_9D_{8-0} для смещений *X* и *Y*, а второй положительный фронт сигнала *LDCK* перепрограммирует смещение *Y*, если это необходимо. Максимальное значение смещения DP_9D_{8-0} = 1023 можно использовать только для одного из смещений *X* или *Y*. Во время программирования загрузка данных в *FIFO* заблокирована. Чтобы использовать значения смещений по умолчанию (*X* = *Y* = 256), при сбросе следует установить значение сигнала $\overline{PEN}$ = 1.

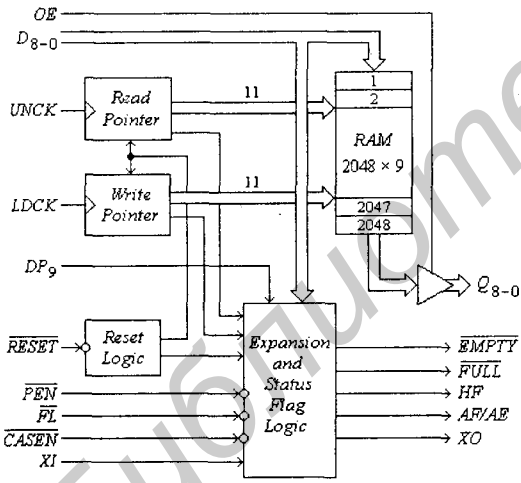


Рис. 2.31. Структурная схема *FIFO* '7808

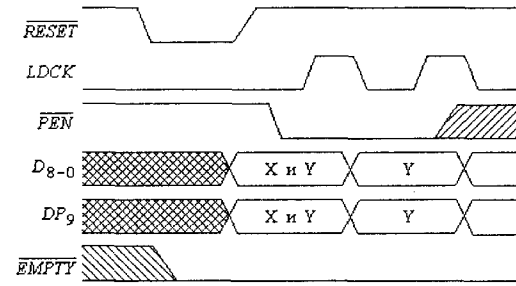


Рис. 2.32. Временны́е диаграммы программирования флага *AF/AE*

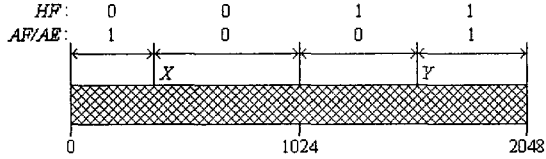


Рис. 2.33. Состояния *FIFO*

Таблица 2.6. **Размеры блоков данных**

AF/AE	*HF*	N_{LD}	N_{UN}
0	0	1024 слова	*X* слов
0	1	*Y* слов	1024 слова
1	0	1024 слова	1 слово
1	1	1 слово	1024 слова

Для достижения максимальной скорости передачи данных флаги *HF* и *AF/AE* следует использовать совместно — при программном квитировании вывода и ввода должны учитываться комбинации значений этих флагов (рис. 2.33). В табл. 2.6 приведены максимальные значения размеров загружаемых N_{LD} и выгружаемых N_{UN} без побайтного квитирования блоков. При размере блока в 1 слово дальнейшая передача данных возможна лишь с побайтным квитированием. При этом, естественно, необходимо учитывать значения флагов $\overline{FULL}$ и $\overline{EMPTY}$. Если флаг *AF/AE* не используется, то размеры блоков данных определяются двумя нижними строками табл. 2.6.

Четыре описанных флага имеет и *FIFO SN74ALS2233A* 64 × 9 бит, но флаг *AF/AE* не программируемый — *AF/AE* = 1, когда *FIFO* содержит меньше девяти или больше 55 слов.

Четыре сигнала *FIFO* — $\overline{CASEN}$ (*Cascade Enable* — разрешение каскадирования), $\overline{FL}$ (*First Load* — вход начала загрузки), *XI* (*Expansion Input* — вход расширения) и *XO* (*Expansion Output* — выход расширения) — используются для их каскадирования с целью увеличения (расширения) глубины *FIFO*. Если расширение *FIFO* по глубине не производится, то следует задать значение сигнала $\overline{CASEN}$ = 1, а при расширении глубины *FIFO* необходимо задать $\overline{CASEN}$ = 0. На рис. 2.34 изображено *FIFO* 6К × 9 бит, построенное из трех *FIFO* 2К × 9 бит.

Для первого *FIFO* в цепочке следует задать значение сигнала $\overline{FL}$ = 0, а для остальных — $\overline{FL}$ = 1. Выход расширения *XO* одного устройства подключается к входу расширения *XI* следующего устройства в цепочке, а выход расширения *XO* последнего устройства в цепочке должен быть связан с входом расширения *XI* первого устройства. Шина данных общая для всех *FIFO* в цепочке. Для индикации граничных условий флаги $\overline{FULL}$ и $\overline{EMPTY}$ всех *FIFO* должны быть объединены по ИЛИ. Таким способом можно соединить любое число *FIFO*.

На рис. 2.35 показан способ каскадирования *FIFO*, позволяющий увеличивать разрядность слов данных. Таким способом можно увеличить разрядность слов в любое число раз. Флаги $\overline{FULL}$ и $\overline{EMPTY}$ можно и не объединять. Для двунаправленной передачи данных необходимо использовать два *FIFO*, включенных в противоположных направлениях (рис. 2.36).

Обнаружение однократных ошибок в *FIFO*. Ошибки в работе *FIFO* происходят не в линиях связи, а при записи и чтении данных, например, из-за импульсных помех источника питания. Для обнаружения однократных ошибок достаточно использовать контроль паритета в 9-разрядных *FIFO* (рис. 2.37).

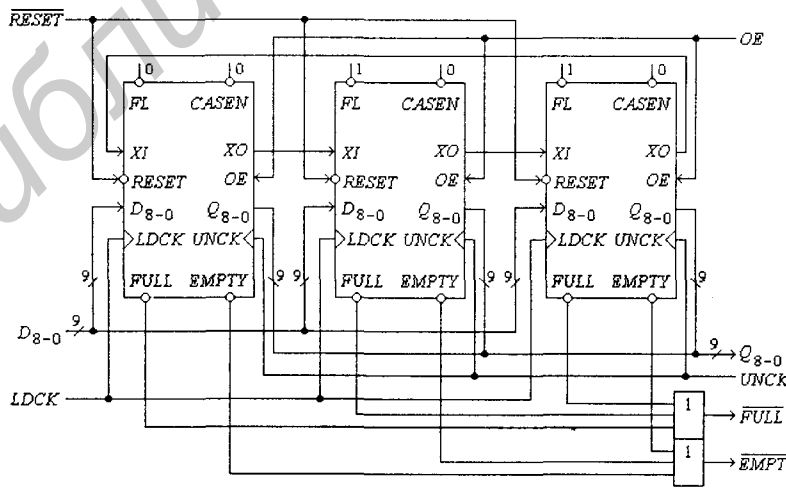


Рис. 2.34. Расширение глубины *FIFO* с 2К до 6К

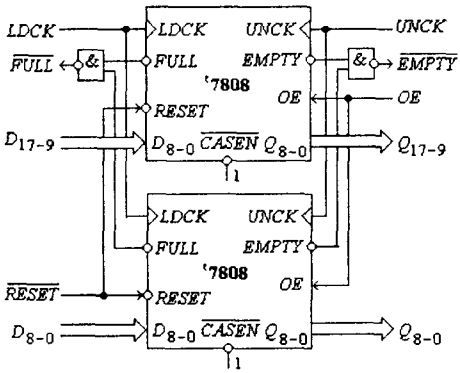


Рис. 2.35. Удвоение разрядности слов данных *FIFO*

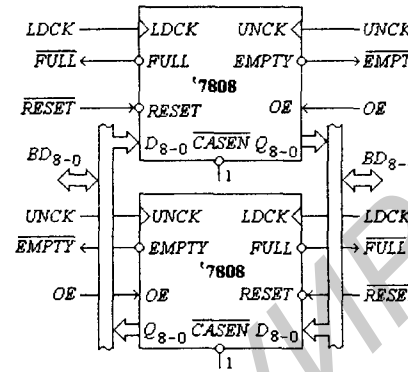


Рис. 2.36. Двунаправленная передача данных

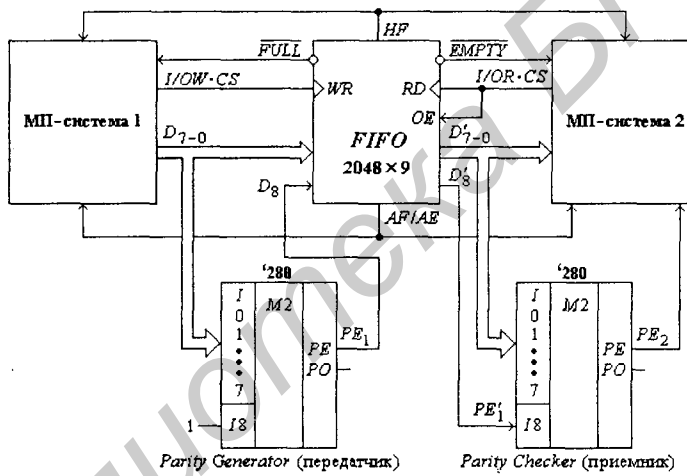


Рис. 2.37. Контроль передачи данных *FIFO*

ИС ‘280 описываются функциями:

$$PE = \overline{\sum_{p=0}^{8} I_p} = \overline{I_8} \oplus \sum_{p=0}^{7} I_p \quad \text{и} \quad PO = \sum_{p=0}^{8} I_p = I_8 \oplus \sum_{p=0}^{7} I_p,$$

где I_p — входные сигналы разрядов данных ($p = 0, 1, \ldots, 8$), *PE* (*Parity Even*) — выходной сигнал четного паритета, *PO* (*Parity Odd*) — выходной сигнал нечетного паритета ($PO = \overline{PE}$), Σ — операция сумма по модулю два.

При передаче 8-разрядных данных D_{7-0} генератор паритета (*Parity Generators*) добавляет девятый проверочный разряд

$$D_8 = PE_1 = \sum_{p=0}^{7} D_p,$$

т. е. в *FIFO* загружаются 9-разрядные двоичные коды $PE_1D_7D_6D_5D_4D_3D_2D_1D_0$, всегда содержащие четное число единиц.

При выгрузке данных из *FIFO* с помощью устройства проверки паритета (*Parity Checker*) производится контроль четности возможно искаженного кода $PE_1'D_7'D_6'D_5'D_4'D_3'D_2'D_1'D_0'$:

$$PE_2 = \overline{PE_1'} \oplus \sum_{p=0}^{7} D_p'.$$

Если ошибки в работе *FIFO* отсутствуют, то код

$$PE_1'D_7'D_6'D_5'D_4'D_3'D_2'D_1'D_0' = PE_1D_7D_6D_5D_4D_3D_2D_1D_0 \text{ и сигнал}$$

$$PE_2 = \overline{PE'}_1 \oplus \sum_{p=0}^{7} D_p' = \overline{\sum_{p=0}^{7} D_p} \oplus \sum_{p=0}^{7} D_p = 1 \text{ — четный паритет (ошибки нет).}$$

Если в *FIFO* произойдет искажение нечетного числа разрядов кода, то при его выгрузке будет получено значение сигнала $PE_2 = 0$ — нечетный паритет (ошибка в передаче данных). Искажение любого четного числа разрядов не обнаруживается.

Классификация *FIFO*. По терминологии фирмы *Texas Instruments* все *FIFO* подразделяются на четыре типа (рис. 2.38):

Asynchronous FIFO — загрузка данных производится потенциальным сигналом низкого уровня (*pulse*) при условии разрешения загрузки сигналом, управляющим записью (*write–enable*); выгрузка данных производится потенциальным сигналом низкого уровня (*pulse*) при условии разрешения выгрузки сигналом, управляющим чтением (*read–enable*); флаги *empty* и *full* не синхронизированы никакими входными сигналами;

Clocked FIFO — загрузка данных производится переходом сигнала записи (*write clock*) с 0 на 1 при условии разрешения загрузки сигналом, управляющим записью (*write–enable*); выгрузка данных производится переходом сигнала чтения (*read clock*) с 0 на 1 при условии разрешения выгрузки сигналом, управляющим чтением (*read–enable*); флаг готовности входов (*input–ready flag*) синхронизирован сигналом записи, а флаг готовности выходов (*output–ready flag*) — сигналом чтения;

Strobed FIFO — запись данных производится переходом сигнала загрузки (*load–clock*) с 0 на 1; чтение данных производится переходом сигнала выгрузки (*unload–clock*) с 0 на 1; флаги *empty* и *full* не синхронизированы никакими входными сигналами;

Synchronous FIFO — загрузка и выгрузка данных производится таким же образом, что и в *Clocked FIFO*; флаг *empty* синхронизирован сигналом чтения, а флаг *full* — сигналом записи; термин "синхронный" относится к методу управления портами и не подразумевает, что запись и чтение должны быть синхронны друг к другу.

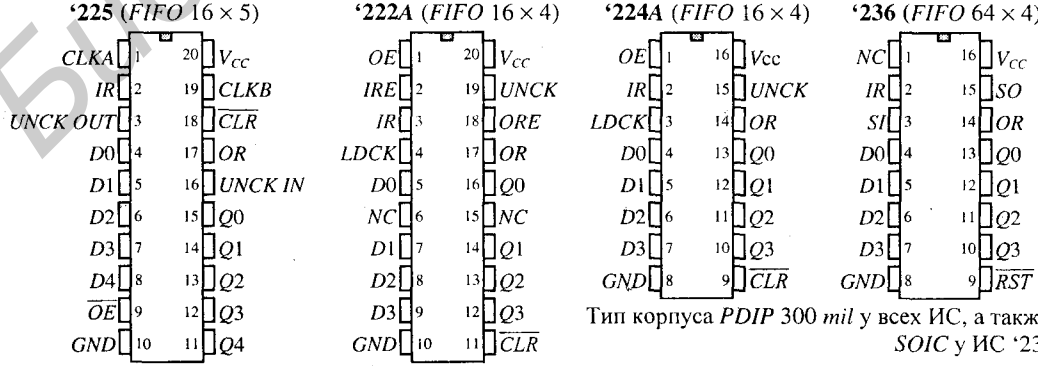
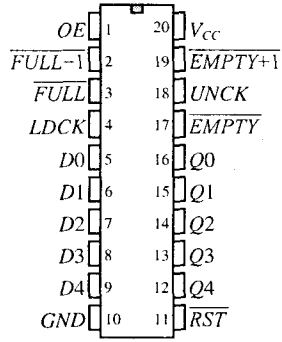
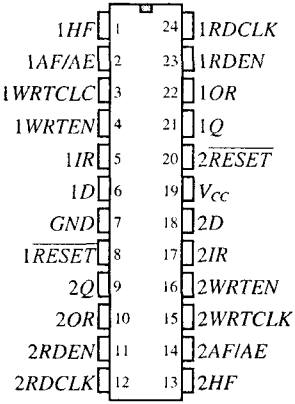


Рис. 2.38. *FIFO* фирмы *Texas Instruments Inc.* (см. также с. 189 – 192)

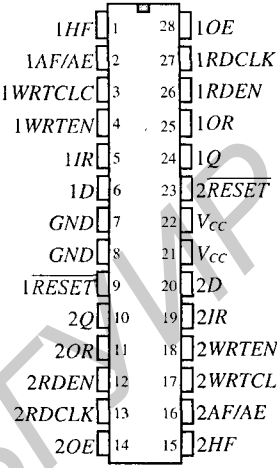
‘233*B* (*FIFO* 16 × 5)



‘2226 (*FIFO Dual* 64 × 1)
‘2228 (*FIFO Dual* 256 × 1)



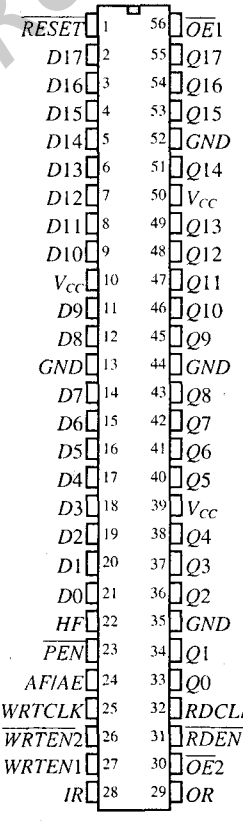
‘2227 (*FIFO Dual* 64 × 1)
‘2229 (*FIFO Dual* 256 × 1)



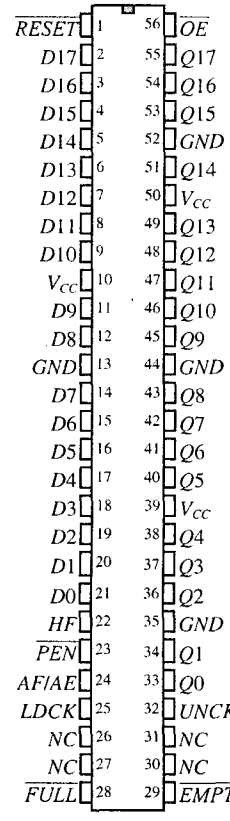
Корпус типа *SOIC* у всех ИС,
а также *PDIP* 300 *mil* у ИС ‘233*B*

Корпус типа *SSOP* 300 *mil* →

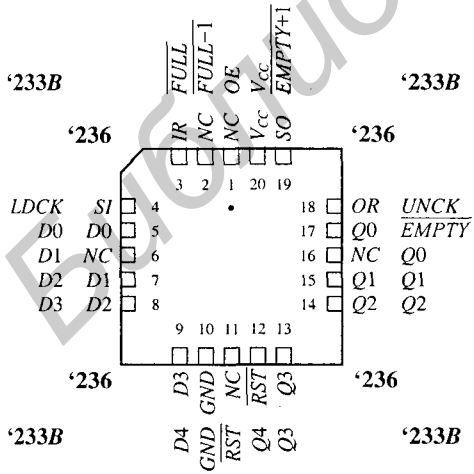
‘7803 (*FIFO* 512 × 18)
‘7805 (*FIFO* 256 × 18)
‘7813 (*FIFO* 64 × 18)



‘7804 (*FIFO* 512 × 18)
‘7806 (*FIFO* 256 × 18)
‘7814 (*FIFO* 64 × 18)

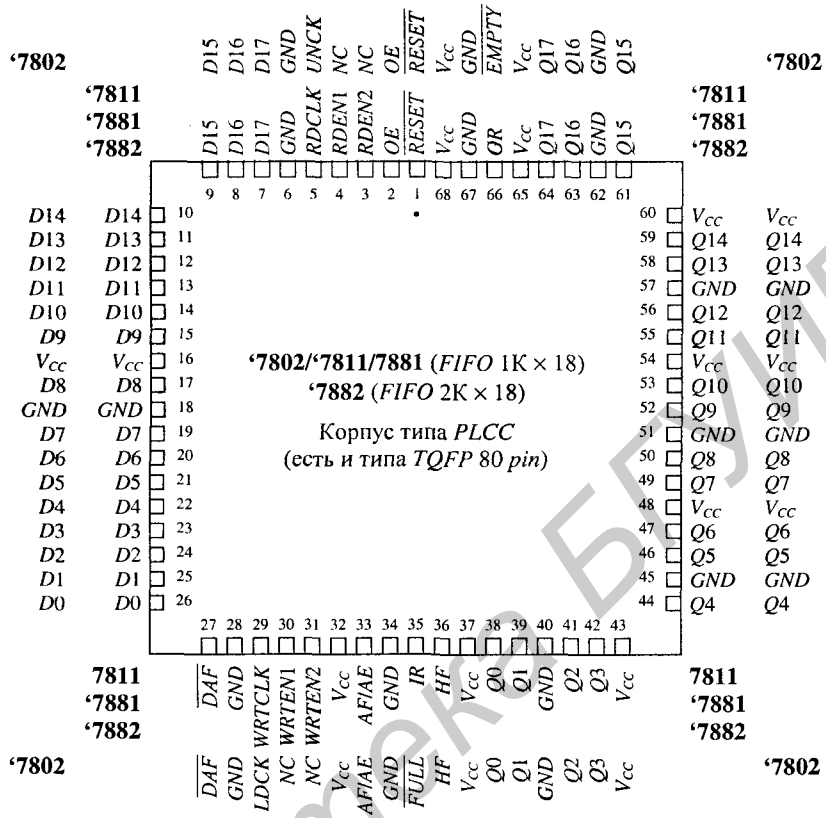


‘233*B* (*FIFO* 16 × 5), **‘236** (*FIFO* 64 × 4)



Корпус типа *PLCC*

Рис. 2.38 (продолжение)



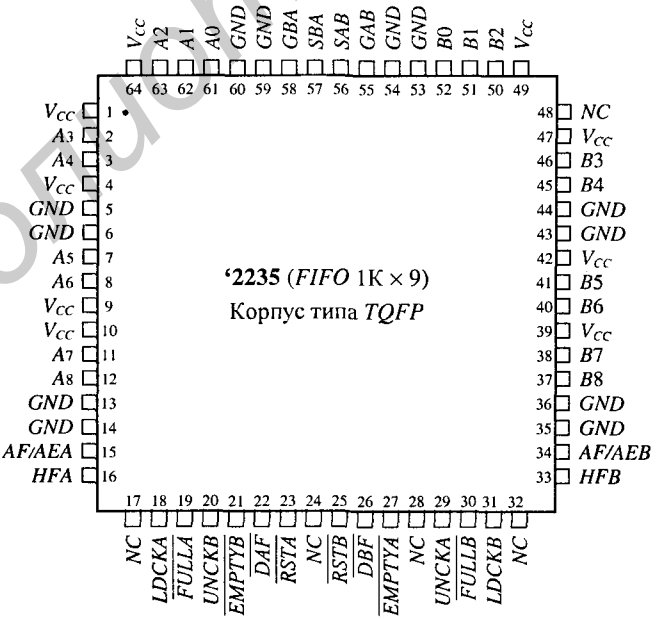
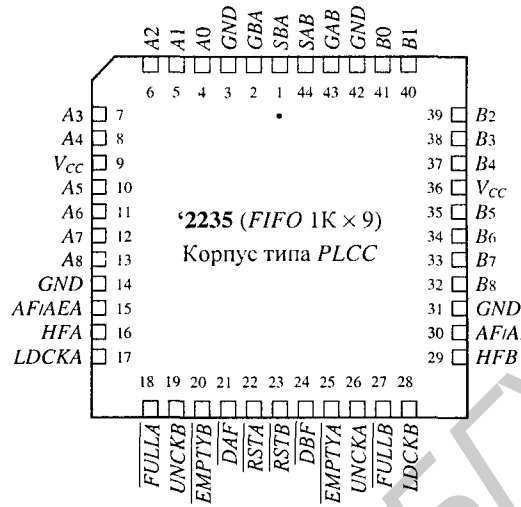


Рис. 2.38 (продолжение)



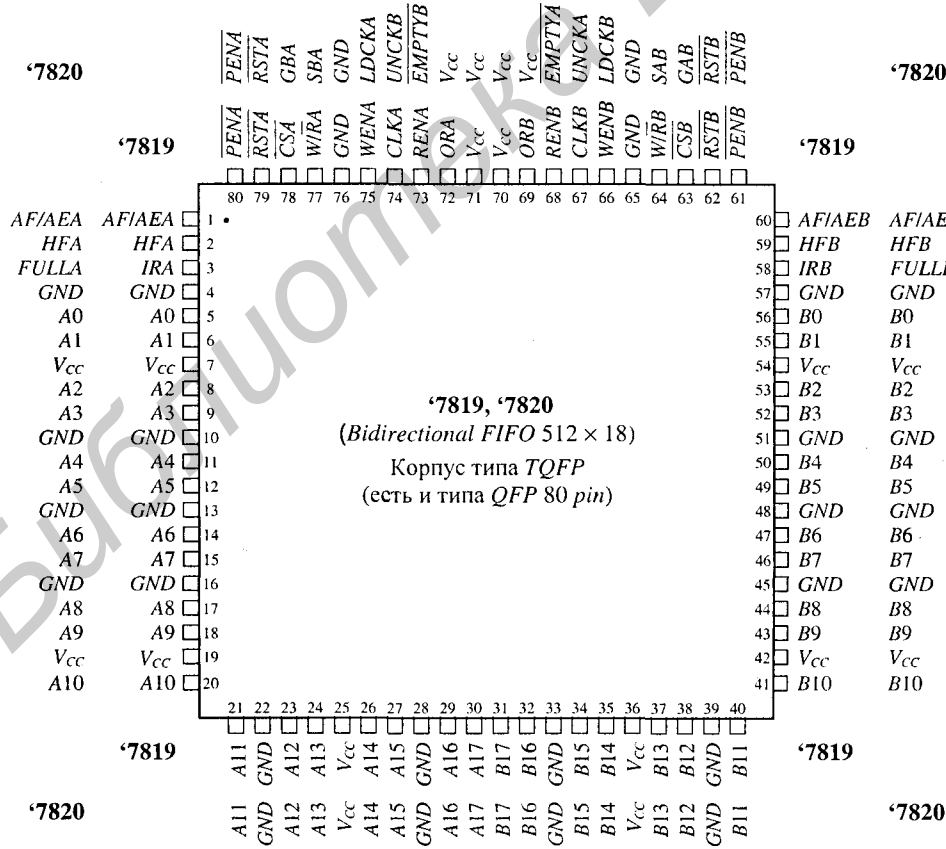


Рис. 2.38 (продолжение)

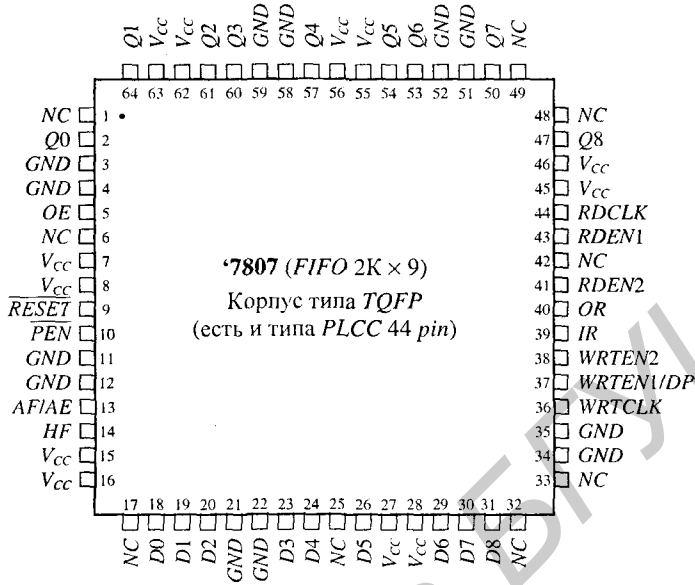


Рис. 2.38 (окончание)

Память типа *FIFO* является идеальным внешним устройством для ввода и вывода блоков данных, так как заполнение *FIFO* может производиться достаточно медленно (в темпе подготовки данных программой одного микропроцессора), а передача накопленных данных в системную память другого микропроцессора может выполняться единым блоком с минимальной затратой времени. Флаг *HF* можно использовать и для запроса прямого доступа к памяти — в этом случае преимущества использования *FIFO* в качестве буферной памяти сказываются еще больше, так как достигается наибольшая скорость передачи данных и не требуется адресация блоков данных, находящихся в *FIFO* (адресный счетчик *Counter*, подобный приведенному на рис. 2.21, не нужен).

Ввод по прерыванию с использованием *FIFO* в качестве буферной памяти необходим в тех случаях, когда МП может быть занят выполнением какой-либо важной программы, а поэтому не способен немедленно отреагировать на запрос обслуживания. По этой причине контроллеры клавиатуры, как правило, содержат *FIFO* небольшого объема для временного сохранения кодов нажатых клавиш, что исключает их потерю из-за "нерасторопности" МП.

Типовыми устройствами ввода данных в МП-системы являются аналого-цифровые преобразователи (*ADC — analog-to-digital converters*), в которые для сопряжения их с МП могут вводиться *FIFO*. Так ИС *THS*1206 фирмы *Texas Instruments* с частотой выборок аналогового сигнала 6 *msps* (*mega samples per second*) содержит *FIFO* 16 × 12 (16 слов по 12 бит).

Глава 3

ИНТЕРФЕЙСНЫЕ БИС

3.1. Общая характеристика интерфейсных БИС фирмы *Intel*

Представление о назначении некоторых семейств БИС фирмы *Intel* дают две первые цифры в их обозначении, например:

27××× — *EPROM* (*Erasable Programmable ROM* — стираемое программируемое ПЗУ),

80××× — микропроцессоры и микроконтроллеры,

82××× — интерфейсные БИС поддержки МП,

87××× — приборы, содержащие *EPROM* или *OTPROM* (*One Time Programmable ROM* — однократно программируемое ПЗУ).

В табл. 3.1 приведены некоторые характеристики интерфейсных БИС, описанных в этой главе.

Таблица 3.1. **Назначение основных интерфейсных БИС фирмы *Intel***

БИС фирмы *Intel*/аналоги	*m*	Назначение
8237/1810ВТ37 8257/580ИК57	40	Программируемый четырехканальный контроллер прямого доступа к памяти
8251*A*/580ВВ51А	28	Программируемый связной интерфейс (*USART*)
8253/580ВИ53 8254/1810ВИ54	24	Программируемый интервальный таймер (три 16-разрядных вычитающих счетчика с шестью режимами работы)
8255*A*/580ВВ55А	40	Программируемый периферийный интерфейс
8259*A*/1810ВН59А	28	Программируемый контроллер прерываний
8279/580ВВ79	40	Программируемый контроллер клавиатуры и дисплея
8155/1821РУ55, 8156	40	*SRAM* 256 × 8, один 14-разрядный таймер, два 8-разрядных и один 6-разрядный программируемые порты ввода-вывода
8355/1821РЕ55	40	*ROM* 2К × 8 бит и два 8-разрядных программируемых порта ввода-вывода
8755/1821РФ55	40	*EPROM* 2К × 8 бит и два 8-разрядных программируемых порта ввода-вывода
Примечание: *m* — число выводов БИС		

3.2. Программируемый периферийный интерфейс 8255*A*

Основное назначение БИС 8255*A* (отечественный аналог 580ВВ55А) и 82*C*55*A* фирмы *Intel* (рис. 3.1) — разработка программируемых устройств ввода-вывода для МП-систем, построенных на основе МП 8080*A*, 8085*A* и 8086/8088/80186/80188. Периферийные интерфейсы 8255*A* и 8255*A*-5 изготовляются по *NMOS* технологии, а 82*C*55*A* — по *CHMOS* III технологии. Различаются эти БИС только быстродействием — БИС 82*C*55*A* имеет наибольшее быстродействие, поэтому ее можно использовать с любым из перечисленных МП.

Максимальные значения токов потребления $I_{CC\,max}$ при V_{CC} = +5 В равны 120 мА и 10 мА для БИС 8255*A*/8255*A*-5 и 82*C*55*A* соответственно. Максимальная рассеиваемая мощность у всех БИС составляет 1 Вт. Выходные сигналы характеризуются параметрами:

$V_{OL\,max}$ = 0,45 В при I_{OL} = 2,5 мА и $V_{OH\,min}$ = 2,4 В при I_{OH} = –0,4 мА для 8255*A* (шина данных),
$V_{OL\,max}$ = 0,45 В при I_{OL} = 1,7 мА и $V_{OH\,min}$ = 2,4 В при I_{OH} = –0,2 мА для 8255*A* (порты вывода),
$V_{OL\,max}$ = 0,4 В при I_{OL} = 2,5 мА и $V_{OH\,min}$ = 3 В при I_{OH} = –2,5 мА для 82*C*55*A*.

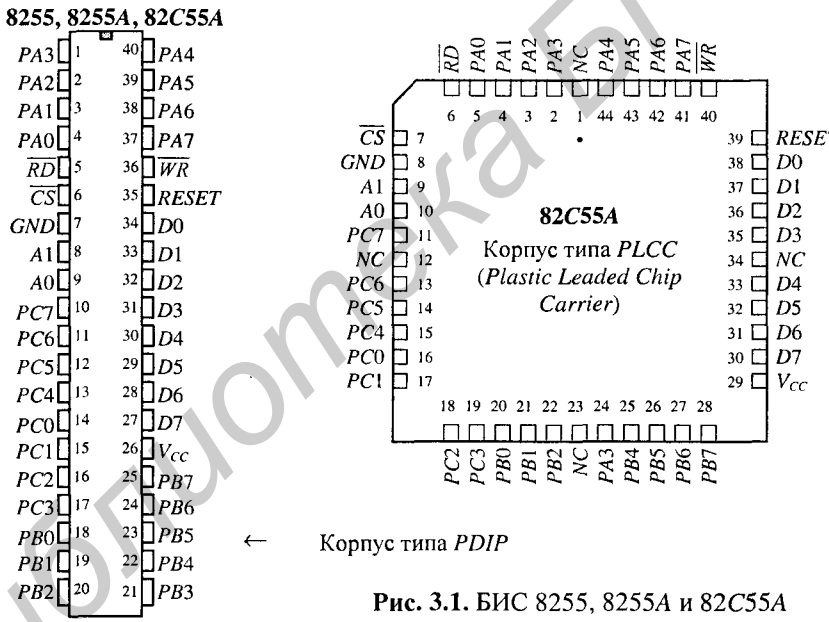


Рис. 3.1. БИС 8255, 8255*A* и 82*C*55*A*

Структурная схема БИС 8255/8255*A*. Структурная схема программируемого периферийного интерфейса *PPI* (*Programmable Peripheral Interface*) показана на рис. 3.2. Обеспечивает *PPI* ввод-вывод по трем 8-разрядным каналам (портам): $PA = PA_{7-0}$, $PB = PB_{7-0}$ и $PC = PC_{7-0}$, причем последний порт может использоваться в качестве двух 4-разрядных портов $PCh = PC_{7-4}$ (*PCh* — старшая тетрада порта *PC*) и $PCl = PC_{3-0}$ (*PCl* — младшая тетрада порта *PC*). БИС состоит из узлов:

Data Bus Buffer — буфер шины данных D_{7-0},

Read/Write Control Logic — схема управления чтением и записью данных D_{7-0} в регистры *PPI*,

Group A Port A — порт ввода-вывода *PA* группы *A*,

Group B Port B — порт ввода-вывода *PB* группы *B*,

Group A Port C — порт ввода-вывода *PCh* группы *A*,

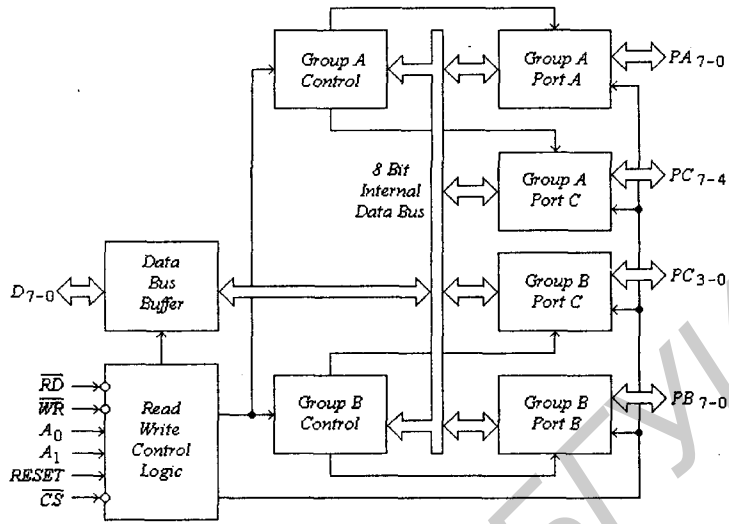


Рис. 3.2. Структурная схема *PPI*

Group B Port C — порт ввода-вывода *PCl* группы *B*,

Group A Control — устройство управления группой портов *A* (*Group A*): порты *PA* и *PCh*,

Group B Control — устройство управления группой портов *B* (*Group B*): порты *PB* и *PCl*; устройства управления группами портов *A* и *B* содержат регистр управления, задающий режимы работы портов.

Программирование режимов работы и управление БИС производится микропроцессором посредством сигналов:

D_{7-0} (*Data*) — сигналы разрядов шины данных МП,

A_1, A_0 и $\overline{CS}$ (*Address and Chip Select*) — сигналы двух разрядов шины адреса МП и сигнал с дешифратора адреса разрядов A_{7-2},

$\overline{RD}$ (*Read*) — сигнал чтения данных из *PPI*, $\overline{WR}$ (*Write*) — сигнал записи данных в регистры *PPI* (для МП 8080/8085 $\overline{RD} = \overline{I/OR}$ и $\overline{WR} = \overline{I/OW}$),

RESET — сигнал установки начального состояния регистра управления, задающего режим ввода без квитирования всех портов.

Адресные сигналы A_1 и A_0 дешифрируются внутри *PPI* для селекции одного из четырех внутренних устройств ввода и вывода.

Режимы работы *PPI*. Устройства ввода-вывода порта *PA* содержат два 8-разрядных регистра памяти (регистр ввода и регистр вывода; см. рис. 2.1), порта *PB* — один 8-разрядный регистр памяти, который может переключаться на ввод и вывод (см. рис. 2.2), порта *PC* — один 8-разрядный регистр вывода. В ранее выпускавшейся БИС 8255 (580ВВ55) регистры памяти были построены на асинхронных потенциальных *D–L*-триггерах, а в БИС 8255*A* (580ВВ55А) — на синхронных *D*-триггерах.

Управление вводом-выводом *PPI* представлено в табл. 3.2 (*RGCW — Control Word Register* — регистр управления). Режимы работы портов программируются записью в 7-разрядный регистр памяти *RGCW* слова управления *CW* (*Control Word*), формат которого показан на рис. 3.3, *а*. При записи слова управления все регистры памяти портов *PD* устанавливаются в 0 (*PD* = *PA*, *PB* или *PC* — общее обозначение портов данных). Разряд D_7 для записи *CW* должен быть установлен в 1 ($D_7 = 1$ — флаг установки режима — *Mode Set Flag*).

Таблица 3.2. **Управление вводом-выводом**

$\overline{CS}$	A_1	A_0	$\overline{WR}$	$\overline{RD}$	Операция	Примечание
0	0	0	0	1	$D_{7-0} \rightarrow PA$	Вывод в порт *PA*
0	0	1	0	1	$D_{7-0} \rightarrow PB$	Вывод в порт *PB*
0	1	0	0	1	$D_{7-0} \rightarrow PC$	Вывод в порт *PC*
0	1	1	0	1	$D_{7-0} \rightarrow RGCW$	Запись *CW* в *RGCW*
0	0	0	1	0	$D_{7-0} \leftarrow PA$	Ввод из порта *PA*
0	0	1	1	0	$D_{7-0} \leftarrow PB$	Ввод из порта *PB*
0	1	0	1	0	$D_{7-0} \leftarrow PC$	Ввод из порта *PC*
0	1	1	1	0	Запрещено	Нет операций
0	×	×	1	1	Z-состояние D_{7-0}	Нет операций
1	×	×	×	×	Z-состояние D_{7-0}	Нет операций

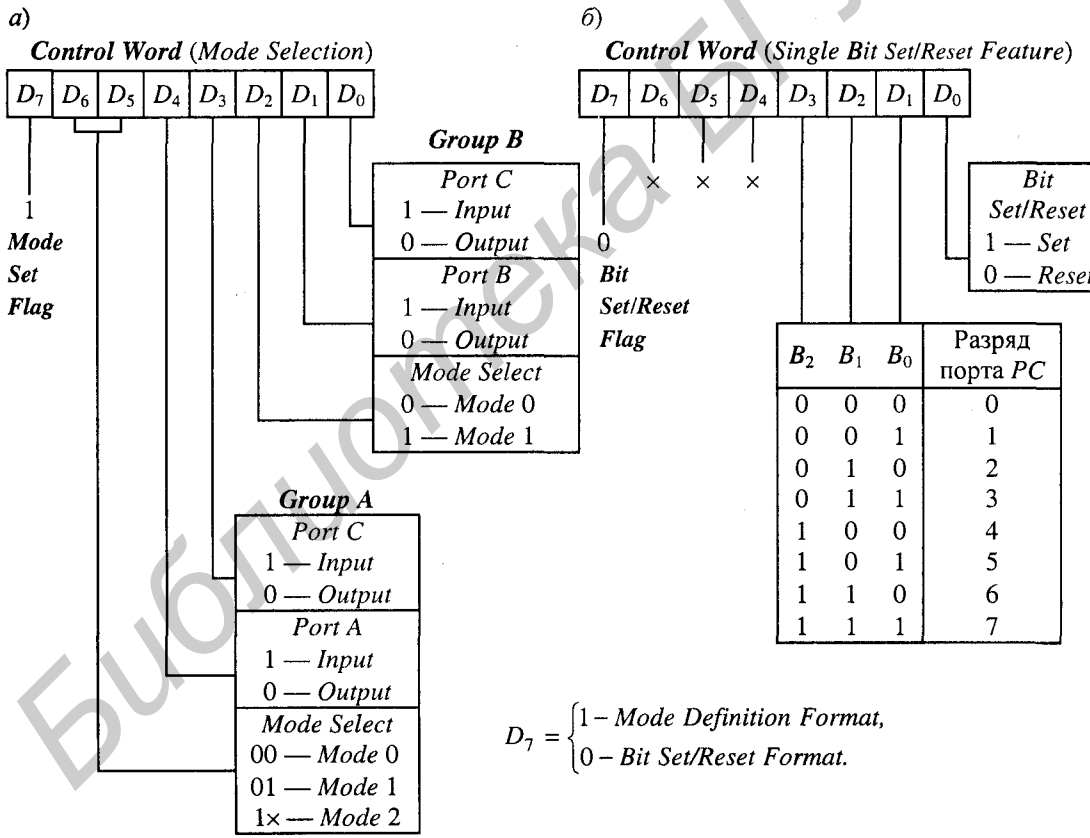


Рис. 3.3. Формат слова управления *CW*

Если разряд $D_7 = 0$ (рис. 3.3, *б*), то слово управления *CW* не записывается в регистр *RGCW*, а производится запись в один из триггеров регистра памяти порта *PC* значения 0 или 1: значения разрядов $D_3D_2D_1 = r = 0 \ldots 7$ задают адрес (номер) триггера в регистре памяти порта *PC*, в который записывается значение 0 или 1, указанное разрядом D_0 ($D_7 = 0$ — флаг поразрядной

установки/сброса — *Bit Set/Reset Flag*). Порт *PC* является в этом случае регистром памяти с адресуемыми разрядами, аналогичным выпускаемому в виде ИС 1533ИР30 (*SN*74*ALS*259) [5].

Итак, разряд D_7 в слове управления *CW* адресует два внутренних регистра: регистр управления *RGCW* и регистр памяти порта *PC*. Такая передача адресных сигналов внутренних узлов БИС по шине данных широко используется для уменьшения числа их внешних выводов.

Разряды D_6D_5 и D_2 в слове управления *CW* при $D_7 = 1$ задают режимы работы M_0 (*Mode* 0), M_1 (*Mode* 1) и M_2 (*Mode* 2) портов ввода и вывода:

M_0 — программный ввод и вывод без квитирования,

M_1 — программный ввод и вывод с квитированием и по прерыванию,

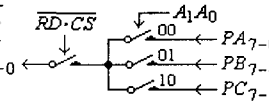
M_2 — программный ввод-вывод (двунаправленный порт) с квитированием и по прерыванию.

Таблица 3.3. **Режимы работы *PPI***

Порт	Режим работы		
	M_0	M_1	M_2
PA	+	+	+
PB	+	+	–
PC	+	–	–

В табл. 3.3 знаками "+" указаны режимы, в которых могут работать порты *PD*. Сигнал *RESET* = 1 устанавливает все порты на ввод в режиме M_0. В этом случае *PPI* будет представлять собой трехканальный 8-разрядный мультиплексор, описываемый функцией

$$D_i = \begin{cases} PA_i\overline{A_1}\cdot\overline{A_0} \vee PB_i\overline{A_1}A_0 \vee PC_iA_1\overline{A_0} \text{ при } \overline{CS}=0 \text{ и } \overline{RD}=0, \\ Z\text{ - состояние при } \overline{CS}=1, \text{ или } \overline{RD}=1, \text{ или } A_1=A_0=1, \end{cases} \Rightarrow$$



где $i = 0, 1, \ldots, 7$.

Режим M_0. В этом режиме данные фиксируются в регистрах памяти портов *PD* только при выводе. При вводе же данные должны удерживаться внешним устройством до тех пор, пока они не будут прочитаны микропроцессором. Регистры вывода портов позволяют производить обратное чтение выведенных данных (см. § 2.1), т. е. командой IN *port_PD* в аккумулятор МП пересылаются данные, записанные в регистр памяти порта *PD* командой OUT *port_PD* (*port_ PD* — адрес порта *PD*).

Пример 1. Пусть в режимах M_0 всех портов требуется записать выданные портом *PA* данные в ячейку памяти M(22AC) и вывести в порты *PB* и *PC* содержимое регистров *D* и *E*. Адреса портов принять равными C0*h* для порта *PA*, C1*h* для порта *PB*, C2*h* для порта *PC* и C3*h* для порта регистра управления.

На рис. 3.4 показано слово управления (составлено на основании рис. 3.3, *а*), задающее ввод из порта *PA* и вывод в порты *PB* и *PC* в режиме M_0 (*GA* — *Group A*, *GB* — *Group B*).

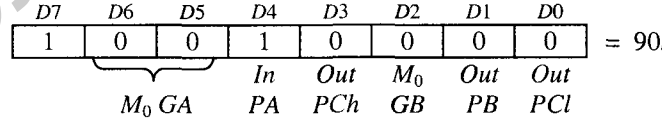


Рис. 3.4. Слово управления для задания режимов M_0 всех портов

Программа, выполняющая указанные операции ввода-вывода, имеет вид:

```
MVI   A, 90h  ; A ← CW = 90h — слово управления для задания режимов работы
OUT   0C3h    ; RGCW ← CW  (C3h = 1100 0011 — адрес RGCW, см. табл. 3.2)
IN    0C0h    ; A ← PA7-0  (C0h = 1100 0000 — адрес порта данных PA)
STA   22ACh   ; M(22AC) ← A = PA7-0
MOV   A, D    ; A ← D
OUT   0C1h    ; PA7-0 ← A = D  (C1h = 1100 0001 — адрес порта данных PB)
```

```
MOV  A, E     ; A ← E
OUT  0C2h     ; PC7-0 ← A = E  (C2h = 1100 0010 — адрес порта данных PC)
```

При использованных адресах портов *PD* и регистра управления *RGCW* БИС 8255*A* должна селектироваться сигналом

$$\overline{CS} = \overline{A_7 A_6 \cdot \overline{A_5} \cdot \overline{A_4} \cdot \overline{A_3} \cdot \overline{A_2}} = \overline{A_7 A_6 \cdot \overline{A_5 \vee A_4 \vee A_3 \vee A_2}}$$

($\overline{CS} = 0$ только при $A_7 = A_6 = 1$, $A_5 = A_4 = A_3 = A_2 = 0$).

Временны́е диаграммы, поясняющие ввод и вывод в режиме M_0, показаны на рис. 3.5 (* — не более нс, остальные — не менее нс).

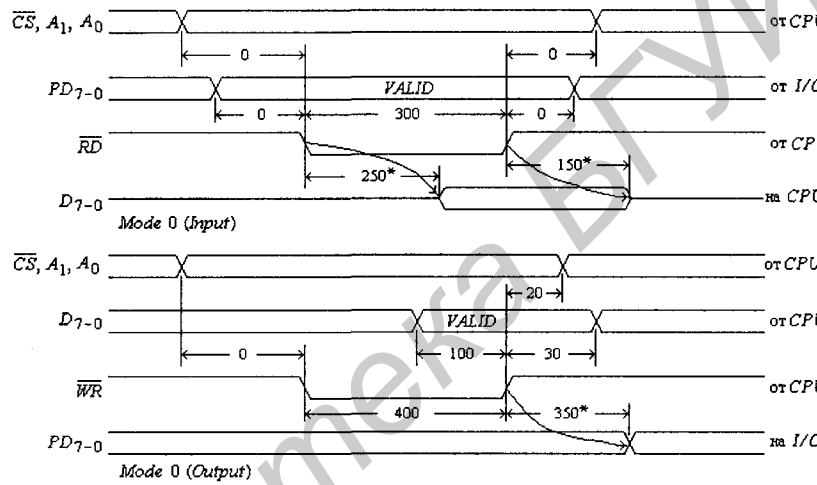


Рис. 3.5. Временны́е диаграммы ввода-вывода в режиме M_0

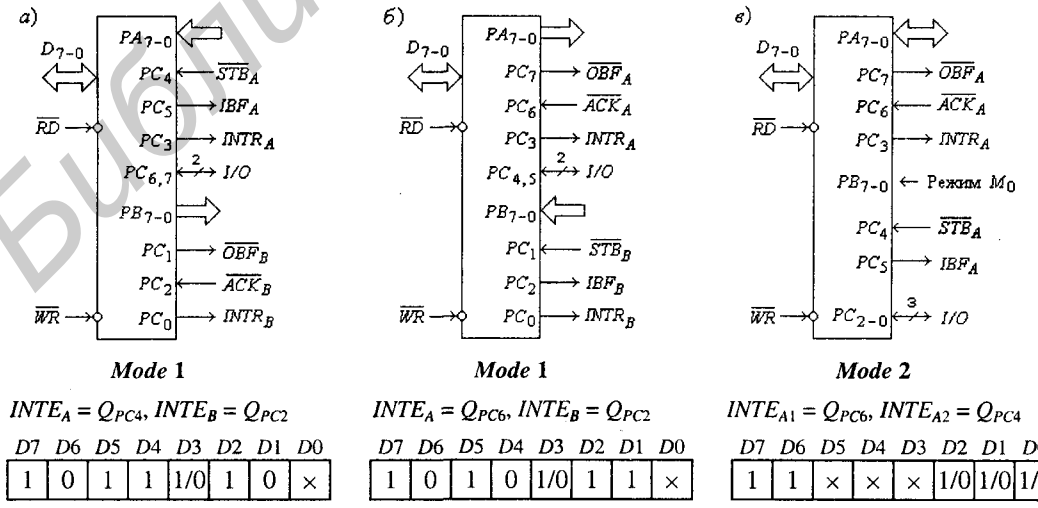


Рис. 3.6. Конфигурации портов ввода-вывода и слово управления *CW*

Режим M_1. В данном режиме могут работать только порты *PA* и *PB*, а порт *PC* обеспечивает сигналы управления для работы с квитированием и по прерыванию. Данные фиксируются в регистрах памяти, как при выводе, так и при вводе. На рис. 3.6 показаны две конфигурации портов ввода/вывода для режима M_1: ввод из порта *PA*, вывод в порт *PB* (рис. 3.6, *а*) и вывод в порт *PA*, ввод из порта *PB* (рис. 3.6, *б*). Можно использовать и другие конфигурации, а также установку режима M_1 только для одного порта.

Часть внешних выводов (контактов) и триггеров регистра памяти порта *PC* обеспечивают сигналы управления для работы с квитированием и по прерыванию:

$\overline{STB_A}$, $\overline{STB_B}$ (*Strobe*) — сигналы загрузки данных в регистры ввода из портов *PA* и *PB*,
IBF_A, IBF_B (*Input Buffer Full*) — сигналы квитирования ввода из портов *PA* и *PB*,
$\overline{OBF_A}$, $\overline{OBF_B}$ (*Output Buffer Full*) — сигналы квитирования вывода в порты *PA* и *PB*,
$\overline{ACK_A}$, $\overline{ACK_B}$ (*Acknowledge Input*) — подтверждение приема данных внешним устройством,

	D7	D6	D5	D4	D3	D2	D1	D0	
CW	1	0	1	1	1/0	1	1	×	PA: Mode 1, Input
SW	I/O	I/O	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$	PB: Mode 1, Input
CW	1	0	1	0	1/0	1	0	×	PA: Mode 1, Output
SW	$\overline{OBF_A}$	$INTE_A$	I/O	I/O	$INTR_A$	$INTE_B$	$\overline{OBF_B}$	$INTR_B$	PB: Mode 1, Output
CW	1	0	1	1	1/0	1	0	×	PA: Mode 1, Input
SW	I/O	I/O	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	$\overline{OBF_B}$	$INTR_B$	PB: Mode 1, Output (рис. 3.6, *а*)
CW	1	0	1	0	1/0	1	1	×	PA: Mode 1, Output
SW	$\overline{OBF_A}$	$INTE_A$	I/O	I/O	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$	PB: Mode 1, Input (рис. 3.6, *б*)

	D7	D6	D5	D4	D3	D2	D1	D0	
CW	1	1	×	×	×	0	1	1/0	PA: Mode 2
SW	$\overline{OBF_A}$	$INTE_1$	IBF_A	$INTE_2$	$INTR_A$	I/O	I/O	I/O	PB: Mode 0, Input
CW	1	1	×	×	×	0	0	1/0	PA: Mode 2
SW	$\overline{OBF_A}$	$INTE_1$	IBF_A	$INTE_2$	$INTR_A$	I/O	I/O	I/O	PB: Mode 0, Output
CW	1	1	×	×	×	1	1	×	PA: Mode 2
SW	$\overline{OBF_A}$	$INTE_1$	IBF_A	$INTE_2$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$	PB: Mode 1, Input
CW	1	1	×	×	×	1	0	×	PA: Mode 2
SW	$\overline{OBF_A}$	$INTE_1$	IBF_A	$INTE_2$	$INTR_A$	$INTE_B$	$\overline{OBF_B}$	$INTR_B$	PB: Mode 1, Output

× — безразличные значения, *I/O* — вводимые или выводимые данные порта *PC*,
1/0 — задание режима ввода (1) или режима вывода (0) порта *PC*.

Рис. 3.7. Форматы слов управления *CW* и состояния *SW*

$INTR_A = INTE_A \cdot IBF_A = Q_{PC4} \cdot IBF_A$ — сигнал запроса прерывания для ввода из порта *PA*,

$INTR_A = INTE_A \cdot \overline{OBF}_A = Q_{PC6} \cdot \overline{OBF}_A$ — сигнал запроса прерывания для вывода в порт *PA*,

$INTR_B = INTE_B \cdot IBF_B = Q_{PC2} \cdot IBF_B$ — сигнал запроса прерывания для ввода из порта *PB*,

$INTR_B = INTE_B \cdot \overline{OBF}_B = Q_{PC2} \cdot \overline{OBF}_B$ — сигнал запроса прерывания для вывода в порт *PB*,

где Q_{PC4}, Q_{PC6} и Q_{PC2} — триггеры 4, 6 и 2 регистра памяти порта *PC*; $INTR_A$, $INTR_B$ (*Interrupt Request*) — сигналы запроса прерывания, подаваемые на контроллер прерываний 8259; $INTE_A$, $INTE_B$ (*Interrupt Enable*) — разрешение прерывания (разрешение генерации сигналов запроса прерывания $INTR_A$ и $INTR_B$).

Значения сигналов квитирования IBF_A, $\overline{OBF}_A$, IBF_B, $\overline{OBF}_B$ и др. дублируются в регистре слова состояния *SW* (*Status Word*), в качестве которого используется регистр памяти порта *PC* (рис. 3.7).

Чтение микропроцессором слова состояния *SW* командой IN *port_PC* , где *port_PC* — адрес порта *PC*, позволяет обеспечить квитирование ввода и вывода с помощью сигналов *IBF* и $\overline{OBF}$. Значения сигналов *INTE* устанавливаются с помощью операции поразрядной установки/сброса триггеров регистра памяти порта *PC*. Незанятые под управляющую информацию разряды регистра памяти порта *PC* могут использоваться для ввода или вывода в режиме M_0 (разряды *I/O* на рис. 3.7).

Временны́е диаграммы ввода и вывода в режиме M_1 показаны на рис. 3.8. Активный уровень сигналов *IBF* = 1 устанавливается сигналами $\overline{STB}$ = 0, а сбрасываются эти флаги в 0 положительным фронтом сигнала $\overline{RD}$. Активный уровень сигналов $\overline{OBF}$ = 0 устанавливается положительным фронтом сигнала $\overline{WR}$, а неактивный уровень $\overline{OBF}$ = 1 — сигналами $\overline{ACK}$ = 0. Сигналы квитирования *IBF* и $\overline{OBF}$ подаются на внешние устройства, подключенные к портам *PA* и *PB*, для синхронизации операций ввода и вывода.

Пример 2. Пусть требуется производить ввод данных из порта *PB* с квитированием, а по остальным портам — вывод в режиме M_0. Адреса портов принять равными C0*h* для порта *PA*, C1*h* для порта *PB*, C2*h* для порта *PC* и C3*h* для порта регистра управления.

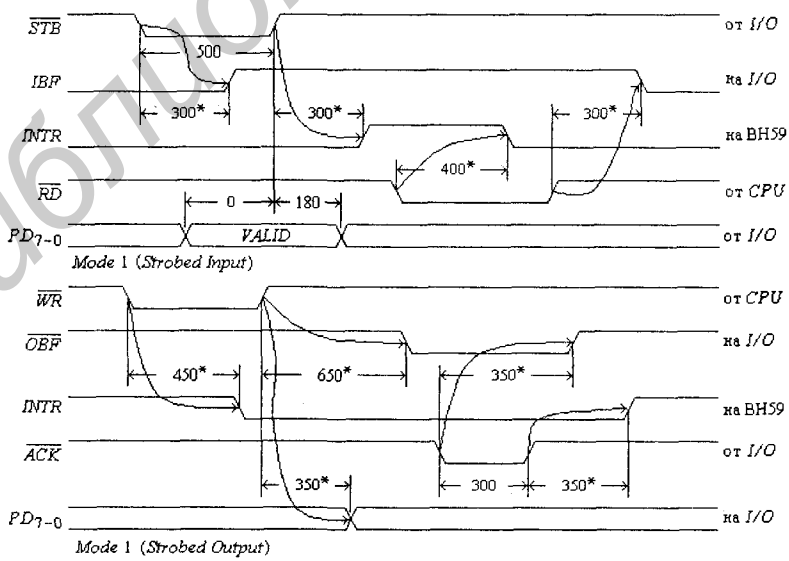


Рис. 3.8. Временны́е диаграммы ввода-вывода в режиме M_1

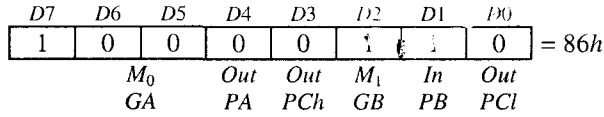


Рис. 3.9. Слово управления *CW*

Режимы работы устанавливаются словом управления *CW*, показанным на рис. 3.9 (*GA — Group A*, *GB — Group B*). Программирование заданных режимов работы портов и ввод данных с квитированием можно реализовать в виде:

```
        MVI   A, 86h   ; A ← CW = 86h — слово управления для задания режимов работы
        OUT   0C3h     ; RGCW ← CW — программирование режимов работы
        CALL  Cin      ; Вызов подпрограммы ввода данных из порта PB
          ∴
Cin:    IN    0C2h     ; A ← SW — чтение регистра состояния
        ANI   2        ; Выделение флага IBF_B (см. рис. 3.7)
        JZ    Cin      ; Переход, если IBF_B = 0
        IN    0C1h     ; A ← PB7–0 — чтение данных из порта PB
        RET
```

Cin — подпрограмма ввода данных из порта *PB* с квитированием

Для ввода и вывода по прерыванию флаги *INTE* устанавливаются в 1 словами управления *CW* при $D_7 = 0$ и $D_0 = 1$ с помощью команд OUT *port_PC* (рис. 3.10). Для сброса этих флагов следует задать $D_0 = 0$.

Control Word (*Single Bit Set/Reset Feature*)

D7	D6	D5	D4	D3	D2	D1	D0	CW	
0	×	×	×	1	0	0	1	= 09	— установка $INTE_A = 1$ для ввода
0	×	×	×	1	0	0	0	= 08	— сброс $INTE_A$ для ввода в 0
0	×	×	×	1	1	0	1	= 0D	— установка $INTE_A = 1$ для вывода
0	×	×	×	1	1	0	0	= 0C	— сброс $INTE_A$ для вывода в 0
0	×	×	×	0	1	0	1	= 05	— установка $INTE_B = 1$
0	×	×	×	0	1	0	0	= 04	— сброс $INTE_B$ в 0

Рис. 3.10. Управление разрешением прерываний

Пример 3. Пусть требуется производить вывод данных блоками по 64 байта из ячеек памяти с начальным адресом 4000*h* в порт *PA* с квитированием по прерыванию при подключении выхода $INTR_A$ БИС 8255*A* к входу RST 7.5 МП 8085*A*. Остальные порты запрограммировать в режим M_0 на вывод. Адреса портов принять равными C0*h* для порта *PA*, C1*h* для порта *PB*, C2*h* для порта *PC* и C3*h* для порта регистра управления.

Режимы работы устанавливаются словом управления $CW = 1010\,0000 = A0h$ (см. рис. 3.3, *а*). Программирование заданных режимов работы портов и вывод данных с квитированием по прерыванию можно реализовать программой:

```
MVI   A, 0A0h  ; A ← CW = A0h — слово управления для задания режимов работы
OUT   0C3h     ; RGCW ← CW — программирование режимов работы
MVI   A, 0Dh   ; A ← CW = 0Dh — слово управления для задания значения INTE_A = 1
OUT   0C3h     ; Q_PC6 ← 1 — установка значения INTE_A = 1 для вывода
  ∴
```

```
003C  PUSH  PSW      ; 003Ch — адрес вызова подпрограммы обслуживания прерываний
      PUSH  H        ;         по входу RST 7.5
      PUSH  B
      LXI   H, 4000h ; rp H ← 4000h — начальный адрес памяти
      MVI   B, 40h   ; 40h — задание вывода 64 байт
COUT: IN    0C2h     ; Чтение регистра состояния SW
      ANI   80h      ; Выделение флага OBF_A (см. рис. 3.7)
      JZ    COUT     ; Переход, если OBF_A = 0
      MOV   A, M     ; A ← M(rp H)
      OUT   0C0h     ; PA7-0 ← A — вывод данных в порт PA
      INX   H
      DCR   B
      JNZ   COUT
      POP   B        ; Восстановление состояния прерванной программы
      POP   H
      POP   PSW
      EI             ; Разрешение прерываний
      RET            ; Возврат из подпрограммы обработки прерывания
```

Режим M_2 порта *PA*. Данный режим позволяет производить ввод и вывод по двунаправленной шине данных без перепрограммирования режимов работы *PPI*. Конфигурация порта *PA* в режиме M_2 (и *Mode* 0 для порта *PB*) показана на рис. 3.6, *в*, а временны́е диаграммы, поясняющие работу порта — на рис. 3.11. Режим M_2 объединяет ввод в режиме M_1 и вывод в режиме M_1. Принципиальная схема подобного порта ввода-вывода (приемопередатчика с буферными регистрами) была приведена на рис. 2.17.

Назначение всех сигналов управления остается прежним (как в режиме M_1). Сигнал запроса прерывания

$$INTR_A = Q_{PC4} \cdot IBF_A \vee Q_{PC6} \cdot \overline{OBF_A}.$$

Форматы слов состояния *SW* для порта *PA* в режиме M_2 показаны на рис. 3.7. Этот режим используется для работы с внешними устройствами, имеющими двунаправленную шину данных.

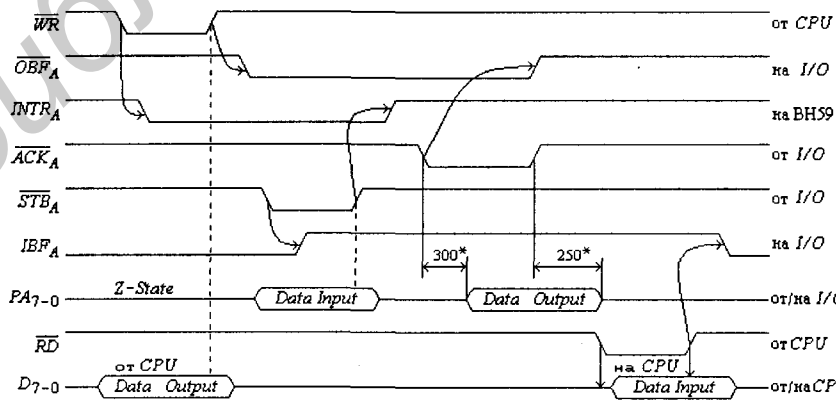


Рис. 3.11. Временны́е диаграммы работы порта *PA* в режиме M_2

3.3. Программатор *EPROM* 573РФ2 и 573РФ5

В настоящее время зарубежными фирмами выпускается несколько типов постоянных запоминающих устройств (ПЗУ), или энергонезависимых ЗУ (*Non-volatile Memories*): *ROM* (*Read Only Memory*) — масочное ПЗУ, программируемое при изготовлении; *OTPROM*, или *PROM* (*One Time Programmable ROM*) — однократно программируемое ПЗУ; *EPROM* (*Erasable Programmable ROM*), или *UV-EPROM* (*Ultra-Violet EPROM*) — стираемое ультрафиолетовыми лучами многократно программируемое ПЗУ (изобретение фирмы *Intel*); *EEPROM* (*Electrically Erasable Programmable ROM*) — электрически стираемое многократно программируемое ПЗУ; *Flash Memory* — флэш-память с внутрисистемным электрическим стиранием и программированием (прямо в оборудовании, в котором она установлена).

Принцип работы *EPROM*. На рис. 3.12 изображена упрощенная модель одной ячейки памяти *EPROM*, поясняющая принцип ее работы. При программировании *EPROM* в плавающий затвор инжектируются генерируемые в канале МОП-транзистора электроны, которые под воздействием приложенного к управляющему затвору высокого напряжения разгоняются в канале, накапливая энергию, достаточную для прохождения сквозь тонкий слой затворного окисла. Накопленные в плавающем затворе электроны экранируют действие управляющего затвора, и МОП-транзистор ток не проводит. При чтении такой ячейки на выходе ПЗУ будет получен логический 0. В исходном состоянии сигналы на всех выходах равны 1. В корпусе БИС имеется окно, закрытое кварцевым стеклом, через которое проходит ультрафиолетовое излучение. При воздействии ультрафиолетовых лучей электроны приобретают энергию и покидают плавающий затвор. Длительность стирания равна примерно 30 мин при длине волны 400 нм и энергетической освещенности 100 Вт/м^2. Для стирания обычно используются кварцевые лампы ДРТ-220 и ДРТ-375. Срок хранения информации составляет 15 000 … 25 000 ч.

На рис. 3.13 изображены *EPROM* 2К × 8 бит типа 2716 (573РФ2, 573РФ5) и 4К × 8 бит типа 2732*A* фирмы *Intel*:

A_j (*Address*) — разряды адреса;

$O_i = DO_i$ (*Data Output*) — разряды данных (при программировании эти выводы двунаправленные);

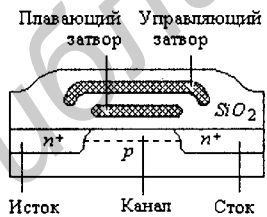


Рис. 3.12. Модель ячейки памяти *EPROM*

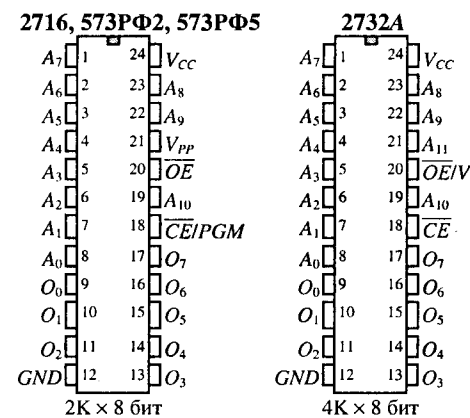


Рис. 3.13. *EPROM* фирмы *Intel*

$\overline{CE}/PGM$ (*Chip Enable/Program Strobe*) — выбор кристалла/строб программирования;

$\overline{OE}$ (*Output Enable*) — разрешение выхода ($\overline{OE} = \overline{MEMR}$ при чтении).

На рис. 3.14, *а* показаны временны́е диаграммы работы *EPROM* 2716 в режиме чтения, а на рис. 3.14, *б* — в режиме программирования и контроля (* — время не более, остальные — не менее). При программировании на контакт 21 подается напряжение питания V_{PP} = 25 В, а на контакт 18 — программирующий импульс $\overline{CE}/PGM$ = 1 длительностью 50 мс. В режиме чтения

на контакт 21 подается напряжение $V_{PP} = V_{CC} = +5$ В. Чтение при контроле (верификации) в режиме программирования производится при $V_{PP} = +25$ В.

Программатор *EPROM* 573РФ2/РФ5. Структурная схема программатора, подключенного к шинам микроЭВМ, показана на рис. 3.15. Для микроЭВМ программатор представляет собой обычное внешнее устройство, в которое производится вывод данных, представляющих собой сигналы управления $\overline{OE}$ и $\overline{CE}/PGM$, адреса ячеек памяти *EPROM* a_{10-0} и записываемые в них данные d_{7-0}. При проверке правильности записи данных в *EPROM* (*Verify* — верификация) производится его чтение — ввод данных в МП. Данные для программирования должны быть предварительно подготовлены в *RAM* (конечно, данные можно брать и из *ROM*).

Оператор с помощью клавиатуры вводит в микроЭВМ три параметра: *Abeg* — начальный адрес массива данных в *RAM*, которые необходимо записать в *EPROM*; *Aend* — конечный адрес массива данных в *RAM*; *Apgm* — начальный адрес *EPROM* (можно вводить 16-разрядный адрес, хотя *EPROM* имеет 11-разрядный адрес). Будем полагать, что эти параметры поступают в регистровые пары *rp H*, *B* и *D*.

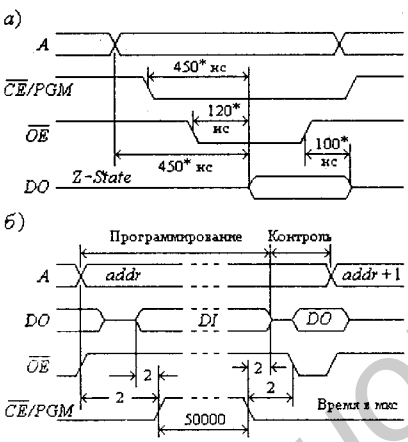


Рис. 3.14. Временные диаграммы для режимов чтения (*а*) и программирования *EPROM* (*б*)

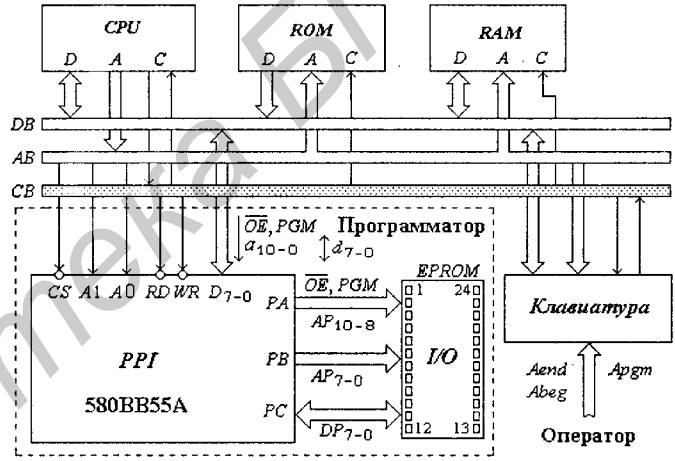


Рис. 3.15. Структурная схема программатора

Принципиальная схема программатора, соответствующая описанной выше структурной схеме, изображена на рис. 3.16. Сигналы адресации ячеек памяти *EPROM* $AP_{10-0} = a_{10-0}$, а программируемых данных $DP_{7-0} = d_{7-0}$ (рис. 3.15). Программатор выполнен на отдельной плате, поэтому он снабжен своим приемопередатчиком (ИС 1533АП6) для подключения к системной шине данных микропроцессорных устройств, приведенных на рис. 1.10 и 1.32.

Программатор снабжен преобразователем напряжения +12 В в напряжение +25 В (10, 10, 31 — число витков трансформатора, выполненного на ферритовом кольце; 142ЕН2Б — стабилизатор напряжения с защитой от перегрузок по току и от короткого замыкания). Мощные ЛЭ 155ЛА18 с открытым коллекторным выходом обеспечивают ток $I_{OL\,max} = 300$ мА. Для *EPROM* 2716 при напряжении программирования $V_{PP} = +25$ В ток потребления $I_{PP} = 30$ мА (в импульсном режиме, когда сигнал $\overline{CE}/PGM = 1$). В основных режимах работы *EPROM* 2716 (при значениях $V_{CC} = +5$ В и $V_{PP} = +5$ В) токи потребления имеют значения: $I_{CC} = 100$ мА — в режиме чтения и $I_{CC} = 25$ мА — в режиме хранения.

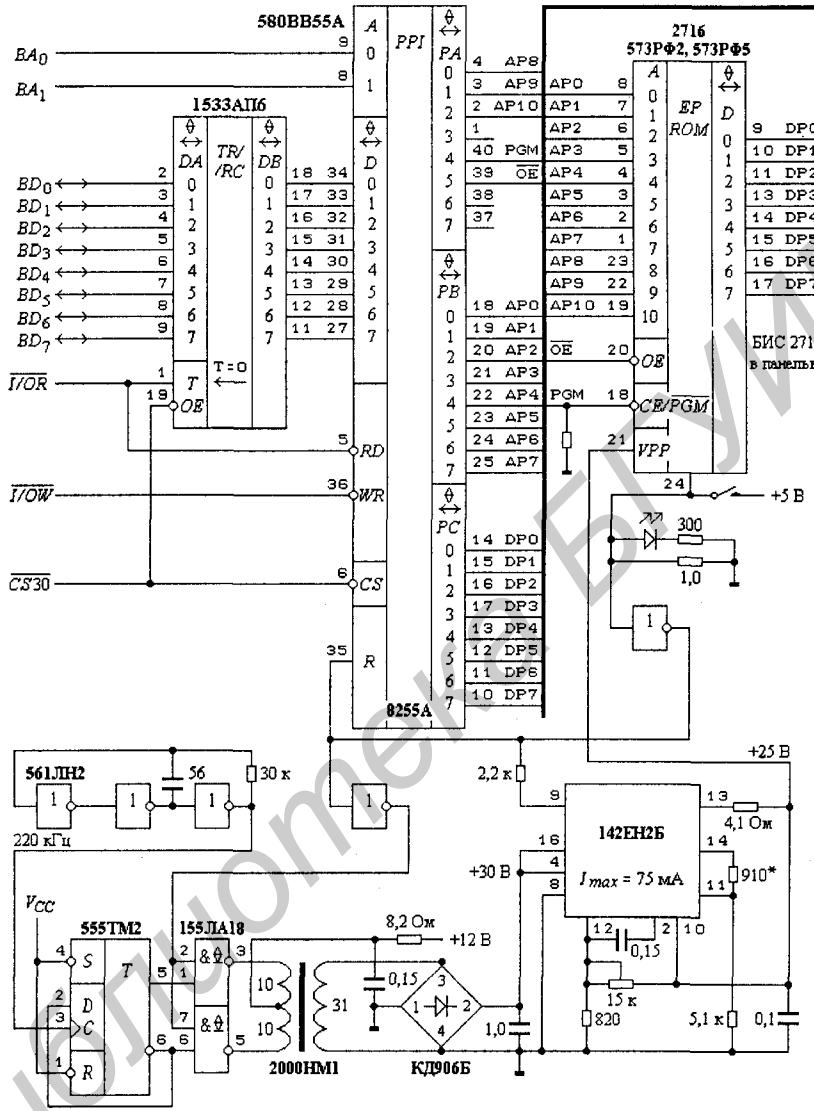


Рис. 3.16. Принципиальная схема программатора

Не составляет труда ввести программное управление включением и выключением напряжения программирования +25 В. Для этого требуется добавить один триггер для программного управления ЛЭ 155ЛА18.

На рис. 3.17 представлена блок-схема алгоритма программирования *EPROM* 573РФ2/РФ5 для программатора, изображенного на рис. 3.16. Основой для этой блок-схемы служат информация, представленная временны́ми диаграммами на рис. 3.14, *б*, и знания, полученные при изучении БИС 580ВВ55А (8255*A*).

Во-первых, требуется выбрать метод ввода-вывода данных. Так как *EPROM* немедленно готово принимать и выдавать данные, то следует использовать самый простой метод ввода-вывода — программный ввод-вывод без квитирования. Это означает, что для *PPI* 580ВВ55А

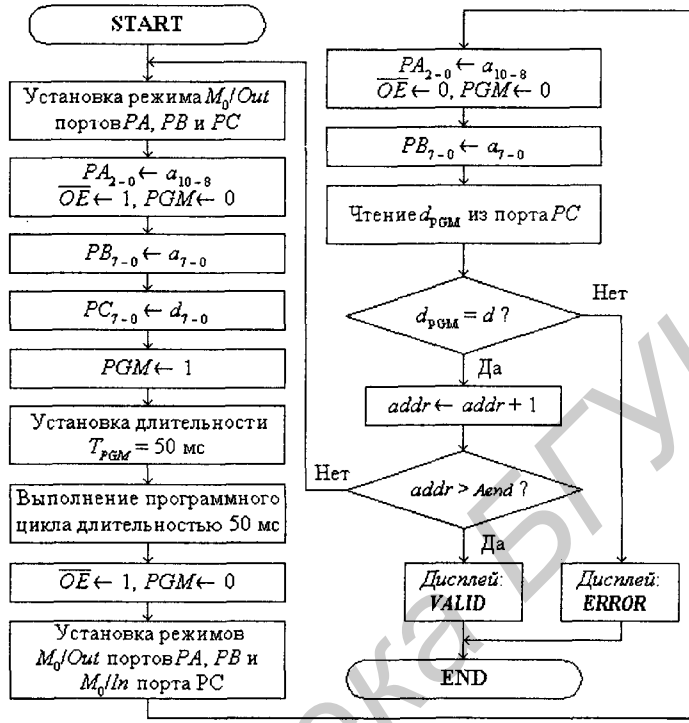


Рис. 3.17. Блок-схема алгоритма программирования *EPROM*

следует выбрать режим работы M_0 для всех портов. Не следует забывать, что при перепрограммировании режима работы порта *PC* на вывод и ввод регистры всех портов сбрасываются в 0.

По блок-схеме алгоритма, представленной на рис. 3.17, не составляет труда написать программу:

```
; Исходные параметры: rp H = Abeg, rp B = Aend, rp D = Apgm,
; port = 30, 31, 32, 33 — адреса портов PA, PB, PC и RGCW соответственно
      MOV   A, C      ; Вычисление объема массива ΔA программируемых ячеек памяти
      SUB   L
      MOV   C, A
      MOV   A, B
      SBB   H
      MOV   B, A
      INX   B         ; rp B = Abeg – Aend + 1 = ΔA — объем массива данных для EPROM
L3:   PUSH  B
      MVI   A, 80h    ; CW = 80h = 1000 0000 — режим M0/вывод для портов PA, PB и PC
      OUT   33h       ; RGCW ← CW, port = 33h — адрес порта RGCW
      MOV   A, D
      ANI   7
      MOV   D, A      ; rp D = 0000 0a10a9a8 a7a6a5a4 a3a2a1a0 — 11-разрядный адрес EPROM
      ORI   20h
      OUT   30h       ; PA ← 00OEPGM 0a10a9a8 (OE = 1, PGM = 0)
```

```
     MOV   A, E
     OUT   31h      ; PB ← a7a6a5a4 a3a2a1a0
     MOV   A, M     ; A ← d = d7d6d5d4 d3d2d1d0 — байт данных для EPROM,
     OUT   32h      ; PC ← d                          rp H — addr RAM
     MOV   A, D
     ORI   30h
     OUT   30h      ; PA ← 00OEPGM0a10a9a8 (OE = 1, PGM = 1)
     LXI   B, 1045h ; Задание длительности T_PGM = 50 мс
L1:  DCX   B
     MOV   A, B
     ORA   C
     JNZ   L1
     MOV   A, D
     ORI   20h
     OUT   30h
     MVI   A, 89h   ; CW = 89h = 1000 1001 — режимы M0/вывод для портов PA, PB и
     OUT   33h      ; M0/ввод для порта PC
     MOV   A, D
     OUT   30h      ; PA ← 00OEPGM 0a10a9a8 (OE = 0, PGM = 0)
     MOV   A, E
     OUT   31h      ; PB ← a7a6a5a4 a3a2a1a0
     IN    32h      ; A ← d_PGM = (d7d6d5d4 d3d2d1d0)PGM — чтение байта данных из EPROM
     CMP   M        ; Сравнение d_PGM с d = d7d6d5d4 d3d2d1d0
     JNZ   L2       ; Переход, если d_PGM ≠ d (ERROR)
     INX   D
     INX   H        ; rp H ← addr + 1
     POP   B        ; rp B ← ΔA
     DCX   B        ; rp B ← ΔA – 1
     MOV   A, B     ; Проверка: addr > Aend ?
     ORA   C
     JNZ   L3
      ∴             ; Вывод на дисплей сообщения VALID
     JMP   L4
L2:   ∴             ; Вывод на дисплей сообщения ERROR и адреса
L4:  HLT            ;       отказавшей ячейки памяти
```

Задать программирование можно от одной до 2048 ячеек памяти. При необходимости можно без стирания перепрограммировать любую ячейку памяти, если требуется значения 1 некоторых разрядов байта данных заменить значениями 0. Время программирования всех ячеек памяти с использованием этой программы составляет 108 с. Время программирования пропорционально числу ячеек памяти и становится неприемлемо большим уже для *EPROM* с объемом памяти 8K × 8 бит. Поэтому фирмой *Intel* были разработаны усовершенствованные (быстрые) методы программирования *Inteligent Programming*™ и *Quick-Pulse Programming*™. Естественно, что и сами *EPROM* должны быть спроектированы с поддержкой этих методов программирования.

***EPROM* фирмы *Intel*.** Дальнейшее усовершенствование *EPROM* происходило с учетом требований: увеличение объема памяти и быстродействия, снижение напряжения программирования (V_{PP} = +12 В) и тока потребления I_{CC} от источника питания V_{CC} = +5 В в расчете на

один бит информации. Данные требования привели, в частности, к разработке новой высококачественной *CMOS*-технологии.

На рис. 3.18 представлены *EPROM* фирмы *Intel*, широко используемые при проектировании МП-систем.

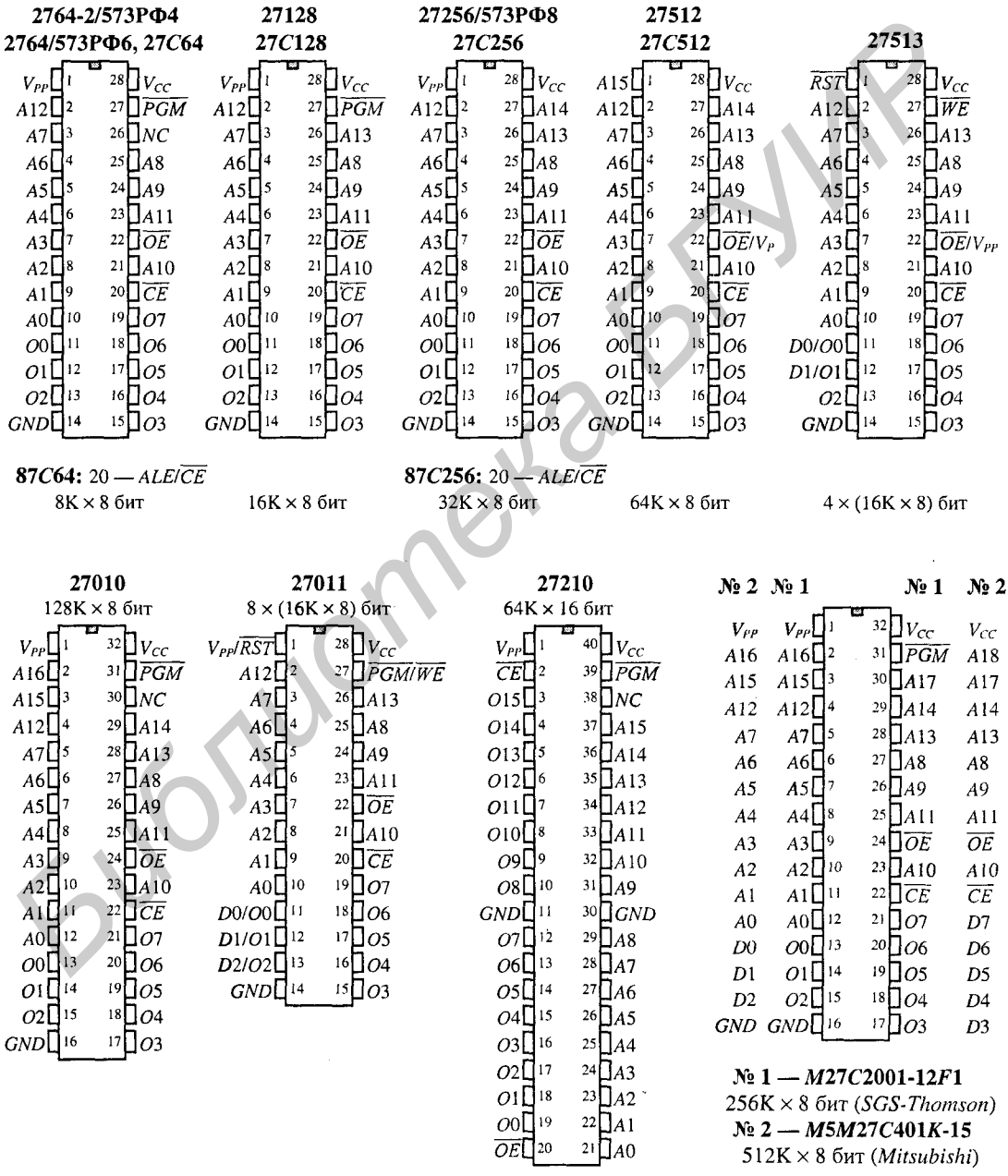


Рис. 3.18. *EPROM* фирмы *Intel*

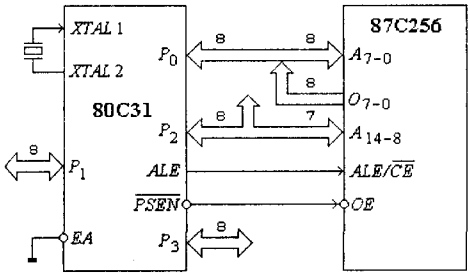


Рис. 3.19. Подключение *EPROM* 87C256 к микроконтроллеру 80*C*31

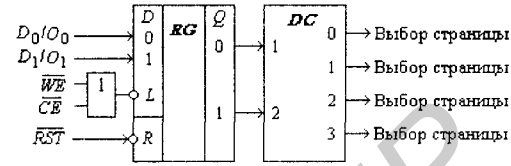


Рис. 3.20. Схема выбора страниц памяти в *EPROM* 27513

В *EPROM* 87*C*××× введены регистры хранения адреса: при обращении к этим БИС фронт ⅂ сигнала выбора кристалла $ALE/\overline{CE}$ = ⅂⌐ записывает установленные значения разрядов адреса во внутренний регистр (разряды A_{12-0} — для *EPROM* 87*C*64 и A_{14-0} — для *EPROM* 87*C*256). Такие *EPROM* позволяют уменьшить число ИС в МП-системах, построенных на основе МП и микроконтроллерах, имеющих мультиплексную шину адреса-данных (в частности, их удобно использовать совместно с МП 8085*A*). На рис. 3.19 изображена структурная схема подключения *EPROM* 87*C*256 к микроконтроллеру 80*C*31 в качестве внешней памяти программ.

В *EPROM* 27513 заложены четыре страницы (банка) памяти, выбор которых производится программным способом — записью во внутренний двухразрядный регистр адреса страницы по линиям двух разрядов данных D_1/O_1 и D_0/O_0, т. е. *EPROM* дополнена функцией внешнего устройства (устройства вывода). Для управления страницами памяти используются сигналы $\overline{WE}$ (*Page-Select Write Enable*) — разрешение записи для селекции страницы, $\overline{RST}$ (*Page* 0 *Reset*) — сброс на страницу 0, D_k/O_k (*Input/Outputs*) — задание номера страницы (рис. 3.20).

Таблица 3.4. **Параметры *EPROM* фирмы *Intel***

Параметр	2732*A*-2	27*C*64-1 87*C*64-1	27128*A*-1	27*C*256-2 [1] 87*C*256-2	27512-2	27010-200	27210-150
V_{OL}/I_{OL}, В/мА	0,45/2,1	0,45/2,1	0,45/2,1	0,45/2,1	0,45/2,1	0,45/2,1	0,45/2,1
V_{OH}/I_{OH}, В/мА	2,4/–0,4	3,5/–2,5	2,4/–0,4	3,5/–2,5	2,4/–0,4	2,4/–0,4	2,4/–0,4
I_{CC1}/I_{CC2}, мА	100/100	20/30	100/100	30/30	125/125	150/150	150/160
I_{SB}, мА	35	1,0	40	1,0	40	50	40
I_{PP1}/I_{PP2}, мА	–/30	0,1/30	5/50	0,2/50	–/40	0,001/50	0,001/50
t_{CE}, нс (*max*)	200	150	150	200	200	200	150
t_{OE}, нс (*max*)	70	75	65	75	75	85	65
t_{DF}, нс (*max*)	60	35	55	55	55	60	50
t_{PW}/t_{PGM}, мс/с	50/205	1/33 [2]	1/66 [2]	1/132 [2]	1/264 [2]	—	—
t_{PW}/t_{PGM}, мкс/с	—	100/1 [3]	100/2 [3]	100/4 [3]	100/8 [3]	100/15 [3]	100/8 [3]
Технология	*HMOS-E*	*HCMOS II-E*	*HMOS II-E*	*HCMOS II-E*	*HMOS II-E*		
Объем памяти	4K × 8	8K × 8	16K × 8	32K × 8	64K × 8	128K × 8	64K × 16

Примечание: [1] t_{CE} = 170 нс, t_{OE} = 70 нс, t_{DF} = 55 нс для 27*C*256–1; [2] *Inteligent Programming*™; [3] *Quick-Pulse Programming*™.

Основные параметры некоторых *EPROM* приведены в табл. 3.4:

V_{OL}/I_{OL} — напряжение выходного сигнала низкого уровня при указанном значении выходного тока,

V_{OH}/I_{OH} — напряжение выходного сигнала высокого уровня при указанном значении выходного тока,

I_{CC1} и I_{PP1} — токи потребления от источников питания V_{CC} и V_{PP} при чтении данных ($V_{PP} = V_{CC}$),

I_{SB} — ток потребления от источника питания V_{CC} невыбранной БИС ($\overline{CE} = 1$),

I_{CC2} и I_{PP2} — токи потребления от источников питания V_{CC} и V_{PP} при программировании,

t_{CE} — задержка от входа $\overline{CE}$ (⎍) до выходов O_i,

t_{OE} — задержка от входа $\overline{OE}$ (⎍) до выходов O_i,

t_{DF} — задержка от входа $\overline{OE}$ (⎍) до перехода выходов O_i в *Z*-состояние (*DF — Delay Float*),

t_{PW} — длительность активного уровня сигнала программирования,

t_{PGM} — теоретическое минимальное время программирования всех ячеек *EPROM* без учета времени верификации.

На рис. 3.21 изображены временны́е диаграммы для режима программирования различных *EPROM*, позволяющие спроектировать общую схему программатора и правильно написать для него программное обеспечение — интервалом времени t_{PW} отмечены сигналы и их значения, которые производят программирование *EPROM* ($\overline{CE} = 1$ — 2716, $\overline{CE} = 0$ — 2732, 27256, 27512 и 27513, $\overline{PGM} = 0$ — 2764 ... 27210).

Блок-схемы алгоритмов усовершенствованных методов программирования *Inteligent Programming*™ и *Quick-Pulse Programming*™ приведены на рис. 3.22 и 3.23 соответственно.

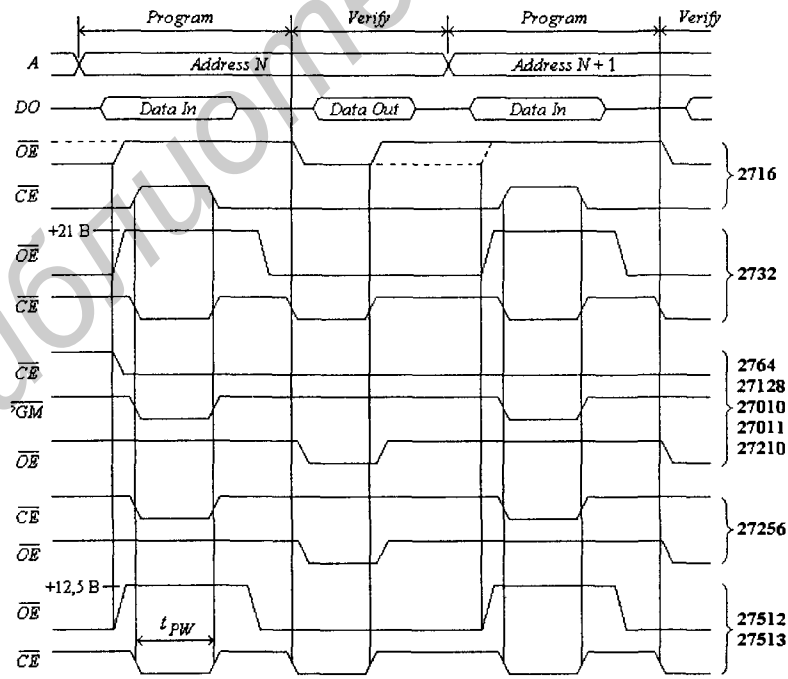


Рис. 3.21. Временны́е диаграммы для режима программирования *EPROM*

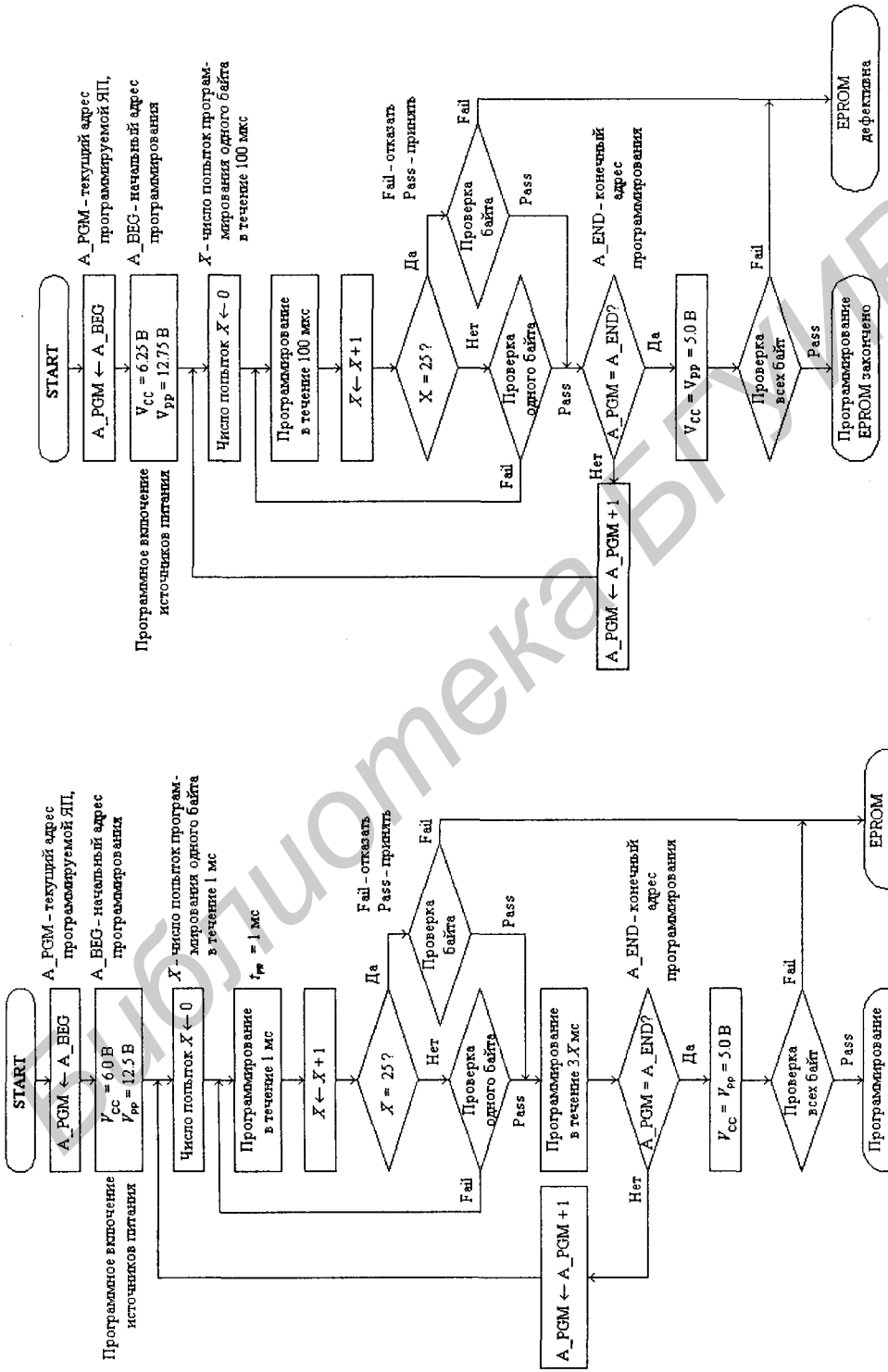


Рис. 3.22. Блок-схема алгоритма *Inteligent Programming*™

Рис. 3.23. Блок-схема алгоритма *Quick-Pulse Programming*™

На рис. 3.24 приведено расположение и назначение контактов *EPROM* фирмы *Intel*, выпускаемых в корпусах типа *PLCC* (*Plastic Leaded Chip Carrier*), отличающихся от корпусов типа *PDIP* (*Plastic Dual-In-Line Package*), изображенных на рис. 3.18. Неиспользуемые контакты обозначаются аббревиатурой *NC* (*No Internal Connection*) или *DU* (*Don't Use*).

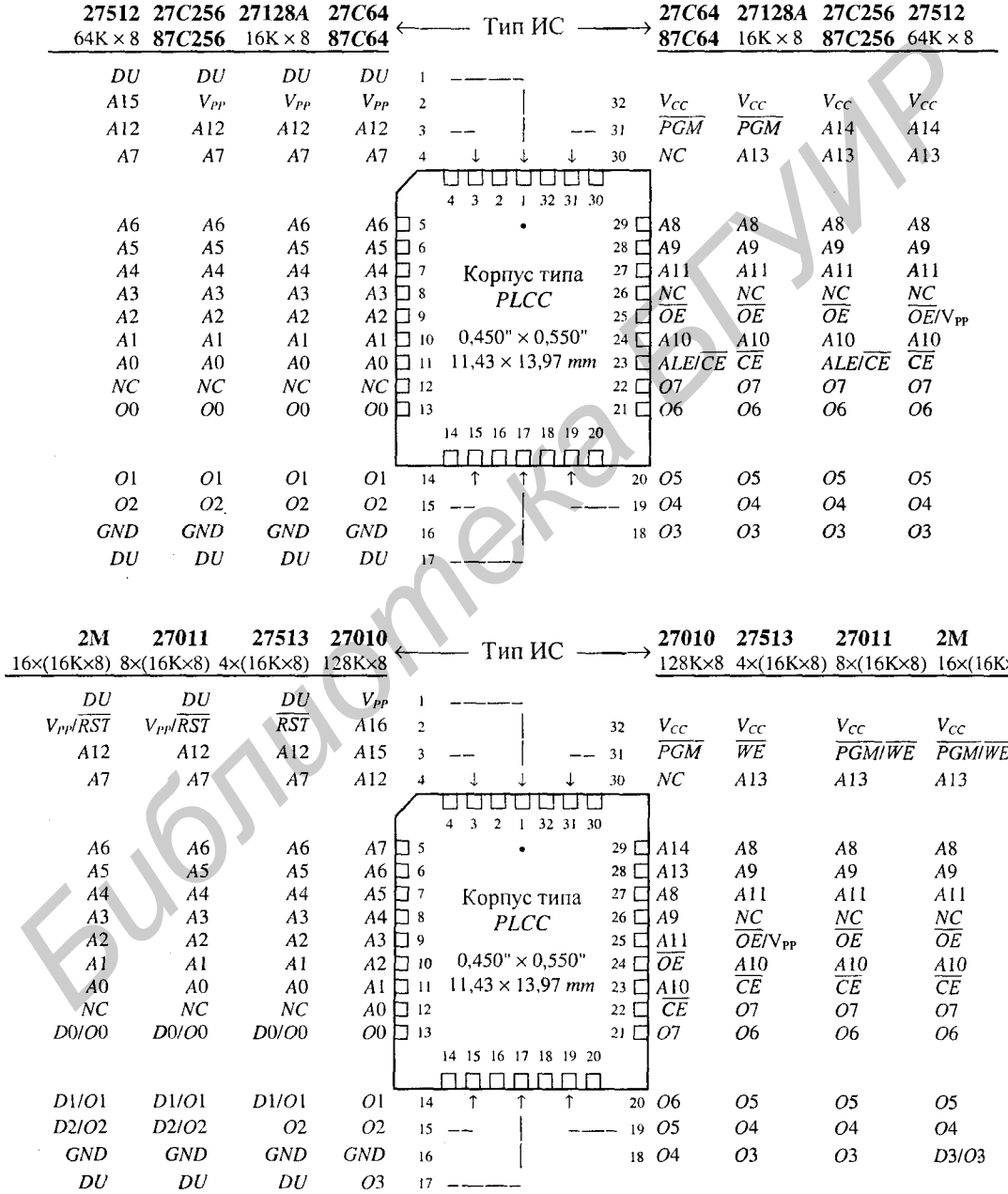


Рис. 3.24. *EPROM* фирмы *Intel*

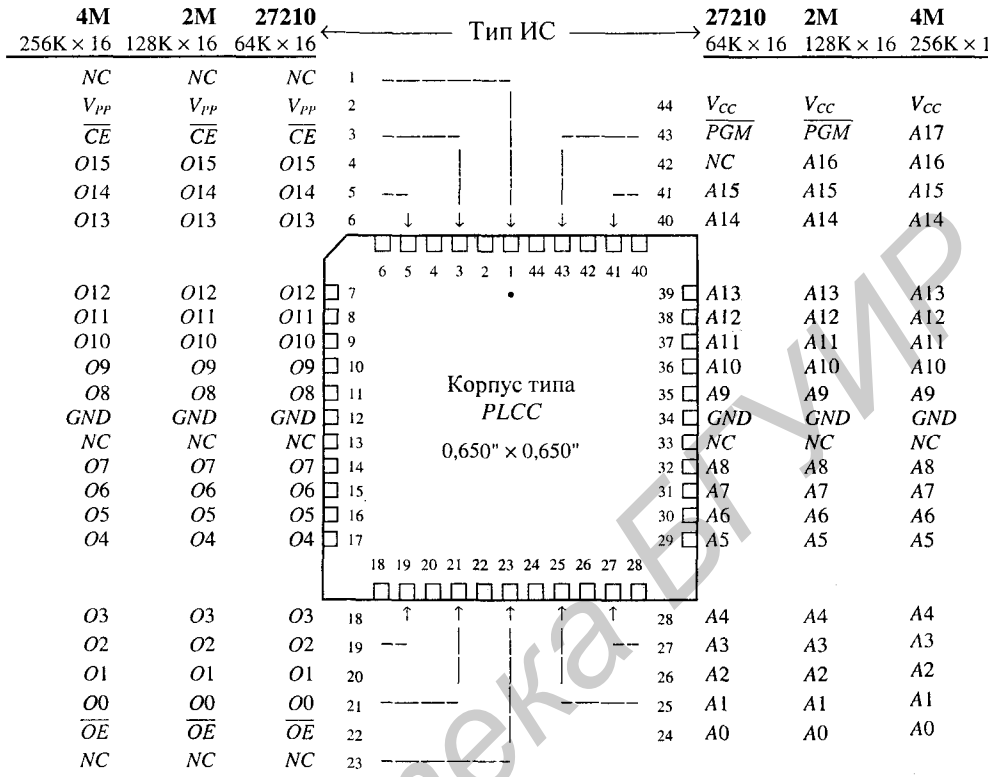


Рис. 3.24 (продолжение)

3.4. Программируемые таймеры 8253 и 8254

Таймеры (рис. 3.25) в МП-системах применяются для создания точных задержек сигналов, генерации временных интервалов и проектирования делителей частоты с программируемым коэффициентом деления под управлением программного обеспечения. Кроме того, таймеры могут использоваться в качестве счетчиков событий, часов реального времени, цифровых ждущих мультивибраторов и др. Таймеры 8253 и 8254 изготовляются по *NMOS* технологии, а 82*C*54 — по усовершенствованной *CHMOS* III технологии, которая обеспечивает малое потребление мощности. Различаются эти БИС только быстродействием — БИС 82×54–2 имеют наибольшее быстродействие. Основные параметры БИС таймеров приведены в табл. 3.5. Максимальная рассеиваемая мощность (*Power Dissipation*) у всех БИС составляет 1 Вт (этот параметр определяется свойствами корпуса).

Структурная схема программируемого таймера. Основные узлы программируемых таймеров *PT* (*Programmable Interval Timer*) 8253 и 82×54 фирмы *Intel* представлены на рис. 3.26. Таймер состоит из трех независимых каналов (трех 16-разрядных программируемых вычитающих счетчиков *Counter* 0, 1 и 2), буфера шины данных (*Data Bus Buffer*), схемы управления чтением и записью (*Read/Write Logic*) и регистра слова управления *RGCW* (*Control Word Register*). Буфер шины данных — это двунаправленный 8-разрядный приемопередатчик с *Z*-состоянием выходов, используемый в качестве шинного интерфейса (обеспечивает связь таймера с системной шиной данных).

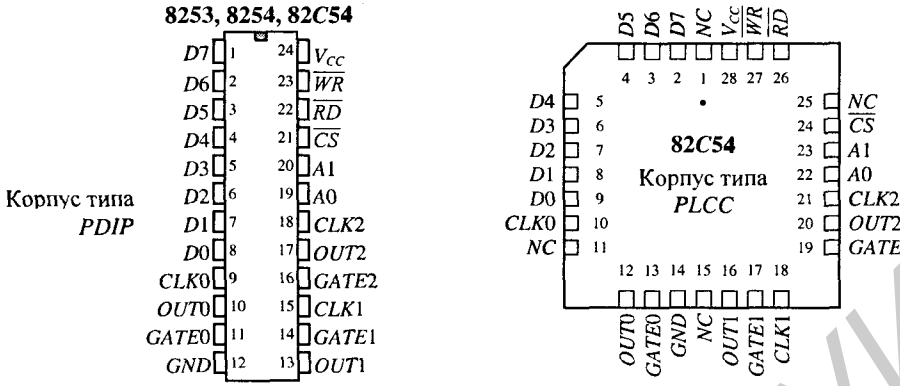


Рис. 3.25. Программируемые таймеры

Таблица 3.5. **Основные параметры БИС таймеров**

БИС	Аналог	$I_{CC\ max}$, мА	V_{CC}, В	$V_{OH\ min}/I_{OH}$, В/мА	$V_{OL\ max}/I_{OL}$, В/мА	F_{CLK}, МГц
8253	КР580ВИ53	140	5 ± 10%	2,4/–0,4	0,45/2,2	2,6
8253–5	КР580ВИ53Д	140	5 ± 10%	2,4/–0,4	0,45/2,2	2,6
8254–5	—	170	5 ± 10%	2,4/–0,4	0,45/2	5
8254	1810ВИ54	170	5 ± 10%	2,4/–0,4	0,45/2	8
8254–2	—	170	5 ± 10%	2,4/–0,4	0,45/2	10
82C54	—	20	4 ÷ 7	3/–2,5	0,4/2,5	8
82C54–2	—	20	4 ÷ 7	3/–2,5	0,4/2,5	10

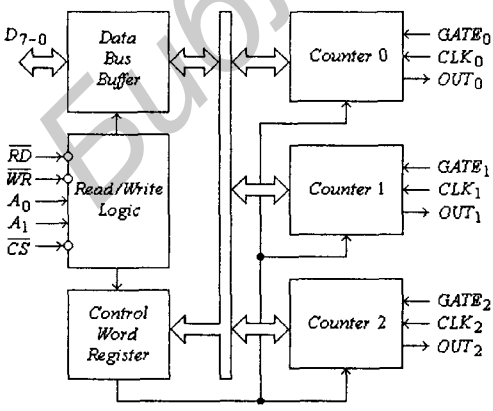


Рис. 3.26. Структурная схема таймера

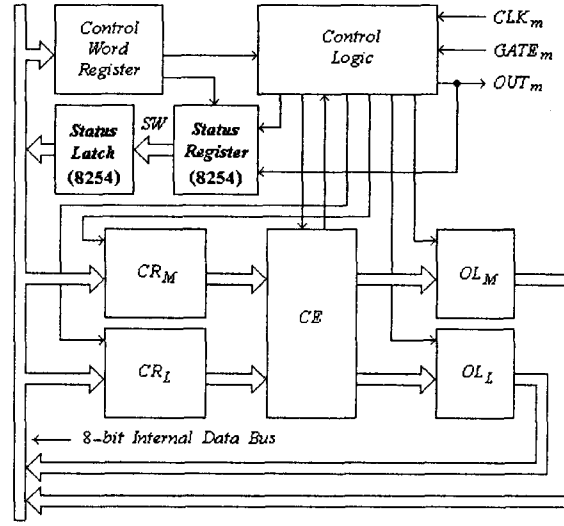


Рис. 3.27. Структурная схема одного канала таймера

Программирование и управление таймером, подключенным к системным шинам, производится микропроцессором и внешними сигналами. Сигналы таймера имеют назначение:

D_{7-0} — сигналы разрядов шины данных МП,

A_1, A_0 и $\overline{CS}$ (*Address and Chip Select*) — сигналы двух разрядов шины адреса МП и сигнал с дешифратора адреса разрядов A_{7-2},

$\overline{RD}$ (*Read*) — сигнал чтения данных из регистров таймера, $\overline{WR}$ (*Write*) — сигнал записи данных в регистры таймера (для МП 8080/8085 $\overline{RD} = \overline{I/OR}$ и $\overline{WR} = \overline{I/OW}$ — сигналы системной шины управления),

OUT_m — выходные сигналы счетчиков (m = 0, 1 и 2 — номер канала таймера),

$GATE_m$ — управляющие сигналы счетчиков (стробы разрешения и запрета счета),

CLK_m — тактовые сигналы счетчиков (максимальные значения частоты F_{CLK} сигналов CLK_m приведены в табл. 3.5).

Адресные сигналы A_1 и A_0 дешифрируются внутри таймера для селекции одного из четырех внутренних устройств — трех каналов счетчиков и регистра слова управления *RGCW*.

Структурная схема одного канала таймера представлена на рис. 3.27 (узлы *Status Latch* и *Status Register* в таймере 8253 отсутствуют):

CE (*Counting Element*) — 16-разрядный синхронный вычитающий счетчик с параллельной записью данных [5];

OL_M и OL_L (*Output Latch*) — 8-разрядные регистры (выходные) для фиксации и хранения текущего состояния счетчика (*M — Most significant byte* — старший байт регистра, *L — Least significant byte* — младший байт регистра);

CR_M и CR_L (*Count Register*) — 8-разрядные регистры хранения модуля пересчета счетчика.

Описанные пары регистров, как единое целое, обозначаются через *OL* и *CR* (или OL_m и CR_m, где m = 0, 1 и 2 — номер канала таймера) — двухбайтовые регистры хранения текущего состояния счетчика и его модуля пересчета. Текущее состояние счетчика m переписывается в регистр OL_m в каждом такте.

Таблица 3.6. **Операции ввода-вывода**

CS	A_1	A_0	$\overline{WR}$	$\overline{RD}$	Операция	Примечание
0	0	0	0	1	$D_{7-0} \rightarrow CR_0$	Запись модуля пересчета *M* счетчика канала 0
0	0	1	0	1	$D_{7-0} \rightarrow CR_1$	Запись модуля пересчета *M* счетчика канала 1
0	1	0	0	1	$D_{7-0} \rightarrow CR_2$	Запись модуля пересчета *M* счетчика канала 2
0	1	1	0	1	$D_{7-0} \rightarrow RGCW$	Запись *CW* в *RGCW* (регистр управления)
					$D_{7-0} \rightarrow Command$	Подача команд чтения "на лету" и обратного чтения
0	0	0	1	0	$D_{7-0} \leftarrow OL_0$	Чтение текущего состояния счетчика канала 0
0	0	1	1	0	$D_{7-0} \leftarrow OL_1$	Чтение текущего состояния счетчика канала 1
0	1	0	1	0	$D_{7-0} \leftarrow OL_2$	Чтение текущего состояния счетчика канала 2
0	1	1	1	0	Нет операций	*Z*-состояние D_{7-0}
0	×	×	1	1	Нет операций	*Z*-состояние D_{7-0}
1	×	×	×	×	Нет операций	*Z*-состояние D_{7-0}

Примечание: × — безразличные значения.

Управление вводом-выводом таймера представлено в табл. 3.6 (*RGCW* — регистр управления).

Для программирования режима работы счетчика канала m МП записывает в регистр *RGCW* слово управления CW_m (*Control Word*) для данного канала. При этом регистр CR_m этого канала сбрасывается в 0. Затем МП записывает в регистр CR_m один или два байта модуля пере-

счета счетчика $M_{Cm} = M_{CMm}M_{CLm}$ (M_{CMm} — старший байт модуля пересчета, M_{CLm} — младший байт модуля пересчета) и запрограммированный канал таймера начинает работать согласно указаниям, содержащимся в слове управления CW_m. Возможность записи в регистр CR_m только одного байта модуля пересчета объясняется сбросом его в 0 при программировании режима работы счетчика (другой байт будет равен 0). Чтение текущего состояния счетчика каждого канала таймера МП может производить в любой момент времени из регистра OL_m.

Далее будет рассмотрен таймер 8253, а затем описаны отличия таймера 8254 от таймера 8253.

Управление режимами работы таймера. Обмен информацией МП с таймером производится по 8-разрядной двунаправленной шине данных D_{7-0} однобайтовыми или двухбайтовыми числами (рис. 3.26). Из табл. 3.6 следует, что номер канала таймера $m = A_1A_0 = 0, 1, 2$ для записи модуля пересчета M_{Cm} и чтения текущего состояния счетчика OL_m. Режимы работы таймера программируются записью в регистр *RGCW* слова управления *CW*, формат которого показан на рис. 3.28.

Разряды $D_7D_6 = SC_1SC_0$ (*SC — Select Counter*) в слове управления *CW* адресуют три части регистра *RGCW* для записи 6-разрядного слова управления $CW_m = D_{5-0}$: $RGCW_0$, $RGCW_1$ и $RGCW_2$, а также производят выбор счетчика для подачи команды чтения "на лету", т. е. в этих случаях $m = SC_1SC_0 = 0$, 1 и 2. Такая передача адресных сигналов внутренних узлов БИС по шине данных используется для увеличения числа портов вывода и уменьшения числа ее контактов. Таким образом, регистр *RGCW* 18-разрядный.

Разряды $D_5D_4 = RL_1RL_0$ (*RL — Read/Load*) управляют режимами чтения из регистра OL_m одного или двух байт состояния счетчика и загрузки в регистр CR_m однобайтовых или двухбайтовых чисел M_{Cm} для программирования модуля пересчета ($D_5D_4 \neq 0$). Если загружается только

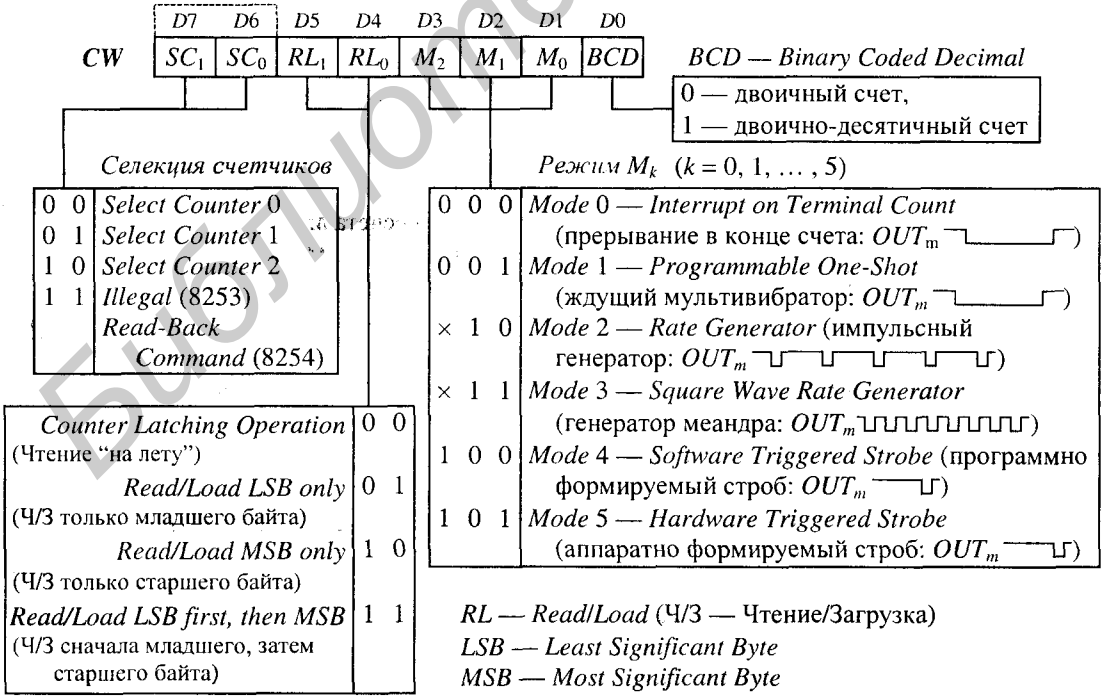


Рис. 3.28. Слово управления *CW* таймера

один байт модуля пересчета, то второй байт равен 0. Если разряды $D_5D_4 = 00$, то слово управления CW_m не записывается в $RGCW_m$ (заданный ранее режим работы счетчика канала m не изменяется). При этом текущее состояние счетчика блокируется в регистре OL_m (содержимое OL_m больше не будет изменяться с каждым тактом). Этот тип слова управления CW_m называется командой чтения "на лету" (назначение команды описано ниже).

Разряд D_0 слова управления CW_m задает двоичный или двоично-десятичный счет. Соответственно максимальный модуль пересчета равен 2^{16} и 10^4 (при значении $M_{Cm} = 0$). Модуль пересчета программируется от 2 до 2^{16} и от 2 до 10^4 с шагом единица ($2^{16} = 65536$). Разряды $D_3D_2D_1$ определяют один из шести режимов работы счетчиков.

Пример 1. Получить выходной сигнал с частотой $f_{OUT} = 1$ Гц из сигнала φ_2, частота которого равна 2,048 МГц. Построить схему, задав адреса портов E0 (канал 0), E1 (канал 1), E2 (канал 2) и E3*h* (*RGCW*).

Заданные адреса портов (E0 ÷ E3 = 111000××) определяют функцию для сигнала выбора БИС таймера:

$$\overline{CS} = \overline{A_7A_6A_5 \cdot \overline{A_4} \cdot \overline{A_3} \cdot \overline{A_2}} = \overline{A_7A_6A_5 \cdot \overline{A_4 \vee A_3 \vee A_2}} = 0$$

только при $A_7A_6A_5A_4A_3A_2A_1A_0 = 111000\times\times$ (конечно, для адресации таймера можно использовать и дешифратор *DC* 3 × 8, выполненный на ИС 555ИД7).

Структурная схема устройства, соответствующая условиям задачи, изображена на рис. 3.29.

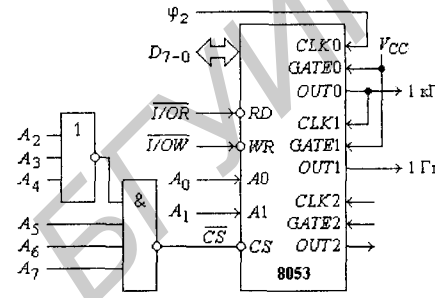


Рис. 3.29. Структурная схема генератора сигнала с частотой 1 Гц

	D7	D6	D5	D4	D3	D2	D1	D0	
	SC1	SC0	RL1	RL0	M2	M1	M0	BCD	CW
CW_0 =	0	0	1	1	0	1	1	0	= 36*h*
CW_1 =	0	1	1	1	0	1	1	1	= 77*h*

Рис. 3.30. Слова управления каналов 0 и 1

Общий коэффициент деления $M = 2048 \cdot 10^3 = M_{C0} \cdot M_{C1}$, можно получить, используя каналы 0 и 1 таймера: $M_{C0} = 2^{11} = 2048d = 800h$ для счетчика канала 0 (двоичный счет, загрузка двух байт модуля пересчета), $M_{C1} = 10^3 = 1000d$ для счетчика канала 1 (двоично-десятичный счет, загрузка двух байт модуля пересчета). Выберем режим работы M_3 обоих счетчиков — генератор меандра (делитель частоты).

На рис. 3.30 показаны слова управления *CW* для каналов 0 и 1 таймера, задающие вышеуказанные режимы работы. Программирование таймера на заданные режимы работы можно выполнить с помощью программы:

```
MVI   A, 36h   ; A ← CW0 = 36h
OUT   0E3h     ; RGCW0 ← CW0
MVI   A, 77h   ; A ← CW1 = 77h
OUT   0E3h     ; RGCW1 ← CW1
SUB   A        ; A ← 0
OUT   0E1h     ; CRL1 ← ML1 = 00d — младший байт модуля пересчета канала 1
OUT   0E0h     ; CRL0 ← ML0 = 00h — младший байт модуля пересчета канала 0
MVI   A, 8     ; A ← 8
```

```
OUT   0E0h   ; CR_M0 ← M_CM0 = 08h — старший байт модуля пересчета канала 0
MVI   A, 10h ; A ← 10h
OUT   0E1h   ; CR_M1 ← M_CM1 = 10d — старший байт модуля пересчета канала 1
```

Режимы работы каналов таймера. В соответствии с рис. 3.28 счетчики таймера могут работать в шести режимах M_k (k = 0, 1, … , 5). В табл. 3.7 приведено функциональное назначение сигналов $GATE_m$ в каждом из этих режимов.

Режим M_0 — прерывание в конце счета (рис. 3.31). В этом режиме сигнал $GATE_m$ разрешает счет ($GATE_m$ = 1) или запрещает его ($GATE_m$ = 0). Запись слова управления CW_m в регистр $RGCW_m$ устанавливает на выходе OUT_m значение 0. Изменение сигнала $GATE_m$ с 0 на 1 запускает счет. По окончании счета на выходе OUT_m устанавливается значение 1. Загрузка счетчика новым значением модуля пересчета M_{Cm} устанавливает на выходе OUT_m значение 0. Если загрузка производится во время счета, то запись младшего байта останавливает счет, а запись старшего байта запускает новый цикл счета. Содержимое регистра CR_m передается в счетчик тактовым сигналом.

Режим M_1 — ждущий мультивибратор (рис. 3.32). Запись слова управления CW_m в регистр $RGCW_m$ устанавливает на выходе OUT_m значение 1. Переход сигнала $GATE_m$ с 0 на 1 запускает счет. Затем первый перепад тактового сигнала с 1 на 0 устанавливает сигнал OUT_m в 0. После окончания счета сигнал OUT_m устанавливается в 1.

Таблица 3.7. **Функциональное назначение сигналов $GATE_m$**

Режим M_k	Низкий уровень или переход на низкий уровень	Положительный фронт	Высокий уровень
0	Запрет счета	—	Разрешение счета
1	—	1. Запуск счета 2. Сброс OUT_m в 0 в следующем такте	—
2	1. Запрет счета 2. Установка OUT_m = 1	1. Перезагрузка счетчика 2. Запуск счета	Разрешение счета
3	1. Запрет счета 2. Установка OUT_m = 1	1. Перезагрузка счетчика 2. Запуск счета	Разрешение счета
4	Запрет счета	—	Разрешение счета
5	—	Запуск счета	—

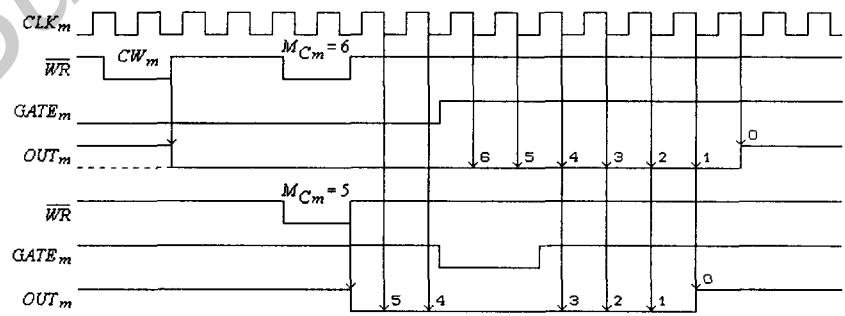


Рис. 3.31. Временны́е диаграммы работы таймера в режиме M_0

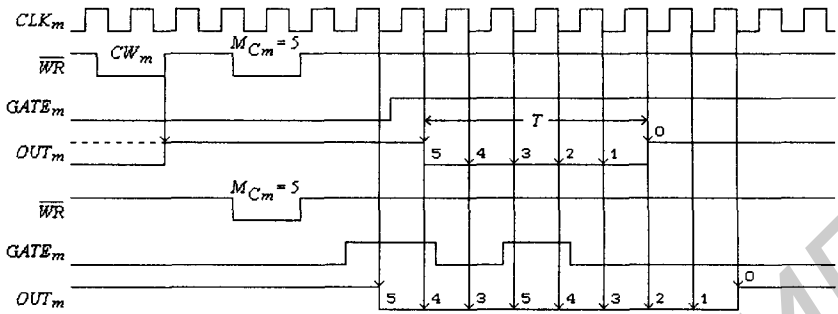


Рис. 3.32. Временны́е диаграммы работы таймера в режиме M_1

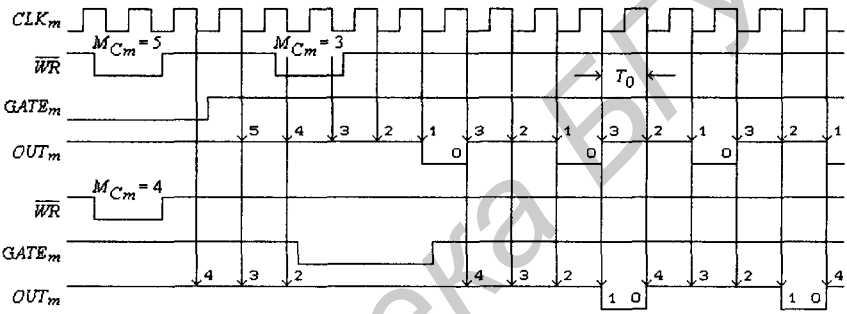


Рис. 3.33. Временны́е диаграммы работы таймера в режиме M_2

Если во время счета будет загружено новое значение модуля пересчета M_{Cm}, то это не повлияет на длительность текущего импульса $OUT_m = 0$ (до следующего запуска). Ждущий мультивибратор является перезапускаемым — каждый перепад сигнала $GATE_m$ с 0 на 1 запускает счет сначала, если счет не завершен до конца. Содержимое регистра CR_m передается в счетчик тактовым сигналом при условии, что поступил перепад сигнала $GATE_m$ с 0 на 1.

Режим M_2 — импульсный генератор (рис. 3.33). В этом режиме таймер является делителем частоты тактового сигнала CLK_m в M_{Cm} раз:

$$f_{OUTm} = f_{CLKm} / M_{Cm}.$$

Если счетчик перезагружается новым значением модуля пересчета M_{Cm} между выходными импульсами $OUT_m = 0$, то на текущем периоде это не скажется, однако следующий период будет соответствовать новой величине M_{Cm}. Значение сигнала $GATE_m = 0$ запрещает счет. Если затем установить значение $GATE_m = 1$, то работа начинается от загруженного ранее значения M_{Cm}. Длительность значения сигнала $OUT_m = 0$ равна одному периоду T_0 сигнала CLK_m.

Режим M_3 — генератор меандра (рис. 3.34). Этот режим аналогичен режиму M_2 во всем, за исключением длительности значений выходного сигнала $OUT_m = 0$ и 1:

$$T_1 = T_2 = T_0 \cdot M_{Cm} / 2 \text{ при } M_{Cm} \text{ четном,}$$

$$T_1 = T_0 \cdot (M_{Cm} + 1) / 2 \text{ и } T_2 = T_0 \cdot (M_{Cm} - 1)/2 \text{ при } M_{Cm} \text{ нечетном}$$

(при значении $M_{Cm} = 3$ счетчики не работают).

Самый младший разряд счетчика независимо от типа счета (двоичный или двоично-десятичный) всегда производит деление частоты тактового сигнала CLK_m в 2 раза (счет ведется

по *mod* 2). В режиме M_3 этот разряд счетчика не участвует в счете, поэтому в каждом такте из содержимого счетчика вычитается не число 1, а число 2. Изъятие из счета младшего разряда счетчика эквивалентно выполнению операции M_{Cm} / 2 при четном значении M_{Cm} и (M_{Cm} – 1) / 2 при нечетном M_{Cm}. Чтобы модуль пересчета не уменьшился в 2 раза, период выходного сигнала OUT_m формируется за два цикла пересчета: в первом цикле значение OUT_m = 1, а во втором цикле значение OUT_m = 0. При нечетном M_{Cm} в первом цикле пересчета в счетчик загружается число M_{Cm} и дополнительно затрачивается один такт на изменение состояния первого триггера с 1 на 0 (после этого триггер из счета исключается), а во втором цикле пересчета в счетчик загружается число M_{Cm} – 1 (четное число). Этим и достигается примерное равенство полупериодов выходного сигнала счетчика OUT_m при нечетном значении M_{Cm} и точное равенство при четном значении M_{Cm}. Грубо говоря, в режиме M_3 самый младший триггер счетчика перемещается на его выход (см. рис. 3.130 в § 3.9). Из-за такой организации счета режим M_3 не рекомендуется использовать в счетчиках событий.

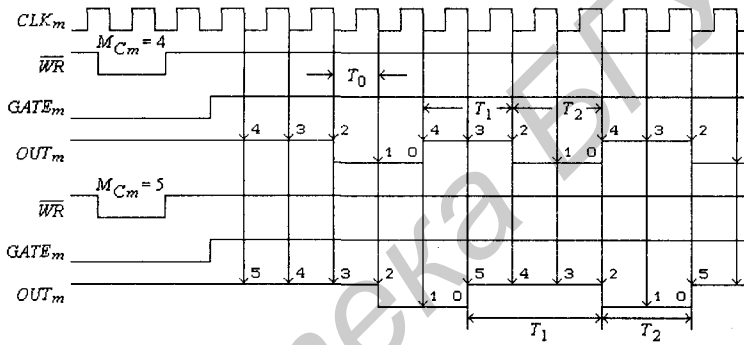


Рис. 3.34. Временны́е диаграммы работы таймера в режиме M_3

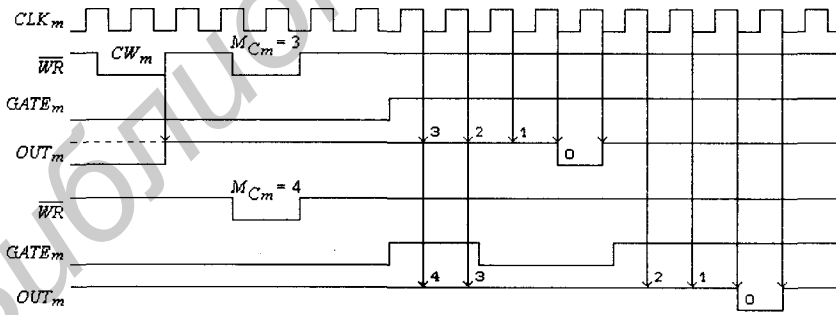


Рис. 3.35. Временны́е диаграммы работы таймера в режиме M_4

Режим M_4 — одиночный программно формируемый строб (рис. 3.35). Запись слова управления CW_m в регистр $RGCW_m$ устанавливает на выходе OUT_m значение 1. Счет разрешает значение строба $GATE_m$ = 1. По действию сигнала $GATE_m$ режим M_4 аналогичен режиму M_0. Если во время счета производится загрузка нового значения модуля пересчета M_{Cm}, то загрузка младшего байта не влияет на текущий цикл счета, а загрузка старшего байта запускает новый цикл счета.

Режим M_5 — одиночный аппаратно формируемый строб (рис. 3.36). Запись слова управления CW_m в регистр $RGCW_m$ устанавливает на выходе OUT_m значение 1. По действию сигнала

$GATE_m$ этот режим аналогичен режиму M_1, т. е. запуск счета осуществляется перепадом сигнала $GATE_m$ с 0 на 1. Счетчик является перезапускаемым — каждый перепад сигнала $GATE_m$ с 0 на 1 запускает счет или перезапускает его сначала, если счет не завершен до конца. Перезагрузка счетчика новым значением модуля пересчета M_{Cm} во время счета не влияет на длительность текущего цикла счета, но следующий цикл будет уже новым.

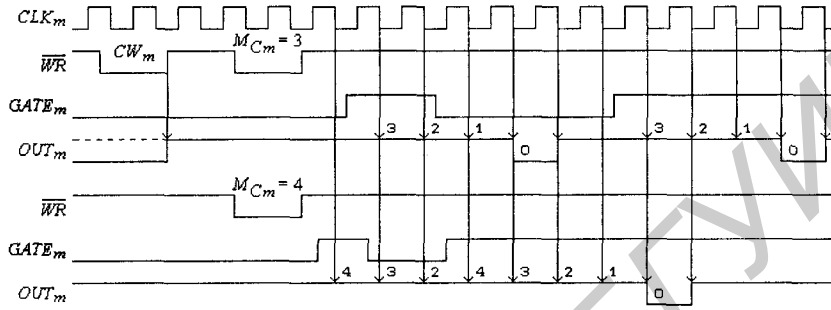


Рис. 3.36. Временны́е диаграммы работы таймера в режиме M_5

Чтение состояния счетчиков. Чтение состояния счетчика требуется для принятия решения на основе анализа числа поступивших на него тактовых импульсов. Во время работы счетчика его текущее состояние передается в регистр OL_m. Регистр OL_m может быть прочитан двумя способами:

путем выполнения обычной операции чтения, но с предварительным остановом счетчика (с нарушением процесса счета),

путем подачи специальной команды чтения "на лету" ("*on the fly*") — без останова счета.

Останов счетчика выполняется подачей значения сигнала $GATE_m$ = 0 или с помощью внешнего вентиля, отключающего тактовый сигнал CLK_m. Иначе, на интервале времени, затрачиваемом на последовательное чтение двух байт, состояние счетчика может измениться и будет получен неверный результат (например, при чтении из регистра OL_m числа 6E 00*h* будет получено число 6D 00*h*).

Чтение "на лету" производится с помощью предварительной подачи в таймер слова управления CW_m, изображенного на рис. 3.37 ($m = SC_1SC_0$ — номер счетчика).

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	×	×	×	×

Рис. 3.37. Слово управления для чтения "на лету" состояний счетчиков

Подача данного слова управления CW_m не влияет на содержимое регистров $RGCW_m$, задающих режимы работы счетчиков, а приводит лишь к блокированию в регистре OL_m текущего состояния счетчика канала *m*. Далее содержимое регистра OL_m не изменяется, пока не будет произведено его чтение командой IN *port* или пока счетчик не будет перепрограммирован. После полного завершения операции чтения (одного или двух байт) блокировка автоматически отключается, и в регистр OL_m снова будут переписываться текущие состояния счетчика *CE*. Чтение "на лету" состояний счетчиков не нарушает процесса счета.

Если состояние счетчика заблокировано, то повторная подача команды чтения "на лету" в этот же канал игнорируется. Чтение регистра OL_m даст результат, заблокированный первой

командой чтения "на лету". Команды чтения "на лету" нельзя подавать подряд в разные каналы без их чтения (без чтения регистров OL_m) после каждой команды.

При любом методе чтения состояния счетчика, оно должно производиться согласно запрограммированному формату; особо следует отметить, что если счетчик запрограммирован для чтения/загрузки двух байт, то оба байта должны быть прочитаны.

Пример 2. Для *примера* 1 выполнить чтение "на лету" счетчика канала 1 и результат поместить в регистры *C* и *B*. Эти операции выполняются программой:

```
MVI   A, 40h   ; A ← CW0 = 40h — чтение "на лету" счетчика канала 1
OUT   0E3h     ; Блокировка записи текущего состояния счетчика в регистр OL1
IN    0E1h     ; A ← OL_L1
MOV   C, A     ; C ← OL_L1
IN    0E1h     ; A ← OL_M1
MOV   B, A     ; B ← OL_M1
```

Особенности таймера 8254. Несколько команд чтения "на лету" могут использоваться для блокирования в регистрах OL_m текущих состояний более чем одного счетчика. Каждое из состояний счетчиков, зафиксированных в регистре OL_m, хранится до тех пор, пока оно не будет прочитано. Минимальные и максимальные значения модулей пересчета таймера 8254 для различных режимов работы приведены в табл. 3.8 (значению $M_C = 0$ соответствуют модули пересчета 2^{16} для двоичного счета и 10^4 для десятичного счета).

Таблица 3.8. **Модули пересчета таймера**

Режим	M_C (*min*)	M_C (*max*)
M_0	1	0
M_1	1	0
M_2	2	0
M_3	2	0
M_4	1	0
M_5	1	0

Другое свойство таймера 8254 — для одного и того же счетчика чтение и запись могут чередоваться, например, если счетчик запрограммирован для чтения/загрузки двух байт, то допустима следующая последовательность операций: 1) чтение младшего байта текущего состояния счетчика, 2) запись младшего байта нового модуля пересчета, 3) чтение старшего байта, 4) запись старшего байта нового модуля пересчета.

Если счетчик запрограммирован для чтения/загрузки двух байт, то используется следующая предосторожность: программа не должна передавать управление между чтением первого и второго байта другой подпрограмме, которая также должна читать текущее состояние того же самого счетчика. Иначе, будет получен неправильный результат.

В функциональном отношении таймер 8254 отличается от таймера 8253 только наличием специального формата слова управления, называемого командой обратного чтения (*Read-Back Command*; см. рис. 3.28), которая связана с узлами, выделенными полужирным шрифтом на рис. 3.27. Эта команда записывается в 6-разрядный регистр, входящий в *RGCW*, и имеет формат, показанный на рис. 3.38.

Рис. 3.38. Формат команды обратного чтения

После подачи команды обратного чтения производятся операции чтения слова состояния каналов SW_m (*Status Word*) и (или) состояния счетчиков OL_m.

Команда обратного чтения может использоваться для блокировки состояний нескольких счетчиков в регистрах OL_m установкой разряда $\overline{COUNT} = D_5 = 0$ и выбором счетчиков заданием значений трех разрядов $CNTm = D_{3-1}$ (1 — счетчик выбран, 0 — счетчик не выбран). Эта одна команда является функциональным эквивалентом нескольких команд чтения "на лету" в таймере 8253. Заблокированные состояния счетчиков сохраняются, пока они не будут прочитаны (или счетчики не будут перепрограммированы). Блокировка состояния счетчика m автоматически отменяется при чтении регистра OL_m (в регистр OL_m тактовым сигналом будут переписываться текущие состояния счетчика CE_m), но состояния других счетчиков остаются зафиксированными, пока они не будут прочитаны. Если несколько команд обратного чтения будут предназначены одному и тому же счетчику без чтения его регистра OL_m, то все, кроме первой команды, игнорируются — будет прочитано состояние счетчика, заблокированное в регистре OL_m первой командой обратного чтения.

Если заблокировано состояние счетчика ($\overline{COUNT} = 0$) и зафиксировано слово состояния канала ($\overline{STATUS} = 0$), то первая операция чтения этого счетчика выдаст зафиксированное слово состояния SW_m (рис. 3.39), независимо от того какая из величин была зафиксирована первой. Следующие одна или две операции чтения (в зависимости от того, запрограммирован счетчик для чтения/загрузки одного или двух байт) выдадут заблокированное состояние счетчика OL_m. Последующие операции чтения выдают незаблокированные (текущие) состояния счетчика.

Формат слова состояния SW_m приведен на рис. 3.39. Команда обратного чтения позволяет пользователю кроме состояния заданных счетчиков (OL_m) проверить также и запрограммированные режимы их работы ($CW_m = D_{5-0}$), текущие состояния выходных сигналов таймера OUT_m и состояния флагов *NULL COUNT*. Например, разряд $D_7 = OUTPUT$ слова состояния позволяет пользователю контролировать выходы счетчиков с помощью программного обеспечения, давая возможность удалить из системы некоторые аппаратные средства.

Поведение флага *NULL COUNT* описывается соотношениями:

запись в регистр слова управления *Read-Back Command* $\Rightarrow$ *NULL COUNT* = 1
(только счетчик, определенный словом управления, будет иметь флаг *NULL COUNT* установленным в 1; флаги *NULL COUNT* других счетчиков не изменяются);

запись модуля пересчета в регистр *CR* $\Rightarrow$ *NULL COUNT* = 1
(если счетчик запрограммирован для чтения/загрузки двух байт, то флаг *NULL COUNT* переходит в 1 после записи второго байта);

новый модуль пересчета загружен в счетчик (*CR* $\rightarrow$ *CE*) $\Rightarrow$ *NULL COUNT* = 0.

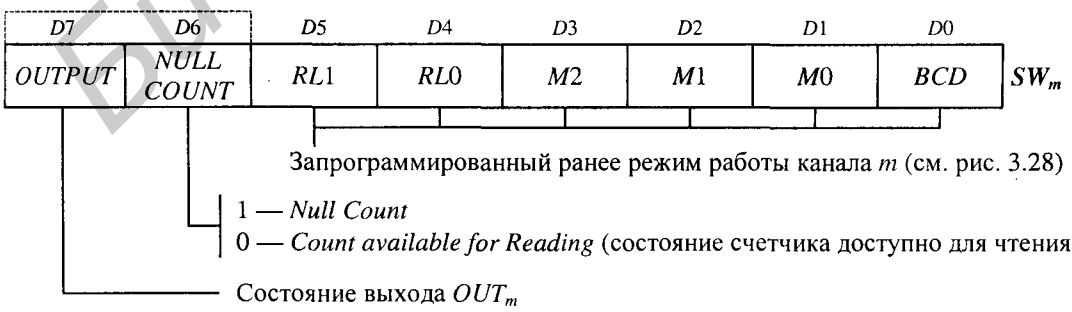


Рис. 3.39. Слово состояния каналов таймера SW_m

3.5. Программируемый контроллер прерываний 8259*A*

Программируемые контроллеры прерываний (*Programmable Interrupt Controller — PIC*) фирмы *Intel* 8259, 8259*A* (отечественные аналоги 580ВН59 и 1810ВН59А) и 82*C*59*A*–2 (рис. 3.40) предназначены для организации обработки приоритетных 8-уровневых запросов прерываний от восьми внешних устройств. Контроллер прерываний 8259 был разработан для МП-систем, построенных только на основе МП 8080 и 8085. Модифицированный вариант этого контроллера 8259*A* предназначен для обслуживания как МП 8080/8085, так и МП 8086/8088. Контроллер 8259*A* полностью совместим сверху вниз с контроллером 8259 — программное обеспечение, написанное для контроллера прерываний 8259, можно использовать и для контроллера прерываний 8259*A* во всех 8259-эквивалентных режимах.

Контроллеры прерываний спроектированы так, чтобы минимизировать программное обеспечение и непроизводительные затраты времени при обработке приоритетных многоуровневых прерываний. Они имеют несколько режимов, позволяющих оптимизировать ввод-вывод по прерыванию для ряда системных требований. Предусмотрена также возможность индивидуального маскирования запросов прерываний. Статус уровней приоритета обслуживания *I/O* устанавливается программным способом (приоритеты, закрепленные за внешними устройствами, можно изменять в процессе выполнения программы). Контроллеры прерываний можно каскадировать для получения 64-уровневой системы прерываний.

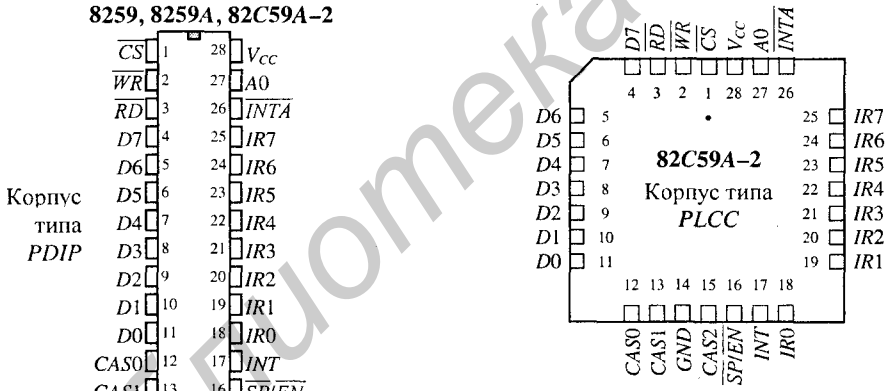


Рис. 3.40. Программируемые контроллеры прерываний

Контроллеры прерываний 8259 и 8259*A* изготовляются по *n*-МОП технологии (*NMOS*), а контроллер прерываний 82*C*59*A*–2 (аналог 8259*A*) — по высокоэффективной *CHMOS* технологии, которая обеспечивает малое потребление мощности.

Максимальные значения токов потребления $I_{CC\,max}$ при V_{CC} = +5 В равны 85 мА и 5 мА для БИС 8259*A* и 82*C*59*A* соответственно. Максимальная рассеиваемая мощность составляет 1 Вт и 0,9 Вт. Выходные сигналы характеризуются параметрами:

$V_{OL\,max}$ = 0,45 В при I_{OL} = 2,2 мА и $V_{OH\,min}$ = 2,4 В при I_{OH} = –400 мкА для 8259*A*,

$V_{OL\,max}$ = 0,40 В при I_{OL} = 2,5 мА и $V_{OH\,min}$ = 3,0 В при I_{OH} = –2,5 мА для 82*C*59*A*.

Структурная схема *PIC*. Имеются определенные отличия в использовании *PIC* в системах, построенных на основе МП 8080/8085 и 8086/8088, поэтому за основу будет взято описание применения *PIC* при работе совместно с МП 8080/8085. Особенности же применения *PIC* при работе совместно с МП 8086/8088 будут указываться особо. Условное графическое обо-

значение *PIC* 8259 и 8259*A* показано на рис. 3.41. На структурной схеме, приведенной на рис. 3.42, изображены узлы:

Data Bus Buffer — буфер шины данных, представляющий собой двунаправленный приемопередатчик с *Z*-состоянием выхода. Буфер используется для приема от МП слов команд инициализации и команд операций, а также для чтения команды CALL *addr* и состояний некоторых регистров *PIC*;

Read/Write Logic — устройство управления чтением и записью при инициализации и управлении *PIC*. Это устройство содержит регистры памяти для приема и хранения слов команд инициализации ICW_k (*Initialization Command Word*), $k = 1, 2, 3, 4$ и слов команд операций OCW_m (*Operation Command Word*), $m = 1, 2, 3$;

Control Logic — устройство управления запросами прерываний МП и чтением команды CALL *addr*;

Interrupt Request Register (*IRR*) — 8-разрядный регистр запросов прерываний, используемый для запоминания в триггерах $T_{IRR\,i}$ ($i = 0, 1, \ldots, 7$) запросов, поступивших на входы $IR_{7\text{–}0}$;

Priority Resolver (*PR*) — схема анализа приоритетов запросов прерываний IR_i (приоритетное решающее устройство), выбирающая запрос с наибольшим приоритетом среди поступивших в *IRR* и уже находящихся в обслуживании (зафиксированных в *ISR*);

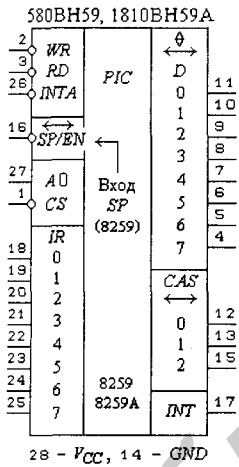


Рис. 3.41. Контроллер прерываний

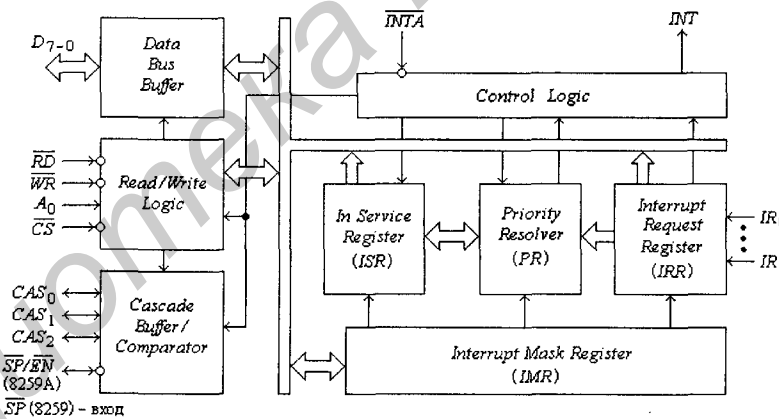


Рис. 3.42. Структурная схема *PIC* 8259 и 8259*A*

In Service Register (*ISR*) — 8-разрядный регистр памяти, хранящий запросы прерываний, находящиеся в обслуживании. В триггер $T_{ISR\,i}$ ($i = 0, 1, \ldots, 7$) регистра записывается 1, если запрос по входу IR_i прошел через схему анализа приоритетов *PR*. Значение $T_{ISR\,i} = 1$ запрещает прохождение на обслуживание последующих запросов от собственного входа IR_i и входов IR_j, имеющих меньший приоритет. Одновременно в обслуживании может находиться до восьми запросов прерываний, так как запросы с меньшим приоритетом могут быть обслужены не до конца (прерваны) при поступлении запросов по входам IR_i, имеющим больший приоритет. Триггер $T_{ISR\,i}$ сбрасывается в 0 командой OCW_2 конца прерывания, которая ставится в конце подпрограммы обслуживания прерывания (ППОП), соответствующей входу IR_i (см. рис. 3.58);

Interrupt Mask Register (*IMR*) — 8-разрядный регистр маски прерываний, программно доступный для записи и чтения байта данных. Если установить значение триггера $T_{IMR\,i} = 1$, то за-

просы прерывания по входу IR_i будут запрещены. Замаскированный вход IR_i более высокого приоритета не запрещает прохождение на обслуживание запросов прерывания по входам IR_j более низкого приоритета;

Cascade Buffer/Comparator — буфер каскадирования/компаратор, используемые для каскадирования до девяти *PIC* 8259A (один ведущий — *Master* и восемь ведомых — *Slave*; *Cascade Buffer* — у ведущего, *Comparator* — у ведомых).

На рис. 3.43 показана структурная схема МП-системы, построенная на МП 8080 и контроллере прерываний 8259 или 8259*A*. Задача контроллера состоит в приеме запросов прерываний, поступающих от I/O_i на входы IR_i, анализе их приоритетов и подачи запроса прерывания $INT = 1$ на МП (рис. 3.44) для обслуживания I/O_i, имеющего наибольший приоритет из всех одновременно поступивших запросов $IR_i = 1$. В ответ контроллер прерываний получает от МП три импульса $\overline{INTA} = 0$, выполняющих чтение из *PIC* трехбайтовой команды CALL *addr*. Сигнал $\overline{INTA}$ больше нигде не используется, поэтому производится безадресное чтение команды CALL *addr*. Это означает, что сигналы $\overline{CS}$ и A_0 могут иметь произвольные значения (это единственный случай чтения команды не из памяти, а из внешнего устройства). Адрес *addr* записывается в *PIC* при его инициализации (программировании).

Приоритеты входов IR_i в процессе работы могут изменяться командой операций OCW_2, подаваемой в *PIC*. Запросы прерываний по входам IR_i, имеющим больший приоритет, чем обслуживаемый в данный момент, могут прервать его обслуживание — максимальное число уровней вложенности ППОП определяется числом подключенных к *PIC* внешних устройств I/O_i.

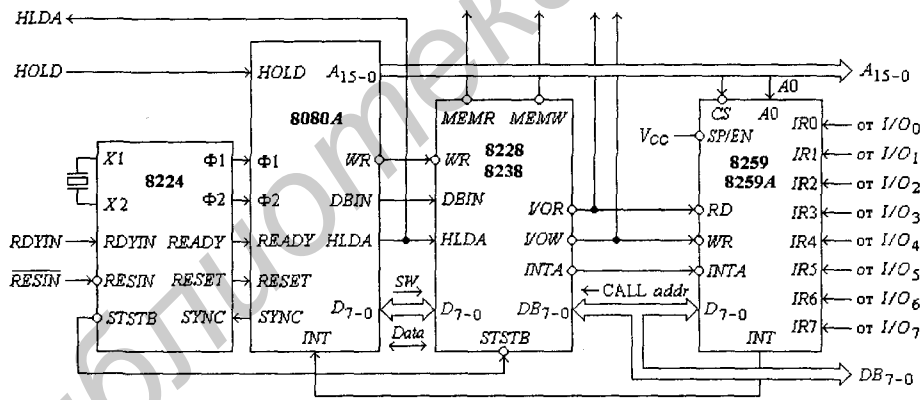


Рис. 3.43. Структурная схема МП-системы (8080 + 8259/8259*A*)

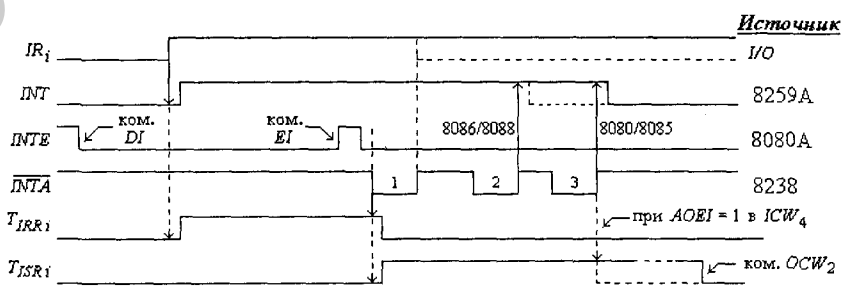


Рис. 3.44. Выполнение запроса прерывания

Назначение сигналов *PIC* 8259 и 8259A. Связь контроллера прерываний с МП и *I/O* осуществляется сигналами:

D_{7-0} (*Data*) — сигналы шины данных МП;

$\overline{RD}$ (*Read*), $\overline{WR}$ (*Write*) — сигналы чтения и записи информации в регистры *PIC*, являющегося для МП обычным внешним устройством ($\overline{RD} = \overline{I/OR}$, $\overline{WR} = \overline{I/OW}$ — для МП 8080/8085);

A_0 (*Address*) — разряд шины адреса МП;

$\overline{CS}$ (*Chip Select*) — сигнал выбора кристалла (поступает от дешифратора адресных сигналов A_{7-1});

IR_{7-0} (*Interrupt Request*) — сигналы запросов прерываний от внешних устройств *I/O*;

INT (*Interrupt*) — сигнал запроса прерывания, подаваемый на МП;

$\overline{INTA}$ (*Interrupt Acknowledge*) — сигнал подтверждения прерывания, поступающий от МП в ответ на значение сигнала *INT* = 1 (связь сигналов IR_i, *INT* и $\overline{INTA}$ показана на рис. 3.44). Этот сигнал используется для чтения из *PIC* трехбайтовой команды CALL *addr* для МП 8080/8085 ($\overline{INTA}$ = ⊓⊓⊓) и одного байта типа прерывания *type* для микропроцессоров 8086/8088 ($\overline{INTA}$ = ⊓ ⊓ ⊓);

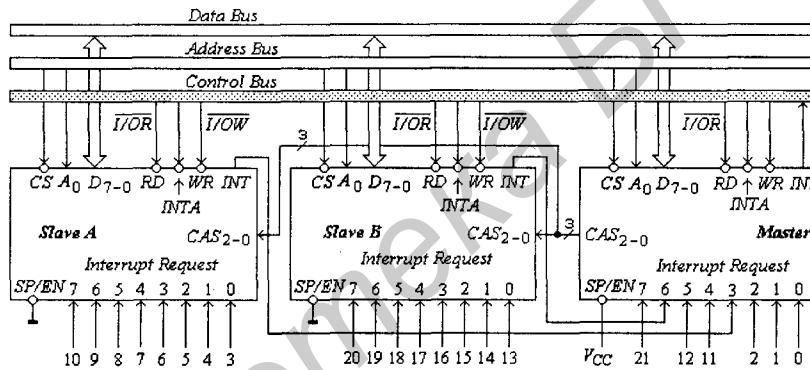


Рис. 3.45. Схема каскадирования *PIC* 8259/8259*A*

$\overline{SP}/\overline{EN}$ (*Slave Program/Enable Buffer*) — контакт для программирования ведомого/разрешения буфера ($\overline{SP}$ — вход, $\overline{EN}$ — выход). При инициализации *PIC* 8259*A* командой ICW_4 (такой команды в *PIC* 8259 нет) можно задать один из двух режимов его работы (см. рис. 3.53): небуферированный или буферированный в зависимости от способа подключения *PIC* к системной шине данных — без приемопередатчика или через приемопередатчик. В небуферированном режиме входной сигнал $\overline{SP}$ используется для задания ведущего *Master* ($\overline{SP}/\overline{EN} \equiv 1$) и ведомых *Slave* ($\overline{SP}/\overline{EN} \equiv 0$) контроллеров прерываний (рис. 3.45). Это справедливо и для *PIC* 8259, у которого вместо сигнала $\overline{SP}/\overline{EN}$ имеется сигнал $\overline{SP}$ (см. рис. 3.41), т. е. *PIC* 8259 имеет только небуферированный режим работы. При задании командой ICW_4 буферированного режима *PIC* 8259*A* программируется и его назначение — ведущий или ведомый. В буферированном режиме выходной сигнал $\overline{EN}$ (только в *PIC* 8259*A*) используется для управления приемопередатчиком шины данных. В схеме, изображенной на рис. 3.43, следует использовать небуферированный режим работы *PIC* 8259*A*;

CAS_{2-0} (*Cascade Lines*) — три линии каскадирования. Эти линии являются выходами буфера каскадирования ведущего *PIC* и входами компараторов ведомых *PIC*. Ведущий *PIC* по линиям CAS_{2-0} посылает трехразрядный двоичный код идентификатора ведомого *PIC*, запрос прерывания которого следует обслужить. Компараторы ведомых *PIC* сравнивают собственный

идентификатор с полученным от ведущего *PIC*. При совпадении этих идентификаторов ведомый *PIC* выдает запрограммированный в нем адрес *addr* вызова ППОП (рис. 3.45 — работа этой схемы станет более понятна после описания слов команды инициализации ICW_3 для ведущего и ведомого контроллеров 8259/8259*A*).

На рис. 3.46 изображена структурная схема МП-системы, построенная на МП 8085 и контроллере прерываний 8259 или 8259*A*. Здесь должен использоваться небуферированный режим работы *PIC*, так как его шина данных непосредственно подключена к системной шине данных МП. В этой схеме приемопередатчик 1533АП6 (буфер между локальной и системной шинами данных МП) вообще можно исключить, конечно, если не требуется усиления выходных токов сигналов AD_{7-0}. Принципиальная схема соответствующего центрального процессорного устройства была приведена на рис. 1.10.

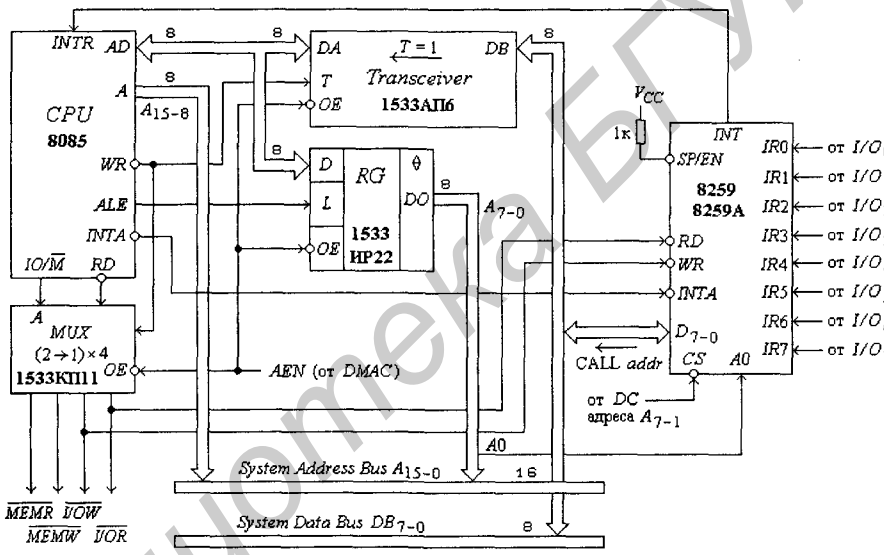


Рис. 3.46. Структурная схема МП-системы для небуферированного режима *PIC*

На рис. 3.47 представлена структурная схема МП-системы, построенная на МП 8085 и контроллере прерываний 8259*A*. Здесь должен использоваться буферированный режим работы *PIC* 8259*A*, так как его шина данных непосредственно подключена к локальной шине данных МП, а к системной шине данных — через приемопередатчик. В буферированном режиме при чтении содержимого регистров *PIC* командами IN *port* и значением сигнала $\overline{INTA} = 0$ будут вырабатываться значения сигналов управления $\overline{RD} = 0$ для команды IN *port* или $\overline{RD} = 1$ для $\overline{INTA} = 0$, $\overline{WR} = 1$ (см. табл. 1.2) и $\overline{SP/EN} = 0$, но при этом данные с системной шины данных не должны поступать.

Для выключения приемопередатчика при чтении данных из *PIC* и предназначен сигнал $\overline{SP/EN}$. Перевод выходных сигналов приемопередатчика в *Z*-состояние производится значением сигнала

$$\overline{OE} = \overline{\overline{SP} / \overline{EN} \cdot \overline{AEN}} = \overline{\overline{SP} / \overline{EN}} \vee AEN = 1,$$

т. е. при прямом доступе к памяти и чтении данных из *PIC*.

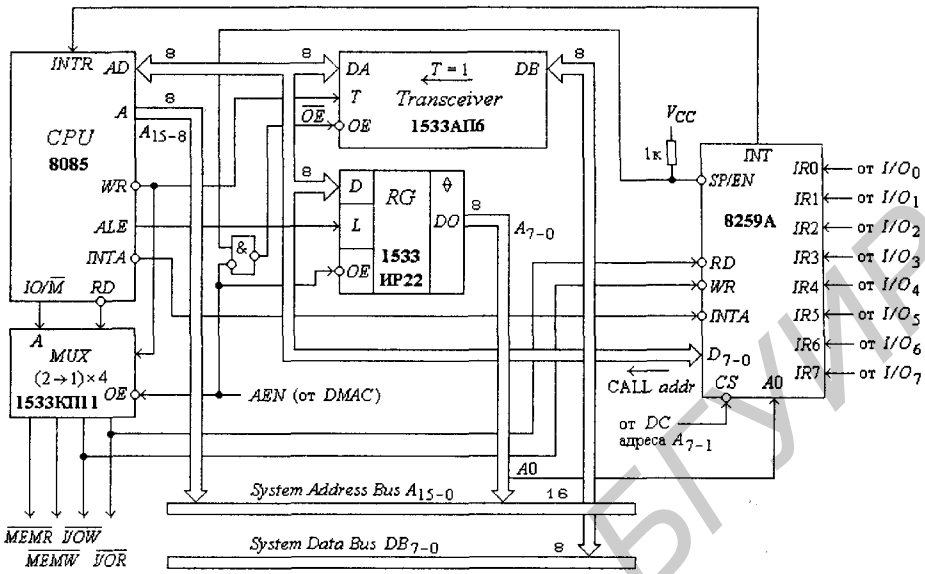


Рис. 3.47. Структурная схема МП-системы для буферированного режима *PIC*

Таблица 3.9. **Операции ввода-вывода**

$\overline{CS}$	A_0	D_4	D_3	$\overline{RD}$	$\overline{WR}$	Операция	Примечание
0	0	–	–	0	1	$D_{7-0} \leftarrow IRR, ISR, IL$	Ввод в МП (чтение состояний регистров)
0	1	–	–	0	1	$D_{7-0} \leftarrow IMR$	
0	0	0	0	1	0	$D_{7-0} \rightarrow OCW_2$	Вывод из МП (инициализация и управление)
0	0	0	1	1	0	$D_{7-0} \rightarrow OCW_3$	
0	0	1	×	1	0	$D_{7-0} \rightarrow ICW_1$	
0	1	×	×	1	0	$D_{7-0} \rightarrow OCW_1, ICW_2, ICW_3, ICW_4$	
0	×	×	×	1	1	Нет операций	Шина D_{7-0} в Z-состоянии
1	×	×	×	×	×	Нет операций	

Если имеются ведомые *PIC* и подключаются они к системной шине данных через индивидуальные приемопередатчики, то следует задать $\overline{OE} \equiv 0$ и $T = \overline{\overline{SP}/\overline{EN}}$ (чтение данных из *PIC* при значении $T = 1$).

Управление вводом-выводом *PIC* 8259/8259A представлено в табл. 3.9:

ICW_k (*Initialization Command Word*), $k = 1, 2, 3, 4$ — слова команд инициализации (инициализация производится один раз сразу после включения питания);

OCW_m (*Operation Command Word*), $m = 1, 2, 3$ — слова команд операций (эти команды позволяют производить выбор и изменение алгоритма обслуживания запросов прерывания в процессе выполнения программ);

IL (*Interrupting Level*) — уровень прерывания (используется в режиме полинга, устанавливаемом командой OCW_3).

Разряды D_4 и D_3 при записи команд ($\overline{WR} = 0$) используются для адресации соответствующих им регистров.

Инициализация *PIC*. Перед началом работы (после каждого включения питания МП-системы) должна производиться инициализация *PIC* командами ICW_k. Блок-схема алгоритма инициализации ведущей и ведомых *PIC* представлена на рис. 3.48. Данный алгоритм реализован с помощью внутреннего управляющего автомата, содержащего счетчик и комбинационную схему анализа условий переходов, которые задаются разрядами D_1 и D_0 слова команды инициализации ICW_1. Слово ICW_3 используется только в том случае, если имеются ведомые *PIC*.

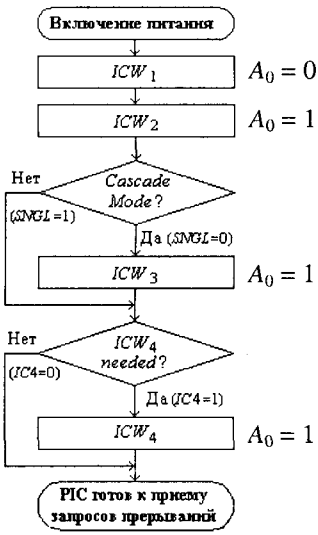


Рис. 3.48. Блок-схема алгоритма инициализации *PIC*

Слово команды инициализации ICW_1. Формат этой команды представлен на рис. 3.49:

$D_0 = IC_4$ — задание использования ($D_0 = 1$) или не использования ($D_0 = 0$) команды ICW_4; для *PIC* 8259 разряд $D_0 \equiv 0$;

$D_1 = SNGL$ — задание использования одного ($D_1 = 1$) или более одного ($D_1 = 0$) *PIC*;

$D_2 = ADI$ — задание интервала ΔA_{BC} адресов *addr* команды CALL, равным 4 ($D_2 = 1$) или 8 ($D_2 = 0$);

$D_3 = LTIM$ — задание запроса прерывания по уровню сигналов IR_i ($D_3 = 1$) или по положительному фронту ($D_3 = 0$);

$D_4 = 1$ — признак ICW_1 (адресация регистра памяти ICW_1);

$D_7D_6D_5 = A_7A_6A_5$ — значения трех разрядов младшего байта базового адреса A_{BC} команды CALL *addr* (в МП 8086/8088 не используются).

Базовый адрес A_{BC} команды CALL — это адрес, сообщаемый *PIC* при инициализации, который присваивается входу запроса прерывания IR_0. Адреса *addr* команд CALL для других входов IR_i вычисляет *PIC* самостоятельно в соответствии с формулой (табл. 3.10):

$$addr = A_{BC} + i \cdot \Delta A_{BC}, \quad i = 0, 1, \ldots, 7.$$

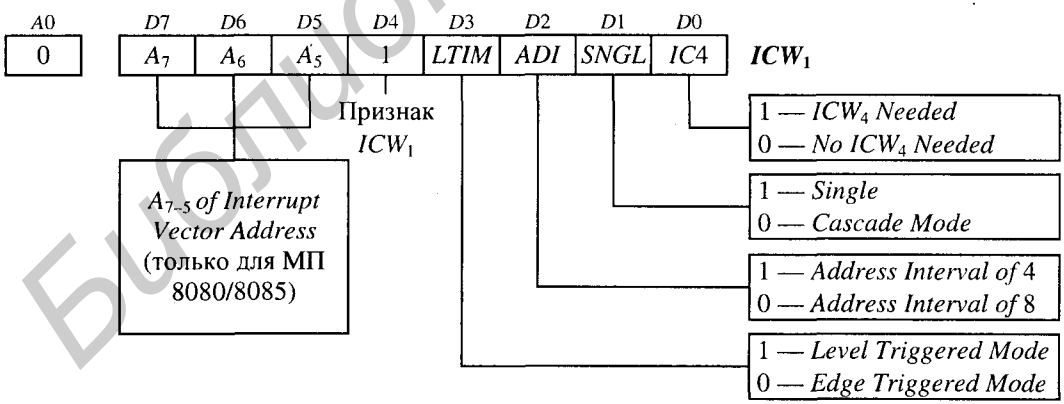


Рис. 3.49. Формат слова команды инициализации ICW_1

Слово команды инициализации ICW_2. Формат данной команды представлен на рис. 3.50:

$A_{15} \ldots A_9A_8$ — значение старшего байта базового адреса A_{BC} команды CALL *addr* для МП 8080/8085;

$T_7T_6T_5T_4T_3$ — значения пяти старших разрядов байта *type* типа прерывания команды вызова подпрограмм INT *type* для МП 8086/8088/80186. Значение байта *type* = $T_7T_6T_5T_4T_3t_2t_1t_0$, где

$t_2t_1t_0 = i$ — номер входа IR_i (табл. 3.10). Чтение байта *type* выполняется вторым импульсом $\overline{INTA}$ ($\overline{INTA}$ = ⎍⎍). По первому импульсу $\overline{INTA} = 0$ никаких действий по шине данных не производится.

Команды INT *type* используются и для программного вызова подпрограмм (прародителем команды INT *type* можно считать команду RST *n* в МП 8080/8085). По типу прерывания *type* = 00 ÷ FF*h* микропроцессор 8086 вычисляет значение 4 × *type*, которое является адресом памяти, по которому находится адрес подпрограммы обслуживания прерывания (два байта для сегмента кода *CS* и два байта для указателя инструкции *IP* — аналога счетчика команд *PC* в МП 8080/8085). Таблица адресов вызова подпрограмм занимает 256 × 4 = 1024 байта и располагается с нулевого адреса памяти (см. рис. 4.26).

Таблица 3.10. **Задание адресов команд CALL и типа прерывания *type***

i	$\Delta A_{BC} = 4$								$\Delta A_{BC} = 8$								*ADI* = × (в ICW_1)							
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	A_7	A_6	A_5	0	0	0	0	0	A_7	A_6	0	0	0	0	0	0	T_7	T_6	T_5	T_4	T_3	0	0	0
1	A_7	A_6	A_5	0	0	1	0	0	A_7	A_6	0	0	1	0	0	0	T_7	T_6	T_5	T_4	T_3	0	0	1
2	A_7	A_6	A_5	0	1	0	0	0	A_7	A_6	0	1	0	0	0	0	T_7	T_6	T_5	T_4	T_3	0	1	0
3	A_7	A_6	A_5	0	1	1	0	0	A_7	A_6	0	1	1	0	0	0	T_7	T_6	T_5	T_4	T_3	0	1	1
4	A_7	A_6	A_5	1	0	0	0	0	A_7	A_6	1	0	0	0	0	0	T_7	T_6	T_5	T_4	T_3	1	0	0
5	A_7	A_6	A_5	1	0	1	0	0	A_7	A_6	1	0	1	0	0	0	T_7	T_6	T_5	T_4	T_3	1	0	1
6	A_7	A_6	A_5	1	1	0	0	0	A_7	A_6	1	1	0	0	0	0	T_7	T_6	T_5	T_4	T_3	1	1	0
7	A_7	A_6	A_5	1	1	1	0	0	A_7	A_6	1	1	1	0	0	0	T_7	T_6	T_5	T_4	T_3	1	1	1

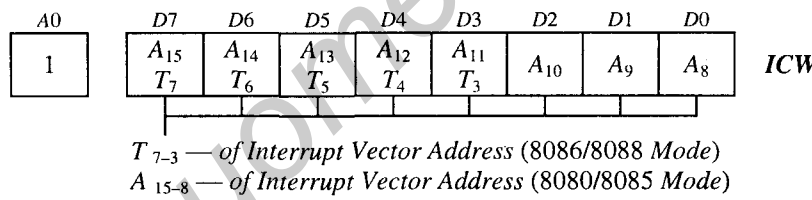


Рис. 3.50. Формат слова команды инициализации ICW_2

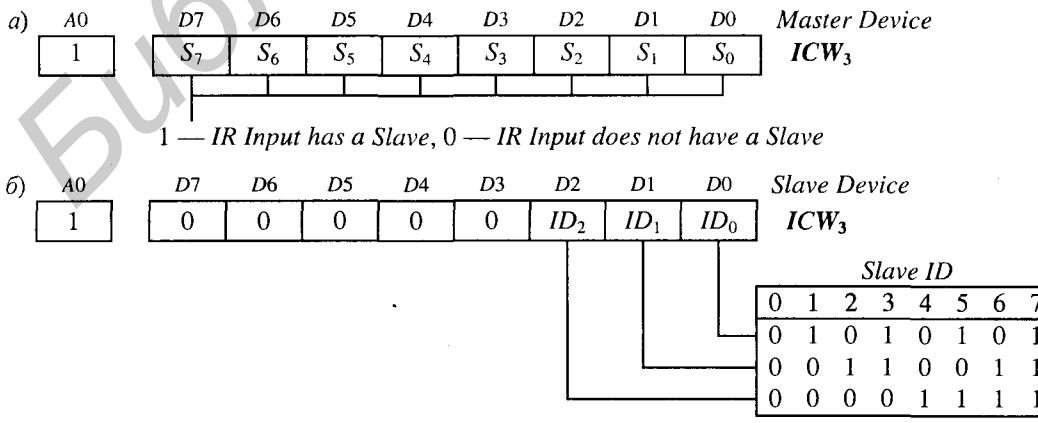


Рис. 3.51. Формат слов команд инициализации ICW_3

Слова команд инициализации ICW_3. Форматы этих команд для ведущего (*Master Device*) и ведомых (*Slave Device*) контроллеров прерываний представлены на рис. 3.51:

$S_r = 1$, если к входу IR_r ведущего *PIC* подключен ведомый *PIC* (рис. 3.51, *а* — инициализация ведущего *PIC*); данная информация используется ведущим *PIC* для выдачи номера *r* по линиям каскадирования CAS_{2-0} для включения ведомого *PIC* (с помощью компараторов) для чтения из него значения *addr* команды CALL; первый байт команды CALL *addr* (машинный код равен CD*h*) всегда читается из ведущего *PIC*;

$ID_2ID_1ID_0 = r$ — номер входа ведущего *PIC*, к которому подключен ведомый *PIC* (рис. 3.51, *б* — инициализация ведомого *PIC*); этот номер используется для сравнения со значением $r = CAS_2CAS_1CAS_0$, выдаваемым ведущим *PIC* — только при совпадении этих номеров соответствующий ведомый *PIC* подключается к шине данных для чтения из него сигналом $\overline{INTA}$ адреса *addr* команды CALL для МП 8080/8085 или типа прерывания *type* для МП 8086/8088.

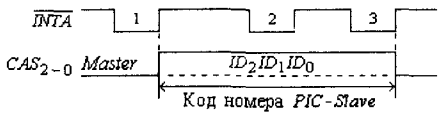


Рис. 3.52. Сигналы каскадирования

На рис. 3.52 изображены временны́е диаграммы, поясняющие работу схемы, приведенной на рис. 3.45, при поступлении запроса прерывания от внешнего устройства на один из ведомых *PIC*:

1) первый импульс $\overline{INTA} = 0$ передает на шину данных из ведущего *PIC* машинный код CD*h* команды CALL и устанавливает на выходах линий каскадирования CAS_{2-0} ведущего (*Master*) *PIC* номер $ID_2ID_1ID_0$, идентифицирующий ведомый *PIC*, на который поступил запрос прерывания от внешнего устройства;

2) компараторы ведомых *PIC* сравнивают номер $ID_2ID_1ID_0$ на линиях CAS_{2-0} с номером $ID_2ID_1ID_0$, записанным командой ICW_3 при их инициализации; второй и третий импульсы $\overline{INTA} = 0$ производят чтение двухбайтового адреса команды CALL из того ведомого *PIC*, у которого указанные номера совпали.

Если к входу IR_r ведущего *PIC* не подключен ведомый *PIC*, то на выходах линий каскадирования CAS_{2-0} устанавливаются значения 0.

Слово команды инициализации ICW_4. Формат данной команды показан на рис. 3.53:

$D_0 = \mu FM$ — задание типа МП (8080/8085 или 8086/8088);

$D_1 = AEOI$ (*Auto End of Interrupt*) — задание автоматического окончания прерывания; при $D_1 = 1$ разряд $T_{ISR\ i}$ сбрасывается в 0 в конце последнего импульса $\overline{INTA}$ (см. рис. 3.44), а при $D_1 = 0$ — командой конца прерывания OCW_2, находящейся в конце ППОП;

$D_2 = M/S$ (*Master/Slave*) — программирование использования *PIC* в качестве ведущего ($D_2 = 1$) или ведомого ($D_2 = 0$) вместо аппаратного программирования сигналом $\overline{SP/EN}$;

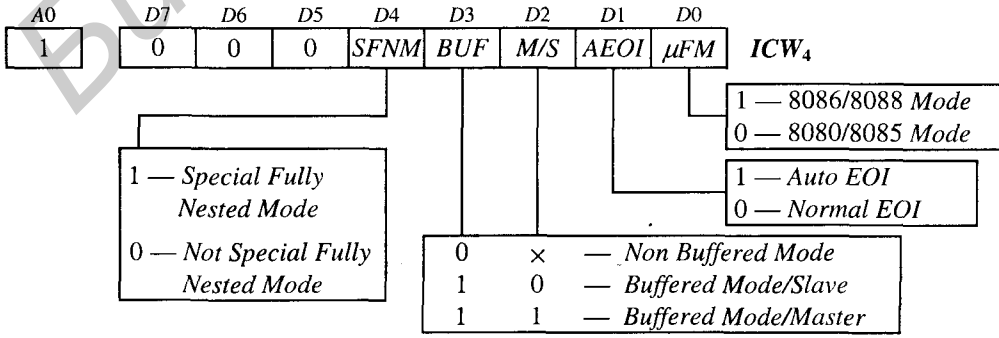


Рис. 3.53. Формат слова команды инициализации ICW_4

$D_3 = BUF$ — задание буферированного ($D_3 = 1$) и небуферированного ($D_3 = 0$) режимов; при $D_3 = 1$ вывод $\overline{SP/EN}$ становится выходом управляющего сигнала для приемопередатчика шины данных, который переводит его в Z-состояние при чтении информации из *PIC*; при $D_3 = 0$ разряд *M/S* игнорируется;

$D_4 = SFNM$ — задание специального режима полной вложенности.

При записи слова ICW_1 автоматически производятся установки: сбрасывается в 0 схема, чувствительная к фронту сигнала IR_i (*Edge sense Latch* — рис. 3.54); все разряды регистра маски *IMR* сбрасываются в 0 (прерывания по всем входам IR_i разрешены); входу IR_7 присваивается наименьший приоритет, а входу IR_0 — наибольший приоритет; сбрасывается режим специального маскирования и устанавливается состояние чтения регистра *IRR* (см. рис. 3.60). Если в слове ICW_1 разряд $IC_4 = 0$, то для *PIC* 8259*A* устанавливаются режимы, соответствующие значению слова $ICW_4 = 00h$ (небуферированный режим, не автоматический конец прерывания, МП 8080/8085). После инициализации *PIC* готов к работе в режиме полной вложенности. Достоинством запроса прерывания по положительному фронту ($LTIM = 0$ в команде ICW_1) является то, что вход IR_i может оставаться в состоянии 1, не генерируя повторного прерывания.

Если при поступлении первого импульса $\overline{INTA}$ запрос прерывания $IR_i = 0$ (значение запроса $IR_i = 1$ было слишком малой длительности), то *PIC* 8259*A* по умолчанию выдаст прерывание уровня 7 без установки состояния триггера $T_{ISR\ 7} = 1$ ($LTIM = \times$). Байты вектора прерывания (*addr* или *type*) и сигналы на линиях каскадирования CAS_{2-0} будут соответствовать уровню прерывания 7. Это может быть использовано для обнаружения прерываний, вызванных помехами на входах IR_i. Для отработки помех ППОП может содержать единственную команду RET, игнорируя таким образом прерывание. Если запрос прерывания IR_7 необходим для других целей, то запрос от помех может быть обнаружен чтением регистра *ISR* с последующим анализом состояния триггера $T_{ISR\ 7}$.

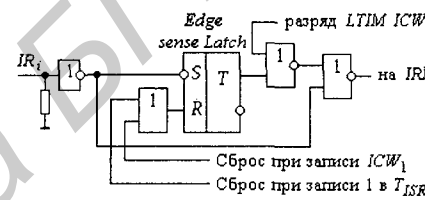


Рис. 3.54. Схема управления запросами прерывания

Примеры инициализации *PIC*. Будем полагать, что МП-система построена на основе МП 8085 и *PIC* 8259*A* по небуферированной схеме и требуется обеспечить запросы прерываний по фронту сигналов IR_i, т. е. в команде ICW_1 можно задать значение разряда $D_0 = 0$, чтобы не использовать команду ICW_4, и требуется задать значение разряда $D_3 = LTIM = 0$.

Пример 1. Пусть базовый адрес A_{BC} = 3A60*h*, интервал адресов $\Delta A_{BC} = 4$ и используется один *PIC*, адреса портов которого равны F8*h* ($A_0 = 0$) и F9*h* ($A_0 = 1$). Из рис. 3.48 следует, что на *PIC* следует подать только два слова инициализации:

$ICW_1 = 76h = 0111\ 0110$ ($A_7A_6A_5 = 011$, $LTIM = 0$, $ADI = 1$, $SNGL = 1$ — см. рис. 3.49);
$ICW_2 = 3Ah = 0011\ 1010$ (см. рис. 3.50).

Инициализация *PIC* выполняется программой:

```
MVI   A, 76h    ; A ← ICW1 = 76h
OUT   0F8h      ; Запись в PIC команды ICW1
MVI   A, 3Ah    ; A ← ICW2 = 3Ah — старший байт базового адреса
OUT   0F9h      ; Запись в PIC команды ICW2
EI              ; Флаг IE ← 1 (прерывания разрешены; триггер INTE ← 1 в МП 8080)
```

В результате входам IR_i будут соответствовать адреса $addr = (3A60 + i \cdot 4)h$ команд CALL, читаемых МП из *PIC*. Обычно ППОП требуют объем памяти более четырех байт, поэтому по адресам *addr* могут быть записаны только трехбайтовые команды JMP AIR_i, а четвертый байт либо не используется (указывается команда NOP), либо перед командой перехода используется однобайтовая команда из ППОП:

```
3A60    JMP   AIR0     ; Начало ППОП для входа IR0
3A63    NOP            ; Один байт не используется
3A64    PUSH  PSW      ; Начало ППОП для входа IR1
3A65    JMP   AIR1
3A68    EI             ; Начало ППОП для входа IR2
3A69    JMP   AIR2
3A6C    JMP   AIR3     ; Начало ППОП для входа IR3
          ∴
3A7C    JMP   AIR7     ; Начало ППОП для входа IR7
          ∴
AIR0:   PUSH  PSW      ; Продолжение ППОП для входа IR0
          ∴
        POP   PSW
        EI             ; Разрешение прерываний
        RET            ; Конец ППОП для входа IR0
AIR1:   PUSH  H        ; Продолжение ППОП для входа IR1
          ∴
        POP   H
        POP   PSW
        EI             ; Разрешение прерываний
        RET            ; Конец ППОП для входа IR1
AIR2:     ∴            ; Продолжение ППОП для входа IR2
```

Если в МП-системе используется несколько *PIC* (ведущий и ведомые), то вышеприведенная программа инициализации должна выполняться для каждого *PIC* с различными базовыми адресами A_{BC} и подачей команды инициализации ICW_3 (см. рис. 3.51).

Пример 2. Для условий предыдущего примера выполнить инициализацию *PIC* в системе, изображенной на рис. 3.45, использовав непрерывные массивы адресов команд CALL и портов *PIC*:

A_{BC} = 3A60*h*, *port* = F8*h* и F9*h* — для ведущего *PIC* (*Master* — рис. 3.55),
A_{BC} = 3A80*h*, *port* = FA*h* и FB*h* — для ведомого *A* (*Slave A* — рис. 3.55),
A_{BC} = 3AA0*h*, *port* = FC*h* и FD*h* — для ведомого *B* (*Slave B* — рис. 3.55).

На основании рис. 3.49 – 3.51 находим:

$ICW_{1\,M}$ = 74*h* = 0111 0100 ($A_7A_6A_5$ = 011, *LTIM* = 0, *ADI* = 1, *SNGL* = 0),
$ICW_{2\,M}$ = 3A*h* = 0011 1010, $ICW_{3\,M}$ = 48*h* = 0100 1000;
$ICW_{1\,A}$ = 94*h* = 1001 0100 ($A_7A_6A_5$ = 100, *LTIM* = 0, *ADI* = 1, *SNGL* = 0),
$ICW_{2\,A}$ = 3A*h* = 0011 1010, $ICW_{3\,A}$ = 03*h* = 0000 0011;
$ICW_{1\,B}$ = B4*h* = 1011 0100 ($A_7A_6A_5$ = 101, *LTIM* = 0, *ADI* = 1, *SNGL* = 0),
$ICW_{2\,B}$ = 3A*h* = 0011 1010, $ICW_{3\,B}$ = 06*h* = 0000 0110.

Инициализация выполняется программой:

```
MVI  A, 74h  ; A ← ICW1 M = 74h                    Инициализация Master PIC
OUT  0F8h    ; Запись в PIC Master команды ICW1 M
MVI  A, 3Ah  ; A ← ICW2 M = 3Ah
OUT  0F9h    ; Запись в PIC Master команды ICW2 M
MVI  A, 48h  ; A ← ICW3 M = 48h
OUT  0F9h    ; Запись в PIC Master команды ICW3 M
MVI  A, 94h  ; A ← ICW1 A = 94h                    Инициализация Slave A
```

```
OUT  0FAh  ; Запись в PIC Slave A команды ICW1 A
MVI  A, 3Ah ; A ← ICW2 A = 3Ah
OUT  0FBh  ; Запись в PIC Slave A команды ICW2 A
MVI  A, 3   ; A ← ICW3 A = 03h
OUT  0FBh  ; Запись в PIC Slave A команды ICW3 A
MVI  A, B4h ; A ← ICW1 B  Инициализация Slave B
OUT  0FCh  ; Запись в PIC Slave B команды ICW1 B
MVI  A, 3Ah ; A ← ICW2 B = 3Ah
OUT  0FDh  ; Запись в PIC Slave B команды ICW2 B
MVI  A, 6   ; A ← ICW3 B = 06h
OUT  0FDh  ; Запись в PIC Slave B команды ICW3 B
EI         ; Флаг IE ← 1 (прерывания разрешены; триггер INTE ← 1 в МП 8080)
```

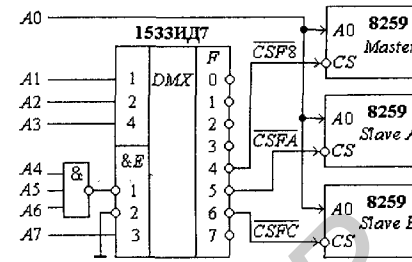


Рис. 3.55. Адресация *PIC*

Задание статуса уровней приоритета. Форма задания статуса представляет собой приоритетное кольцо (рис. 3.56), положение которого по отношению к входам IR_i определяет их уровни приоритета. Приоритетное кольцо можно вращать по отношению к входам IR_i командой OCW_2. Самым высоким уровнем приоритета обладает вход IR_i, которому соответствует номер 0 приоритетного кольца (вход IR_5 на рис. 3.56), а самым низким — вход IR_i, которому соответствует номер 7 приоритетного кольца (вход IR_4 на рис. 3.56), называемый дном приоритетного кольца (ДПК). Присвоить ДПК любому входу IR_i можно командой OCW_2 в процессе выполнения основной программы или ППОП.

По присвоению приоритетов *PIC* может работать в трех режимах.

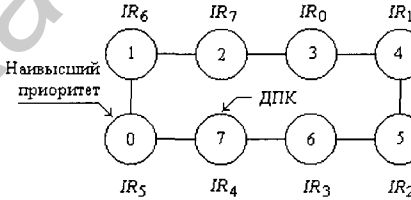


Рис. 3.56. Приоритетное кольцо

1. *Режим полной вложенности.* Входу IR_0 присваивается наивысший приоритет, а входу IR_7 — самый низкий приоритет. Запрос прерывания IR_i с более высоким приоритетом может прервать ППОП внешнего устройства с меньшим приоритетом. Прерванная ППОП будет продолжена по окончании выполнения ППОП внешнего устройства с большим приоритетом (после команды RET). Этот режим устанавливается сразу же после окончания инициализации *PIC*.

2. *Режим циклического приоритета.* Приоритеты циклически сдвигаются командой OCW_2, находящейся в конце ППОП, так что обслуженному I/O_i присваивается ДПК. Этот режим исключает возможность захвата МП одним внешним устройством, предоставляя всем *I/O* равные возможности по обслуживанию. Применение этого режима решает также проблему множества *I/O*, имеющих равный приоритет.

3. *Режим заданного приоритета.* Положение ДПК командой OCW_2 присваивается некоторому входу IR_i в произвольные моменты времени в зависимости от потребностей главной программы или ППОП.

Каждый режим может быть видоизменен применением масок, которые задаются командой OCW_1 и позволяют замаскировать любой вход IR_i независимо от его приоритета.

Управление работой *PIC*. Для задания операций и алгоритмов работы *PIC* предназначены слова команд операций OCW_m (*Operation Command Word*), $m = 1, 2, 3$. В отличие от команд инициализации эти команды используются многократно как при выполнении ППОП, так и при выполнении основной программы.

Слово команды операции OCW_1. Формат команды OCW_1 представлен на рис. 3.57: если $M_i = 1$, то запрос прерывания IR_i фиксируется в регистре *IRR*, но прохождение его в схему *PR* блокируется. Как только маска будет снята ($M_i = 0$), то запрос прерывания IR_i будет обслужен, если нет запросов с более высоким приоритетом.

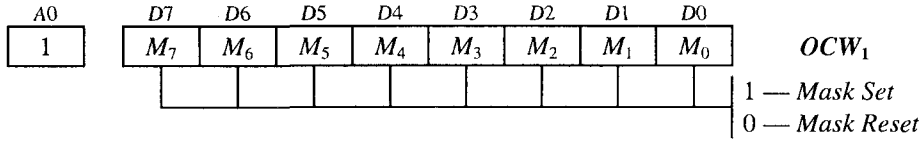


Рис. 3.57. Формат слова команды операции OCW_1

Пример 3. Замаскировать входы запросов прерываний IR_6, IR_4 и IR_2, снять маску с входов IR_5 и IR_1, оставив маску остальных входов неизменной; считаем, что адреса портов *PIC* равны FE*h* и FF*h*.

Эта задача решается программой:

```
IN    0FFh    ; A ← IMR — чтение исходной маски
ORI   54h     ; 01010100 = 54h — маскирование входов 6, 4 и 2
ANI   0DDh    ; 11011101 = DDh — снятие маски с входов 5 и 1
OUT   0FFh    ; IMR ← (IMR ∨ 54h) & DDh
```

Слово команды операции OCW_2. Данная команда используется для задания конца обработки прерывания (сброс триггера $T_{ISR\ i}$ в 0 обслуженного входа IR_i) и циклических сдвигов приоритетов. Формат команды OCW_2 показан на рис. 3.58:

$D_2D_1D_0 = L_2L_1L_0 = i$ — номер входа IR_i. Эти разряды определяют уровень запроса прерывания $i = 0 \ldots 7$, на который воздействуют операции, заданные разрядами *EOI*, *SL*, и *R*. Если разряд $SL = 0$, то разряды $L_2L_1L_0$ игнорируются;

$D_4D_3 = 00$ — признак команды OCW_2;

$D_5 = EOI$ (*End of Interrupt*) — конец прерывания для режима не автоматического конца прерывания; если разряд $EOI = 1$, то тип конца прерывания будет определяться состоянием разрядов *SL* и *R*. Если разряд $EOI = 0$, то команда конца прерывания не будет выполнена (триггер $T_{ISR\ i}$ не сбрасывается в 0);

$D_6 = SL$ (*Specific Level*) — специальный конец прерывания; если разряд $SL = 1$, то разряды $L_2L_1L_0$ используются для задания уровня i прерывания, для которого будет выполнена операция, задаваемая разрядами *EOI* и *R*. Если разряд $SL = 0$, то разряды $L_2L_1L_0$ игнорируются;

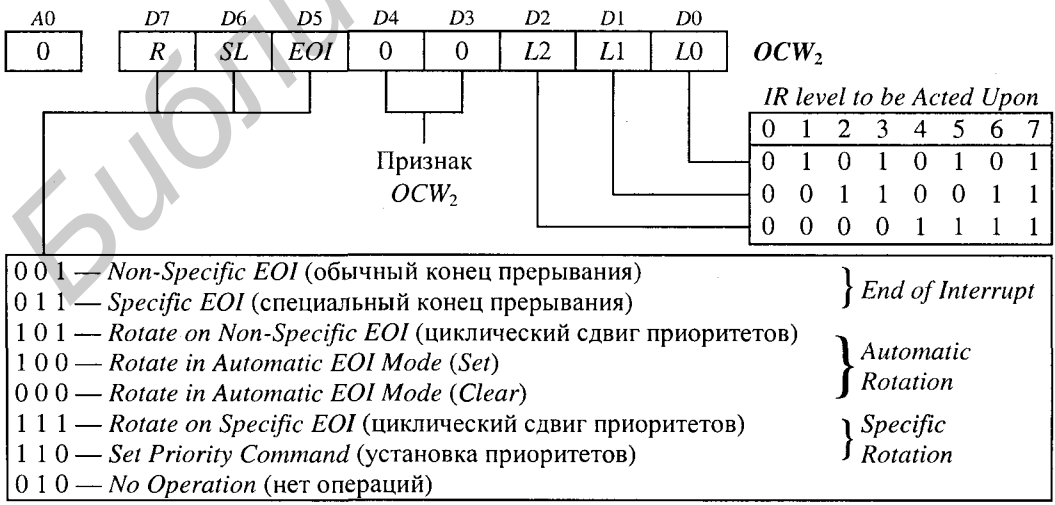


Рис. 3.58. Формат слова команды операции OCW_2

D_7 = R (*Rotate*) — циклический сдвиг уровней приоритетов; если разряд R = 1, то тип циклического сдвига будет определяться состоянием разрядов *SL* и *EOI* (при значении R = 0 сдвига нет).

Команда OCW_2 выполняет функции:

$D_7D_6D_5$ = 001 — обычный конец прерывания (фиксированные приоритеты и сброс в 0 триггера $T_{ISR\,i}$ обслуженного запроса прерывания; $L_2L_1L_0$ = ×××);

$D_7D_6D_5$ = 011 — специальный конец прерывания (фиксированные приоритеты и сброс в 0 триггера $T_{ISR\,i}$, указываемого двоичным кодом $L_2L_1L_0 = i$);

$D_7D_6D_5$ = 101 — обычный конец прерывания (циклический сдвиг приоритетов с присвоением ДПК обслуженному запросу и сбросом в 0 его триггера $T_{ISR\,i}$; $L_2L_1L_0$ = ×××);

$D_7D_6D_5$ = 100 — установка режима циклического сдвига приоритетов при автоматическом конце прерывания, заданного командой ICW_4 (разряд *AEOI* = 1);

$D_7D_6D_5$ = 000 — сброс режима циклического сдвига приоритетов при автоматическом конце прерывания, заданного командой ICW_4;

$D_7D_6D_5$ = 111 — специальный конец прерывания ($L_2L_1L_0 = i$ — двоичный код сбрасываемого в 0 триггера $T_{ISR\,i}$ и присвоение ДПК входу IR_i с циклическим сдвигом приоритетов);

$D_7D_6D_5$ = 110 — циклический сдвиг приоритетов без завершения прерывания ($L_2L_1L_0 = i$ — двоичный код номера входа IR_i, которому присваивается ДПК).

Ниже приведено типовое оформление ППОП (написано продолжение *примера* 1, начиная с метки AIR_0):

```
AIR0: PUSH   PSW      ; ППОП для входа запроса прерывания IR0
      PUSH   H        ; Запись в стек состояния прерываемой программы
      PUSH   D
      PUSH   B
        ∴             ; Собственно ППОП
      POP    B        ; Восстановление состояния прерванной программы
      POP    D
      POP    H
      MVI    A, 20h   ; A ← 20h = OCW2 — Non Specific EOI
      OUT    0F8h     ; PIC ← OCW2 — конец прерывания
      POP    PSW
      EI              ; Флаг IE ← 1 (прерывания разрешены; триггер INTE ← 1 в МП 8080)
      RET             ; Возврат на адрес прерванной программы
```

Команда *EI* может располагаться в любом месте ППОП. Если она расположена в конце ППОП, то ее не могут прервать запросы прерывания с более высоким приоритетом, так как флаги *IE* в МП 8085 и *INTE* в МП 8080 сбрасываются в 0 при каждом переходе к обслуживанию запроса прерывания. В стек можно записывать содержимое только тех регистровых пар, которые используются в ППОП.

Если в МП-системе используются ведомые *PIC*, то команда *EOI* должна подаваться дважды: в ведущий *PIC* и в соответствующий ведомый *PIC*. Если при инициализации был задан запрос прерывания по уровню (*LTIM* = 1 в команде ICW_1), то для предотвращения повторного прерывания запрос IR_i должен быть сброшен до подачи команды *EOI*.

При работе в режиме полной вложенности команда *EOI* обычного конца прерывания (R = 0, *SL* = 0, *EOI* = 1) автоматически сбрасывает триггер $T_{ISR\,i}$, соответствующий запросу прерывания, имеющего наибольший приоритет из всех находящихся в обслуживании запросов прерывания (рис. 3.59 — принцип работы подобной схемы описан в § 6.6 книги [5]). В этом режиме сбрасываемый запрос прерывания всегда соответствует обслуженному запросу прерывания.

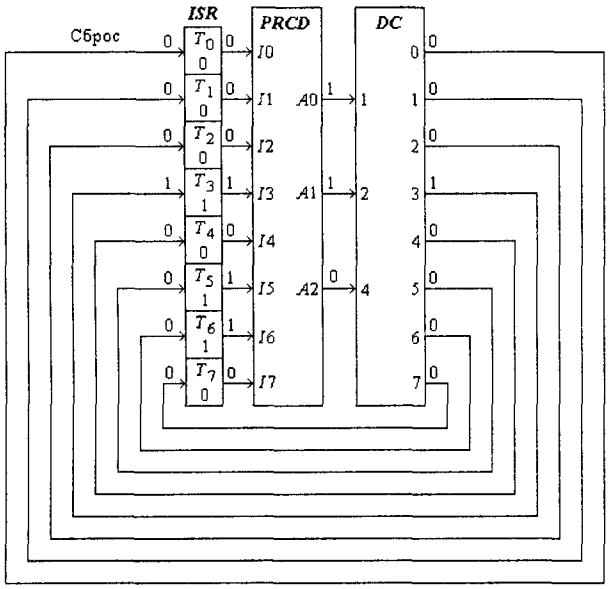


Рис. 3.59. Схема сброса триггеров регистра *ISR*

При нарушении режима полной вложенности, например командой циклического сдвига приоритетов, это соответствие отсутствует. В этом случае следует применять команду *EOI* специального конца прерывания (R = 0 или 1, SL = 1, EOI = 1) с явным заданием разрядами $L_2L_1L_0$ команды OCW_2 сбрасываемого триггера $T_{ISR\ i}$. Полная схема сброса триггеров регистра *ISR* должна содержать двухканальный 3-разрядный мультиплексор, переключающий входные сигналы дешифратора *DC*: при значении разряда SL = 0 должны подаваться сигналы A_2, A_1 и A_0, а при значении SL = 1 — сигналы L_2, L_1 и L_0.

Слово команды операции OCW_3. Эта команда используется для задания режима работы *PIC* по опросу (режим полинга) и режима специального маскирования, а также для управления чтением регистров *IRR* и *ISR*. Формат этой команды изображен на рис. 3.60:

D_0 = *RIS* (*Read ISR*) — чтение регистров *ISR* при *RIS* = 1 и *IRR* при *RIS* = 0; чтение возможно только при значении разряда *RR* = 1;

D_1 = *RR* (*Read Register*) — чтение состояния регистров *ISR* и *IRR*;

D_2 = *P* (*Polling* — опрос) — режим работы *PIC* по опросу (*P* = 1 — режим полинга); если одновременно установлены значения *P* = 1 и *RR* = 1, то выполняется режим полинга;

D_5 = *SMM* (*Special Mask Mode*) — установка (*SMM* = 1) и отмена (*SMM* = 0) режима специального маскирования при значении разряда *ESMM* = 1;

D_6 = *ESMM* (*Enable SMM*) — разрешение режима специального маскирования.

A0	*D7*	*D6*	*D5*	*D4*	*D3*	*D2*	*D1*	*D0*	
0	0	*ESMM*	*SMM*	0	1	*P*	*RR*	*RIS*	OCW_3
0C*h* =	0	0	0			1	0	0	— *Poll Mode*
0B*h* =	0	0	0	Признак		0	1	1	— *Read ISR*
0A*h* =	0	0	0	OCW_3		0	1	0	— *Read IRR*
68*h* =	0	1	1			0	0	0	— *Set Special Mask Mode*
48*h* =	0	1	0			0	0	0	— *Reset Special Mask Mode*

Рис. 3.60. Формат слова команды операции OCW_3

Режим полинга. После подачи на *PIC* команды OCW_3 = 0C*h* командой IN *port* в МП вводится уровень прерывания *IL* (см. табл. 3.9), формат которого изображен на рис. 3.61:

I — запрос прерывания (*I* = 1, если был запрос прерывания хотя бы по одному входу IR_i);

$W_2W_1W_0 = i$ — номер входа IR_i, имеющего набольший приоритет, при одновременном запросе обслуживания по нескольким входам; $W_2W_1W_0$ = 111 при *I* = 0.

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	I	—	—	—	—	W_2	W_1	W_0

Рис. 3.61. Формат уровня прерывания *IL*

Прочитав уровень прерывания *IL*, МП по номеру *i* программным способом определяет, какое внешнее устройство следует обслужить. Импульс $\overline{RD}$ = 0, генерируемый командой IN *port* при чтении уровня прерывания *IL*, воспринимается *PIC* как подтверждение прерывания (вместо сигнала $\overline{INTA}$), устанавливая, если имеется запрос прерывания, состояние соответствующего триггера $T_{ISR\,i} = 1$.

Инициализацию *PIC* 8259*A*, предназначенных для работы в режиме полинга, тоже нужно производить, но только двумя командами ICW_1 и ICW_2 = 00*h*, где 00*h* — "фиктивный" адрес (*можно задать и любое другое число, так как адрес команды* CALL *в режиме полинга не используется*).

Режим полинга (OCW_3 = 0C*h*) позволяет включить в МП-систему любое число *PIC* 8259*A* в пределах адресного пространства ввода-вывода МП. В режиме полинга производится последовательный опрос *PIC* по инициативе МП (сигналы *INT* и $\overline{INTA}$ не используются). Например, если в МП-системе необходимо обслуживать 80 внешних устройств, то следует использовать один ведущий *PIC*, 8 ведомых *PIC* (обслуживание 64 внешних устройств по прерыванию с использованием сигналов *INT* и $\overline{INTA}$) и два *PIC*, работающих в режиме полинга.

Пример 4. Выполнить обслуживание запроса от внешнего устройства, если он поступил на *PIC* 8259*A*, работающий в режиме полинга и имеющего адреса портов E0*h* и E1*h*. Эта задача решается программой:

```
     MVI  A, 0Ch ; A ← 0Ch — OCW3 (режим полинга)
     OUT  0E0h   ; 8259A ← команда OCW3
     IN   0E0h   ; A ← IL = I ×××× W2W1W0 — чтение уровня прерывания IL
     ANI  87h    ; A ← I 0000 W2W1W0
     CPI  07h    ; Проверка разряда D7 = I
     JZ   L1     ; Переход, если I = 0 (не было запроса обслуживания — "холостая" проверка)
      ∴          ; Далее анализ числа i = W2W1W0 для перехода на обслуживание I/O–i,
      ∴          ;      подобный анализу числа m = A2A1A0 в программах
L1:   ∴          ;      для рис. 2.12, б (см. § 2.4)
```

Использование дополнительных *PIC* 8259*A*, не связанных с *PIC*, обслуживающих 64 внешних устройства по прерыванию с использованием сигналов *INT* и $\overline{INTA}$, из-за непроизводительных потерь времени на опрос оправдано только в тех случаях, когда они расположены на другой плате и сигналы *INT* и $\overline{INTA}$ недоступны. Если все *PIC* 8259*A* расположены на одной плате, то расширение системы прерываний на более чем 64 уровня, можно выполнить более эффективно, использовав третий уровень *PIC* 8259*A*.

На рис. 3.62 изображен контроллер прерываний с 512-уровневой системой прерываний при совместном использовании режима полинга и векторной системы прерываний. Данный контроллер содержит 73 БИС 8259*A* — в первом ярусе одна БИС, во втором ярусе восемь БИС и в третьем ярусе 64 БИС, предназначенных для работы в режиме полинга. Первые два яруса представляют собой каскадированный контроллер прерываний с 64 уровнями прерываний, подобный изображенному на рис. 3.45. Адреса *addr* всех команд *CALL* вызова подпрограмм обслуживания прерываний запрограммированы при инициализации в *PIC* 8259*A* второго яруса, поэтому при инициализации ведущего *PIC* (*Master*), как и всех *PIC* третьего яруса, используется команда ICW_2 = 00*h* с "фиктивным" значением 00*h* старшего байта адреса *addr*. В третьем

ярусе задаются значения сигналов $\overline{SP/EN}$ = $\overline{INTA}$ = 1, т. е. все *PIC* этого яруса установлены в режим ведущего (*MP* 63 ÷ 0 — *Master Polling*) и чтение адреса *addr* из них не производится.

При инициализации рассматриваемого контроллера прерываний можно задать специальный режим полной вложенности. Для этого в каждую БИС первого и второго ярусов следует подать по четыре команды инициализации (ICW_1, ICW_2, ICW_3 и ICW_4), а в БИС третьего яруса — только команды ICW_1 и ICW_2. Для БИС второго яруса следует задать ICW_1 = 15*h*, 35*h*, 55*h*, 75*h*, 95*h*, B5*h*, D5*h*, F5*h* и ICW_2 = 10*h* для получения базовых адресов A_{BC} = 1000*h*, 1020*h*, 1040*h*, 1060*h*, 1080*h*, 10A0*h*, 10C0*h*, 10E0*h* (конечно можно задать и другие значения базовых адресов).

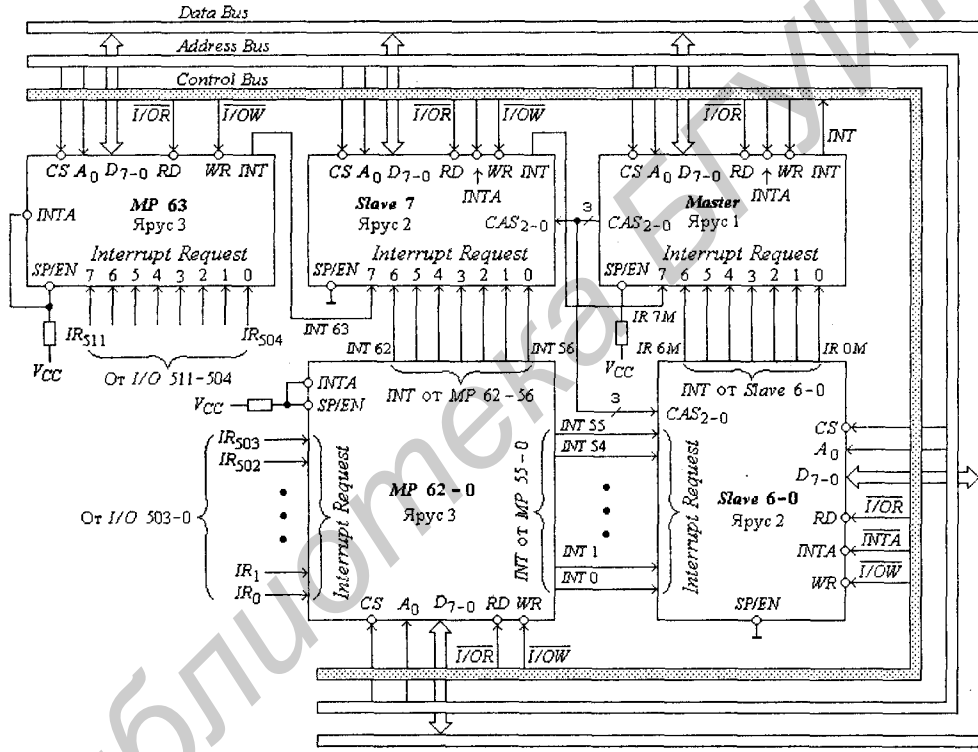


Рис. 3.62. Трехъярусная схема каскадирования *PIC* 8259*A*

Значение запроса прерывания IR_i = 1 от любого внешнего устройства *I/O* 0 ÷ 511 установит значение сигнала *INT* = 1 ведущего *PIC*, что вызовет чтение команды CALL *addr* значениями сигналов $\overline{INTA}$ = 0 (адрес *addr* поступает на шину данных из БИС второго яруса, соответствующей входу IR_i). В вызванной командой CALL *addr* подпрограмме обработки прерывания производится чтение уровня прерывания *IL*, как и в *примере* 4, но проверку разряда D_7 = *1* выполнять не нужно, так как заранее известно, что запрос обслуживания поступил ("холостые" проверки исключаются). Команду OCW_3 нужно подавать каждый раз перед чтением уровня прерывания *IL*.

Чтение состояния регистров. Чтение регистра *ISR* производится командой IN *port* после подачи команды OCW_3 = 0B*h* и регистра *IRR* — после подачи команды OCW_3 = 0A*h*. Каждый из регистров после подачи соответствующей команды OCW_3 можно читать многократно.

Пример 5. Чтение состояния регистра *ISR* (считаем, что адреса портов *PIC* равны E0*h* и E1*h*):

```
MVI  A, 0Bh   ; A ← 0Bh = OCW3 — команда чтения состояния регистра ISR
OUT  0E0h     ; 8259A ← команда OCW3
IN   0E0h     ; A ← состояние ISR
```

Режим специального маскирования. Данный режим задается командой $OCW_3 = 68h$ в тех случаях, когда требуется обслужить запрос прерывания, который блокируется старшим по уровню приоритета обслуживаемым запросом прерывания IR_i, хранящимся в триггере $T_{ISR\ i}$, не сбрасывая его. Эта команда используется совместно с командой OCW_1, предварительно маскирующей запрос прерывания IR_i.

Пример 6. Пусть требуется установить спецмаскирование входа IR_2. ППОП обслуживания запроса прерывания по входу IR_2 должна содержать фрагмент (адреса портов *PIC* равны FE*h* и FF*h*):

```
EI            ; Флаг IE ← 1 (прерывания разрешены; триггер INTE ← 1 в МП 8080)
 ∴            ; Запросы прерываний по входам IR7-3 (с более низким приоритетом)
 ∴            ;      блокированы
DI            ; Флаг IE ← 0 (прерывания запрещены; триггер INTE ← 0 в МП 8080)
IN   0FFh     ; A ← IMR — чтение регистра маски
ORI  4        ; 4 = 00000100 — маскирование входа IR2
OUT  0FFh     ; IMR ← OCW1 — запись новой маски
MVI  A, 68h   ; A ← 68h = OCW3
OUT  0FEh     ; PIC ← OCW3 — установка режима специального маскирования
EI            ; Флаг IE ← 1 (прерывания разрешены; триггер INTE ← 1 в МП 8080)
 ∴            ; На этом участке запросы прерываний по входам IR7-3 вызывают
 ∴            ;      прерывание данной ППОП
DI            ; Флаг IE ← 0 (прерывания запрещены; триггер INTE ← 0 в МП 8080)
IN   0FFh     ; Чтение регистра маски IMR
ANI  0FBh     ; FBh = 11111011 — снятие маски с входа IR2
OUT  0FFh     ; IMR ← OCW1 — восстановление исходной маски
MVI  A, 48h   ; A ← 48h = OCW3
OUT  0FEh     ; PIC ← OCW3 — сброс режима специального маскирования
EI            ; Флаг IE ← 1 (прерывания разрешены; триггер INTE ← 1 в МП 8080)
 ∴            ; Запросы прерываний по входам IR7-3 блокированы
```

Команда специального маскирования $OCW_3 = 68h$ действует на все разряды i, которые замаскированы командой OCW_1, до момента отмены режима специального маскирования командой $OCW_3 = 48h$. Если *PIC* находится в режиме специального маскирования, то триггер $T_{ISR\ i}$, замаскированный командой OCW_1, не сбрасывается операцией *EOI* обычного конца прерывания, задаваемой командой OCW_2.

Режим автоматического конца прерывания (*AEOI*). Если в команде ICW_4 разряд $AEOI = 1$, то *PIC* 8259*A* функционирует в режиме *AEOI* пока не перепрограммировано слово ICW_4. В этом режиме *PIC* автоматически выполняет операцию *EOI* обычного конца прерывания по заднему фронту последнего импульса подтверждения прерывания $\overline{INTA}$ (третий импульс в МП 8080/8085 и второй в МП 8086/8088). Очевидное преимущество режима *AEOI* — не требуется подавать команду OCW_2 с указанием операции *EOI*, что упрощает программирование.

Так как в режиме *AEOI* триггер $T_{ISR\ i}$ сбрасывается автоматически (см. рис. 3.44), то этот режим может использоваться только в тех случаях, когда не требуется вложенная многоуров-

невая система прерываний. Это означает, что в режиме *AEOI* необходимо запретить прерывания выполняемой ППОП другими запросами прерывания (команду EI следует помещать в самом конце всех ППОП). Режим *AEOI* допустим только для ведущего *PIC* (если в МП-системе имеется только один *PIC*, то он тоже является ведущим).

Специальный режим полной вложенности. При использовании в МП-системе ведомых *PIC* поддержка вложенной структуры запросов прерываний, вызывающих ППОП для обслуживания внешних устройств, усложняется. В каждую ППОП, вызываемую ведомыми *PIC*, необходимо включать команды управления как ведомым, так и ведущим *PIC*. Например, для рассмотренного выше *примера* 2 (с. 234) оформление ППОП для входа IR_2 ведомого *PIC Slave A*, подключенного к входу IR_3 ведущего (*Master*) *PIC*, можно выполнить так:

```
SL_A2: PUSH PSW    ; ППОП для входа запроса прерывания IR2 Slave A (см. рис. 3.45)
       PUSH H      ; Запись в стек состояния прерываемой программы
       PUSH D
       PUSH B
       IN   0F9h   ; A ← IMR — чтение маски PIC Master
       MOV  E, A   ; E ← IMR — сохранение маски в регистре E
       ORI  0F0h   ; C0h = 1111 0000 = F0h — маскирование запросов IR7-4 ведущего PIC
       OUT  0F9h   ; IMR Master ← модифицированная маска
       MVI  A, 63h ; A ← OCW2 (R = 0, SL = 1, EOI = 1, L2L1L0 = 011 = 3)
       OUT  0F8h   ; Сброс триггера TISR 3 ведущего PIC
       EI          ; Флаг IE ← 1 (прерывания разрешены; триггер INTE ← 1 в МП 8080)
         ∴         ; Собственно ППОП. Теперь PIC Slave A будет вызывать ППОП
         ∴         ;     при запросах прерываний по входам IR1 и IR0
       DI          ; Флаг IE ← 0 (прерывания запрещены; триггер INTE ← 0 в МП 8080)
       MVI  A, 20h ; 20h = OCW2 — Non Specific EOI (обычный конец прерывания)
       OUT  0FAh   ; PIC Slave A ← OCW2 — конец прерывания
       MOV  A, E   ; A ← E — сохраненная ранее маска
       OUT  0F9h   ; Восстановление маски PIC Master
       POP  B      ; Восстановление состояния прерванной программы
       POP  D
       POP  H
       POP  PSW
       EI          ; Флаг IE ← 1 (прерывания разрешены; триггер INTE ← 1 в МП 8080)
       RET         ; Возврат на адрес прерванной программы
```

Сброс триггера $T_{ISR\ 3}$ ведущего *PIC* необходим потому, что значение $T_{ISR\ 3} = 1$ блокирует дальнейшие запросы прерываний от ведомого *PIC Slave A* и по входам IR_{7-4} ведущего *PIC*, имеющим меньший приоритет, чем вход IR_3. Чтобы после сброса триггера $T_{ISR\ 3}$ ведущего *PIC* его входы IR_{7-4} остались заблокированными, их следует предварительно замаскировать. Наложение маски на входы IR_{7-4} ведущего *PIC* позволяет сохранить при запросах прерываний присвоенные входам приоритеты (для восстановления исходной маски ведущего *PIC* она была сохранена в регистре *E*).

Для упрощения управления прерываниями в *PIC* 8259*A* введен специальный режим полной вложенности, задаваемый значением разряда *SFNM* = 1 в слове команды инициализации ICW_4 (см. рис. 3.53). Этот режим программируется только для ведущего *PIC*. В данном режиме ведущий *PIC* игнорирует только запросы прерывания более низкого приоритета, чем находящиеся в обслуживании, и принимает все запросы *равного* (для ведомых *PIC*) и более *высокого* приоритета. Таким образом, если на ведомый *PIC* поступит запрос прерывания более высокого приоритета, чем находящийся в обслуживании запрос того же ведомого *PIC*, то он будет принят

ведущим *PIC* и вызовет прерывание выполняемой ППОП более низкого приоритета. Это обеспечивает правильную полностью вложенную структуру прерываний для ведомых *PIC* 8259*A*. Сброс триггера $T_{ISR\,i}$ ведущего *PIC*, к входу IR_i которого подключен ведомый *PIC*, следует производить только после завершения всех вложенных ППОП, вызванных этим ведомым *PIC*. Для этого в каждой такой ППОП после подачи команды обычного конца прерывания *EOI* ($OCW_2 = 20h$) необходимо проверять состояние регистра *ISR* ведомого *PIC*. Если в нем нет ни одной 1, то производится сброс триггера $T_{ISR\,i}$ ведущего *PIC* соответствующей командой OCW_2. Для предыдущего примера эти операции выполняются командами, расположенными в конце ППОП:

```
SL_A2: PUSH  PSW     ; ППОП для входа запроса прерывания IR2 Slave A (см. рис. 3.45)
       PUSH  H       ; Запись в стек состояния прерываемой программы
       PUSH  D
       PUSH  B
       EI            ; Флаг IE ← 1 (прерывания разрешены; триггер INTE ← 1 в МП 8080)
         ∴           ; Собственно ППОП
       DI            ; Флаг IE ← 0 (прерывания запрещены; триггер INTE ← 0 в МП 8080)
       MVI   A, 20h  ; 20h = OCW2 — Non Specific EOI (обычный конец прерывания)
       OUT   0FAh    ; PIC Slave A ← OCW2 — конец прерывания
       MVI   A, 0Bh  ; OCW3 = 0Bh — селекция операции чтения ISR (см. рис. 3.60)
       OUT   0FAh    ; PIC Slave A ← OCW3
       IN    0FAh    ; A ← ISR (чтение ISR)
       ANA   A
       JNZ   L1
       MVI   A, 20h  ; A ← OCW2 (R = 0, SL = 0, EOI = 1) — Non-Specific EOI
       OUT   0F8h    ; Сброс триггера T_ISR 3 ведущего PIC
L1:    POP   B       ; Восстановление состояния прерванной программы
       POP   D
       POP   H
       POP   PSW
       EI
       RET           ; Возврат на адрес прерванной программы
```

3.6. Контроллеры прямого доступа к памяти 8257 и 8237*A*

Ввод-вывод по прямому доступу к памяти (ПДП) больших блоков данных является самым быстродействующим из всех методов ввода-вывода и используется, например, для обслуживания НГМД и винчестеров. Данный метод ввода-вывода часто требуется использовать также в МП-системах, работающих в реальном масштабе времени при пересылках даже небольших блоков данных. Для аппаратной реализации ПДП фирма *Intel* разработала БИС программируемых контроллеров ПДП (*DMAC — Programmable Direct Memory Access Controller*) 8257 (580ВТ57) и 8257–5, совместимый с МП 8085*A*, 8237*A* (1810ВТ37), 8237*A*–4, 8237*A*–5 и 82*C*37*A*–5.

Контроллеры ПДП 8257 и 8237*A* изготовляются по *n*-МОП технологии (*NMOS*), а контроллер ПДП 82*C*37*A*–5 (аналог 8237*A*–5) — по *CHMOS* технологии, которая обеспечивает малое потребление мощности. Основные параметры контроллеров ПДП приведены в табл. 3.11.

Таблица 3.11. **Основные параметры контроллеров ПДП**

БИС	Аналог	$I_{CC\ max}$, мА	V_{CC}, В	$V_{OH\ min}/I_{OH}$, В/мА	$V_{OL\ max}/I_{OL}$, В/мА	F_{CLK}, МГц
8257	580ВТ57	120	5 ± 10%	2,4/–0,15	0,45/1,6	2,5
8257–5	—	120	5 ± 10%	2,4/–0,15	0,45/1,6	2,5
8237*A*	1810ВТ37	130	5 ± 5%	2,4/–0,20	0,4/2	3
8237*A*–4	—	130	5 ± 5%	2,4/–0,20	0,4/2	4
8237*A*–5	—	130	5 ± 5%	2,4/–0,20	0,4/2	5
82*C*37*A*–5	—	—	—	—	—	5

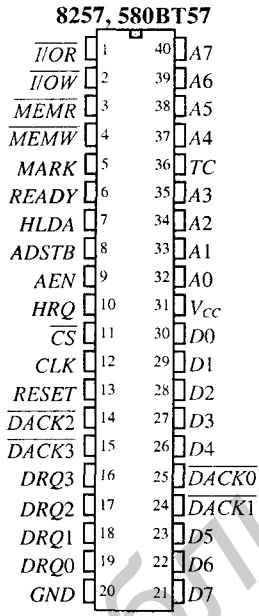


Рис. 3.63. Контроллер ПДП 8257

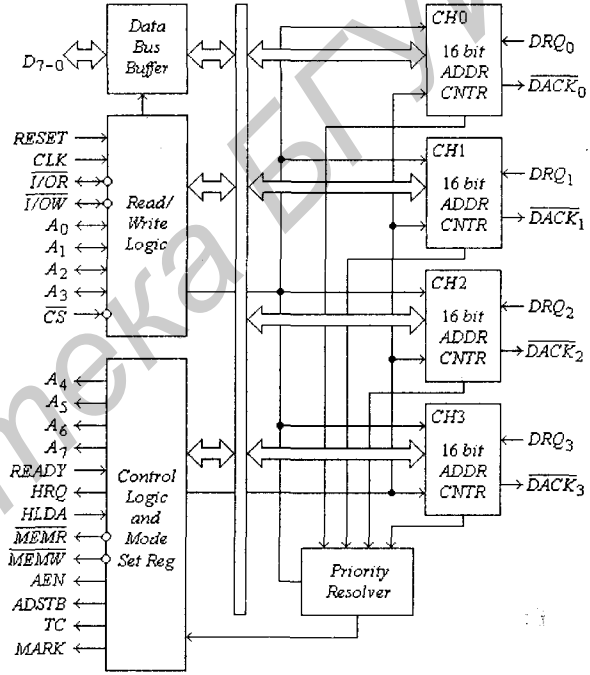


Рис. 3.64. Структурная схема контроллера ПДП 8257

Структурная схема контроллера ПДП 8257. Этот контроллер (рис. 3.63) содержит 4 одинаковых канала управления ПДП, приоритеты которых можно задать постоянными или изменять циклически [11]. Структурная схема *DMAC*, изображенная на рис. 3.64, содержит узлы:

Data Bus Buffer — буфер шины данных (8-разрядный приемопередатчик с *Z*-состоянием выходов);

Read/Write Logic — схема управления чтением и записью;

Control Logic and Mode Set Register — схема управления и регистр установки режима *MSRG*;

CH m, 16 *bit ADDR CNTR* (*Channel m*, 16 *bit Address Counter*; *m* = 0, 1, 2, 3) — каналы ПДП, содержащие два 16-разрядных регистра ARG_m (*Address Register*) и $TCRG_m$ (*Terminal Count Regi-*

ster — счетчик циклов *DMA*), в которые МП при инициализации каждого канала записывает 16-разрядный начальный адрес системной памяти A_{Bm} (базовый адрес) и 14-разрядный размер пересылаемого блока данных ΔA_m (число циклов *DMA*). Два разряда регистра $TCRG_m$ используются для указания типа *DMA*-пересылок (*DMA*-чтение или *DMA*-запись в системную память);

Priority Resolver — приоритетная решающая схема, выбирающая канал ПДП, имеющий наибольший приоритет при одновременном запросе ПДП сигналами DRQ_m от нескольких I/O_m.

Принцип работы *DMAC* 8257. Циклом *DMA* называется пересылка одного байта между системной памятью и внешним устройством. Краткое описание принципа работы МП-системы с вводом-выводом по прямому доступу к памяти было приведено в § 1.1 и 2.5. В такой МП-системе по очереди шинами могут управлять как МП, так и *DMAC*, т. е. *DMAC* может быть ведомым системной шины (шинами управляет МП) и ведущим системной шины (шинами управляет *DMAC*). Для краткости эти режимы работы *DMAC* будем называть *пассивным* и *активным* режимами соответственно. В пассивном режиме *DMAC* является для МП обычным внешним устройством, как и другие интерфейсные БИС. В активном режиме *DMAC* представляет собой специализированный процессор ввода-вывода. На время работы *DMAC* в активном режиме МП отключается от системных шин, переведя свои шины в Z-состояние.

В пассивном режиме *DMAC* производится инициализация одного или нескольких его каналов — запись во внутренние регистры ARG_m и $TCRG_m$ начального (базового) адреса системной памяти A_{Bm} и размера ΔA_m пересылаемого по этому каналу блока данных с указанием типа *DMA*-пересылок (направления передачи данных). Если требуется переслать блок данных размером ΔA_m, то следует загрузить в регистр $TCRG_m$ значение $N_m = \Delta A_m - 1$ (для пересылки одного байта данных следует задать $N_m = 0$).

После инициализации, перейдя в активный режим, *DMAC* может передать блок данных от 1 до 16384 байт (16 Кбайт) между памятью и периферийным устройством непосредственно без дальнейшего вмешательства МП.

Назначение сигналов *DMAC* 8257. Характерной особенностью *DMAC*, отличающей его от обычных интерфейсных БИС, является использование *двунаправленных* адресных сигналов A_{3-0} и системных сигналов управления $\overline{I/OW}$ и $\overline{I/OR}$. Входные и выходные сигналы *DMAC* имеют назначение:

D_{7-0} (*Data*) — сигналы двунаправленной шины данных с *Z*-состоянием выходов. Помимо традиционного назначения шины данных по ней в первом такте каждого *DMA*-цикла *DMAC* выдает старший байт A_{15-8} адреса системной памяти для фиксации его во внешнем регистре сигналом *ADSTB* (см. рис. 3.65);

$\overline{CS}$ (*Chip Select*) — сигнал с дешифратора адреса разрядов A_{7-4}. При *DMA*-пересылках вход $\overline{CS}$ автоматически блокируется, чтобы предотвратить БИС от случайного выбора;

A_{3-0} (*Address*) — двунаправленные линии шины адреса с *Z*-состоянием выходов (входы в пассивном и выходы в активном режимах *DMAC*). В пассивном режиме сигналы A_{3-0} используются для адресации внутренних регистров *DMAC*, а в активном режиме — для выдачи младшей тетрады младшего байта адреса системной памяти;

A_{7-4} (*Address*) — адресные линии, на которые *DMAC* в активном режиме выдает старшую тетраду младшего байта адреса системной памяти (в пассивном режиме эти линии находятся в *Z*-состоянии);

ADSTB (*Address Strobe*) — сигнал строба адреса, используемый для фиксации во внешнем регистре старшего байта A_{15-8} адреса системной памяти, выдаваемого *DMAC* по шине данных D_{7-0} в каждом *DMA*-цикле (см. рис. 3.65);

CLK (*Clock*) — тактовый сигнал (обычно используется сигнал φ_2 от генератора 8224 при работе с МП 8080*A* или сигнал *CLK* от МП 8085*A*);

RESET — сигнал сброса (подается от генератора 8224 при работе с МП 8080*A* или от МП 8085*A*). Значение сигнала *RESET* = 1 отключает все каналы ПДП, сбрасывая в 0 регистр

установки режима *MSRG* (при этом выходы всех системных сигналов управления переводятся в Z-состояние);

$\overline{I/OR}$ (*I/O Read*) — двунаправленная линия с Z-состоянием (вход в пассивном режиме и выход в активном режиме *DMAC*). В пассивном режиме сигнал $\overline{I/OR}$ предназначен для чтения регистра состояния *SRG* и регистров ARG_m и $TCRG_m$. В активном режиме сигнал $\overline{I/OR}$ используется для чтения байта данных из внешнего устройства и передачи его в системную память в каждом *DMA*-цикле записи;

$\overline{I/OW}$ (*I/O Write*) — двунаправленная линия с Z-состоянием (вход в пассивном режиме и выход в активном режиме *DMAC*). В пассивном режиме сигнал $\overline{I/OW}$ предназначен для записи слова управления в регистр установки режима *MSRG* и записи данных в регистры ARG_m и $TCRG_m$. В активном режиме сигнал $\overline{I/OW}$ используется для записи во внешнее устройство байта данных, полученного из системной памяти, в каждом *DMA*-цикле чтения;

$\overline{MEMR}$ и $\overline{MEMW}$ (*Memory Read, Memory Write*) — сигналы управления, выдаваемые *DMAC* в активном режиме для чтения и записи данных в системную память при выполнении *DMA*-циклов (в пассивном режиме эти выходы находятся в Z-состоянии);

READY — сигнал готовности, используемый в активном режиме (имеет такое же назначение, что и одноименный входной сигнал МП — увеличение длительности активных уровней системных сигналов управления $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$ и $\overline{I/OW}$ при выполнении *DMA*-циклов);

HRQ (*Hold Request*) — сигнал запроса системной шины (запрос *DMA*, запрос захвата шин), который подается на вход *HOLD* микропроцессора;

HLDA (*Hold Acknowledge*) — сигнал подтверждения *DMA*, поступающий от МП в ответ на запрос *HRQ* = *HOLD* = 1 и указывающий, что управление системными шинами передано *DMAC*, а МП отключился от системных шин, переведя свои шины данных, адреса и управления в Z-состояние. Сигналы *HRQ* и *HLDA* являются сигналами квитирования ввода-вывода по прямому доступу к памяти для *DMAC* и МП;

AEN (*Address Enable*) — сигнал разрешения адреса (разрешения доступа к системным шинам). Значение *AEN* = 1 указывает на переход *DMAC* в активный режим работы. Сигнал *AEN* используется для перевода в Z-состояние всех буферов центрального процессорного устройства (см. рис. 1.10 и 1.32) и запрета выбора всех внешних устройств, не участвующих в *DMA*-пересылках (см. рис. 1.33), чтобы не было их ложного выбора при работе *DMAC* в активном режиме. Сигнал *AEN* можно использовать и для изоляции *DMAC* от системной шины данных при выдаче старшего байта адреса, чтобы снять любые ограничения по синхронизации передач данных по системной шине данных (в этом случае *DMAC* должен быть подключен к системной шине данных через свой приемопередатчик типа 1533АП6, управляемый сигналом *AEN*);

DRQ_{3-0} (*DMA Request*) — сигналы запросов *DMA* от внешних устройств I/O_m ($m = 0 \ldots 3$). Если не используется циклический сдвиг приоритетов, то вход DRQ_0 имеет наибольший, а вход DRQ_3 наименьший приоритет. Значение $DRQ_m = 1$ должно удерживаться до выполнения последнего *DMA*-цикла;

$\overline{DACK}_{3-0}$ (*DMA Acknowledge*) — сигнал подтверждения *DMA* для внешнего устройства I/O_m. Каждое значение сигнала $\overline{DACK}_m$ = ⊔ сообщает внешнему устройству I/O_m, запросившему прямой доступ к памяти, что оно выбрано для выполнения текущего *DMA*-цикла. Сигнал $\overline{DACK}_m$ используется в качестве сигнала выбора кристалла внешнего устройства I/O_m (или для его формирования — см. ИС 1533ИД7 на рис. 2.23). Пары сигналов DRQ_m и $\overline{DACK}_m$ являются сигналами квитирования ввода-вывода по прямому доступу к памяти для *DMAC* и I/O_m;

TC (*Terminal Count*) — сигнал окончания *DMA*-циклов при передаче блоков данных любого допустимого размера. Значение сигнала *TC* = 1 сообщает обслуженному внешнему устройству, что выполняемый *DMA*-цикл является последним для передаваемого блока данных. Этот сигнал можно использовать для сброса сигналов DRQ_{3-0} запроса *DMA* от внешних устройств;

MARK (*Modulo* 128 *Mark*) — маркерные импульсы, информирующие обслуживаемое внешнее устройство об окончании передачи секторов данных размером 128 байт. Значение сигнала *MARK* = 1 при нулевом состоянии семи младших разрядов регистра $TCRG_m$ — вычитающего счетчика циклов *DMA*. Например, если задан размер блока данных ΔA = 259, то первое значение *MARK* = 1 появится в третьем *DMA*-цикле, а два других — последовательно через 128 *DMA*-циклов.

Переключение *DMAC* из пассивного режима в активный, работа в активном режиме и возврат в пассивный режим описывается последовательностью изменения сигналов:

$$DRQ_m \uparrow \Rightarrow HRQ \uparrow \Rightarrow HLDA \uparrow \Rightarrow AEN \uparrow \Rightarrow \overline{DACK} \sqcap\!\sqcup\!\sqcap \ldots \sqcap\!\sqcup\!\sqcap \Rightarrow$$

от I/O_m — на МП — от МП — на *BZ* — активный режим *DMAC*

$$\Rightarrow HRQ \downarrow \Rightarrow HLDA \downarrow \Rightarrow AEN \downarrow$$

на МП — от МП — на *BZ*

(*BZ* — все устройства МП-системы, переводимые в *Z*-состояние или запрещаемые для адресации значением сигнала *AEN* = 1; это устройства, не участвующие в *DMA*-пересылках). Каждое значение сигнала $\overline{DACK}_m = \sqcap$ соответствует пересылке одного байта данных (*DMA*-цикл). В активном режиме при передаче каждого байта *DMAC* выдает активные уровни пар сигналов $\overline{MEMR}$ и $\overline{I/OW}$ или $\overline{MEMW}$ и $\overline{I/OR}$ в зависимости от направления передачи данных.

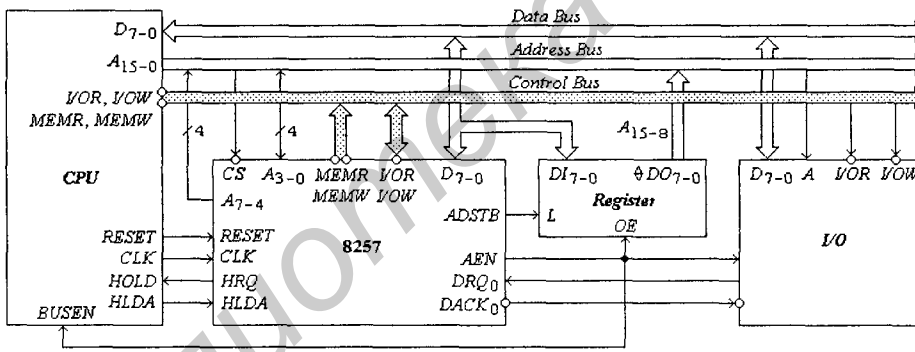


Рис. 3.65. Схема подключения *DMAC* к системным шинам

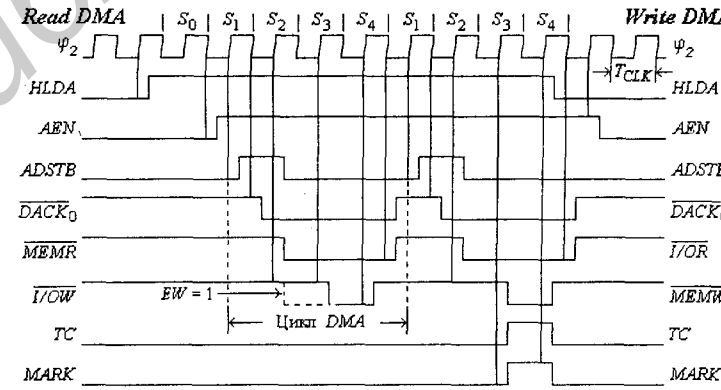


Рис. 3.66. Временны́е диаграммы пересылки двух байт данных

На рис. 3.65 показана структурная схема подключения *DMAC* к системным шинам. Младший байт адреса памяти *DMAC* выдает непосредственно, а старший байт — по шине данных D_{7-0} в первом такте каждого *DMA*-цикла. Старший байт адреса фиксируется во внешнем регистре (*Register*, например, 555ИР22) сигналом *ADSTB* (рис. 3.66). В пассивном режиме работы *DMAC* этот регистр находится в *Z*-состоянии (сигнал $OE = AEN = 0$ — запрет выходов DO_{7-0} регистра памяти). Работа *DMAC* для случая пересылки двух байт данных поясняется временны́ми диаграммами, изображенными на рис. 3.66.

Длительность одного цикла *DMA* при значении сигнала $READY \equiv 1$ равна $4 \cdot T_{CLK}$, скорость пересылки данных — 500 Кбайт/с при частоте тактового сигнала $CLK = \varphi_2$ равной 2 МГц.

Программирование *DMAC* 8257. Операции ввода-вывода, которые может производить МП в пассивном режиме работы *DMAC*, представлены в табл. 3.12. Для адресации двухбайтовых регистров ARG_m и $TCRG_m$ дополнительно используется внутренний триггер *F/L* (*First/Last Flip-Flop*) для указания записи/чтения младшего и старшего байт (рис. 3.67). Триггер *F/L* сбрасывается в 0 значением сигнала $RESET = 1$, а также при каждой загрузке регистра режима *MSRG*. Первым записывается младший байт, и триггер *F/L* из состояния 0 переходит в состояние 1, которое адресует старший байт при неизменных значениях адресных сигналов A_{3-0}. Например, для первой строки $D_{7-0}D_{7-0} = A_{15-8}A_{7-0}$ задает базовый адрес A_{B0}, загружаемый в регистр ARG_0 двумя командами OUT *port* — сначала младший $D_{7-0} = A_{7-0}$, затем старший байт адреса $D_{7-0} = A_{15-8}$. На время загрузки двухбайтовых регистров прерывания следует запретить, если в вызываемых подпрограммах могут инициализироваться другие каналы *DMA* (иначе значение $F/L = \times$ — неопределенное значение).

Таблица 3.12. **Операции ввода-вывода**

$\overline{CS}$	A_3	A_2	A_1	A_0	$\overline{I/OW}$	$\overline{I/OR}$	Операция	Примечание
0	0	0	0	0	0	1	$D_{7-0}D_{7-0} = A_{B0} \rightarrow ARG_0$	Вывод (программирование начального адреса $ARG_m = A_{Bm}$ и размера блока $TCRG_m = \Delta A_m$ для четырех каналов *DMA*; $m = A_2A_1$)
0	0	0	0	1	0	1	$D_{7-0}D_{7-0} = \Delta A_0 \rightarrow TCRG_0$	
0	0	0	1	0	0	1	$D_{7-0}D_{7-0} = A_{B1} \rightarrow ARG_1$	
0	0	0	1	1	0	1	$D_{7-0}D_{7-0} = \Delta A_1 \rightarrow TCRG_1$	
0	0	1	0	0	0	1	$D_{7-0}D_{7-0} = A_{B2} \rightarrow ARG_2$	
0	0	1	0	1	0	1	$D_{7-0}D_{7-0} = \Delta A_2 \rightarrow TCRG_2$	
0	0	1	1	0	0	1	$D_{7-0}D_{7-0} = A_{B3} \rightarrow ARG_3$	
0	0	1	1	1	0	1	$D_{7-0}D_{7-0} = \Delta A_3 \rightarrow TCRG_3$	
0	0	0	0	0	1	0	$D_{7-0}D_{7-0} \leftarrow ARG_0$	Ввод (чтение текущих значений адреса памяти ARG_m и терминального счетчика $TCRG_m$ четырех каналов *DMA*; $m = A_2A_1$)
0	0	0	0	1	1	0	$D_{7-0}D_{7-0} \leftarrow TCRG_0$	
0	0	0	1	0	1	0	$D_{7-0}D_{7-0} \leftarrow ARG_1$	
0	0	0	1	1	1	0	$D_{7-0}D_{7-0} \leftarrow TCRG_1$	
0	0	1	0	0	1	0	$D_{7-0}D_{7-0} \leftarrow ARG_2$	
0	0	1	0	1	1	0	$D_{7-0}D_{7-0} \leftarrow TCRG_2$	
0	0	1	1	0	1	0	$D_{7-0}D_{7-0} \leftarrow ARG_3$	
0	0	1	1	1	1	0	$D_{7-0}D_{7-0} \leftarrow TCRG_3$	
0	1	0	0	0	0	1	$D_{7-0} \rightarrow MSRG$	Установка режима
0	1	0	0	0	1	0	$D_{7-0} \leftarrow SRG$	Чтение состояния
0	×	×	×	×	1	1	Нет операции	D_{7-0} в *Z*-состоянии
1	×	×	×	×	×	×	Нет операции	D_{7-0} в *Z*-состоянии

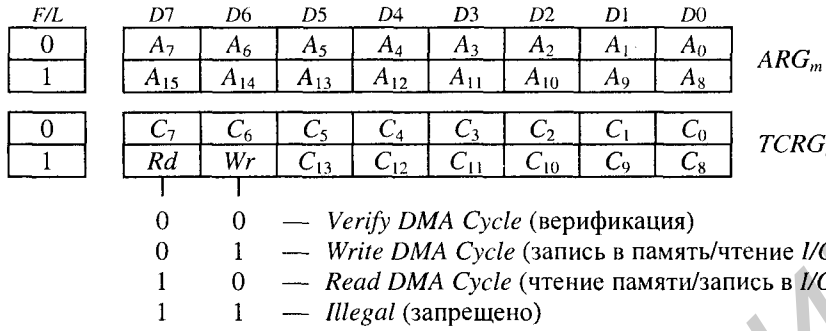


Рис. 3.67. Форматы регистров ARG_m и $TCRG_m$

Разряды A_{15-0} регистра ARG_m задают начальный (базовый) адрес A_{Bm} системной памяти, а разряды C_{13-0} регистра $TCRG_m$ — размер ΔA_m пересылаемого блока данных:

$$\Delta A_m = N_m + 1 = C_{13} \dots C_1C_0 + 1 = 1 \text{ байт} \dots 16 \text{ Кбайт}$$

(этим значениям соответствуют 14-разрядные двоичные числа $N_m = C_{13} \dots C_1C_0$ от 0 до $2^{14} - 1$). После каждого цикла *DMA* производится инкремент содержимого регистра ARG_m и проверка числа $C_{13} \dots C_1C_0$ на нуль. Если число $C_{13} \dots C_1C_0 \neq 0$, то производится его декремент и выполняется следующий цикл *DMA*, а если $C_{13} \dots C_1C_0 = 0$, то сеанс *DMA* заканчивается.

Разряды *Rd* и *Wr* регистра $TCRG_m$ (рис. 3.67) задают тип циклов *DMA*. При выполнении верификации (*Rd Wr* = 0 0) активные уровни сигналов $\overline{I/OW}$, $\overline{MEMR}$, $\overline{MEMW}$ и $\overline{I/OR}$ не вырабатываются, т. е. пересылок данных не производится, а изменения всех остальных сигналов происходят так же, как и при пересылках данных.

Регистр установки режима *MSRG*. Значение сигнала *RESET* = 1 сбрасывает регистр *MSRG* в 0, отключая все каналы ПДП для предотвращения конфликтов шин при включении питания. Формат команды установки режима (слова управления *CW — Control Word*) показан на рис. 3.68:

$D_m = CH_m$ (*Enables DMA Channel*) — разрешение работы канала *m* ПДП (*m* = 0, 1, 2, 3). Только при задании хотя бы одного значения $CH_m = 1$ будет вырабатываться запрос ПДП *HRQ* = 1, подаваемый на МП. Сигнал *RESET* = 1 устанавливает $CH_m = 0$ для всех каналов;

$D_4 = RP$ (*Rotation Priority*) — установка режима циклического приоритета. Значение разряда *RP* = 0 задает режим фиксированных приоритетов (канал 0 имеет наибольший приоритет, а канал 3 — наименьший). При значении *RP* = 1 обслуженному каналу ПДП присваивается наименьший приоритет, что предотвращает захват ПДП одним каналом. Приоритеты задаются приоритетным кольцом, подобно тому, как это сделано в контроллере прерываний 8259*A* (см. рис. 3.56);

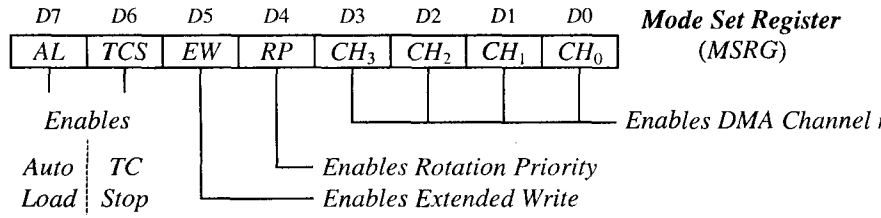


Рис. 3.68. Формат команды установки режима (*CW*)

D_5 = *EW* (*Extended Write*) — задание расширенной длительности активных уровней (0) системных сигналов управления $\overline{MEMW}$ и $\overline{I/OW}$ в активном режиме работы *DMAC* (длительность этих сигналов при значении *EW* = 1 показана на рис. 3.66 штриховой линией);

D_6 = *TCS* (*Terminal Count Stop*) — разрешение автоматического сброса в 0 разряда CH_m по окончании пересылки всего блока данных по каналу *m* при *TCS* = 1, что предотвращает продолжение дальнейших *DMA*-операций по этому каналу, если внешнее устройство I/O_m не сняло запрос DRQ_m = 1. Если разряд *TCS* = 0, то прекращение *DMA*-операций возлагается на внешнее устройство I/O_m, информируемое о конце пересылок значением сигнала *TC* = 1;

D_7 = *AL* (*Auto Load*) — разрешение автоматической загрузки значений ARG_3 и $TCRG_3$ канала 3 в соответствующие регистры канала 2 после его обслуживания. Если разряд *AL* = 1, то при загрузке параметров A_{B2} и ΔA_2 в регистры ARG_2 и $TCRG_2$ они автоматически дублируются в регистрах ARG_3 и $TCRG_3$ канала 3. Затем эти параметры используются для автоматического восстановления исходных значений ARG_2 и $TCRG_2$ (цикл обновления) после передачи блока данных без непосредственного программного вмешательства между блоками. Такой режим работы *DMAC* используется, например, при регенерации изображения на дисплее (блок данных в памяти содержит экранную информацию). Разряд CH_2 не сбрасывается автоматически в 0 после обслуживания канала 2 при значении разрядов *AL* = 1 и *TCS* = 1.

До установки значения разряда CH_m = 1 следует произвести загрузку регистров ARG_m и $TCRG_m$ канала *m* значениями базового адреса A_{Bm} системной памяти и размером пересылаемого блока данных ΔA_m для предотвращения разрушения хранящихся в памяти данных при случайном запросе ПДП внешним устройством I/O_m.

Инициатором ПДП может быть как *I/O*, так и МП. Если значение сигнала $DRQ_m \equiv 1$, то ПДП вызывает МП подачей команды установки режима, в которой разряд CH_m = 1. Перед подачей этой команды должен быть запрограммирован начальный адрес $ARG_m = A_{Bm}$ и размер блока $TCRG_m = \Delta A_m$.

Регистр состояния *DMAC* 8257. Формат слова состояния *SW* показан на рис. 3.69. Разряды TC_m устанавливаются в 1 при обнулении терминального счетчика (TC_m = 1 означает, что канал *m* обслужен). Разряд *UP* (флаг обновления) устанавливается в 1 при автозагрузке (при значении разряда *AL* = 1). При чтении слова состояния *SW* все разряды TC_m сбрасываются в 0, при этом флаг *UP* не сбрасывается. Сброс этого флага производится при подаче сигнала системного сброса или при сбросе режима автозагрузки (разряд *AL* в команде установки режима сбрасывается в 0), а также при окончании цикла обновления.

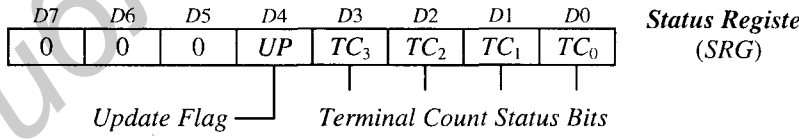


Рис. 3.69. Формат слова состояния *DMAC*

Управление электронным диском. Объем системной памяти МП-систем, построенных на основе МП 8080*A*/8085*A*, равен всего 64 Кбайта. Это иногда вызывает необходимость использовать внешнюю память для хранения больших объемов оперативных данных или загружаемых в системную память прикладных программ. Внешнюю память можно реализовать как на магнитных дисках (требуется некоторая дисковая операционная система), так и на БИС постоянной и оперативной памяти (на электронном диске). Принципиальная схема контроллера прямого доступа к памяти, управляющего обменом данными между системной памятью и электронным диском была приведена на рис. 2.23.

Пересылка блоков размером до 256 байт из электронного диска в системную память (*DMA*-запись) инициируется программой:

```
  ∴          ; Инициализация счетчика адресов электронного диска (см. рис. 2.23)
MVI  A, Aled  ; A ← Aled — младший байт адреса электронного диска
OUT  90h      ; 1533ИЕ7 ← Aled
MVI  A, Ahed  ; A ← Ahed — старший байт адреса электронного диска
OUT  0A0h     ; 1533ИЕ7 ← Ahed
MVI  A, Alm   ; A ← Alm — младший байт базового адреса системной памяти A_B0
OUT  80h      ; ARG_0 ← Alm
MVI  A, Ahm   ; A ← Ahm — старший байт базового адреса системной памяти A_B0
OUT  80h      ; ARG_0 ← Ahm
MVI  A, Vm    ; A ← Vm — младший байт размера пересылаемого блока ΔA_0
OUT  81h      ; TCRG_0 ← Vm
MVI  A, 40h   ; A ← RdWrC13C12C11C10C9C8 = 0100 0000 (RdWr = 01 — DMA-запись)
OUT  81h      ; TCRG_0 ← RdWrC13C12C11C10C9C8
MVI  A, 41h   ; A ← CW = 0100 0001 (D6 = TCS = 1, D0 = CH0 = 1)
OUT  88h      ; MSRG ← CW — передача управления шинами контроллеру DMA
```

Пересылка блоков размером до 256 байт из системной памяти в электронный диск (*DMA*-чтение) инициируется такой же программой при задании значения $TCRG_0 = 80h$ вместо $40h$ в старшем байте регистра $TCRG_0$. Для пересылок блоков большего размера следует задать значения старших шести разрядов размера блока $C_{13}C_{12}C_{11}C_{10}C_9C_8 \neq 000000$.

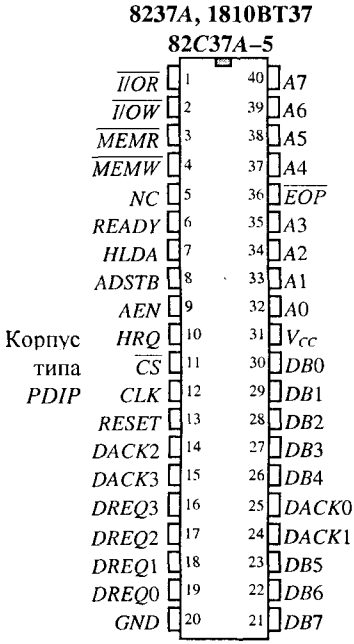
Структурная схема контроллера ПДП 8237A. Этот контроллер (рис. 3.70) обладает значительно большими возможностями, чем рассмотренный выше. В частности, в контроллере ПДП 8237*A* предусмотрены независимая автоинициализация всех четырех приоритетных каналов ПДП, передачи типа "память-память", инкремент и декремент адреса, каскадирование для увеличения числа каналов ПДП и независимое управление полярностью сигналов $DREQ_m$ и $DACK_m$ (значением активного уровня сигналов). В контроллере ПДП 8237*A* введены три базовых режима передачи данных и значительно расширены возможности программного управления, что позволяет увеличить производительность МП-системы при передаче данных и производить системную оптимизацию [12, 13]. Кроме того, допускается динамическая реконфигурация под управлением программного обеспечения. Каждый канал может управлять пересылкой блоков данных размером от одного байта до 64 Кбайт.

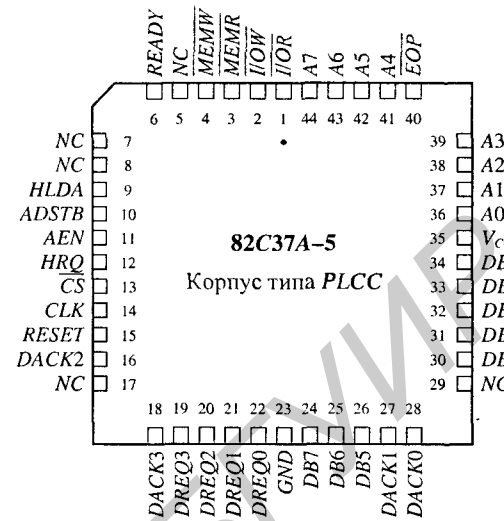
Принцип работы *DMAC* 8237*A* такой же, что и *DMAC* 8257 (см. выше). Передачи типа "*Memory-I/O*" и "*I/O-Memory*" выполняются за четыре такта сигнала синхронизации *CLK*, а передачи типа "*Memory-Memory*" — за восемь тактов (первые четыре такта на чтение памяти с сохранением байта данных во внутреннем регистре и следующие четыре такта на запись байта данных в память по новому адресу).

На структурной схеме *DMAC* 8237*A*, изображенной на рис. 3.71, многие управляющие сигналы между блоками не показаны. Контроллер содержит 27 регистров памяти (всего 344 триггера). Основные узлы контроллера имеют назначение:

Timing and Control — устройство синхронизации и управления, генерирующее внутренние и внешние сигналы управления;

Priority Encoder and Rotating Priority Logic — приоритетная решающая схема с циклическим изменением приоритетов, выбирающая один из четырех каналов ПДП, имеющий наибольший приоритет (при одновременном запросе ПДП сигналами $DREQ_m$ от нескольких I/O_m, где $m = 0, 1, 2, 3$);





Контакт 5 — рекомендуется подключать к V_{CC}, хотя и есть внутренний нагрузочный резистор, подключенный к питанию

Рис. 3.70. Контроллеры ПДП 8237*A*

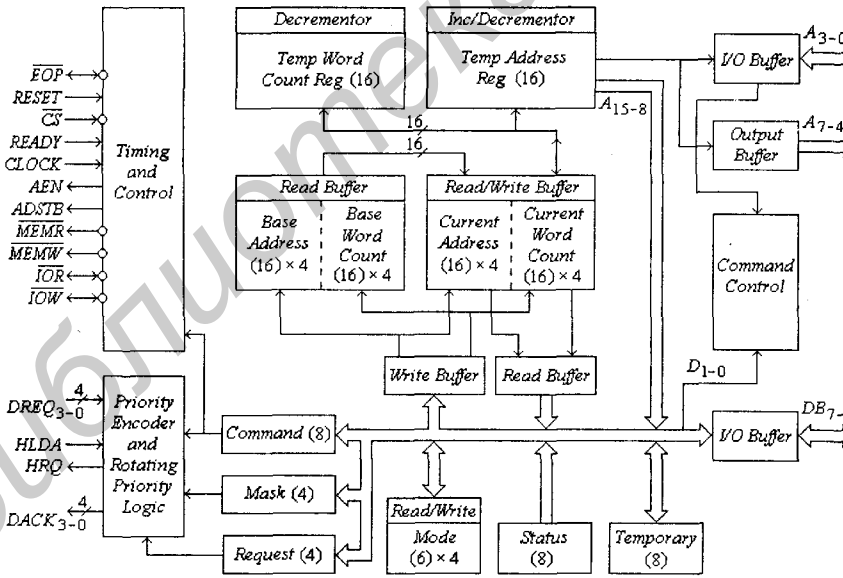


Рис. 3.71. Структурная схема контроллера ПДП 8237*A*

Base Address — четыре 16-разрядных регистра хранения базовых адресов A_{Bm}, в которые МП при инициализации каждого канала записывает 16-разрядный начальный адрес системной памяти;

Base Word Count — четыре 16-разрядных регистра хранения размеров ΔA_m пересылаемых блоков данных (число циклов *DMA*);

Inc/Decrementor, Temp Address Reg (инкрементор/декрементор и регистр временного хранения текущего значения адреса системной памяти) — устройство, вычисляющее значение $A_{Bm} + 1$ или $A_{Bm} - 1$ в каждом *DMA*-цикле;

Decrementor, Temp Word Count Reg (декрементор и регистр временного хранения текущего значения числа оставшихся *DMA*-циклов) — устройство, в каждом *DMA*-цикле, вычисляющее значение $\Delta A_m - 1$;

Current Address — четыре 16-разрядных регистра хранения текущих значений адресов A_m системной памяти;

Current Word Count — четыре 16-разрядных регистра хранения текущих значений числа оставшихся *DMA*-циклов;

Command — 8-разрядный регистр хранения команд;

Mask — 4-разрядный регистр хранения маски запросов *DMA* от внешних устройств;

Request — 4-разрядный регистр хранения запросов *DMA* по требованию МП;

Mode — четыре 6-разрядных регистра хранения режимов работы каналов *DMA*;

Command Control — устройство управления, декодирующее различные команды, полученные от МП до обслуживания запроса *DMA*, а также декодирующее слово управления режимами, задающее тип передач *DMA* при обслуживании;

Status — 8-разрядный регистр слова состояния;

Temporary — 8-разрядный регистр временного хранения байта данных при передачах типа "память-память".

Назначение сигналов *DMAC* 8237А. Обозначения и назначение большинства сигналов *DMAC* 8257 и 8237*A* совпадают, поэтому нет смысла описывать их заново (сигналы $DREQ_{3-0}$ соответствуют сигналам DRQ_{3-0} — *DMA Request*). Отличительные особенности некоторых сигналов *DMAC* 8257:

$DREQ_{3-0}$ (*DMA Request*) — сигналы запросов *DMA* от внешних устройств I/O_m ($m = 0 \ldots 3$). Если не используется циклический сдвиг приоритетов, то вход DRQ_0 имеет наибольший, а вход DRQ_3 наименьший приоритет. Полярность (значение активного уровня) сигналов $DREQ_m$ программируется пользователем — можно задать прямые $DREQ_m$ или инверсные $\overline{DREQ_m}$ сигналы (активные уровни 1 и 0 соответственно). Значение сигнала сброса *RESET* = 1 задает прямые сигналы $DREQ_m$. Значение $DREQ_m = 1$ должно удерживаться до тех пор, пока соответствующий сигнал подтверждения $DACK_m$ не перейдет в активное состояние;

$DACK_{3-0}$ (*DMA Acknowledge*) — сигнал подтверждения *DMA* для внешнего устройства I/O_m ($m = 0 \ldots 3$). Полярность сигналов $DACK_m$ программируется пользователем. Значение сигнала сброса *RESET* = 1 задает инверсные сигналы $\overline{DACK_m}$. Пары сигналов $DREQ_m$ и $DACK_m$ (или $\overline{DACK_m}$) являются сигналами квитирования ввода-вывода по прямому доступу к памяти для *DMAC* и I/O_m;

$\overline{EOP}$ (*End of Process*) — двунаправленная линия, по которой передается сигнал, активный уровень которого (0) указывает окончание процесса передачи данных (завершение обслуживания по прямому доступу к памяти). Контакт $\overline{EOP}$ (вход) можно использовать для подачи внешнего сигнала $\overline{EOP} = 0$, требующего окончания процесса. Если контакт $\overline{EOP}$ не используется для подачи внешнего сигнала, то его следует подключить к источнику напряжения питания +5 В через нагрузочный резистор, чтобы предотвратить ошибочное окончание процесса помехами. Значение выходного сигнала $\overline{EOP} = 0$ генерируется при достижении конца счета *DMA*-циклов, когда текущее значение ΔA_m станет равным 0. Длительность низкого уровня сигнала $\overline{EOP}$ равна одному такту сигнала *CLK* в момент пересылки последнего байта данных, т. е. в этом случае сигнал $\overline{EOP}$ выполняет функцию сигнала *TC* контроллера 8257 (см. рис. 3.66). Значение сигнала $\overline{EOP} = 0$ (внутреннее или внешнее) заставляет контроллер прекратить обслуживание внешнего устройства, сбрасывает запрос и если разрешена автоинициализация, содержимое базовых (*Base*) регистров (A_{Bm} и ΔA_m) перезаписывается в текущие (*Current*) регистры обслуженного

канала. Если же автоинициализация не разрешена, то устанавливается в 1 разряд обслуженного канала в регистре маски и соответствующий разряд TC_m в регистре слова состояния;

RESET — сигнал сброса *DMAC* в исходное состояние. Значение сигнала *RESET* = 1 сбрасывает в 0 триггеры регистра команды (*Command*), регистра состояния (*Status*), регистра запросов *DMA* (*Request*) и регистра временного хранения (*Temporary*), а также сбрасывает в 0 триггер *First/Last Flip-Flop* (триггер *F/L* в *DMAC* 8257 — см. рис. 3.67) и устанавливает в 1 триггеры регистра маски запросов *DMA* (*Mask*).

Подключается контроллер прямого доступа к памяти 8237*A* к системным шинам точно так же, как и контроллер 8257 (см. рис. 3.65). На рис. 3.72 показано каскадирование контроллеров 8237*A*:

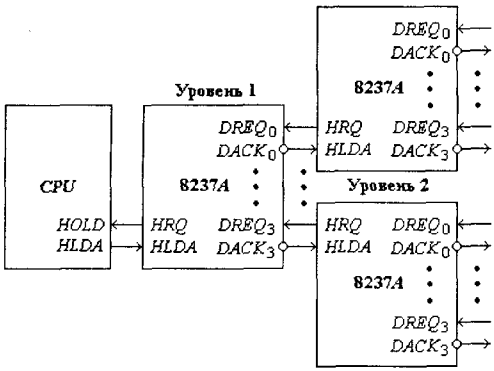


Рис. 3.72. Каскадирование БИС 8237*A*

DMAC уровня 1 командой задания режимов работы устанавливается в режим каскадирования;

запросы *DMA* контроллеров уровня 2 проходят через приоритетную решающую схему *DMAC* уровня 1, что обеспечивает назначение приоритетов всем 16 каналам *DMA* уровня 2 в соответствии с назначенными приоритетами *DMAC* уровня 1;

в активном режиме работы *DMAC* уровня 2 сигналы *DMAC* уровня 1, связанные с системными шинами адреса, данных и управления, блокируются, так как его назначением является только обслуживание *DMAC* уровня 2;

вход готовности *DMAC* уровня 1 игнорируется;

при необходимости можно добавить третий уровень *DMAC*.

Программирование *DMAC* 8237*A*. Операции ввода-вывода, которые может производить МП в пассивном режиме работы *DMAC*, представлены в табл. 3.13. Для адресации двухбайтовых регистров *Base Address* и *Base Word Count* дополнительно используется внутренний триггер *First/Last Flip-Flop* для указания записи/чтения младшего и старшего байт. Первым записывается младший байт, и триггер *First/Last Flip-Flop* из состояния 0 переходит в состояние 1, которое адресует старший байт при неизменных значениях адресных сигналов $A_{3\text{-}0}$. Данный триггер сбрасывается в 0 значением сигнала *RESET* = 1, а также командами *Master Clear* и *Clear First/Last Flip-Flop*.

Отличие табл 3.13 от табл 3.12 вызвано в основном расширением возможностей управления контроллером 8237*A* с помощью программного обеспечения — добавлено семь команд управления и одна команда чтения. Команды *Clear First/Last Flip-Flop* (*port* = ×Ch), *Master Clear* (*port* = ×Dh) и *Clear Mask Register* (*port* = ×Eh) для выполнения операций не используют значения разрядов шины данных ($D_{7\text{-}0}$ = ×).

Слово управления регистра команд. Данное слово управления (рис. 3.73) задает параметры всех четырех каналов *DMA* (*port* = ×8*h*). Все разряды регистра команд устанавливаются в 0 значением сигнала *RESET* = 1 и командой *Master Clear*. Некоторые разряды слова управления требуют дополнительных пояснений:

D_0 — значение D_0 = 1 задает использование для передач данных двух каналов *DMA*: канала 0 для чтения памяти с сохранением байта данных в регистре *Temporary* и канала 1 для записи байта данных из регистра *Temporary* по новому адресу. При этом пересылки типа "память-память" инициируются микропроцессором подачей слова управления в регистр запросов *DMA* при значении разрядов $D_2D_1D_0$ = 100 (см. рис. 3.75) — программный запрос в отличие от аппаратного запроса сигналами $DREQ_m$, поступающими от внешних устройств. Пересылки типа

"память-память" в режиме блоковой передачи (*Block mode transfer* — см. рис. 3.74), выполняемые с помощью *DMAC*, позволяют сократить программное обеспечение и увеличить производительность МП-системы. Пересылки типа "память-память" можно использовать для быстрого поиска в памяти заданного кода байта с помощью аппаратного цифрового компаратора [5], подающего значение сигнала $\overline{EOP}$ = 0 на вход контроллера $\overline{EOP}$ для завершения поиска (завершения передач данных);

D_1 — значение D_1 = 1 отключает операцию инкремента/декремента текущего адреса системной памяти канала 0, что позволяет заполнять константами блоки памяти, адресуемые каналом 1;

D_3 — значение D_3 = 1 задает исключение из *DMA*-циклов такта S_1 (фиксация старшего байта адреса памяти во внешнем регистре) и такта S_3, задающего нормальную длительность активных уровней сигналов $\overline{MEMR}$ и $\overline{I/OR}$ (см. рис. 3.66). Такт S_1 вводится только при изменении старшего байта адреса памяти A_{15-8}, т. е. один раз за 256 *DMA*-циклов. Возможность использования режима *Compressed timing* зависит от быстродействия системы.

Таблица 3.13. **Операции ввода-вывода**

$\overline{CS}$	A_3	A_2	A_1	A_0	$\overline{I/OW}$	$\overline{I/OR}$	Операция
0	0	0	0	0	0	1	$D_{7-0}D_{7-0} = A_{B0} \rightarrow$ *Base and Current Address Channel* 0
0	0	0	0	1	0	1	$D_{7-0}D_{7-0} = \Delta A_0 \rightarrow$ *Base and Current Word Count Channel* 0
0	0	0	1	0	0	1	$D_{7-0}D_{7-0} = A_{B1} \rightarrow$ *Base and Current Address Channel* 1
0	0	0	1	1	0	1	$D_{7-0}D_{7-0} = \Delta A_1 \rightarrow$ *Base and Current Word Count Channel* 1
0	0	1	0	0	0	1	$D_{7-0}D_{7-0} = A_{B2} \rightarrow$ *Base and Current Address Channel* 2
0	0	1	0	1	0	1	$D_{7-0}D_{7-0} = \Delta A_2 \rightarrow$ *Base and Current Word Count Channel* 2
0	0	1	1	0	0	1	$D_{7-0}D_{7-0} = A_{B3} \rightarrow$ *Base and Current Address Channel* 3
0	0	1	1	1	0	1	$D_{7-0}D_{7-0} = \Delta A_3 \rightarrow$ *Base and Current Word Count Channel* 3
0	0	0	0	0	1	0	$D_{7-0}D_{7-0} \leftarrow$ *Current Address Channel* 0
0	0	0	0	1	1	0	$D_{7-0}D_{7-0} \leftarrow$ *Current Word Count Channel* 0
0	0	0	1	0	1	0	$D_{7-0}D_{7-0} \leftarrow$ *Current Address Channel* 1
0	0	0	1	1	1	0	$D_{7-0}D_{7-0} \leftarrow$ *Current Word Count Channel* 1
0	0	1	0	0	1	0	$D_{7-0}D_{7-0} \leftarrow$ *Current Address Channel* 2
0	0	1	0	1	1	0	$D_{7-0}D_{7-0} \leftarrow$ *Current Word Count Channel* 2
0	0	1	1	0	1	0	$D_{7-0}D_{7-0} \leftarrow$ *Current Address Channel* 3
0	0	1	1	1	1	0	$D_{7-0}D_{7-0} \leftarrow$ *Current Word Count Channel* 3
0	1	0	0	0	0	1	$D_{7-0} \rightarrow$ *Command Register* (управление параметрами *DMAC*)
0	1	0	0	1	0	1	$\times D_{2-0} \rightarrow$ *Request Register* (запрос *DMA* по требованию МП)
0	1	0	1	0	0	1	$\times D_{2-0} \rightarrow$ *Single Mask Register Bit* (поразрядное управление маской)
0	1	0	1	1	0	1	$D_{7-0} \rightarrow$ *Mode Register* (управление режимами работы *DMAC*)
0	1	1	0	0	0	1	$\times \Rightarrow$ *Clear First/Last Flip-Flop* (команда очистки триггера *F/L*)
0	1	1	0	1	0	1	$\times \Rightarrow$ *Master Clear* (команда программного сброса)
0	1	1	1	0	0	1	$\times \Rightarrow$ *Clear Mask Register* (команда очистки маски)
0	1	1	1	1	0	1	$\times D_{3-0} \rightarrow$ *All Mask Register Bits* (управление разрядами маски)
0	1	0	0	0	1	0	$D_{7-0} \leftarrow$ *Read Status Register* (чтение слова состояния)
0	1	1	0	1	1	0	$D_{7-0} \leftarrow$ *Read Temporary Register* (чтение байта данных)
0	×	×	×	×	1	1	Нет операции (D_{7-0} в *Z*-состоянии)
1	×	×	×	×	×	×	Нет операции (D_{7-0} в *Z*-состоянии)

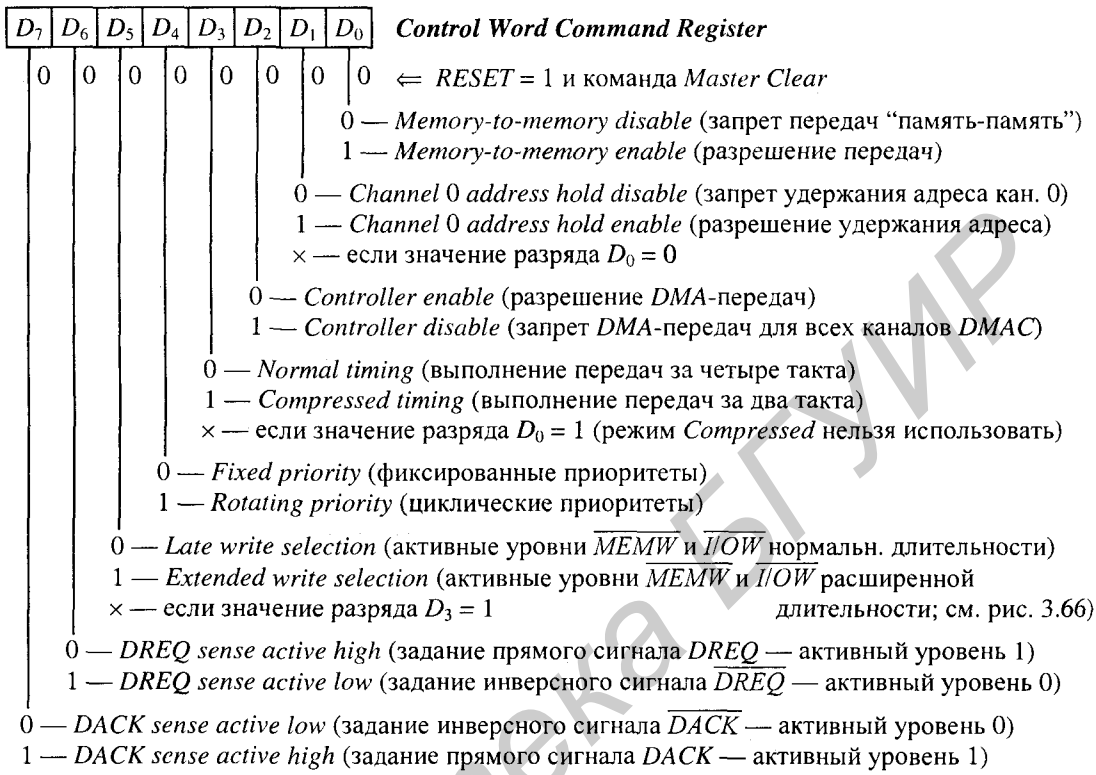


Рис. 3.73. Формат слова управления регистра команд

Слово управления регистра режимов. Данное слово управления (рис. 3.74) задает индивидуальные режимы работы для каждого канала *DMA* (*port* = ×B*h*) — шесть разрядов слова управления $D_{7\text{-}2}$ записываются в выбранный разрядами $D_1D_0 = m$ регистр режима канала *m*. Назначение разрядов слова управления:

$D_1D_0 = m$ — выбор регистра режима канала *m*;

D_3D_2 — выбор режима работы канала *m*. Эти разряды по назначению эквивалентны разрядам *RdWr* для контроллера 8257 (см. рис. 3.67);

D_4 — запрет ($D_4 = 0$) и разрешение ($D_4 = 1$) автоинициализации канала *m*. Если значение разряда $D_4 = 1$, то после передачи текущего блока данных содержимое A_{Bm} регистра хранения базового адреса *Base Address* и содержимое ΔA_m регистра хранения размеров пересылаемых блоков *Base Word Count* автоматически переписываются в регистры хранения текущих значений этих величин *Current Address* и *Current Word Count* (см. рис. 3.71);

D_5 — выбор направления передачи данных памяти для канала *m*;

D_7D_6 — выбор режима передачи данных для канала *m*.

В *режиме одиночной передачи* ($D_7D_6 = 01$) после каждого *DMA*-цикла контроллер освобождает шины микропроцессору минимум на один цикл шины данных (снимается запрос захвата шин *HRQ* = 1), но сразу же начинает проверку входов $DREQ_m$, и как только обнаруживает активный уровень, инициирует следующий *DMA*-цикл (подается новый запрос захвата шин *HRQ* = 1). Так продолжается до тех пор, пока не будут пересланы все ΔA_m байт данных и не будет получено значение $\overline{EOP} = 0$. Значение активного уровня сигнала $DREQ_m$ должно поддерживаться до получения активного уровня сигнала $DACK_m$.

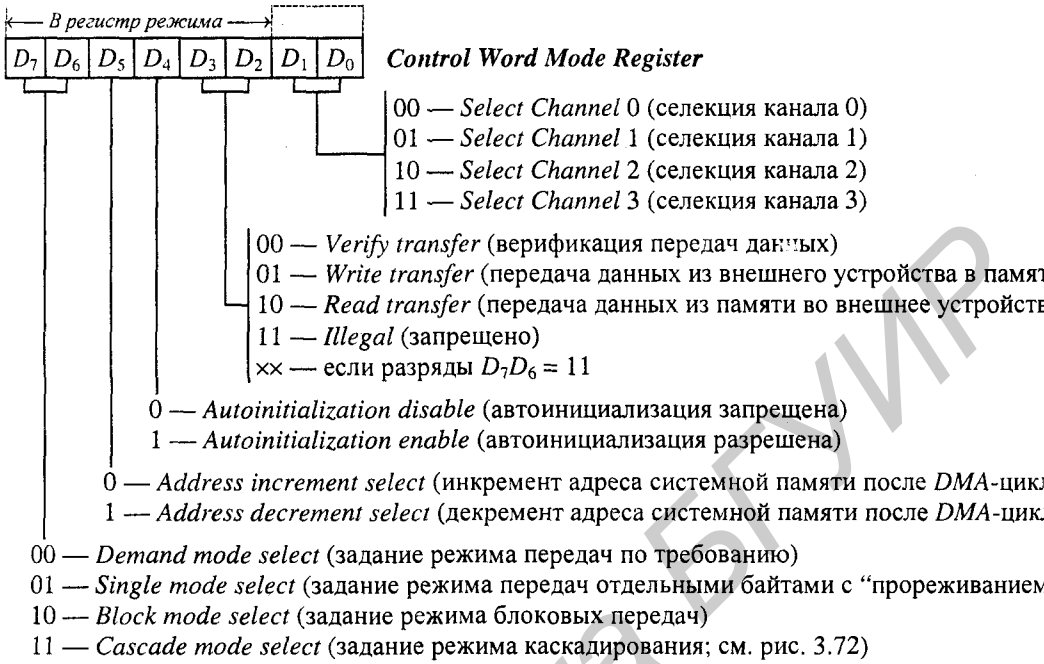


Рис. 3.74. Формат слова управления регистра режимов

В *режиме блоковой передачи* (D_7D_6 = 10) активный уровень сигнала $DREQ_m$ должен сохраняться только до получения активного уровня сигнала $DACK_m$, после чего шины не освобождаются до завершения передачи всего блока. Передача может быть прервана внешним сигналом $\overline{EOP} = 0$.

Режим передачи по требованию (D_7D_6 = 00) аналогичен режиму блоковой передачи, но после каждого *DMA*-цикла проверяется значение сигнала $DREQ_m$ — если значение сигнала $DREQ_m$ неактивное, то передачи приостанавливаются до тех пор, пока не будет получено активное значение сигнала $DREQ_m$. После этого передача продолжается с того адреса памяти, на котором она была приостановлена. Данный режим позволяет внешнему устройству остановить передачу, если оно не может продолжить ее.

В любом режиме при достижении значения $\Delta A_m = 0$ выдается значение сигнала $\overline{EOP} = 0$, передача данных прекращается и выполняется автоинициализация, если разряд D_4 = 1 в регистре режима.

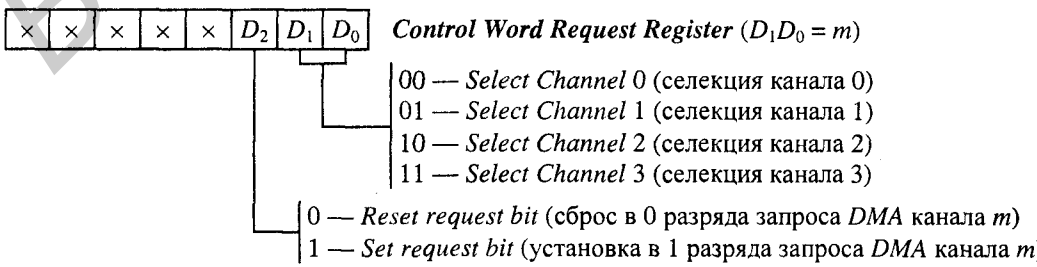


Рис. 3.75. Формат слова управления регистра запросов и регистра маски

Слово управления регистра запросов. Операция, выполняемая при подаче этого слова управления (рис. 3.75), аналогична операции поразрядной установки/сброса в БИС 8255*A* (см. рис. 3.3) — производится запись значения 0 или 1 только в один из разрядов 4-разрядного регистра программных запросов *DMA* (*port* = ×9*h*).

Установка в 1 разряда запроса *DMA* какого-либо канала *m* эквивалентна подачи активного уровня сигнала $DREQ_m$ от внешнего устройства, т. е. прямым доступом к памяти можно управлять программным способом. Этот разряд автоматически сбрасывается в 0 при завершении передачи значением сигнала $\overline{EOP} = 0$.

Слово управления регистра маски. Операция, выполняемая при подаче этого слова управления (рис. 3.76), аналогична операции поразрядной установки/сброса в БИС 8255*A* (см. рис. 3.3) — производится запись значения 0 или 1 только в один из разрядов 4-разрядного регистра маски *DMA* (*port* = ×A*h*).

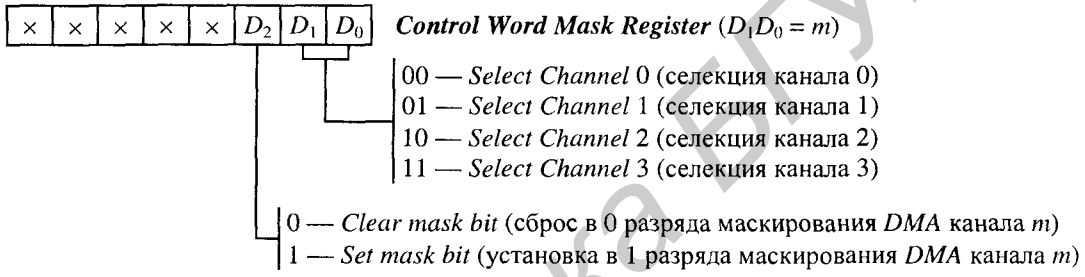


Рис. 3.76. Формат слов управления регистра запросов и регистра маски

Установка в 1 разряда маски *DMA* какого-либо канала *m* блокирует аппаратные запросы *DMA* этого канала $DREQ_m$ (программные запросы не блокируются). Разряд маски обслуживаемого канала *m* автоматически устанавливается в 1 при завершении передачи значением сигнала $\overline{EOP} = 0$, если канал *m* не запрограммирован на автоинициализацию.

Слово управления установки маски. Это слово управления (рис. 3.77) используется для установки маски *DMA* сразу для всех четырех каналов *m* (*port* = ×F*h*). Установку в 0 всех разрядов маски можно выполнить командой *Clear Mask Register* (см. табл. 3.13).

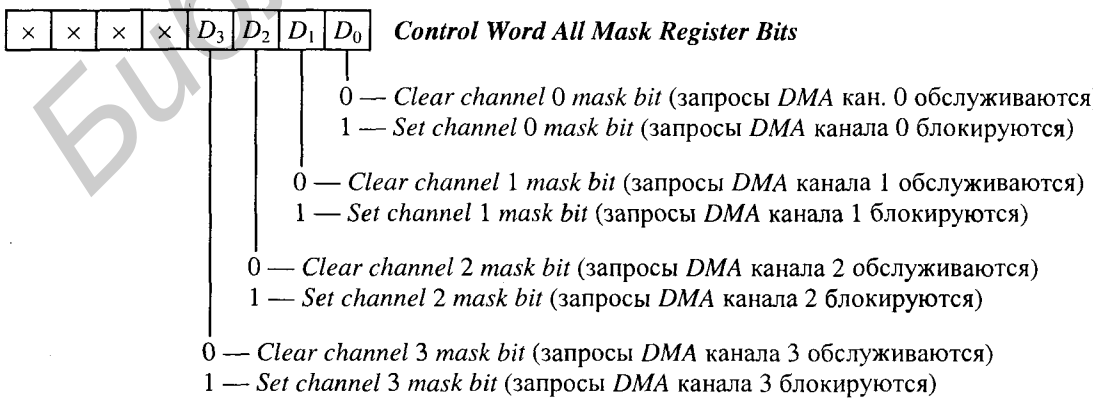


Рис. 3.77. Формат слова управления установки маски

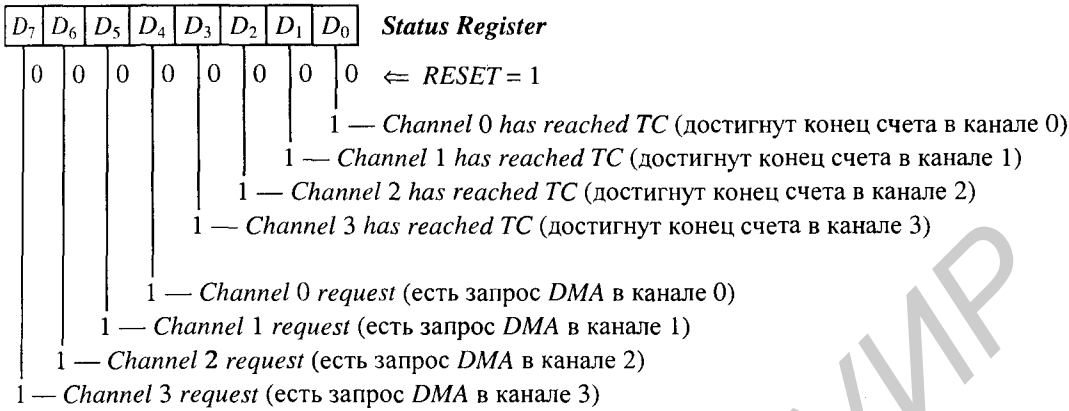


Рис. 3.78. Формат слова состояния *SW*

Регистр состояния. Содержимое этого регистра (рис. 3.78) доступно для чтения микропроцессором командой IN *port* (*port* = ×8*h*). Значение разряда $D_m = 1$ ($m = 0, 1, 2, 3$) указывает, что обслуживание канала *m* завершено, а значения старших четырех разрядов $D_{m+4} = 1$ сигнализируют о наличии не обслуженных запросов *DMA* от каналов *m*.

3.7. Программируемый связной интерфейс 8251*A*

Программируемые связные интерфейсы 8251/8251*A* (*Programmable Communication Interface — PCI*) фирмы *Intel* используются для передачи данных по последовательным каналам связи. Эти интерфейсы представляют собою универсальные синхронные/асинхронные приемопередатчики — *USART* (*Universal Synchronous/Asynchronous Receiver/Transmitter*). БИС 8251*A* является усовершенствованным вариантом БИС 8251 (отечественные аналоги 580ВВ51 и 580ВВ51А). Данные БИС предназначены для организации последовательного канала связи при проектировании МП-систем на основе микропроцессорных семейств фирмы *Intel MSC*–48, 80, 85 и *iAPX*–86, 88 (1816ВЕ48, 580ВМ80А, 1821ВМ85А и 1810ВМ86, 1810ВМ88).

Изготавливаются БИС по *n*-МОП технологии, причем входы и выходы совместимы с ТТЛ ИС. Кристалл БИС 8251 содержит 3350 транзисторов. Допустимые значения частоты тактового сигнала *CLK* равны: F_{CLK} = 0,74 ÷ 2,4 МГц (БИС 8251) и F_{CLK} = 0,74 ÷ 3,1 МГц (БИС 8251*A*). Максимальные значения токов потребления: $I_{CC\,max}$ = 80 мА (БИС 8251) и $I_{CC\,max}$ = 100 мА (БИС 8251*A*). Максимальная рассеиваемая мощность составляет 1 Вт. Выходные сигналы МП характеризуются параметрами:

$V_{OL\,max}$ = 0,45 В при I_{OL} = 1,6 мА, $V_{OH\,min}$ = 2,2 В при I_{OH} = –100 мкА для БИС 8251;

$V_{OL\,max}$ = 0,45 В при I_{OL} = 2,2 мА, $V_{OH\,min}$ = 2,4 В при I_{OH} = –400 мкА для БИС 8251*A*.

Классификация последовательных каналов связи. От МП на *USART* символы данных передаются в параллельном формате (байтами), а *USART* преобразует их в непрерывный последовательный поток бит для передачи по линии связи. Независимо от передачи *USART* может получать последовательные потоки бит по линии связи и преобразовывать их в параллельные коды данных для МП. *USART* сообщает МП всякий раз, когда он может принять новый символ для передачи или когда он принял символ для МП по линии связи. В качестве данных по каналу связи могут приниматься и передаваться 5-, 6-, 7- и 8-разрядные символы.

При приеме данных по линии связи *USART* в слове состояния фиксирует обнаруженные ошибки четности, переполнения и кадра. Прочитать слово состояния МП может в любой момент времени. Максимальная скорость передачи данных в синхронном режиме равна 64 Кбод, а в асинхронном режиме — 19,2 Кбод. На практике используются три типа каналов связи:

1. Симплексный канал (*Simplex Channel*) — передача данных производится в одном направлении (используется, например, для связи компьютера с удаленным печатающим устройством, для подключения мыши к персональному компьютеру и др.).

2. Полудуплексный канал (*Half-Duplex Channel*) — передача данных производится в двух направлениях по одной линии связи с разделением во времени приема и передачи.

3. Дуплексный канал (*Duplex Channel* или *Full-Duplex Channel*) — передача данных в двух направлениях производится по независимым линиям (прием и передача могут осуществляться одновременно); дуплексный канал эквивалентен двум симплексным каналам. БИС 8251*A* предназначена для построения дуплексного канала связи и может быть использована для организации полудуплексного канала связи.

Принцип асинхронной передачи данных. Формат кадра передаваемого по линии связи символа D_{7-0} изображен на рис. 3.79 (высокий уровень напряжения называется маркером, а низкий уровень — пробелом). Кадр начинается с передачи старт-бита (пробела), затем передаются разряды данных, начиная с младшего разряда D_0, бит паритета P (контрольный разряд для обнаружения однократной ошибки с помощью проверки на четность или нечетность) и один, полтора или два стоп-бита (маркеры). Контрольный разряд P при передаче 5-, 6-, 7- и 8-разрядных символов определяется одним из двух выражений (см. § 1.11):

$$P = \sum_{p=0}^{k-1} D_p \text{ — контроль четности, } P = \overline{\sum_{p=0}^{k-1} D_p} \text{ — контроль нечетности, } k = 5, 6, 7, 8.$$

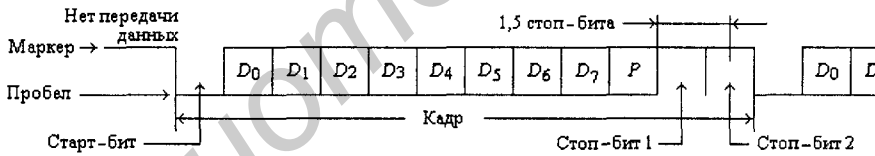


Рис. 3.79. Формат кадра при асинхронной передаче данных

Скорость передачи данных. Номинальная скорость передачи — это скорость передачи данных, определяемая количеством элементов двоичной информации, передаваемых за 1 секунду. Эффективная (реальная) скорость — это скорость передачи с учетом необходимости передачи служебной информации (старт-биты, стоп-биты и биты контроля четности), что уменьшает скорость передачи данных.

Скорость передачи измеряется в бодах (*бод* — единица скорости телеграфирования, названная в честь французского ученого Жана Мориса Эмиля Бодо в 1927 г.). Иногда вместо бод употребляют обозначение *bps* (*bit per second* — бит/с). Однако это немного разные вещи. Величина в бодах указывает количество передаваемых за секунду разрядов с учетом служебных разрядов (старт-биты, стоп-биты и биты контроля четности). А величина, указанная в *bps*, подразумевает скорость передачи самих данных. Типовые значения скорости передачи данных через последовательный интерфейс персональных компьютеров (*PC*) периферийных устройств, таких как модемы, составляют 1200, 2400, 4800, 9600, 19200 бод и выше.

Принцип асинхронного приема данных. Приемник должен быть согласован с передатчиком по всем параметрам формата передаваемого символа, включая и время передачи одного разряда. Для оптимальной защищенности от искажений и шумов в линии связи приемник дол-

жен считывать каждый принимаемый разряд в середине его длительности. Это можно сделать с помощью чтения разрядов с частотой в m раз большей скорости передачи разрядов. Обычно m = 16 или 64 (каждый разряд делится на m элементарных интервалов). Любой перепад входного сигнала приемника с 1 на 0 воспринимается как начало старт-бита (рис. 3.79). Истинность этого разряда проверяется вторично его стробированием через $m/2 - 1$ элементарных интервалов (в середине разряда). Если значение 0 в середине старт-бита не будет обнаружено, то приемник прекращает прием и переходит в состояние ожидания перепада входного сигнала с 1 на 0. Если значение 0 на входе приемника подтверждается, то запускается счетчик разрядов для считывания их значений в середине интервала передачи каждого разряда и для определения конца кадра.

Прием кадра заканчивается чтением значения стоп-бита. Если считан пробел (0), то фиксируется *ошибка кадра*. Если же обнаружен маркер (1), то приемник преобразует принятый последовательный код символа в параллельный и информирует МП о готовности данных. Если очередной символ, принятый *PCI* по линии связи, не будет вовремя прочитан микропроцессором, то он будет замещен новым принятым символом и в слове состояния будет зафиксирована *ошибка переполнения*. Приемник производит проверку принятого символа на четность или нечетность. Если будет обнаружена *ошибка паритета*, то она, как и ошибка кадра, фиксируется в регистре слова состояния. Обнаружение любой ошибки, вызванной помехами в канале связи и "нерасторопностью" МП, не останавливает работу приемника.

Структурная схема *PCI*. Расположение и обозначение контактов БИС **8251, 8251A** показано на рис. 3.80. Функциональные узлы, изображенные на структурной схеме *PCI* (рис. 3.81), имеют назначение:

Data Bus Buffer — буфер (приемопередатчик) шины данных с *Z*-состоянием выхода;

Read/Write Logic — схема управления чтением и записью, содержащая 8-разрядные регистры инструкции режима *MI* (*Mode Instruction*), инструкции команды *CI* (*Command Instruction*) и слова состояния *SW* (*Status Word*);

Modem Control — устройство управления модемом (модулятором-демодулятором). БИС может использоваться и без модема, например, для связи двух компьютеров с помощью кабеля, соединяющего последовательные порты;

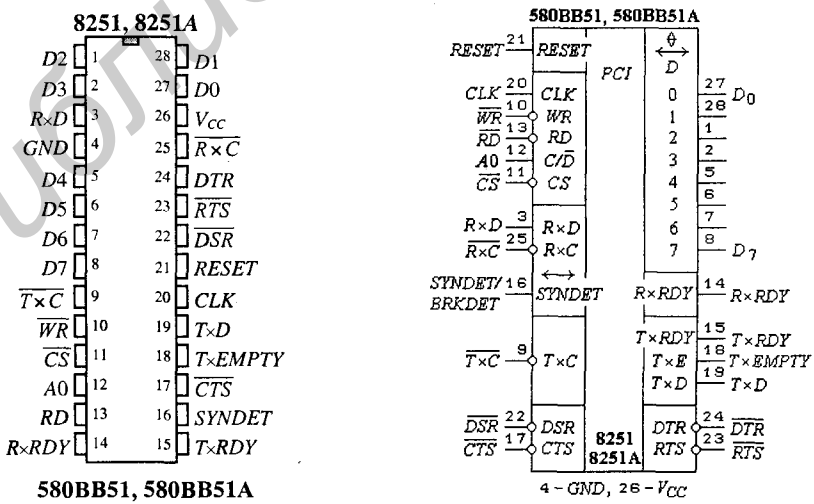


Рис. 3.80. БИС *USART* 8251/8251*A*

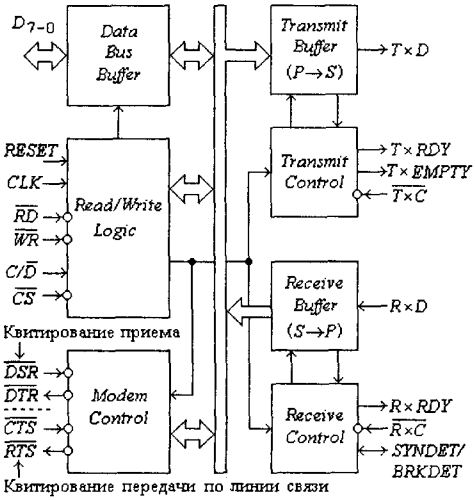


Рис. 3.81. Структурная схема *PCI* 8251*A*

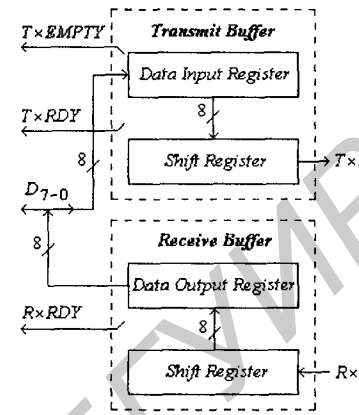


Рис. 3.82. Структурная схема буферов передатчика и приемника

Transmit Buffer ($P \rightarrow S$) — буфер передатчика (рис. 3.82), содержащий регистр ввода данных из МП (*Data Input Register*) и сдвигающий регистр типа *PI/SO* (*Parallel Input/Serial Output*), служащий для преобразования принимаемых от МП в параллельном формате символов в последовательный поток разрядов $T{\times}D$ (преобразование $P \rightarrow S$). Символ из регистра ввода данных передается в сдвигающий регистр после его освобождения. Сдвиг разрядов символа для передачи по линии связи производится по спадающему фронту тактового сигнала $\overline{T{\times}C}$. В последовательный поток данных автоматически вставляются служебные разряды или символы в соответствии с запрограммированным режимом работы;

Transmit Control — устройство управления передачей;

Receive Buffer ($S \rightarrow P$) — буфер приемника (рис. 3.82), содержащий сдвигающий регистр типа *SI/PO* (*Serial Input/Parallel Output*), служащий для преобразования в параллельный формат принимаемых из линии связи по входу $R{\times}D$ последовательных данных (преобразование $S \rightarrow P$), и регистр вывода данных в МП (*Data Output Register*). Полностью сформированный в параллельном формате символ из сдвигающего регистра передается в регистр вывода данных. Последовательные данные на входе $R{\times}D$ тактируются нарастающим фронтом тактового сигнала $\overline{R{\times}C}$. Проверка служебных разрядов и символов производится автоматически в соответствии с запрограммированным режимом работы;

Receive Control — устройство управления приемом, автоматически фиксирующее в регистре слова состояния *SW* (см. рис. 3.90) ошибку паритета *PE* (*Parity Error*), ошибку переполнения (*Overrun Error*) и ошибку кадра *FE* (*Framing Error*).

Назначение сигналов *PCI*. Программирование и управление БИС производится микропроцессором. Входные и выходные сигналы *PCI* имеют назначение:

D_{7-0} — сигналы 8-разрядной двунаправленной шины данных;

CLK (*Clock*) — тактовый сигнал внутренней синхронизации *PCI*. Обычно сигнал *CLK* подается от МП 8085 или $CLK = \varphi_2$ (от генератора 8224 для МП 8080). Период этого сигнала должен быть не менее чем в 30 раз меньше времени передачи одного разряда по последовательному каналу связи;

$C/\overline{D}$ (*Control/Data*) — обычно $C/\overline{D} = A_0$ (разряд шины адреса; см. табл. 3.14);

$\overline{CS}$ — сигнал с дешифратора адресных разрядов A_{7-1};

RESET — сигнал сброса *PCI* в состояние ожидания инструкции режима *MI* (*Mode Instruction*). Длительность значения сигнала *RESET* = 1 должна быть не менее 6 периодов тактового сигнала *CLK*;

$\overline{RD}$ (*Read*), $\overline{WR}$ (*Write*) — сигналы чтения и записи информации ($\overline{RD} = \overline{I/OR}$, $\overline{WR} = \overline{I/OW}$ при использовании МП 8080*A*/8085*A*);

T×D (*Transmitter Data*) — выходные последовательные данные передатчика. Сигнал *RESET* = 1 устанавливает выход *T×D* в состояние маркера;

$\overline{T \times C}$ (*Transmitter Clock*) — тактовый сигнал передатчика, частота которого определяет скорость передачи по последовательному каналу связи (бит/сек);

T×RDY (*Transmitter Ready* — готовность передатчика) — выходной управляющий сигнал, указывающий МП на готовность буфера передатчика принять байт данных для передачи по последовательному каналу связи. Сигнал *T×RDY* дублируется одним разрядом в слове состояния *SW* (см. рис. 3.90). Разряд *T×RDY* в *SW* используется для квитирования программного вывода данных из МП в буфер передатчика. Сигнал *T×RDY* используется для запроса прерывания для обслуживания вывода данных из МП в буфер передатчика (сигнал *T×RDY* в этом случае подается на один из входов IR_i контроллера прерываний 8259*A*). Запрос прерывания можно замаскировать разрядом *T×EN* в инструкции команды (см. рис. 3.89). Разряд *T×RDY* в *SW* не маскируется разрядом *T×EN*, но только указывает, пуст или заполнен регистр данных в буфере передатчика. МП записывает байт данных в регистр памяти буфера передатчика по требованию значения *T×RDY* = 1, сбрасывая его в 0 передним фронтом сигнала $\overline{WR}$, а *PCI* после преобразования предыдущего символа в последовательный код переписывает вновь поступивший символ в сдвигающий регистр и устанавливает значение *T×RDY* = 1 (рис. 3.83);

T×EMPTY (*Transmitter Empty* — передатчик пустой) — выходной сигнал управления *PCI*, информирующий МП о завершении передачи. Значение *T×EMPTY* = 1 устанавливается при отсутствии в регистре данных буфера передатчика символа для передачи после окончания передачи предыдущего символа по последовательному каналу связи (рис. 3.83). Значение *T×EMPTY* сбрасывается в 0 после получения символа от МП при условии, что передача разрешена. Сигнал *T×EMPTY* можно использовать для определения конца передачи при работе в полудуплексном режиме. В синхронном режиме значение *T×EMPTY* = 1 указывает, что символ для передачи от МП не поступил, и в последовательный поток данных в качестве заполнителя автоматически вводятся символы синхронизации *Sync*;

R×D (*Receiver Data*) — входные последовательные данные приемника;

$\overline{R \times C}$ (*Receiver Clock*) — тактовый сигнал приемника, частота которого определяется скоростью передачи по последовательному каналу связи (бит/сек). В большинстве систем связи для передатчика и приемника используется один генератор тактовых сигналов $\overline{T \times C} = \overline{R \times C}$;

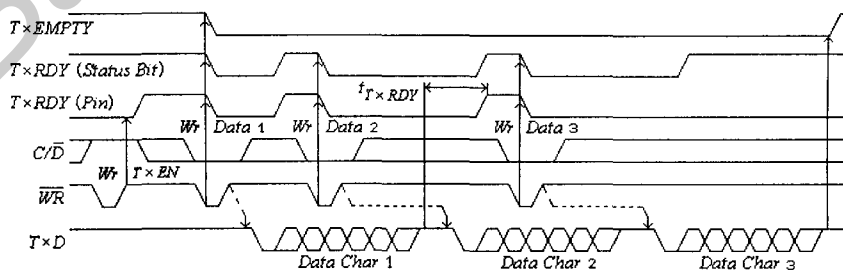


Рис. 3.83. Временны́е диаграммы асинхронной передачи данных

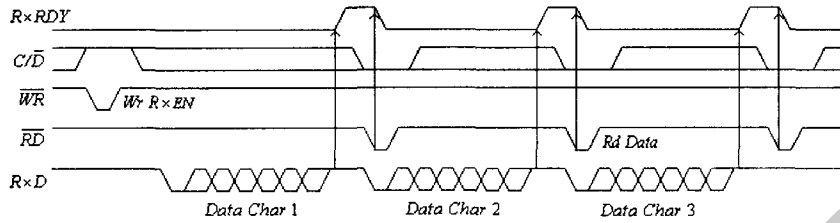


Рис. 3.84. Временны́е диаграммы асинхронного приема данных

R×RDY (*Receiver Ready* — готовность приемника) — выходной сигнал *PCI*, который дублируется одним разрядом в слове состояния *SW* (см. рис. 3.90). Разряд *R×RDY* в *SW* используется для квитирования программного ввода в МП данных, принятых по последовательному каналу связи и преобразованных приемником в параллельный формат. Сигнал *R×RDY* используется для запроса прерывания для обслуживания ввода данных в МП из буфера приемника *PCI* (сигнал *R×RDY* в этом случае подается на один из входов IR_i контроллера прерываний 8259*A*). Устанавливается значение *R×RDY* = 1 по окончании приема очередного символа по линии связи и перемещении его в регистр вывода данных буфера приемника (рис. 3.84). Значение *R×RDY* сбрасывается в 0 при чтении микропроцессором символа из регистра вывода данных буфера приемника значением сигнала $\overline{RD}$ = 0. Если очередной символ, принятый по линии связи, не будет вовремя прочитан микропроцессором, то он будет замещен новым принятым символом и в слове состояния *SW* будет зафиксирована ошибка переполнения *OE* (*Overrun Error* — см. рис. 3.90);

SYNDET/BRKDET (*Sync Detect/Break Detect* — обнаружение синхронизации/обнаружение паузы) — в синхронном режиме приема является выходным сигналом (*SYNDET*) при задании инструкцией команды *CI* режима внутренней синхронизации и входным сигналом при задании этой инструкцией режима внешней синхронизации (см. рис. 3.88). В асинхронном режиме приема *BRKDET* является выходным сигналом. Сигнал *RESET* = 1 устанавливает значение *SYNDET/BRKDET* = 0 (выход).

Сигналы управления модемом. Эти сигналы используются для квитирования передачи последовательных данных между *PCI* и модемом (рис. 3.85). Оборудование, участвующее в передаче данных по последовательным каналам связи, принято делить на два типа: терминальное (*DTE — Data Terminal Equipment*) и связное (*DCE — Data Communications Equipment*). Примерами терминального оборудования могут служить компьютеры, принтеры, сканеры и т. п. (*PCI* входит в состав терминального оборудования). Типовым примером связного оборудования является модем, который служит соединительным звеном, например, между компьютером и телефонной линией. При передаче данных по длинным линиям связи на обоих концах линии включаются модемы, преобразующие логические уровни передаваемого сигнала (цифровой сигнал) в физический (аналоговый) сигнал с использованием того или иного вида модуляции (при приеме производится обратное преобразование). Для управления приемом и передачей данных между *PCI* и модемом предназначены пары сигналов $\overline{DTR}$, $\overline{DSR}$ и $\overline{RTS}$, $\overline{CTS}$:

$\overline{DTR}$ (*Data Terminal Ready* — готовность данных для терминала) — выходной сигнал *PCI*, подаваемый в модем и используемый для запроса его готовности передать данные в *PCI*. Запрос готовности (значение $\overline{DTR}$ = 0) посылает МП подачей в *PCI* инструкции команды *CI* (см. рис. 3.89);

$\overline{DSR}$ (*Data Set Ready* — данные для терминала подготовлены) — входной сигнал *PCI*, поступающий из модема в ответ на сигнал $\overline{DTR}$ и указывающий на готовность модема передать

данные ($\overline{DSR}$ = 0). Значение этого сигнала фиксируется в регистре слова состояния *SW* (см. рис. 3.90) и может быть прочитано микропроцессором;

$\overline{RTS}$ (*Request To Send* — запрос передачи данных) — выходной сигнал *PCI*, передаваемый в модем и используемый для запроса его готовности принять данные из *PCI*. Запрос готовности (значение $\overline{RTS}$ = 0) посылает МП подачей в *PCI* инструкции команды *CI* (см. рис. 3.89);

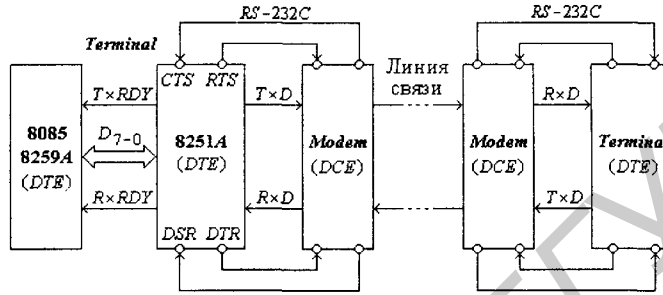


Рис. 3.85. Структурная схема дуплексного канала связи

$\overline{CTS}$ (*Clear To Send* — отсутствие передачи и готовность модема принять данные) — входной сигнал *PCI*, поступающий из модема в ответ на сигнал $\overline{RTS}$ и используемый для разрешения передачи данных из *PCI* в модем.

Сигналы управления модемом имеют универсальное назначение и при необходимости могут использоваться для квитирования передачи последовательных данных при отсутствии в МП-системе модема.

Итак, сигналы *T×RDY*, *R×RDY* и их копии в слове состояния используются для ввода и вывода данных с квитированием и по прерыванию в МП-системе (терминале), а сигналы $\overline{DTR}$, $\overline{DSR}$ и $\overline{RTS}$, $\overline{CTS}$ — для квитирования передачи последовательных данных между терминальным и связным оборудованием.

Управление данными и режимами работы *PCI*. Операции ввода-вывода описаны в табл. 3.14: *MI* (*Mode Instruction*) — инструкция режима, *CI* (*Command Instruction*) — инструкция команды, *SW* (*Status Word*) — слово состояния, $A_0 = C/\overline{D}$. При значении $A_0 = 0$ выполняется чтение символа из регистра ввода данных (*Data Input Register*) буфера приемника и запись символа в регистр вывода данных (*Data Output Register*) буфера передатчика (рис. 3.82). При значении $A_0 = 1$ производятся операции по управлению работой *PCI* и чтение слова состояния, которое используется для управления процессом ввода-вывода символов.

Таблица 3.14. **Таблица ввода-вывода *PCI***

$\overline{CS}$	A_0	$\overline{RD}$	$\overline{WR}$	Операция	Примечание
0	0	0	1	$D_{7-0} \leftarrow$ *PCI Data*	Ввод в МП данных из буфера приемника
0	0	1	0	$D_{7-0} \rightarrow$ *PCI Data*	Вывод из МП данных в буфер передатчика
0	1	0	1	$D_{7-0} \leftarrow$ *SW*	Чтение слова состояния *SW*
0	1	1	0	$D_{7-0} \rightarrow$ *MI*, *CI*	Запись слов управления *MI* и *CI*
0	×	1	1	Нет операций	*Z*-состояние D_{7-0}
1	×	×	×	Нет операций	*Z*-состояние D_{7-0}

Из табл. 3.14 видно, что *PCI* обеспечивает для МП "прозрачность" интерфейса — требуется выполнять только команды ввода и вывода (IN *port* и OUT *port*), а вставку служебных разрядов и символов при передаче и их удаление при приеме выполняет сам *PCI* без участия МП. Полезно сравнить описанный выше дуплексный канал связи с последовательными каналами ввода и вывода, реализованными непосредственно в МП 8085*A*, где вся рутинная работа по преобразованию форматов данных выполняется программным способом, что ведет к существенной потере производительности МП-системы.

Программирование *PCI*. Законченное функциональное назначение *PCI* 8251*A* задается программным обеспечением МП-системы: синхронный или асинхронный режим работы, скорость приемопередачи в бодах, длина символа, число стоп-бит, контроль четности/нечетности или его отсутствие, внутренняя или внешняя синхронизация в синхронном режиме и т. д.

После подачи сигнала *RESET* = 1 (внешний сброс) выход *T×D* устанавливается в состояние маркера и *PCI* переходит в режим ожидания инструкции режима *MI*. Последовательность подачи слов управления *MI*, *CI* и блоков данных *Data* (запись в буфер передатчика и чтение из буфера приемника) показана на рис. 3.86 (блок-схема алгоритма поясняет работу внутреннего цифрового автомата, адресующего регистры *MI*, *CI*, *SYNC* 1 и *SYNC* 2).

Программирование *PCI* производится записью в регистры управляющих слов: инструкции режима *MI* и инструкции команды *CI*. Инструкция режима *MI* подается на *PCI* только один раз. После записи *MI* на выходе *T×RDY* устанавливается значение 1 — требование к МП загрузить байт данных в буфер передатчика.

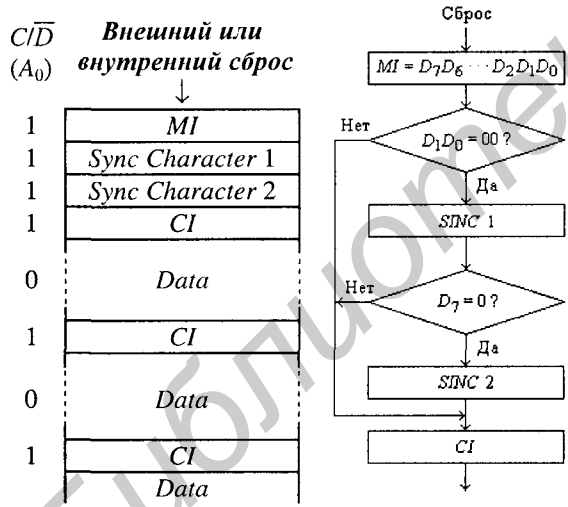


Рис. 3.86. Управление работой *PCI* 8251*A*

Если в инструкции *MI* задан синхронный режим, то в регистры *SYNC* записываются еще одно или два слова, представляющих собой синхросимволы (*Sync Character*). После этого подается инструкция команды *CI*, а затем МП посылает в буфер передатчика данные байт за байтом для передачи по последовательному каналу связи. При записи каждого байта данных в буфер передатчика сигнал *T×RDY* и одноименный флаг в слове состояния сбрасываются в 0. Передача может начаться только при условии, что в инструкции команды *CI* задано значение *T×EN* = 1 (см. рис. 3.89) и от модема получено подтверждение его готовности принимать данные, т. е. получено значение сигнала $\overline{CTS} = 0$.

Инструкция команды *CI* должна следовать за инструкцией режима *MI* и символами синхронизации *Sync*. Все слова управления, записываемые в *PCI* после инструкции режима *MI*, загружают инструкцию команды *CI*. Инструкция команды может быть подана в *PCI* в любое время в течение передачи или приема блока данных. Для изменения инструкции режима *MI* необходимо подать инструкцию команды *CI* = 40*h*, в которой задано значение разряда *IR* = 1 (*Internal Reset* — см. рис. 3.89) внутреннего сброса, переводящего *PCI* в режим ожидания инструкции режима *MI*.

После включения питания *PCI* может оказаться в неопределенном состоянии (в состоянии ожидания инструкции режима *MI*, ввода синхросимволов *Sync* или инструкции команды *CI*). Поэтому перед подачей команды внутреннего сброса (значения *CI* = 40*h*) необходимо произвести инициализацию *PCI* для самого плохого случая, требующего записи байта *MI* и двух байт

Sync, что гарантирует переход *PCI* в состояние ожидания инструкции команды *CI*. Последовательная запись в *PCI* трех байт 00*h* (первый байт задает синхронный режим работы, а остальные — фиктивные значения двух синхросимволов *Sync*) обеспечивает инициализацию *PCI* для всех его исходных состояний. После этого можно подавать команду внутреннего сброса, переводящую *PCI* в состояние ожидания инструкции режима *MI*.

Формат инструкции асинхронного режима. Инструкция режима *MI* определяет основные характеристики *PCI*. Она должна следовать сразу же после внешнего или внутреннего сброса. Формат *MI* для установки асинхронного режима показан на рис. 3.87:

$D_1D_0 = B_2B_1 \neq 0$ — асинхронный режим; код $\overline{B_2B_1}$ задает коэффициент деления частоты тактовых сигналов передатчика $\overline{T \times C}$ и приемника $\overline{R \times C}$;

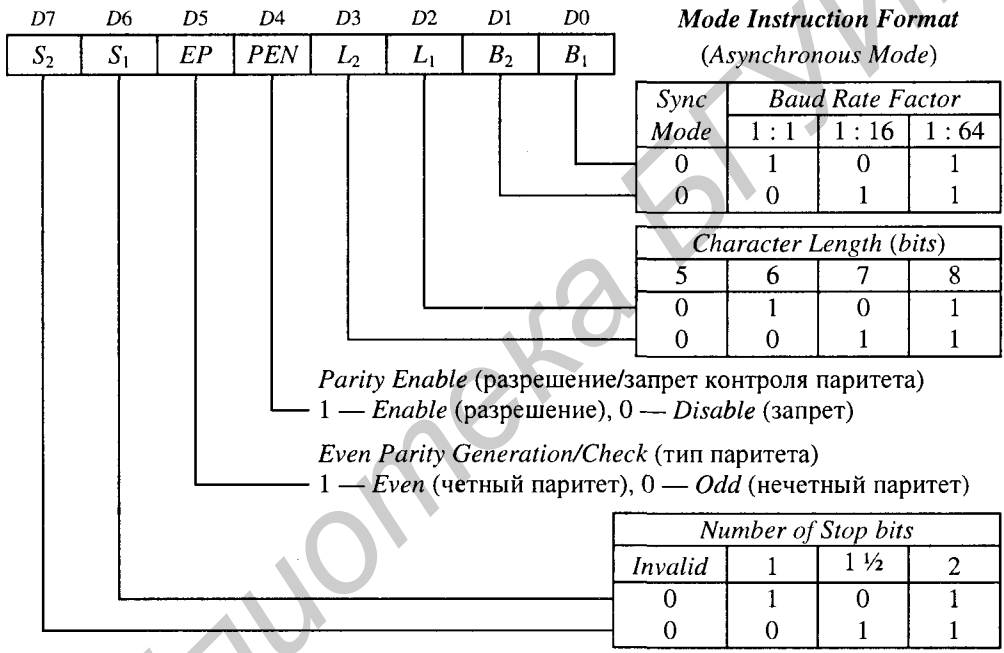


Рис. 3.87. Формат инструкции *MI* асинхронного режима

$D_3D_2 = L_2L_1$ — число разрядов в символе (длина символа в битах); если число разрядов в символе меньше 8 бит, то неиспользуемыми будут старшие разряды (при чтении регистра буфера приемника они будут равны 0);

$D_4 = PEN$ — разрешение/запрет генерации контрольного разряда *P*;

$D_5 = EP$ — контроль четности/нечетности (*Even*/*Odd*);

$D_7D_6 = S_2S_1$ — число стоп-бит.

Передатчик автоматически добавляет к разрядам данных старт-бит, бит паритета (если $PEN = 1$) и заданное число стоп-бит при формировании кадра в соответствии с рис. 3.79.

Формат инструкции синхронного режима. Формат *MI* для установки синхронного режима показан на рис. 3.88:

$D_1D_0 = 00$ — синхронный режим;

D_6 — задание режима внешней ($D_6 = 1$) или внутренней ($D_6 = 0$) синхронизации;

D_7 — число синхросимволов (остальные разряды имеют то же самое назначение, что и для асинхронного режима).

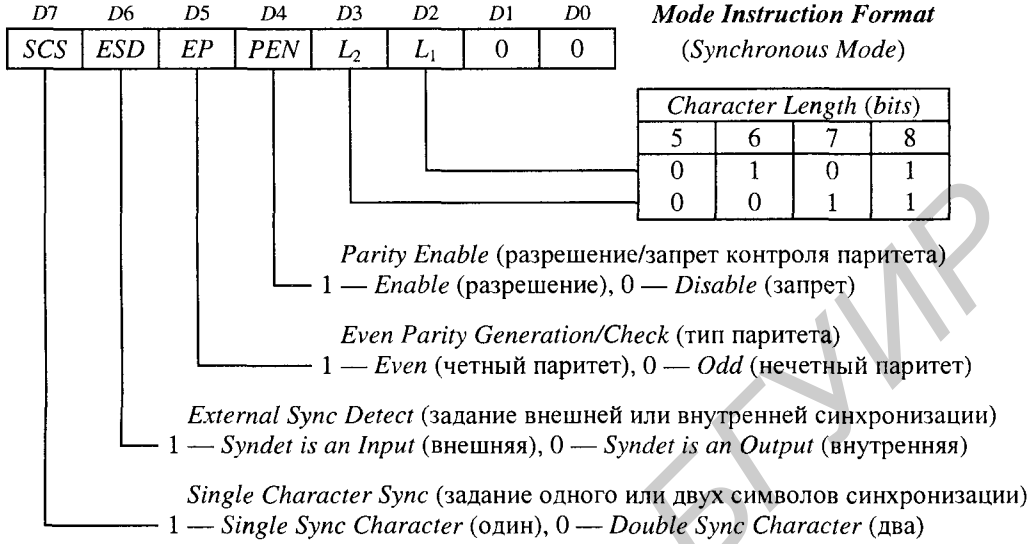


Рис. 3.88. Формат инструкции *MI* синхронного режима

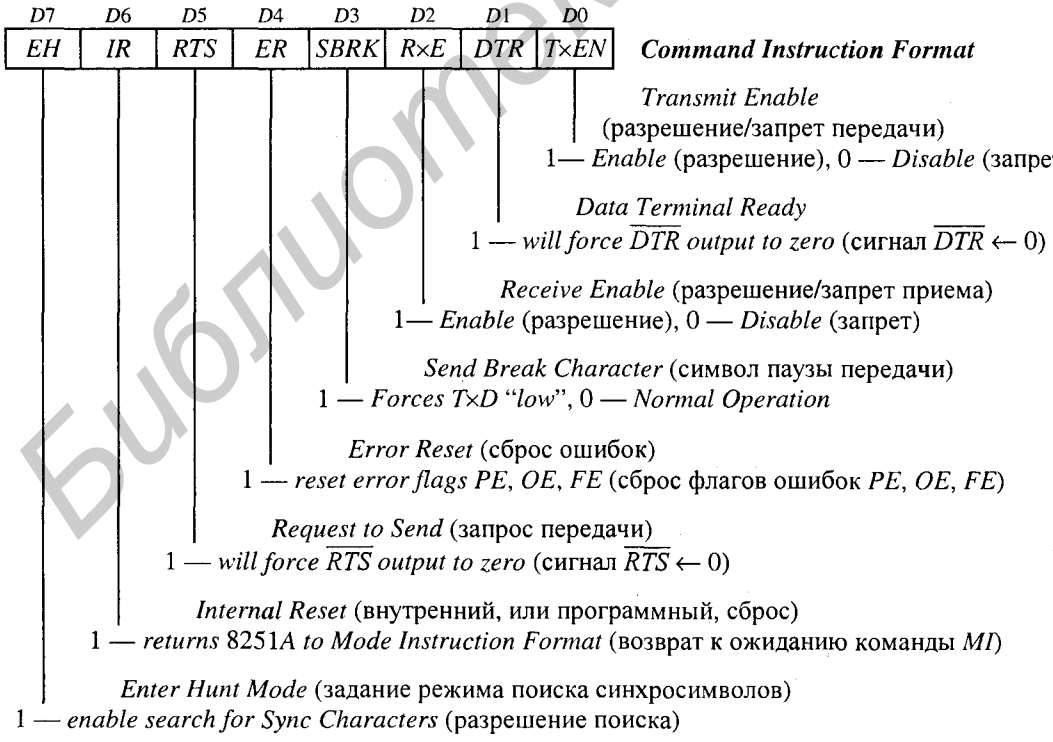


Рис. 3.89. Формат инструкции команды *CI*

Формат инструкции команды. Инструкция команды *CI* подается после программирования режима работы *PCI* и записи в регистры приемника синхросимволов при синхронном режиме работы (см. рис. 3.86). Формат *CI* изображен на рис. 3.89:

$D_0 = T{\times}EN$ — разрешение передачи данных;

$D_1 = DTR$ — задание значения $\overline{DTR} = 0$ сигнала запроса готовности данных терминала, если $D_1 = 1$;

$D_2 = R{\times}E$ — разрешение приема;

$D_3 = SBRK$ — задание нормального режима асинхронной передачи ($D_3 = 0$) или паузы ($D_3 = 1$) при отсутствии данных для передачи (выход передатчика $T{\times}D$ устанавливается при этом в состояние 0);

$D_4 = ER$ — сброс флагов ошибок в слове состояния;

$D_5 = RTS$ — задание значения $\overline{RTS} = 0$ сигнала запроса передачи данных, если $D_5 = 1$;

$D_6 = IR$ — внутренний сброс *PCI* для перевода его в режим ожидания инструкции режима *MI*;

$D_7 = EH$ — разрешение поиска синхросимволов.

Слово состояния *PCI*. Формат слова состояния (*SW* — *Status Word*) показан на рис. 3.90:

$D_0 = T{\times}RDY$ — в отличие от выхода $T{\times}RDY$ значение данного разряда не переводится в 0 сигналом $\overline{CTS} = 1$ и значением разряда $T{\times}EN = 0$ в инструкции команды *CI*;

$D_1 = R{\times}RDY$ — состояние выхода $R{\times}RDY$;

$D_2 = T{\times}EMPTY$ — состояние выхода $T{\times}EMPTY$;

$D_3 = PE$ — ошибка паритета, обнаруженная при приеме символа из последовательного канала связи;

$D_4 = OE$ — ошибка переполнения (МП не успел прочитать из буфера приемника предыдущий принятый из последовательного канала связи символ и он был замещен новым принятым символом);

$D_5 = FE$ — ошибка кадра (принято неверное значение стоп-бита; флаг *FE* используется только в асинхронном режиме). Обновление разрядов ошибок на время чтения слова состояния *SW* запрещается;

$D_6 = SYNDET/BRKDET$ — состояние вывода *SYNDET/BRKDET*;

$D_7 = DSR$ — состояние входа $\overline{DSR}$ (данные для терминала подготовлены).

Для лучшего восприятия назначения сигналов квитирования ввода-вывода в табл. 3.15 показано соответствие этих сигналов для БИС 8251*A* и 8255*A* (разряды $R{\times}E$ и $T{\times}EN$ в инструкции команды *CI* разрешают и запрещают прием и передачу как при программном выводе с квитированием, так и при выводе по прерыванию). Хотя обозначение сигналов квитирования у БИС разное, но служат они одним и тем же целям.

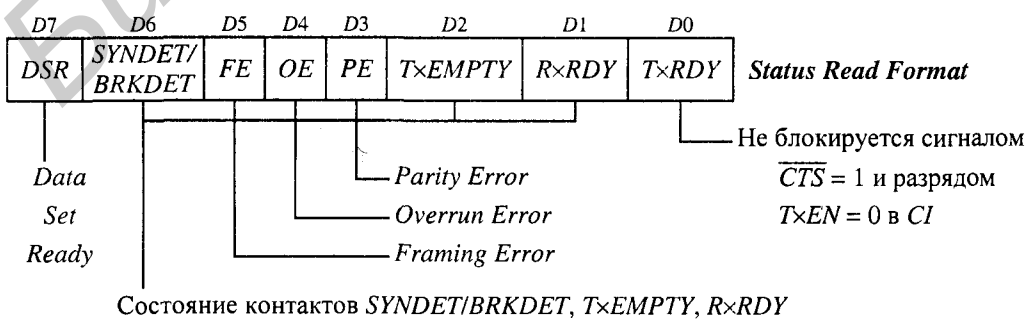


Рис. 3.90. Формат слова состояния *SW*

Таблица 3.15. **Соответствие сигналов квитирования БИС 8251*A* и 8255*A***

8251*A*	8255*A*	Примечание
R×*RDY* в *SW* *T*×*RDY* в *SW*	*IBF* *OBF*	Значения этих флагов МП получает при чтении слова состояния *SW* для программного ввода-вывода с квитированием
R×*RDY* *T*×*RDY*	*INTR* *INTR*	Эти сигналы с выходов БИС 8251*A* подключаются к входам IR_i БИС 8259*A* при вводе-выводе по прерыванию
R×*E* в *CI* *T*×*EN* в *CI*	*INTE* *INTE*	Программирование значений этих внутренних сигналов БИС 8251*A* используется для разрешения/запрета ввода-вывода по прерыванию

Асинхронная передача. Всякий раз, как символ данных поступает от МП, перед информационными разрядами *PCI* автоматически добавляет старт-бит, а в конце кадра — запрограммированное число стоп-бит (см. рис. 3.79). Если в инструкции режима *MI* контроль четности/нечетности разрешен (разряд *PEN* = 1), то после информационных разрядов вставляется бит паритета *P*. Передача символа данных начинается с младшего разряда D_0.

Последовательные данные выдаются на выход *T*×*D* по спадающему фронту тактового сигнала $\overline{T \times C}$ с частотой, заданной в инструкции режима *MI* (1, 1/16 или 1/64 частоты сигнала $\overline{T \times C}$).

Передача кадров начинается только при условии, что в инструкции команды *CI* задано значение *T*×*EN* = 1 и от модема получено подтверждение его готовности принимать данные (значение сигнала $\overline{CTS} = 0$). Если очередной символ в буфер передатчика не загружен, то на выходе *T*×*D* устанавливается значение 1 (маркер) при задании в инструкции команды *CI* значения разряда *SBRK* = 0 (нормальная работа). Если же в команде *CI* задано значение разряда *SBRK* = 1 (пауза), то на выход *T*×*D* непрерывно выдается значение 0 (рис. 3.91) — символы паузы (*Break Characters*). При отсутствии данных для передачи и при подаче команды паузы устанавливается значение сигнала *T*×*EMPTY* = 1 — передатчик пустой.

Асинхронный прием. В нормальном режиме работы *PCI* значение входного сигнала приемника *R*×*D* = 1 до момента поступления данных. Спадающий фронт этого сигнала взводит триггер начала старт-бита. Истинность старт-бита проверяется стробированием его значения в номинальном центре (только при задании режимов 1 : 16 и 1 : 64 в инструкции режима *MI*). Если при стробировании будет обнаружено значение 0 (правильное значение), то включается счетчик разрядов, входящих в состав кадра (в противном случае приемник возвращается в исходное состояние). С помощью этого счетчика определяется центр каждого разряда для принятия решения о его значении 0 или 1. Разряды символа данных и бит паритета вводятся по нарастающему фронту тактового сигнала $R \times C$. При обнаружении ошибки паритета в разряд *PE* слова состояния *SW* записывается 1. Стоп-бит сообщает о конце символа. Для управления работой приемника требуется только один стоп-бит независимо от числа запрограммированных стоп-бит. Если будет обнаружена ошибка при приеме стоп-бита (0 вместо 1), то в слове состояния *SW* будет зафиксирована ошибка кадровой синхронизации (*FE* = 1).

В режиме 1 : 1 при использовании общего генератора тактовых сигналов $\overline{T \times C} = \overline{R \times C}$ для всего канала связи (в двух терминалах) нарастающий фронт тактового сигнала $\overline{R \times C}$ автоматически приходится на центр всех разрядов, так как передача ведется по спадающему фронту тактового сигнала $\overline{T \times C}$ (предполагается, что скважность тактовых сигналов, равна 2).

Принятый символ загружается в параллельном формате в регистр вывода данных буфера приемника (см. рис. 3.82) в момент времени, соответствующий середине первого стоп-бита, и сигнал *R*×*RDY* изменяется с 0 на 1, информируя МП о готовности данных (рис. 3.92). Если МП очередной символ не прочитает из буферного регистра приемника, то он будет заменен следующим принятым символом и в слове состояния *SW* будет зафиксирована ошибка переполне-

ния (*OE* = 1 — предыдущий символ потерян). Флаги всех ошибок (*PE*, *FE* и *OE*) могут быть сброшены инструкцией команды *CI*, если в ней задано значение разряда *ER* = 1. Обнаружение любой из этих ошибок не останавливает работу приемника.

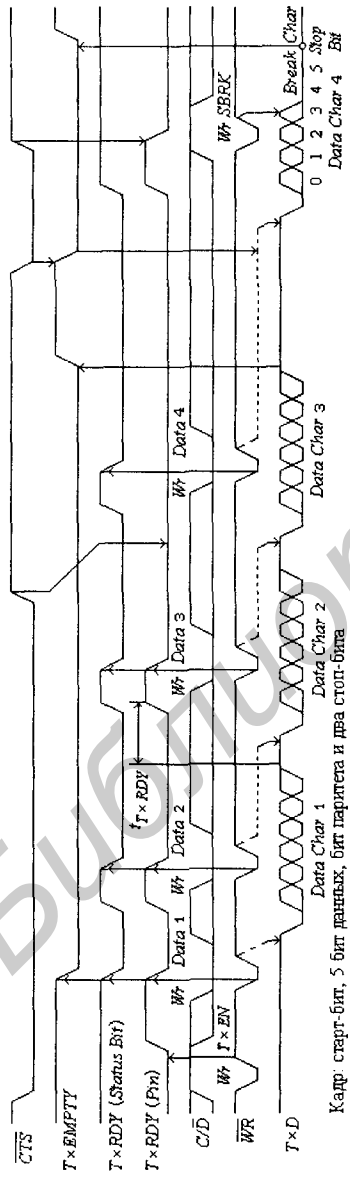


Рис. 3.91. Временны́е диаграммы асинхронного режима передачи данных

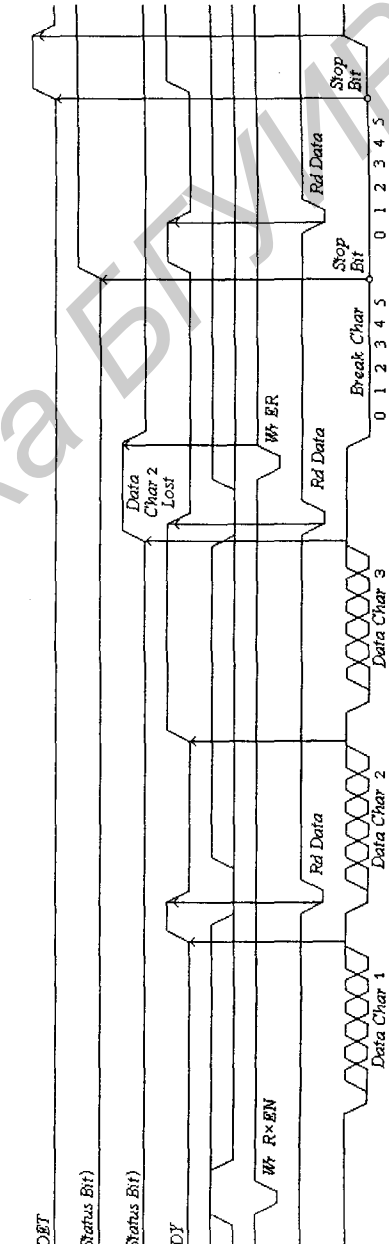


Рис. 3.92. Временны́е диаграммы асинхронного режима приема данных

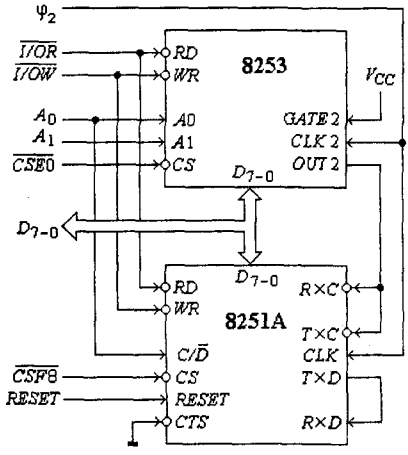


Рис. 3.93. Тестирование *PCI*

При приеме символов паузы (*Break Characters*) после обнаружения двух последовательных нулевых значений стоп-бит устанавливаются значения 1 сигнала *BRKDET* (рис. 3.92) и разряда *SYNDET/BRKDET* в слове состояния. Сбрасывают их в 0 первый переход входного сигнала приемника $R{\times}D$ с 0 на 1 и значение сигнала *RESET* = 1.

В качестве примера использования последовательного дуплексного асинхронного канала связи при использовании программного ввода данных с квитированием рассмотрим тестирование работоспособности *PCI* 8251*A*. Структурная схема для тестирования *PCI* изображена на рис. 3.93 — для получения скорости передачи 9600 бод используется программируемый таймер 8253 для формирования тактовых сигналов $\overline{T{\times}C} = \overline{R{\times}C}$ из тактового сигнала φ_2 генератора 8224 (580ГФ24). Если в инструкции режима *MI* задать коэффициент деления равным 16 ($D_1D_0 = B_2B_1 = 10$; см. рис. 3.87), то модуль пересчета M таймера может быть найден из соотношения:

$$f_{T\times C} = f_{R\times C} = f_{\varphi 2} / (16 \cdot M) = 2^{11} \cdot 10^3 / (2^4 \cdot M),$$

т. е. модуль пересчета M = 13, что дает скорость передачи ≈9846 бод (стандартная скорость передачи равна 9600 бод).

Если использовать адреса портов E0, E1, E2, E3*h* для таймера 8253 ($\overline{CS} = \overline{CSE0}$) и F8, F9*h* для *PCI* 8251*A* ($\overline{CS} = \overline{CSF8}$), то можно предложить для тестирования программу:

```
        MVI   A, 96h    ; Программирование таймера 8253: CW = 96h
        OUT   0E3h     ;      (R/L LSB, режим работы M3, двоичный счет)
        MVI   A, 0Dh    ; Модуль пересчета M = 0Dh = 13d
        OUT   0E2h     ; Загрузка модуля пересчета в канал 2 таймера
        SUB   A        ; Инициализация PCI 8251A — последовательная запись
        OUT   0F9h     ;                                  трех байт 00h
        OUT   0F9h
        OUT   0F9h
        MVI   A, 40h    ; Инструкция команды внутреннего сброса CI = 40h = 0100 0000
        OUT   0F9h     ; Подача в PCI команды внутреннего сброса
        MVI   A, 0BEh   ; MI = BEh (1:16, 8 бит; 1,5 стоп-бита)
        OUT   0F9h     ; Запись MI
        MVI   A, 15h    ; CI = 15h = 0001 0101 (R×E = 1, T×EN = 1)
        OUT   0F9h     ; Запись CI
        MVI   A, Data   ; Data = 00 … FFh
        CALL  Tr_Rec
        JNZ   Error
        CMA
        CALL  Tr_Rec
        JNZ   Error
          ∴             ; Вывод на дисплей сообщения VALID
        HLT
Error:    ∴             ; Вывод на дисплей сообщения ERROR
```

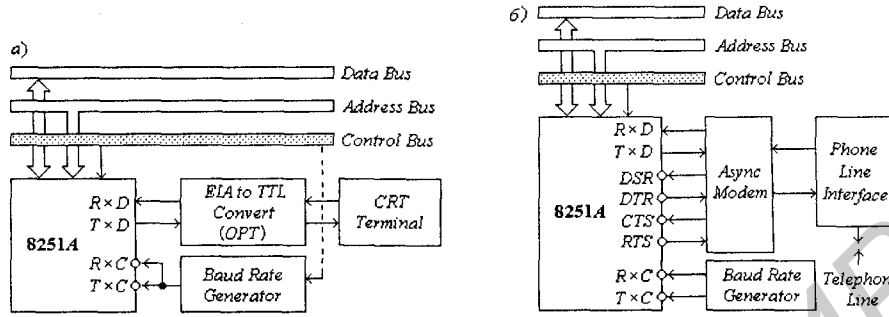


Рис. 3.94. Асинхронные интерфейсы

```
            HLT
             ∴
Tr_Rec:  MOV  B, A     ; Подпрограмма ввода с квитированием Tr_Rec
         OUT  0F8h     ; Запись данных в буфер передатчика
L1:      IN   0F9h     ; Чтение слова состояния SW
         ANI  2        ; Выделение разряда D1 = R×RDY из SW
         JZ   L1
         IN   0F8h     ; Чтение принятых данных из буфера приемника
         CMP  B        ; Верификация
         RET           ;      (проверка равенства переданного и принятого символов)
```

В этой программе используется только квитирование ввода, так как вывод байта данных всегда производится в заведомо пустой буфер передатчика. В реальной системе вместо останова МП (команда HLT) после вывода на дисплей сообщения управление должно передаваться операционной системе.

На рис. 3.94, *а* изображен последовательный асинхронный интерфейс для передачи данных между микроЭВМ и монитором (*CRT — Cathode-ray Tube* — электронно-лучевая трубка). Здесь использован *EIA*-интерфейс с оптоэлектронной гальванической развязкой оборудования (*EIA — Electronic Industries Association* — ассоциация электронной промышленности США, регламентирующая электрические и функциональные характеристики интерфейсного оборудования и кабельных систем). Асинхронный интерфейс для телефонной линии показан на рис. 3.94, *б*.

Синхронная передача. Выход передатчика $T{\times}D = 1$ до тех пор, пока он не начнет передавать первый символ, которым обычно является символ синхронизации *Sync*. Передача начинается только при установке сигнала $\overline{CTS}$ в 0 (готовность модема принять данные) и при $T{\times}EN = 1$ в инструкции команды *CI*. Сдвиг разрядов символов производится спадающим фронтом тактового сигнала $\overline{T \times C}$. Частота передачи разрядов равна частоте сигнала $\overline{T \times C}$.

При отсутствии данных для передачи в поток данных $T{\times}D$ автоматически вставляются синхросимволы (рис. 3.95). Это позволяет принимающему терминалу не терять синхронизацию при временном отсутствии данных для передачи. При отсутствии данных для передачи сигнал $T{\times}EMPTY$ устанавливается в 1 в момент времени, соответствующий номинальному центру последнего разряда передаваемого символа. Значение сигнала $T{\times}EMPTY = 1$ сигнализирует, что регистр ввода буфера передатчика пуст и передаются синхросимволы. При записи данных в буфер передатчика сигнал $T{\times}EMPTY$ сбрасывается в 0.

Синхронный прием. При приеме последовательных данных по входу $R{\times}D$ в *PCI* предусмотрены два способа установления символьной синхронизации, называемых внутренней и

внешней синхронизацией. Если в инструкции режима *MI* задана внутренняя синхронизация (разряд *ESD* = 0), то в первой инструкции команды *CI* следует указать режим поиска синхросимволов (разряд *EH* = 1).

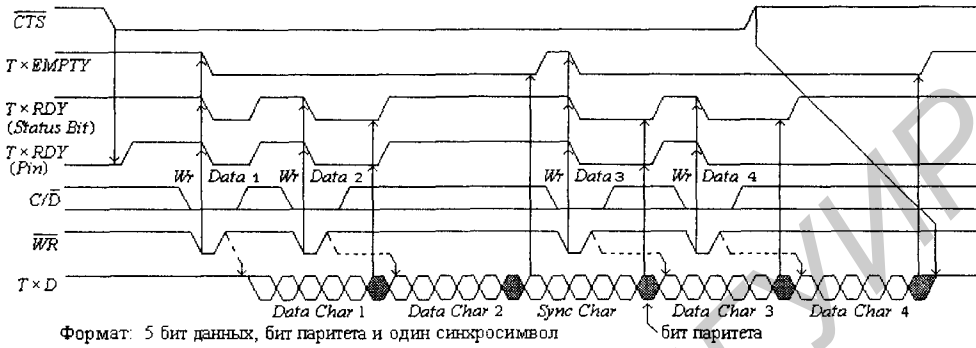


Рис. 3.95. Временны́е диаграммы синхронного режима передачи данных

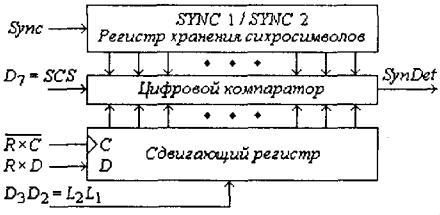


Рис. 3.96. Поиск синхросимволов

Данные на входе *R×D* вводятся в сдвигающий регистр по нарастающему фронту тактового сигнала $\overline{R \times C}$. В каждом такте цифровой компаратор сравнивает принятые разряды с образцом синхросимвола *Sync* 1, записанным в регистр *SYNC* 1, пока не будет установлено их равенство, сигнализирующее об установлении символьной синхронизации (рис. 3.96; число синхросимволов D_7 = *SCS* и длина символов $D_3D_2 = L_2L_1$ задаются инструкцией синхронного режима — см. рис. 3.88).

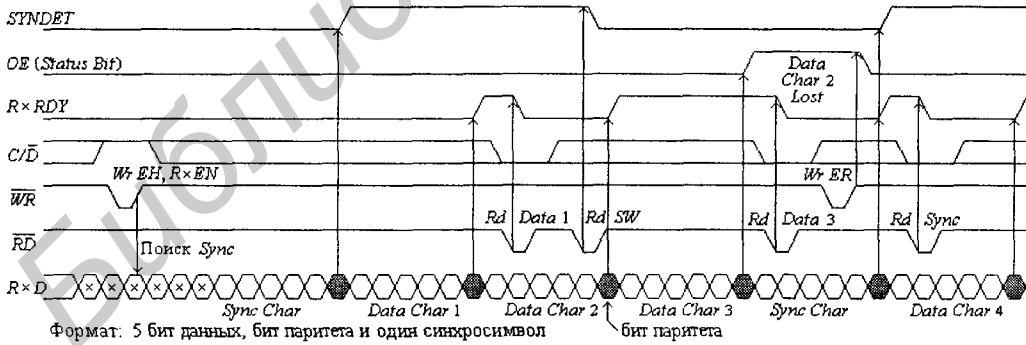


Рис. 3.97. Временны́е диаграммы синхронного режима приема данных

Если инструкцией режима *MI* запрограммирована работа с двумя синхросимволами (*bi-sync* — метод синхронизации фирмы *IBM*), то сравнение производится двух последовательно принятых символов с образцами двух синхросимволов *Sync* 1 и *Sync* 2, записанных в регистры *SYNC* 1 и *SYNC* 2. При обнаружении синхросимволов *PCI* заканчивает режим поиска, установив символьную синхронизацию, и изменяет выходной сигнал *SYNDET* с 0 на 1 в середине последнего информационного разряда или в середине бита четности, если в инструкции

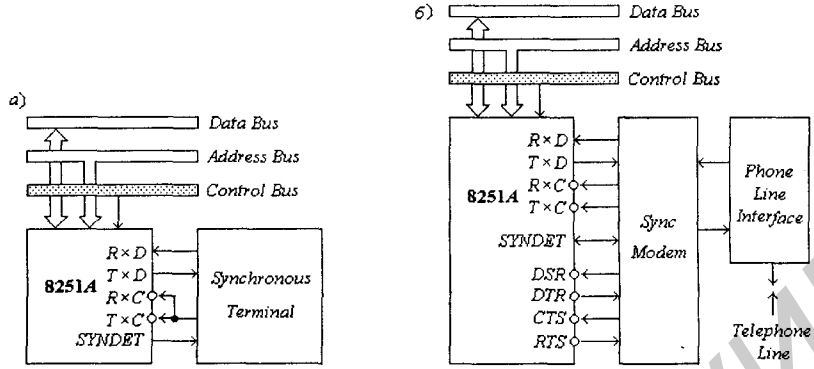


Рис. 3.98. Синхронные интерфейсы

режима *MI* запрограммирован контроль паритета (рис. 3.97). Это означает, что *PCI* определил время поступления первого разряда каждого последующего символа.

Значение сигнала *SYNDET* = 1 сигнализирует передающему терминалу об установлении синхронизации (рис. 3.98, *а*) и терминал начинает передавать данные. Сбрасывается сигнал *SYNDET* в 0 автоматически при чтении слова состояния *SW*. Ошибки паритета и переполнения устанавливаются таким же образом, как и в асинхронном режиме. На рис. 3.98, *б* показан синхронный интерфейс для телефонной линии.

Второй способ установления символьной синхронизации (разряд *EH* = 0 в инструкции режима *MI*) заключается в подаче на вход *SYNDET* высокого уровня сигнала — переход этого сигнала с 0 на 1 задает начало приема по первому же нарастающему фронту тактового сигнала $\overline{R \times C}$. Поиск синхросимволов при этом блокируется. Входной сигнал *SYNDET* может быть сброшен в 0 после одного такта сигнала $\overline{R \times C}$.

При потере символьной синхронизации МП должен подать в *PCI* команду поиска синхросимволов (разряд *EH* = 1 в инструкции команды *CI*). Эта команда устанавливает все разряды символов, находящихся в буфере приемника, в 1, что предотвращает возможное ложное обнаружение синхронизации из-за случайных данных на входе приемника во время поиска. Сигнал *SYNDET* сбрасывается в 0 при каждом чтении слова состояния *SW* независимо от запрограммированного режима внутренней или внешней синхронизации. Это не переводит *PCI* в режим поиска синхросимволов. После перехода к приему символов данных обнаружение синхросимволов производится при известном местоположении их первого разряда.

Если при первом и втором чтении слова состояния *SW* будут получены значения разряда D_6 = *SYNDET/BRKDET* = 1, то это означает, что после выполнения первого чтения были приняты вставленные в поток данных символы синхронизации *Sync*.

3.8. Последовательные интерфейсы

Интерфейс (*Interface*) — совокупность средств и правил, обеспечивающих взаимодействие устройств вычислительной системы и (или) программ; совокупность унифицированных технических и программных средств, используемых для сопряжения устройств в вычислительной системе или сопряжения между системами. Интерфейс ввода-вывода (*Input-Output Interface*) — стандартное сопряжение устройств управления внешними устройствами и каналов ввода-вывода.

Стандарты средств связи и интерфейсов ЭВМ. К числу наиболее широко распространенных видов интерфейсов, используемых в микроЭВМ, относятся следующие:

а) последовательные интерфейсы для локальных терминалов;

б) параллельные интерфейсы для локальных периферийных устройств (таких, как быстродействующие принтеры) и других приборов, подключаемых посредством шины *IEEE* 488;

в) интерфейсы локальных сетей, делающие возможным обмен информацией с терминалами, периферийными устройствами, другими процессорами и удаленными сетями (через станции связи с внешними сетями);

г) дистанционные связи и протяженные сети, обеспечиваемые последовательными синхронными интерфейсами, подключаемыми через модемы и использующими тот или иной протокол связи.

Применение стандартных интерфейсов позволяет изготовителю ЭВМ обходиться небольшим количеством типов интерфейсов, обеспечивающих возможность обмена информацией с разнообразными терминалами, периферийными устройствами, приборами и другими типами ЭВМ. Первыми были разработаны стандарты электрических параметров последовательных и параллельных интерфейсов. В 1969 г. был опубликован стандарт последовательного интерфейса *RS*-232*C*, а в 1975 г. — стандарты последовательных быстродействующих интерфейсов *RS*-422*A* и *RS*-423*A* (*RS — Recommended Standard* — рекомендуемый стандарт). Универсальный стандарт на функциональные и механические характеристики *RS*-449, который совместно с интерфейсами *RS*-422*A* и *RS*-423*A* предназначался для замены интерфейса *RS*-232*C* (однако этого не произошло), опубликован в 1977 г.

Шинный стандарт *IEEE* 488, опубликованный в 1978 г., был разработан на основе универсальной приборной шины *GPIB* (*General-Purpose Interface Bus*) фирмы *Hewlett Packard*. Он представляет собой в первую очередь описание электрических и механических параметров и не содержит подробной характеристики сообщений и форматов, посредством которых эти сообщения должны передаваться по шине. Шина *IEEE* 488 имеет 8 двунаправленных линий данных, 5 линий управления шиной и 3 линии квитирования. Институт инженеров по электротехнике и радиоэлектронике — ИИЭР (*IEEE — Institute of Electrical and Electronic Engineers*) — создан в США в 1963 г. для разработки стандартов, проведения конференций и выпуска специализированных изданий в области электротехники и радиоэлектроники.

Появление в 70-х годах локальных и протяженных вычислительных сетей стимулировало разработку стандартов на физические (включая электрические) параметры систем и протоколы передачи данных:

а) стандарт *X*25 Международного консультативного комитета по телефонии и телеграфии (*CCITT — Comite' Consultatif Internationale de Telegraphique et Telephonique*) для терминалов с коммутацией пакетов сетей общественного пользования; *ITU-T* (*ITU-Telecommunications*) — новое название комитета в составе Международного союза *ITU* (*International Telecommunications Union* — Международный союз по электросвязи);

б) локальная сеть *Ethernet*, положившая впоследствии начало стандарту *IEEE* 802.3.

К концу 70-х годов стало очевидно, что для полного использования потенциала вычислительных сетей необходимы открытые международные стандарты, охватывающие наряду с протоколами физических и информационных связей все аспекты пересылки данных между ЭВМ. До этого все стандарты носили закрытый характер (разрабатывались и использовались отдельными изготовителями ЭВМ, такими, как *IBM* или *DEC*). В 1977 г. Международная Организация по стандартизации (*ISO — International Standards Organization*; основана в 1946 г.) выступила инициатором формулирования требований к внутренним соединениям открытых систем (*OSI — Open System Interconnection*) — была предложена семиуровневая модель протоколов передачи данных, предназначенная для сопряжения различных видов коммуникационного оборудования независимых производителей (состоит из следующих уровней: физического, канального, сетевого, транспортного, сеансового, представления данных, прикладного).

1 — закрепленный 25-контактный разъем типа *D* (штырьки), 2 — съемный разъем типа *D* (гнезда);
3 — кабель интерфейса *RS*-232*C*, 4 — съемный разъем типа *D* (штырьки);
5 — закрепленный 25-контактный разъем типа *D* (гнезда), 6 — телефонная линия

Рис. 3.99. Типичная последовательная линия связи между компьютерами

Последовательный интерфейс *RS*-232*C*. Стандарт *RS*-232*C*, введенный Американской ассоциацией электронной промышленности (*EIA — Electronic Industries Association*), был опубликован в 1969 г. и до начала 80-х годов практически оставался единственным стандартом на последовательный интерфейс для подключения ЭВМ и терминалов к системам связи через модемы, а также для непосредственного подключения терминалов к ЭВМ. Затем были введены уточняющие стандарты *RS*-232*D* (1987 г.) и *ANSI EIA/TIA*-232-*E* (*RS*-232*E*, 1991 г.), не затрагивающие характеристик стандарта *RS*-232*C* (*RS*-232 — общее обозначение этих стандартов; *ANSI — American National Standards Institute* — Американский национальный институт стандартов; *TIA — Telecommunications Industries Association* — ассоциация производителей средств телекоммуникации).

Интерфейс, определенный стандартом *EIA*, подразумевает наличие оборудования двух видов: терминального (*DTE — Data Terminal Equipment*) и связного (*DCE — Data Communications Equipment*). Термин *DTE* (оконечное оборудование данных) — применяется для обозначения любых устройств, использующих канал связи (компьютеров, принтеров, сканеров, мышей и т. д.), а термин *DCE* (аппаратура передачи данных) — для обозначения аппаратных средств, обеспечивающих установление, поддержание и разрыв соединения с каналом связи (рис. 3.99). Примером связного оборудования является модем, который служит соединительным звеном между компьютером и телефонной линией.

Европейскими аналогами стандарта *RS*-232, разработанными *CCITT*, являются стандарт *V*24, который охватывает механические характеристики, детальное описание разъемов и функциональное описание протоколов обмена, и стандарт *V*28, охватывающий электрические характеристики сигналов (ГОСТ 18145–81 — соответствующий отечественный стандарт).

Типовым разъемом, соответствующим стандарту *RS*-232, является 25-контактный разъем типа *D*, вилка которого размещается на *DTE*, а розетка — на модеме (*DCE*; табл. 3.16).

Уровни всех входных и выходных сигналов, используемые в интерфейсе *RS*-232, отличаются от логических уровней ИС: логический 0 (*Space* — пробел) представляется положительным напряжением в диапазоне от +3 до +25 В, а логическая 1 (*Mark* — маркер) — отрицательным напряжением в диапазоне от –3 до –25 В (минимальное входное напряжение приемника равно ±3 В). Интерфейс *RS*-232 увеличивает помехозащищенность канала связи. Так, при использовании уровней напряжений ±12 В данные передаются без потерь по кабелю длиною 50 м и более.

Пример линии связи на основе интерфейса *RS*-232. Для преобразования уровней сигналов используются специальные ИС (рис. 3.100, *а*). Передатчик *SN*75150 (*TR — Transmitter*) выполняет логическую функцию $DO_i = E \cdot DI_i$ ($i = 0$ и 1) с преобразованием входных *TTL*-уровней сигналов DI_i в выходные *RS*-232-уровни сигналов DO_i (рис. 3.100, *б*). Линия связи обычно реализуется с помощью двух проводников, образующих витую пару. Приемник *SN*75154 (*RC — Receiver*) производит обратное преобразование уровней напряжений — *RS*-232-уровни входных сигналов DI_i преобразуются в *TTL*-уровни выходных сигналов DO_i ($i = 0, 1, 2, 3$). Передатчик и приемник соответствуют стандарту *ANSI EIA/TIA*-232-*E*.

Таблица 3.16. **Разъем для интерфейса *RS*-232**

Контакт разъема *DB*25	Направление передачи сигнала	Сигнал	Назначение сигнала
1	—	*FG*	Основная (защитная) земля
2	к *DCE*	*T×D*	Передаваемые данные
3	к *DTE*	*R×D*	Принимаемые данные
4	к *DCE*	*RTS*	Запрос передачи данных
5	к *DTE*	*CTS*	Сброс передачи (готовность модема принять данные)
6	к *DTE*	*DSR*	Готовность модема передавать данные
7	—	*SG*	Сигнальная земля
8	к *DTE*	*DCD*	Обнаружение несущей данных
9	к *DTE*	—	Положительное контрольное напряжение (+12 В, 20 мА)
10	к *DTE*	—	Отрицательное контрольное напряжение (–12 В, 20 мА)
11	к *DCE*	*QM*	Выбор частоты передачи
12	к *DTE*	*SDCD*	Обнаружение несущей дополнительного канала
13	к *DTE*	*SCTS*	Сброс передачи дополнительного канала
14	к *DCE*	*ST×D*	Передаваемые данные дополнительного канала
15	к *DTE*	*T×C*	Синхронизация передачи
16	к *DTE*	*SR×D*	Принимаемые данные дополнительного канала
17	к *DTE*	*R×C*	Синхронизация приема
18	к *DCE*	*DCR*	Свободный (местный шлейф)
19	к *DCE*	*SRTS*	Запрос передачи дополнительного канала
20	к *DCE*	*DTR*	Готовность терминала принимать данные
21	к *DTE*	*SQ*	Детектор качества сигнала
22	к *DTE*	*RI*	Индикатор вызова (звонка)
23	к *DTE*	*DRS*	Переключатель скорости передачи данных
24	к *DCE*	*TC*	Внешняя синхронизация передачи
25	к *DTE*	—	Свободный (индикатор тестирования)

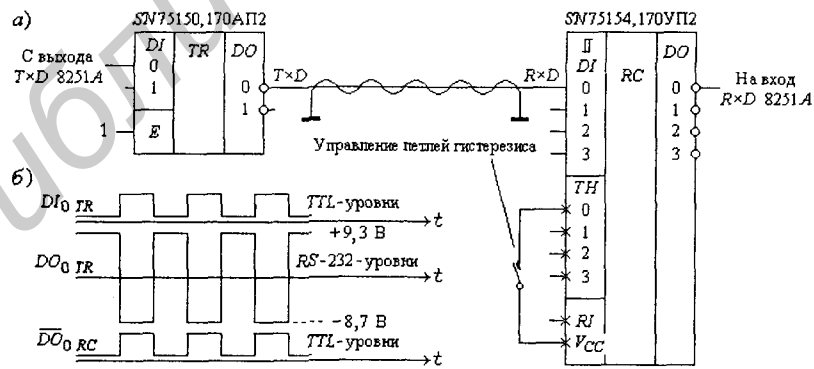


Рис. 3.100. Линия связи на основе интерфейса *RS*-232

На вход DI_0 передатчика *TR* подается сигнал с последовательного выхода данных *T×D PCI* 8251*A*, а выходной сигнал $\overline{DO}_0$ приемника *RC* — на последовательный вход данных *R×D* другого *PCI* 8251*A*.

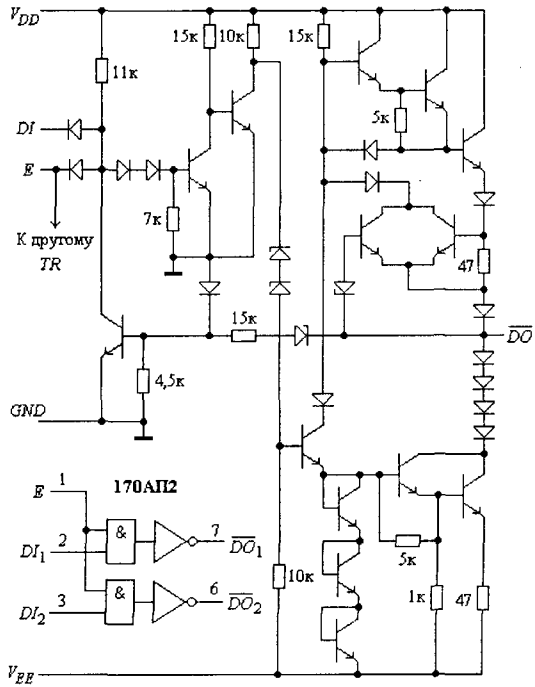


Рис. 3.101. Принципиальная схема одного передатчика типа *SN*75150

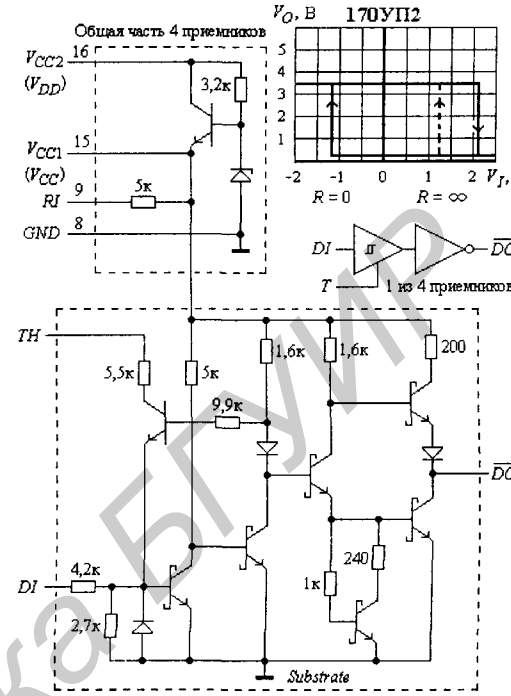


Рис. 3.102. Принципиальная схема одного приемника типа *SN*75154

На рис. 3.101 изображена принципиальная схема одного передатчика типа *SN*75150 (170АП2) — для конвертирования *TTL*-уровней входного сигнала *DI* в *RS*-232-уровни выходного сигнала $\overline{DO}$ требуется использовать два напряжения питания V_{DD} = +12 В и V_{EE} = –12 В (стандартные напряжения питания). Скорость передачи 20 Кбит/с обеспечивается при емкостной нагрузке 2500 пФ. При напряжениях питания V_{DD} = +10,8 В и V_{EE} = –10,8 В (±12 В – 10%) гарантируются уровни V_O = ±8 В выходного сигнала. Токи потребления: $I_{DD\ typ}$ = +10 мА, $I_{DD\ max}$ = +22 мА и $I_{EE\ typ}$ = –9 мА, $I_{EE\ max}$ = –20 мА.

Принципиальная схема одного приемника типа *SN*75154 показана на рис. 3.102 — обычно на ИС подается напряжение питания V_{CC1} = +5 В, но при необходимости можно использовать и напряжение питания V_{CC2} = +12 В (внутренний стабилизатор преобразует +12 В в +5 В). Контакт *TH* (*Threshold Control*) предназначен для задания порога чувствительности приемника — ширины петли гистерезиса. Его можно оставить открытым (R = ∞ — отказоустойчивый режим работы) или подключить к контакту V_{CC1} (R = 0) — нижний порог срабатывания изменяется с +1,4 В на –1,1 В (см. график петли гистерезиса на рис. 3.102). Контакт *TH* можно подключить также к контакту *RI* — получится промежуточное значение нижнего порога срабатывания.

Последовательные порты *PC IBM*. Персональный компьютер *PC IBM* имеет не менее двух последовательных портов — *COM*1 и *COM*2. Каждый порт обеспечивает работу дуплексного канала связи, все входные и выходные сигналы которого соответствуют стандарту *RS*-232 (используются напряжения питания ±12 В). Для портов *COM* используются 9- и 25-контактные разъемы. В табл. 3.17 показано назначение контактов таких разъемов, предназначенных для подключения к компьютеру терминального оборудования (в том числе и других компьютеров) по асинхронному каналу связи.

Таблица 3.17. **Назначение контактов разъема порта *COM***

Контакт разъема *DB*9	Контакт разъема *DB*25	Вход/Выход	Название сигнала
1	8	Вход	*DCD* (*Data Carrier Detect*)
2	3	Вход	*R*×*D* (*Receive Data*)
3	2	Выход	*T*×*D* (*Transmit Data*)
4	20	Выход	*DTR* (*Data Terminal Ready*)
5	7	—	*GND* (*Signal Ground*)
6	6	Вход	*DSR* (*Data Set Ready*)
7	4	Выход	*RTS* (*Request to Send*)
8	5	Вход	*CTS* (*Clear to Send*)
9	22	Вход	*RI* (*Ring Indicate*)

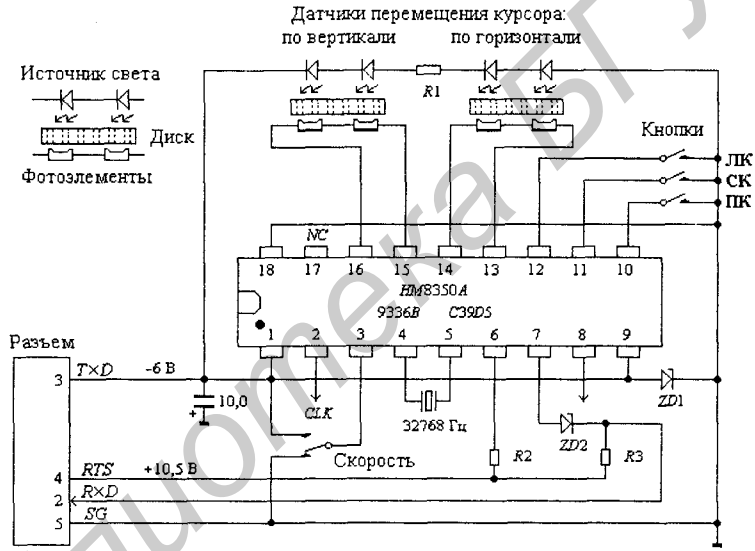


Рис. 3.103. Контроллер последовательной мыши

Контроллер последовательной мыши. С начала триумфального шествия графических оболочек, таких как *Windows* фирмы *Microsoft*, мышь стала типовым устройством ввода информации в компьютеры. Принципиальная схема последовательной *Microsoft*-совместимой мыши (*Serial MS-Mouse*) для *PC IBM* изображена на рис. 3.103 — используется асинхронный симплексный канал связи (передача данных только от мыши в *PC IBM*). Принцип работы оптико-механической мыши предполагается читателю известным (перемещение мыши по коврику вращает диски с радиальными прорезями датчиков вертикального и горизонтального перемещения курсора; контроллер мыши производит счет числа импульсов, поступивших от датчиков, и передает их в МП).

Контроллер такой мыши выполняется в одной ИС по КМОП-технологии (например, ИС *HM*8350*A* или *HT*6510), что позволяет передавать на мышь двуполярное питание, соответствующее интерфейсу *RS*-232, по сигнальным линиям передатчика (контроллер непрограммируемый, поэтому передавать информацию от *PC IBM* на мышь не нужно). Для питания мыши используются сигналы-константы *RTS* и *T*×*D*, в ненагруженном состоянии имеющие уровни

+11,5 В и –11,5 В. Стабилитрон *ZD*1 преобразует напряжение сигнала *T*×*D* в стабилизированное напряжение –6 В для питания ИС контроллера. Временны́е диаграммы на выводах 7 и 8 контроллера и выходного сигнала *R*×*D* изображены на рис. 3.104, *а* (стабилитрон *ZD*2 использован для формирования двуполярного сигнала ±3 В). Каскад формирования выходного сигнала *R*×*D* с большими уровнями (+9,5 В и –5,5 В) показан на рис. 3.104, *б*.

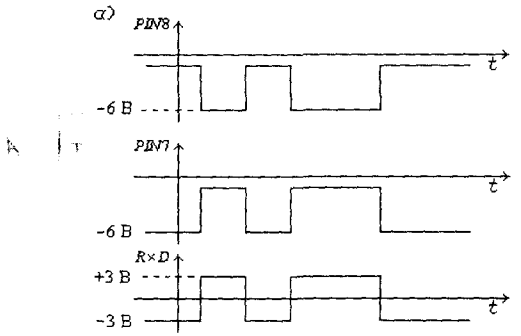


Рис. 3.104. Временны́е диаграммы мыши (*а*) и выходной каскад мыши для интерфейса *RS*-232 (*б*)

Контроллер содержит генератор тактового сигнала *CLK*, синхронизирующий его работу (этот сигнал имеет частоту 32768 Гц и во внешних цепях не используется). Ввод данных с помощью мыши производится под управлением драйвера *mouse*.com для *MS-DOS*, который является резидентной программой.

Кабель нуль-модема. Два терминала, например два компьютера, можно соединить между собой с помощью интерфейса *RS*-232 и без модема. В этом случае необходим только кабель, называемый кабелем нуль-модема (*null-modem cable*). Такой кабель, соединяющий *PC IBM* через последовательные порты COM_m (m = 1 или 2; см. табл. 3.17), показан на рис. 3.105.

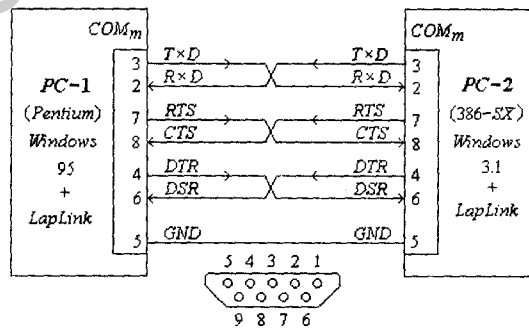


Рис. 3.105. Кабель нуль-модема

Для обмена данными между компьютерами используются программы *interlnk*.exe *MS-DOS* 6.2 (и выше) или *ll*3.exe при работе в оболочке *DOS*. При работе в *Windows* 3.1 и *Windows* 95 удобен программный пакет *LapLink for Windows*.

Последовательные интерфейсы *RS*-422*A*, *RS*-423*A* и *RS*-485. В начале 70-х годов были разработаны новые последовательные интерфейсы, предназначенные для снятия ограничений по скорости и длине кабеля, присущие интерфейсу *RS*-232. Стандарты интерфейсов *RS*-422*A* и *RS*-423*A* ассоциации электронной промышленности (*EIA*) были опубликованы в 1975 г. (стандарты *V*27 и *V*26 — Европейские аналоги стандартов *RS*-422*A* и *RS*-423*A*). Стандарт *RS*-449 является дополнением к стандартам интерфейсов *RS*-422*A* и *RS*-423*A* (стандарт *V*36 — Европейский аналог стандарта *RS*-449). Стандарт *RS*-422*A* определяет электрические характеристики симметричных цепей интерфейса (передатчики имеют парафазные выходы, приемники — дифференциальные входы), а стандарт *RS*-423*A* — несимметричных цепей интерфейса, как и стандарт *RS*-232.

Таблица 3.18. **Сравнительные параметры интерфейсов *RS*-232, *RS*-422*A*, *RS*-423*A* и *RS*-485**

Параметр	*RS*-232	*RS*-422*A*	*RS*-423*A*	*RS*-485
Тип линии связи	Несимметричная	Симметричная	Несимметричная	Симметричная
Максим. длина линии связи, м	15	1200	1200	1200
Максим. скорость передачи данных, Мбод	0,02	10	0,10	10
Маркер (данные 1)	–3 В	$A < B$	A отрицат.	$A < B$
Пробел (данные 0)	+3 В	$A > B$	A положит.	$A > B$
Максим. входн. напряжение приемника, В	±25	±7	±12	–7 ÷ +12
Выходное напряжение передатчика:				
минимальное, В	±5	±2	±3,6	±1,5
максимальное, В	±15	±5	±5,4	±5
Чувствительность приемника, В	±3	±0,2	±0,2	±0,2
Миним. входн. сопротивл. приемника, кОм	3 ÷ 7	4	4	12
Максим. ток короткого замыкания, мА	500	150	150	250

В появившемся позднее интерфейсе *RS*-485 используются передатчики и приемники, имеющие выходы с тремя состояниями для работы с многоточечными и секционированными шинами, обслуживающими до 32 пар передатчик/приемник. В остальных аспектах стандарт *RS*-485 аналогичен стандарту *RS*-422*A*, допускающему скорость передачи до 10 Мбит/с. Большинство ИС передатчиков, соответствующих стандарту *RS*-485, совместимо по контактам с передатчиками, отвечающими более раннему стандарту *RS*-422*A*, что облегчает совершенствование характеристик интерфейсов. Передатчики и приемники интерфейса *RS*-485 можно использовать и для интерфейса *RS*-422*A*.

Основные параметры интерфейсов *RS*-232, *RS*-422*A*, *RS*-423*A* и *RS*-485 сопоставлены в табл. 3.18.

Примеры линий связи на основе интерфейсов *RS*-422*A*, *RS*-423*A* и *RS*-485. Пример реализации симметричной и несимметричной последовательных линий связи на основе интерфейсов *RS*-422*A* и *RS*-423*A* показан на рис. 3.106, *а*.

В симметричном интерфейсе для представления сигнала маркера используется разность уровней напряжения на двух линиях, а для представления пробела изменяется знак этой разности (изменяется полярность сигналов). Благодаря тому, что при этом отсутствует потребность в опорных напряжениях, симметричные интерфейсы устойчивы по отношению к ошибкам, возникающим из-за различия опорных напряжений, а вследствие того, что электрические помехи одинаково воздействуют на обе линии (синфазные помехи), обеспечивается повышенная устойчивость интерфейса к шумам. На рис. 3.106, *б* показана симметричная линия связи, построенная на ИС серии 559 (управление *Z*-состоянием выходов не используется; Ф = 0 или 1).

На рис. 3.107, *а* изображена структурная схема последовательного полудуплексного канала связи на основе интерфейса *RS*-485. Подсоединить к линии связи можно до 32 ЭВМ — каждая ЭВМ может передавать данные на остальные ЭВМ. Каждым передатчиком *TR* управляет сигнал *DE* (*Driver Enable*): значение *DE* = 0 переводит оба выхода передатчика в *Z*-состояние. Каждым приемником *RC* управляет сигнал $\overline{RE}$ (*Receiver Enable*): значение $\overline{RE}$ = 1 переводит выход приемника *DO* в *Z*-состояние.

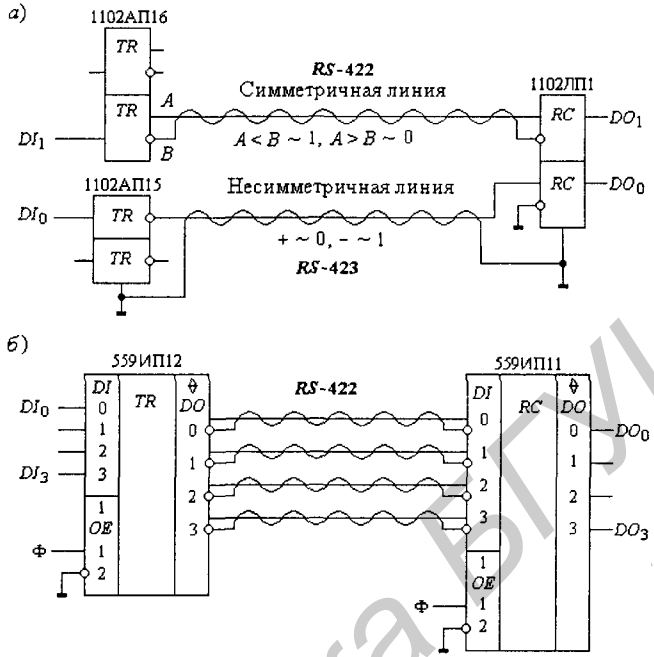


Рис. 3.106. Линии связи на основе интерфейсов *RS*-422*A* и *RS*-423*A*

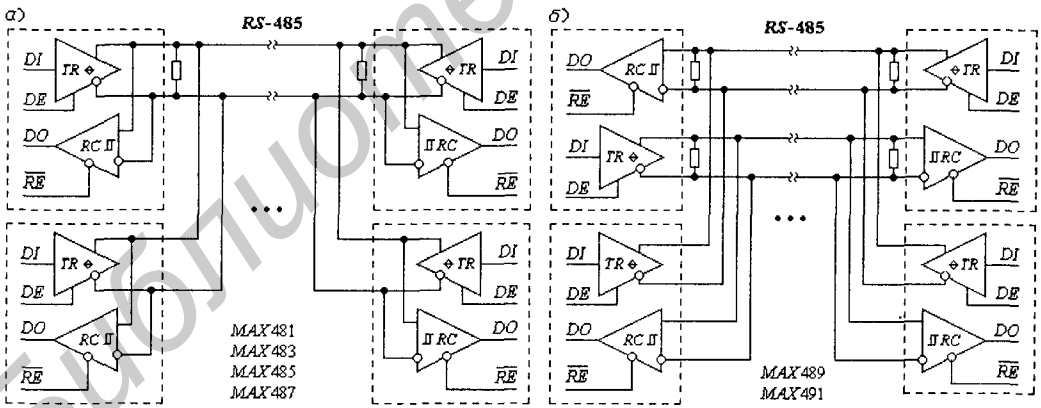


Рис. 3.107. Линии связи на основе интерфейса *RS*-485

На рис. 3.107, *б* показана структурная схема последовательного полного дуплексного канала связи на основе интерфейса *RS*-485. По этому каналу один приемопередатчик (левый верхний) может установить связь с любым другим приемопередатчиком. На рис. 3.107 указаны типы приемопередатчиков *MAX*××× фирмы *Maxim Integrated Products, Inc.*, которые можно использовать для построения полудуплексных и полных дуплексных каналов связи.

Передатчики и приемники интерфейса *RS*-232. Для преобразования уровней напряжений разработаны два типа интерфейсных ИС: передатчики (*TR* — *Transmitter*), преобразующие *TTL*-уровни напряжений в уровни стандарта *RS*-232, и приемники (*RC* — *Receiver*), преобра-

зующие уровни напряжений стандарта *RS*-232 в *TTL*-уровни. Некоторые из этих ИС приведены на рис. 3.108 (*SN* — префикс ИС фирмы *Texas Instruments*):

*SN*75150 (170АП2) — два стробируемых сигналом *E* передатчика (*E* — *Enable* — разрешение; *E* = 0 — передача запрещена, *E* = 1 — передача разрешена). Принципиальная схема одного передатчика типа *SN*75150 была приведена на рис. 3.101;

*SN*75154 (170УП2), *DS*75154 — четыре приемника с регулируемой напряжением шириной петли гистерезиса (*DS* — префикс ИС фирмы *National Semiconductor*). Контакт 15 — выход стабилизированного напряжения +5 В, *RI* — внутренний резистор 3 … 7 кОм. Источник питания V_{DD} и стабилизатор можно не использовать, подав на контакт 15 напряжение +5 В — нормальный режим работы. Принципиальная схема одного приемника типа *SN*75154 была приведена на рис. 3.102;

*SN*75188 (559ИП19), *MC*1488, *DS*1488, *SN*75*C*188, *DS*14*C*88 — четыре передатчика (буква *C* означает *CMOS*-технологию, обеспечивающую малое потребление мощности). Принципиальная схема одного передатчика типа *SN*75188 изображена на рис. 3.109, *а*;

*SN*75189, *SN*75189*A* (559ИП20), *MC*1489, *MC*1489*A*, *DS*1489 — четыре приемника с перемещаемой петлей гистерезиса (рис. 3.110) для оптимизации фильтрации помех. Принципиальная схема одного приемника типа *SN*75189*A* изображена на рис. 3.109, *б*. Выпускаются также ИС *SN*75*C*189 и *SN*75*C*189*A* функционально и по расположению контактов совместимые с ИС *SN*75189 и *SN*75189*A*;

*DS*14*C*89 — четыре приемника (*C* — *CMOS*-технология) с постоянной шириной петли гистерезиса;

*SN*75185, *SN*75*C*185 — три передатчика (контакты *D*×× — *Drivers* — передатчики) типа *SN*75188 и пять приемников (контакты *R*×× — *Receivers*) типа *SN*75188, но с не перемещаемой петлей гистерезиса, позволяющие осуществить простую взаимосвязь *UART INS*8250 фирмы *National Semiconductor* с разъемом последовательного порта *IBM PC/AT* (рис. 3.111). Типовые значения нижнего и верхнего порогов срабатывания равны 1,9 В и 0,97 В соответственно. ИС имеет защиту от разряда электростатического электричества (до 10000 В). Обеспечивается скорость передачи до 120 Кбит/с. Токи потребления:

$$I_{DD\,typ}/I_{DD\,max} = +9/+15 \text{ мА}, I_{EE\,typ}/I_{EE\,max} = -9/-15 \text{ мА}, I_{CC\,max} = +5{,}5 \text{ мА } (SN\ 75185);$$

$$I_{DD\,typ}/I_{DD\,max} = +115/+200 \text{ мкА}, I_{EE\,typ}/I_{EE\,max} = -115/-200 \text{ мкА}, I_{CC\,max} = +750 \text{ мкА } (SN\ 75C185).$$

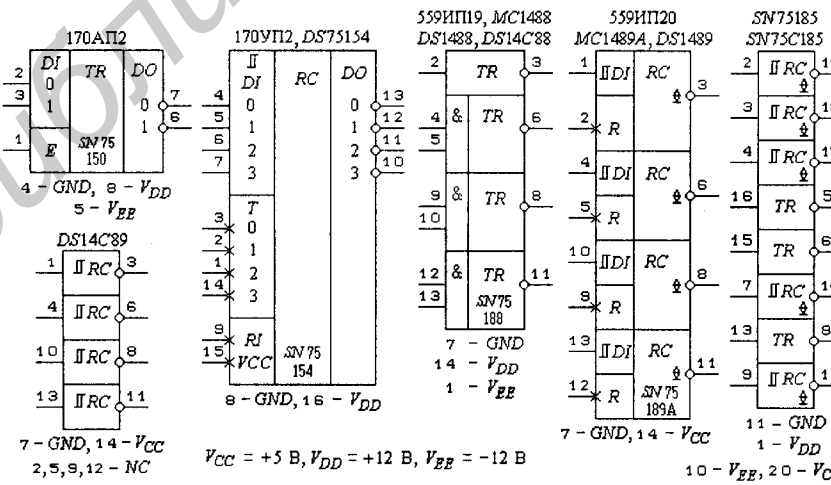


Рис. 3.108. Передатчики и приемники для интерфейса *RS*-232

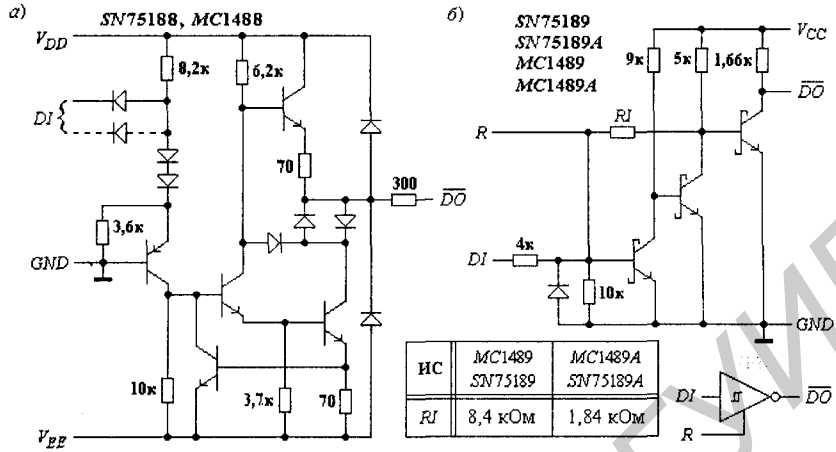


Рис. 3.109. Принципиальные схемы передатчика и приемника

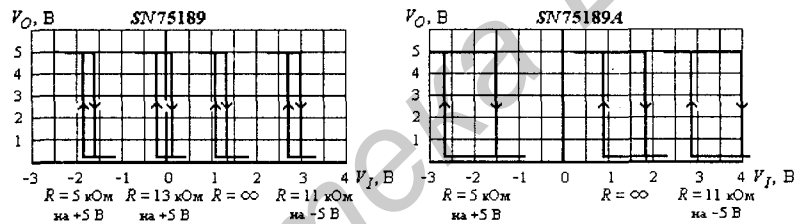


Рис. 3.110. Петли гистерезиса приемников

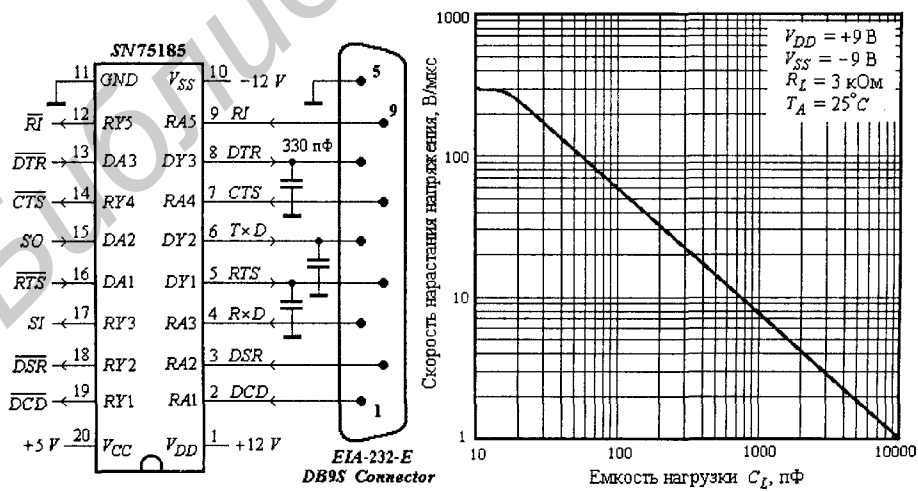


Рис. 3.111. Типовое подключение ИС *SN*75185 к разъему последовательного порта и график для выбора емкости нагрузки

На рис. 3.112 приведены ИС фирм *Maxim Integrated Products Inc.*, *Linear Technology*, *Newport* и *Motorola* (префиксы *MAX*, *LT*, *NM* и *MC*), содержащие как приемники *RC*, так и передатчики *TR* для интерфейса *RS*-232. Отличительной чертой этих ИС является использование только одного напряжения питания V_{CC} = +5 В и специальной схемы преобразования этого напряжения в напряжения +10 В и –10 В, необходимые для питания передатчиков. Преобразователь напряжения содержит автогенератор на 200 кГц и ключевые схемы удвоителя напряжения (*Voltage Double*) и инвертора напряжения (*Voltage Inverter*). Для накопления энергии и фильтрации напряжений +10 В и –10 В требуются внешние конденсаторы *C*1 ÷ *C*5 = 0,1 мкФ (*MAX*202/208/211/222/242), *C*1/*C*2/*C*5 = 4,7 мкФ и *C*3/*C*4 = 10 мкФ (*MAX*220), *C*1 ÷ *C*4 = 1 мкФ (*MAX*232/238/241). Передатчики обеспечивают уровни выходных сигналов в среднем ±7,5 В.

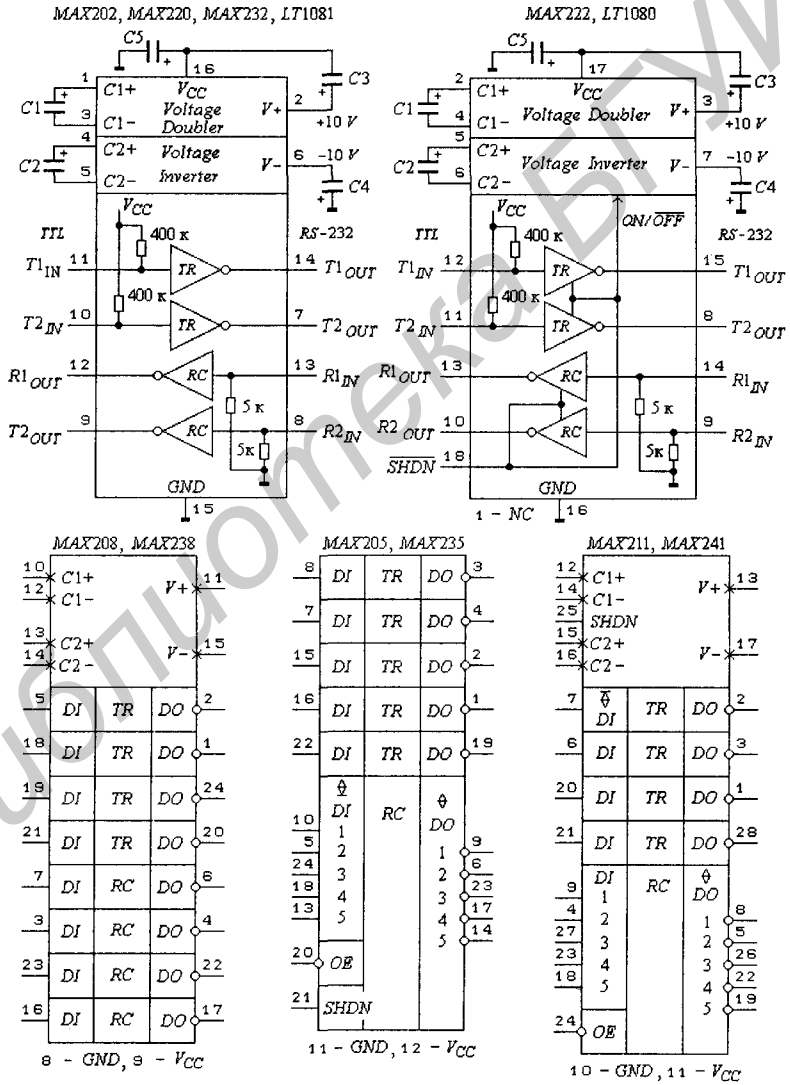


Рис. 3.112. Передатчики и приемники

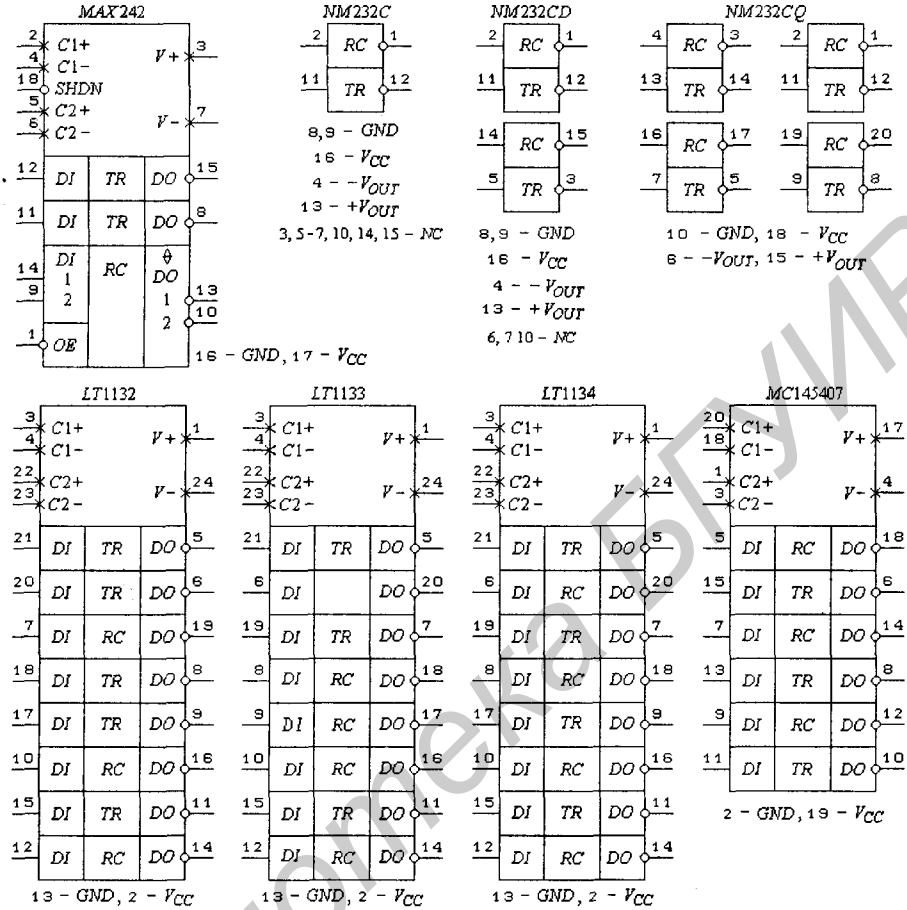


Рис. 3.112 (продолжение)

Краткая характеристика ИС, приведенных на рис. 3.112:

*MAX*202, *MAX*220, *MAX*232, *LT*1081 — два передатчика и два приемника. Скорость передачи равна 120 Кбит/с;

*MAX*222, *LT*1080 — два передатчика и два приемника, управляемые сигналом $\overline{SHDN}$ (*Shutdown* — отключение). Выходы передатчиков и приемников значением сигнала $\overline{SHDN} = 0$ переводятся в *Z*-состояние. Типовой ток потребления $I_{CC\ typ}$ в отключенном состоянии равен 0,1 мкА ($\overline{SHDN} = 0$), а в рабочем состоянии — 4 мА ($\overline{SHDN} = 1$). Скорость передачи для ИС *MAX*222 равна 200 Кбит/с;

*MAX*208, *MAX*238 — четыре передатчика и четыре приемника. Скорость передачи равна 120 Кбит/с;

*MAX*205, *MAX*235 — пять передатчиков и пять приемников без внешних конденсаторов (для приложений, в которых свободная площадь печатной платы ограничена — не нужно размещать внешние конденсаторы). ИС имеют сигналы управления *SHDN* (*Shutdown* — отключение) и $\overline{OE}$ (*Output Enable* — разрешение выходов приемников). Выходы передатчиков и приемников значением сигнала *SHDN* = 1 переводятся в *Z*-состояние. Выходы приемников переводятся в *Z*-состояние также значением сигнала $\overline{OE} = 1$. Типовой ток потребления $I_{CC\ typ}$

в отключенном состоянии равен 1 мкА (*SHDN* = 1), а в рабочем состоянии — 11 мА для *MAX*205 и 7 мА для *MAX*235 (*SHDN* = 0). Скорость передачи равна 120 Кбит/с;

*MAX*211, *MAX*241 — четыре передатчика и пять приемников. Сигналы управления *SHDN* и $\overline{OE}$ имеют то же самое назначение, что и в ИС *MAX*205/235. Типовой ток потребления $I_{CC\ typ}$ в отключенном состоянии равен 1 мкА (*SHDN* = 1), а в рабочем состоянии — 11 мА для *MAX*211 и 7 мА для *MAX*241 (*SHDN* = 0). Скорость передачи равна 120 Кбит/с;

*MAX*242 — два передатчика и два приемника, управляемые сигналами $\overline{SHDN}$ и $\overline{OE}$, имеющими то же самое назначение, что и в ИС *MAX*205/235. Токи потребления $I_{CC\ typ}$ такие же, что и в ИС *MAX*222. Скорость передачи для ИС *MAX*222 равна 200 Кбит/с;

*NM*232*C*, *NM*232*CD*, *NM*232*CQ* — интерфейсные модули, содержащие одну, две и четыре пары узлов передатчик-приемник (*Transmit-Receive Modules*). В корпусы модулей встроены и дискретные компоненты преобразователей напряжения +5 В (конденсаторы и др.). Уровни выходных сигналов передатчиков равны ±15 В, выходной ток приемников равен 30 мА. Токи потребления $I_{CC\ typ}$ = 14, 22 и 25 мА для соответствующих ИС. Габаритные размеры модулей 25,4 × 17 × 9 мм (+ 4 мм — длина выводов). Выходы $+V_{OUT}$ и $-V_{OUT}$ преобразованных напряжений обеспечивают токи 10 мА.

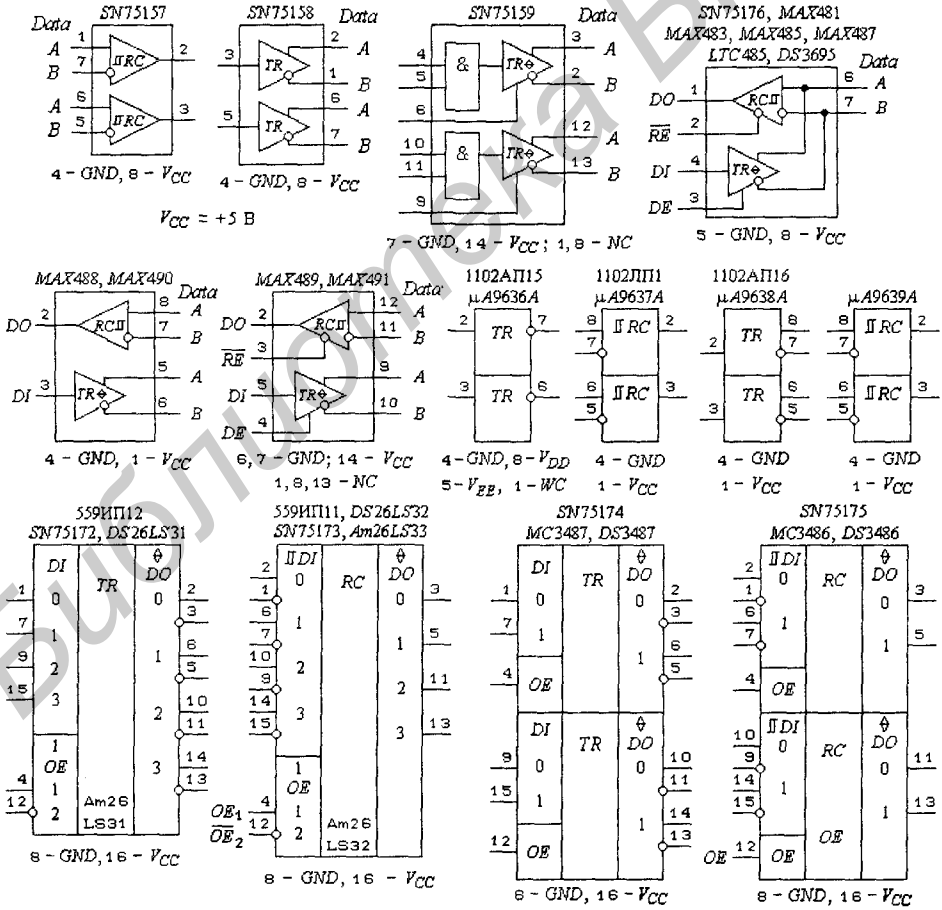


Рис. 3.113. Передатчики и приемники (интерфейсы *RS*-422*A* и *RS*-423*A*)

Передатчики и приемники для интерфейсов *RS*-422*A*, *RS*-423*A* и *RS*-485. Для преобразования *TTL*-уровней напряжений в уровни стандартов *RS*-422*A*, *RS*-423*A* и *RS*-485 разработано большое число интерфейсных ИС, в частности, ИС, представленных на рис. 3.113 (*Am* — префикс в обозначениях ИС фирмы *Advanced Micro Devices*):

*SN*75157 — два приемника для интерфейса *RS*-422*A*;

*SN*75158 — два передатчика для интерфейса *RS*-422*A*;

*SN*75159 — два передатчика для интерфейса *RS*-422*A* с *Z*-состоянием выходов;

*SN*75176, *MAX*481, *MAX*483, *MAX*485, *MAX*487, *LTC*485, *DS*26*LS*31 — приемопередатчик для полудуплексных (двунаправленных) каналов связи (интерфейсы *RS*-422*A* и *RS*-485). Передатчиком *TR* управляет сигнал *DE* (*Driver Enable*): значение *DE* = 0 переводит оба выхода передатчика в *Z*-состояние. Приемником *RC* управляет сигнал $\overline{RE}$ (*Receiver Enable*): значение $\overline{RE}$ = 1 переводит выход приемника *DO* в *Z*-состояние;

*MAX*488, *MAX*490 — приемник и передатчик для полного дуплексного канала связи (интерфейс *RS*-422*A*). Скорость передачи равна 250 кГц для *MAX*488 и 2,5 МГц для *MAX*490;

*MAX*489, *MAX*491 — приемник и передатчик для полного дуплексного канала связи (интерфейсы *RS*-422*A* и *RS*-485). Скорость передачи равна 250 кГц для *MAX*489 и 2,5 МГц для *MAX*491;

μ*A*9636*A*, 1102АП15 — два передатчика для интерфейсов *RS*-232 и *RS*-423*A*;

μ*A*9637*A*, 1102ЛП1 — два приемника для интерфейсов *RS*-422*A* и *RS*-423*A*;

μ*A*9638, 1102АП16 — два передатчика для интерфейсов *RS*-422*A*;

μ*A*9639 — два приемника для интерфейсов *RS*-422*A* и *RS*-423*A*.

*SN*75172, 559ИП12, *DS*26*LS*31 — четыре передатчика для интерфейсов *RS*-422*A* и *RS*-485;

*SN*75173, 559ИП11, *DS*26*LS*32, *Am*26*LS*33 — четыре приемника для интерфейсов *RS*-422*A* и *RS*-485. Максимально допустимое напряжение $V_{I\,max}$ на входе приемников типа '*LS*32 равно ±7 В, а для приемников типа '*LS*33 — ±15 В;

*SN*75174, *MC*3487, *DS*3487 — четыре передатчика для интерфейсов *RS*-422*A* и *RS*-485;

*SN*75175, *MC*3486, *DS*3486 — четыре приемника для интерфейсов *RS*-422*A* и *RS*-485.

На рис. 3.114 изображены петли гистерезиса некоторых приемников.

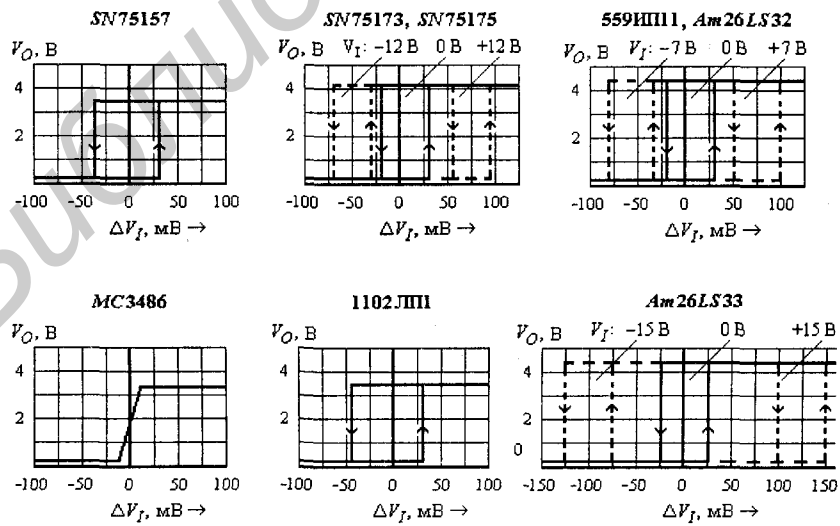


Рис. 3.114. Петли гистерезиса приемников (интерфейс *RS*-422*A*)

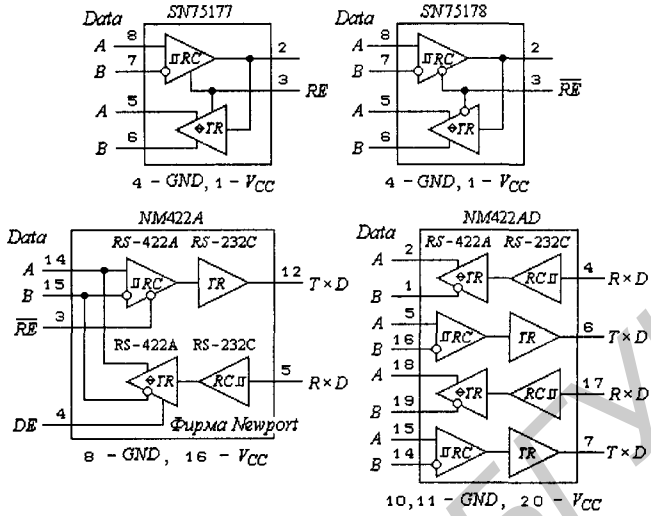


Рис. 3.115. Передатчики, приемники и трансляторы (интерфейсы *RS*-232, *RS*-422*A* и *RS*-423*A*)

На рис. 3.115 представлены ИС:

*SN*75177, *SN*75178 — повторители (*Repeater*), используемые для увеличения длины линий связи (включаются в промежуточных точках линии);

*NM*422*A* — модуль приемопередатчика (*Transceiver Module*) для сопряжения интерфейсов *RS*-232 и *RS*-422*A* (при одном напряжении питания +5 В обеспечиваются полноценные уровни выходных сигналов интерфейса *RS*-232); габаритные размеры этого модуля 25,4 × 17 × 9 мм (+4 мм — длина выводов).

3.9. Программируемые БИС поддержки МП 8085

Для построения небольших МП-систем на основе МП 8085 были специально разработаны БИС поддержки этого МП: 8155*H* (1821РУ55), 8156*H*, 8755*A* (1821РФ55), 8355 (1821РЕ55) и 8185. Данные БИС могут быть использованы и в МП-системах, построенных на основе МП 8088. Назначение контактов этих БИС представлено на рис. 3.116. Характерной особенностью каждой БИС является то, что она содержит 10-разрядный регистр, фиксирующий значение адресной информации (A_{7-0}, $\overline{CE}$/*CE* и *IO*/$\overline{M}$) по спадающему фронту сигнала *ALE*. Это позволяет исключить внешний адресный регистр памяти, изображенный на рис. 1.8, *б*, а значит и разделение сигналов мультиплексной шины адреса-данных AD_{7-0} на шины адреса A_{7-0} и данных D_{7-0}. Основные параметры БИС поддержки МП 8085 приведены в табл. 3.19 (рассеиваемая корпусами мощность P_{DIS} = 1,5 Вт).

Структурная схема БИС 8155*H*/8156*H*. Данные БИС содержит ОЗУ 256 × 8 бит, три порта ввода-вывода, 14-разрядный таймер и изготовляются по *HMOS* технологии. ОЗУ имеет максимальное время доступа 400 нс и используется без режима ожидания с МП 8085*AH*. Более быстродействующий вариант БИС 8155*H*-2 имеет время доступа к ОЗУ 330 нс и предназначен для использования с МП 8085*AH*-2 и 8088, имеющими тактовую частоту 5 МГц. Условное графическое обозначение БИС 8155*H* приведено на рис. 3.117.

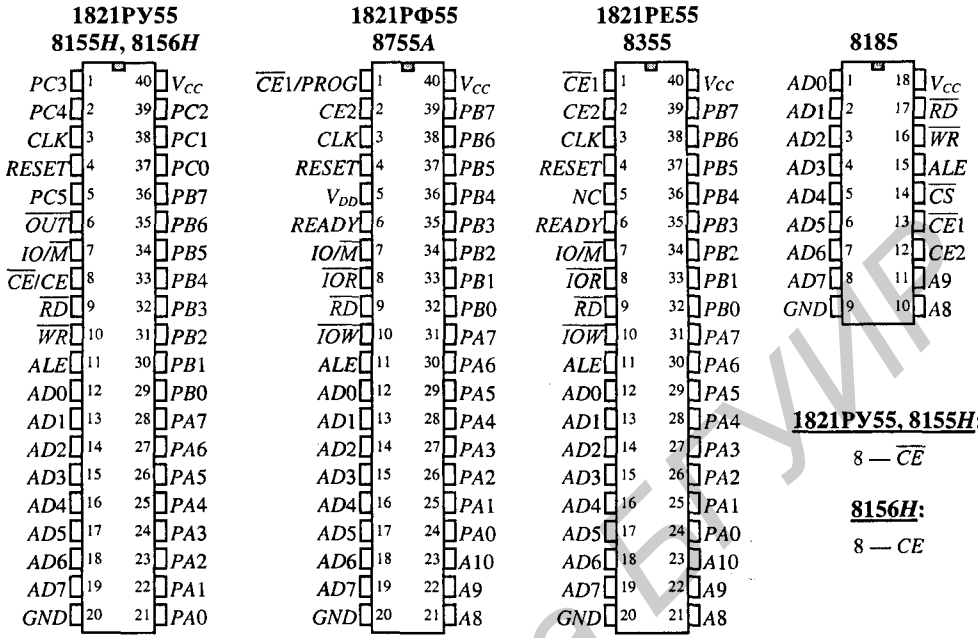


Рис. 3.116. БИС поддержки МП 8085*A*

Таблица 3.19. **Основные параметры БИС поддержки МП 8085*A***

БИС	Аналог	$I_{CC\ max}$, мА	V_{CC}, В	$I_{DD\ typ}/V_{DD\ typ}$, мА/В	$V_{OH\ min}/I_{OH}$, В/мА	$V_{OL\ max}/I_{OL}$, В/мА	F_{CLK}, МГц
8155*H*	1821РУ55	125	5 ± 10%	—	2,4/–0,4	0,45/2,0	4,1
8155*H*-2	—	125	5 ± 10%	—	2,4/–0,4	0,45/2,0	7,1
8156*H*	—	125	5 ± 10%	—	2,4/–0,4	0,45/2,0	4,1
8156*H*-2	—	125	5 ± 10%	—	2,4/–0,4	0,45/2,0	7,1
8755*A*	1821РФ55	180	5 ± 5%	15/25	2,4/–0,4	0,45/2,0	3,1
8355	1821РЕ55	180	5 ± 5%	—	2,4/–0,4	0,45/2,0	3,1
8185, 8185-2	—	100	5 ± 10%	—	2,4/–0,4	0,45/2,0	—

БИС 8156*H* отличается от БИС 8155*H* только полярностью сигнала выбора (разрешения) кристалла ($\overline{CE}$ — для 8155*H* и *CE* — для 8156*H*). Эти БИС содержат узлы (рис. 3.118):

Control — схема управления чтением и записью данных в *RAM* и внутренние регистры БИС;

Address Latch — 10-разрядный регистр, фиксирующий значения сигналов A_{7-0}, $\overline{CE}/CE$ и $IO/\overline{M}$ по спадающему фронту сигнала *ALE*. Зафиксированные значения сигналов A_{2-0}, $\overline{CE}/CE$ и $IO/\overline{M}$ подаются на схему управления, а значения сигналов A_{7-0} — на адресные входы *RAM*;

256 × 8 *Static RAM* — статическое ОЗУ 256 × 8 бит;

Timer — 14-разрядный таймер (программируемый вычитающий двоичный счетчик). Таймер содержит два 8-разрядных регистра для хранения 14-разрядного модуля пересчета и режима работы. В зависимости от заданного режима работы таймер может формировать выходной сигнал $\overline{OUT}$ как в виде меандра (с равными или почти равными полупериодами), так и в виде

импульса при переходе счетчика в нулевое состояние (в конце счета), длительность нулевого значения которого равна одному периоду сигнала *CLK*;

RG CW/SW — 8-разрядный регистр слова управления и 7-разрядный регистр слова состояния;

RG A — два 8-разрядных регистра (ввода и вывода) порта ввода-вывода *PA*;

RG B — два 8-разрядных регистра (ввода и вывода) порта ввода-вывода *PB*;

RG C — 6-разрядный регистр вывода порта ввода-вывода *PC*, возможности которого переключаются на обслуживание портов *PA* и *PB* в режиме их работы с квитированием и по прерыванию.

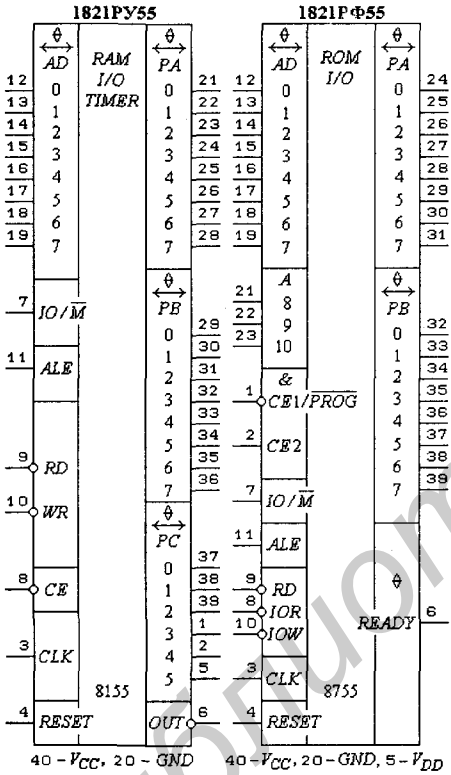


Рис. 3.117. БИС 8155*H* и 8755*A*

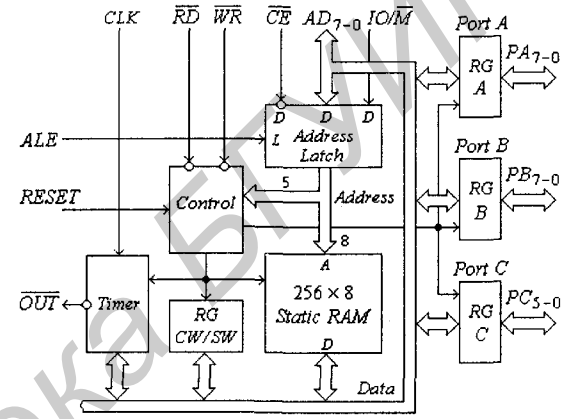


Рис. 3.118. Структурная схема БИС 8155*H*

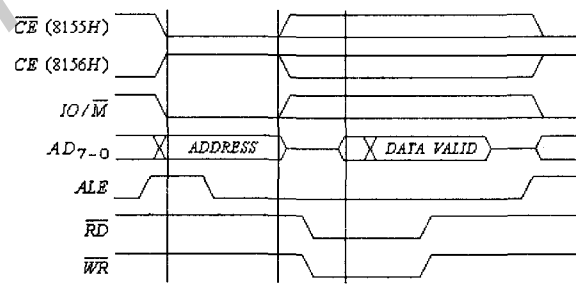


Рис. 3.119. Временны́е диаграммы чтения и записи данных

Сигналы БИС 8155*H*/8156*H*. Сигналы системных шин адреса-данных и управления этих БИС имеют назначение:

AD_{7-0} (*Address/Data*) — мультиплексная шина адреса-данных МП с *Z*-состоянием выходов. Адрес A_{7-0} фиксируется во внутреннем регистре БИС сигналом *ALE* = 1 (адрес *RAM* или внешних устройств *I/O* в зависимости от значения сигнала $IO/\overline{M}$);

$\overline{CE}$ (*Chip Enable*) — сигнал, поступающий с дешифратора адресных сигналов A_{15-11} и позволяющий селектировать необходимое число однотипных БИС (прямой сигнал *CE* для БИС 8156*H*);

ALE (*Address Latch Enable*) — сигнал фиксации по спадающему фронту в 10-разрядном регистре значений сигналов A_{7-0}, $\overline{CE}$ (8155*H*) или *CE* (8156*H*) и $IO/\overline{M}$;

$\overline{RD}$, $\overline{WR}$ (*Read, Write*) — сигналы чтения и записи данных во внутренние регистры памяти и *RAM*;

IO/$\overline{M}$ (*I/O or Memory*) — сигнал, указывающий на обращение к любому из регистров (регистры слов управления и состояния, регистры таймера и портов *PA*, *PB* и *PC*) или к памяти БИС. Значение сигнала *IO*/$\overline{M}$ = 1 при обращении к внешним устройствам и значение *IO*/$\overline{M}$ = 0 при обращении к памяти;

PA_{7-0} и PB_{7-0} — 8-разрядные порты ввода-вывода данных, направление передачи и метод ввода-вывода которых программируются записью слова управления *CW* (*Command Word*);

PC_{5-0} — 6-разрядный порт ввода-вывода данных. Эти шесть линий используются или как порт ввода-вывода, или как управляющие сигналы для портов *PA* и *PB* при задании их работы в режиме ввода-вывода с квитированием и по прерыванию (аналогично тому, как это сделано в БИС 8255 — см. § 3.2);

CLK (*Clock*) — тактовый сигнал таймера (обычно используется выходной сигнал *CLK* МП 8085);

$\overline{OUT}$ (*Timer* $\overline{OUT}$) — выходной сигнал таймера (меандр или импульсный сигнал в зависимости от запрограммированного режима работы);

RESET — сигнал установки начального состояния (обычно на этот вход подается сигнал *RESET OUT* с выхода МП 8085*A*). Значение *RESET* = 1 останавливает счет таймера, производит сброс в 0 регистров вывода портов данных *PA* и *PB* и устанавливает все порты ввода-вывода в режим ввода без квитирования. Типовая длительность активного уровня этого сигнала равна двум тактам *CLK*.

Временны́е диаграммы чтения и записи данных в память для БИС 8155*H*/8156*H* изображены на рис. 3.119 (для ввода и вывода данных из регистров следует изменить только уровень сигнала управления *IO*/$\overline{M}$ во время действия значения сигнала *ALE* = 1).

Управление вводом-выводом БИС 8155*H*/8156*H*. Программная модель регистров ввода и вывода БИС 8155*H*/8156*H* показана на рис. 3.120. Она содержит регистр команды (*Command Register*, или *RGCW* — регистр слова управления), регистр состояния (*Status Register*, или *RGSW* — регистр слова состояния), регистры портов ввода-вывода *PA*, *PB* и *PC* и регистры таймера (*Timer MSB* и *Timer LSB*; *MSB* — *Most Significant Byte* — старший байт и *LSB* — *Least Significant Byte* — младший байт).

Управление вводом-выводом БИС приведено в табл. 3.20 — используется по шесть адресов портов для ввода данных и для вывода данных.

Регистры слов управления *CW* и состояния *SW* имеют один и тот же адрес ×××××000. В регистр *RGCW* записывается слово управления *CW*, задающее режимы работы БИС. Содержимое этого регистра для чтения недоступно. Из регистра *RGSW* производится чтение слова состояния *SW*, в котором фиксируется текущее состояние БИС.

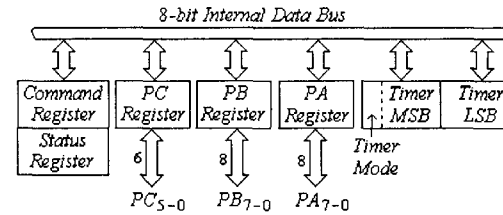


Рис. 3.120. Внутренние регистры БИС 8155*H*

Слово управления БИС 8155*H*/8156*H*. Четыре разряда AD_{3-0} регистра *RGCW* (рис. 3.121) определяют режимы работы портов ввода-вывода *PA*, *PB* и *PC*, разряды AD_5 и AD_4 управляют разрешением/запретом ввода-вывода по прерыванию для портов *PA* и *PB*, а разряды AD_7 и AD_6 задают режимы работы таймера.

В табл. 3.21 приведено назначение линий порта *PC*, программируемое разрядами AD_3 и AD_2, а в табл. 3.22 — значения сигналов управления, которые устанавливаются при записи слова *CW*, задающего режимы работы портов *PA* и *PB* с квитированием и по прерыванию.

Таблица 3.20. **Управление вводом-выводом БИС 8155*H*/8156*H***

$\overline{CE}$	A_2	A_1	A_0	$\overline{WR}$	$\overline{RD}$	Операция	Примечание
0	0	0	0	0	1	$AD_{7-0} \rightarrow RGCW$	Запись слова управления в *RGCW*
0	0	0	1	0	1	$AD_{7-0} \rightarrow PA$	Запись данных в регистр вывода порта *PA*
0	0	1	0	0	1	$AD_{7-0} \rightarrow PB$	Запись данных в регистр вывода порта *PB*
0	0	1	1	0	1	$AD_{7-0} \rightarrow PC$	Запись данных в регистр вывода порта *PC*
0	1	0	0	0	1	$AD_{7-0} \rightarrow Counter$	Запись младшего байта модуля пересчета
0	1	0	1	0	1	$AD_{7-0} \rightarrow Counter$	Запись 6 старших разрядов модуля пересчета и 2-разрядного кода режима работы таймера
0	0	0	0	1	0	$AD_{7-0} \leftarrow RGSW$	Чтение слова состояния из *RGSW*
0	0	0	1	1	0	$AD_{7-0} \leftarrow PA$	Чтение данных из порта *PA*
0	0	1	0	1	0	$AD_{7-0} \leftarrow PB$	Чтение данных из порта *PB*
0	0	1	1	1	0	$AD_{7-0} \leftarrow PC$	Чтение данных из порта *PC*
0	1	0	0	1	0	$AD_{7-0} \leftarrow Counter$	Чтение младшего байта состояния счетчика
0	1	0	1	1	0	$AD_{7-0} \leftarrow Counter$	Чтение 6 старших разрядов состояния счетчика
0	1	1	×	×	×	*Z*-состояние AD_{7-0}	Нет операций
0	×	×	×	1	1	*Z*-состояние AD_{7-0}	Нет операций
1	×	×	×	×	×	*Z*-состояние AD_{7-0}	Нет операций

Примечание: $IO/\overline{M} = 1$ при раздельных пространствах адресов памяти и внешних устройств, $IO/\overline{M} = 0$ при отображении внешних устройств на память (см. § 2.1).

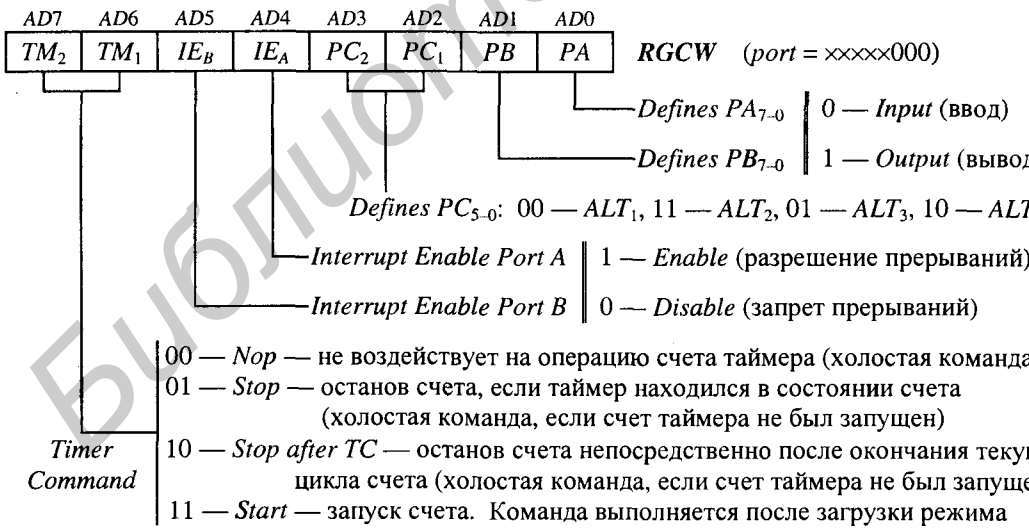


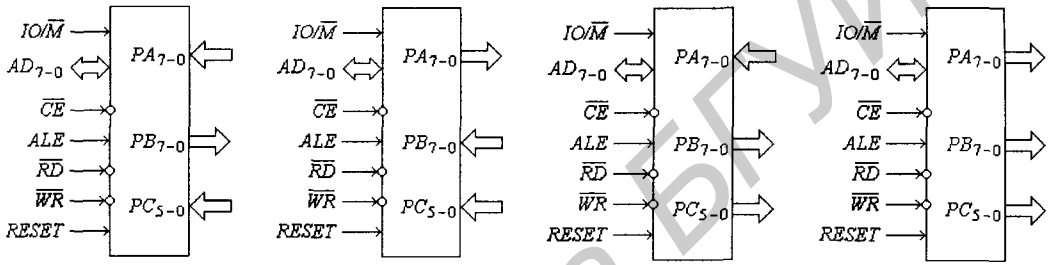
Рис. 3.121. Формат слова управления *CW*

Таблица 3.21. **Назначение линий порта *PC***

Pin	ALT_1	ALT_2	ALT_3	ALT_4
PC_0	*Input Port*	*Output Port*	$INTR_A$	$INTR_A$
PC_1	*Input Port*	*Output Port*	BF_A	BF_A
PC_2	*Input Port*	*Output Port*	$\overline{STB}_A$	$\overline{STB}_A$
PC_3	*Input Port*	*Output Port*	*Output Port*	$INTR_B$
PC_4	*Input Port*	*Output Port*	*Output Port*	BF_B
PC_5	*Input Port*	*Output Port*	*Output Port*	$\overline{STB}_B$

Таблица 3.22. **Инициализация сигналов управления в режимах ALT_3 и ALT_4**

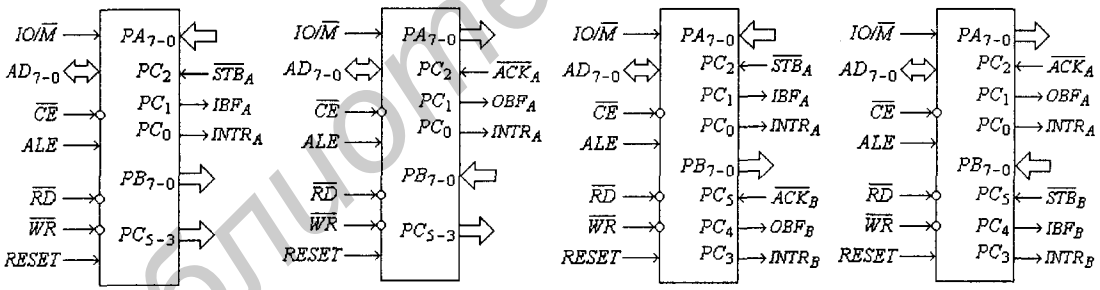
Control	*Input Mode*	*Output Mode*
BF	0	0
INTR	0	1
$\overline{STB}/\overline{ACK}$	*Input Control*	*Input Control*



Режим ALT_1 Режим ALT_2

$CW = 0000\ 0010 = 02h$ $CW = 0000\ 0001 = 01h$ $CW = 0000\ 1110 = 0Eh$ $CW = 0000\ 1111 = 0Fh$

Рис. 3.122. Конфигурации портов ввода-вывода в режимах ALT_1 и ALT_2



Режим ALT_3 Режим ALT_4

$CW = 0001\ 0110 = 16h$ $CW = 0001\ 0101 = 15h$ $CW = 0011\ 1010 = 3Ah$ $CW = 0011\ 1001 = 39h$

Рис. 3.123. Конфигурации портов ввода-вывода в режимах ALT_3 и ALT_4

Сигналы управления имеют такое же назначение, что и в БИС 8255*A* (см. § 3.2):

$INTR_A$ (*Port A Interrupt*), $INTR_B$ (*Port B Interrupt*) — сигналы запросов прерываний, подаваемые на входы *RST* ×.5 МП 8085;

BF_A (*Port A Buffer Full*), BF_B (*Port B Buffer Full*) — сигналы для программного ввода-вывода с квитированием (IBF_A, IBF_B — для режима ввода, OBF_A, OBF_B — для режима вывода);

$\overline{STB}_A$ (*Port A Strobe*), $\overline{STB}_B$ (*Port B Strobe*) — сигналы, поступающие от внешних устройств ($\overline{STB}_A$, $\overline{STB}_B$ — для режима ввода, $\overline{ACK}_A$, $\overline{ACK}_B$ — для режима вывода).

На рис. 3.122 изображены некоторые конфигурации портов ввода-вывода в режимах ALT_1 и ALT_2, а на рис. 3.123 — несколько конфигураций портов ввода-вывода в режимах ALT_3 и ALT_4, которые могут быть заданы словом управления *CW*, записываемым в регистр *RGCW*.

Задача 1. Запрограммировать работу порта *PA* для ввода с квитированием и по прерыванию и порта *PB* для вывода без квитирования. *Решение*:

```
MVI   A, 16h     ; CW = 0001 0110 = 16h, IE_A = AD_4 = 1 — прерывания разрешены
OUT   p_RGCW     ; p_RGCW — имя адреса регистра RGCW (p_RGCW = ×××××000)
```

Временны́е диаграммы ввода и вывода с квитированием и по прерыванию показаны на рис. 3.124 (PD_{7-0} — порты данных PA_{7-0} и PB_{7-0}, * — не более нс, остальные — не менее нс). Сигналы запросов прерывания $INTR_A$ и $INTR_B$ генерируются только в том случае, если прерывания разрешены, т. е. если в слове управления *CW* заданы значения $IE_A = 1$ и $IE_B = 1$.

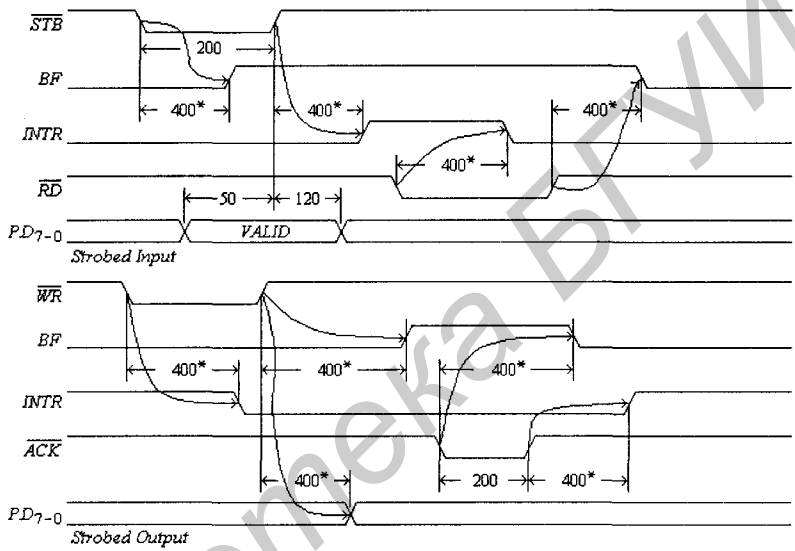


Рис. 3.124. Временны́е диаграммы для БИС 8155*H*/8156*H*

Функционирование портов ввода-вывода БИС 8155*H*/8156*H*. 8-разрядные порты ввода-вывода *PA* и *PB* имеют адреса ×××××001 и ×××××010. Они могут программироваться словом управления *CW* на программный ввод или вывод без квитирования или на программный ввод или вывод с квитированием и по прерыванию, как и одноименные порты в БИС 8255*A* (режимы работы M_0 и M_1 — см. § 3.2). На рис. 3.125 изображена структурная схема одного разряда данных портов ввода-вывода *PA* и *PB*, поясняющая управление их режимами работы:

1) в режиме вывода данные, поступающие из МП, всегда записываются в регистры вывода портов (положение 1 переключателя *MUX*). Эти регистры выполнены по схеме с обратным чтением, т. е. выведенные в них данные в любой момент можно прочитать командой IN *port*, например, для контроля операций вывода;

2) в режиме ввода без квитирования входные данные портов в регистрах ввода не фиксируются (положение 2 переключателя *MUX*);

3) в режиме ввода с квитированием входные данные портов фиксируются в регистрах ввода (положение 3 переключателя *MUX*);

4) значения сигнала сброса регистров вывода портов в нулевое состояние $\overline{R} = Mode = 1$ в режиме вывода и $\overline{R} = 0$ в режиме ввода (активное значение). Этим обеспечиваются значения 0 (детерминированные значения) на выходах портов вывода при перепрограммировании их режимов работы с ввода на вывод.

Сигналы чтения *Rd* и записи *Wr* данных в порты *PA* и *PB* описываются выражениями:

$Rd = IO/\overline{M} \cdot RD \cdot CE \cdot \overline{A}_2\overline{A}_1A_0$ — для порта *PA*,

$Wr = IO/\overline{M} \cdot WR \cdot CE \cdot \overline{A}_2\overline{A}_1A_0$ — для порта *PA*,

$Rd = IO/\overline{M} \cdot RD \cdot CE \cdot \overline{A}_2A_1\overline{A}_0$ — для порта *PB*,

$Wr = IO/\overline{M} \cdot WR \cdot CE \cdot \overline{A}_2A_1\overline{A}_0$ — для порта *PB*

(при задании режима ввода запись в регистр вывода блокирована значением сигнала *Mode* = 0).

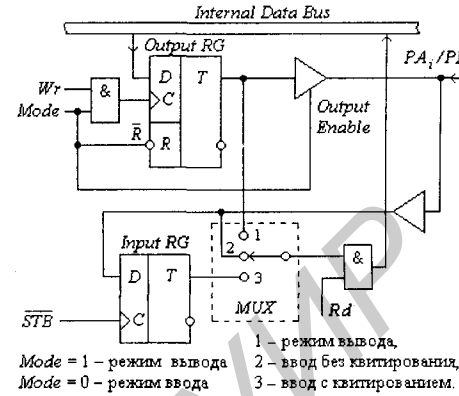


Рис. 3.125. Схема управления режимами работы

Регистр порта *PC* имеет адрес ××××x011 и содержит только 6 разрядов. Разряды AD_3 и AD_2 слова управления *CW* программируют этот порт на работу в режиме ввода и вывода или на управление работой портов *PA* и *PB* (см. рис. 3.121 и табл. 3.21). Когда порт *PC* используется для управления работой портов *PA* и *PB*, то по три его разряда предназначаются для управления каждым из этих портов:

$PC_0 = INTR_A$ ($PC_3 = INTR_B$) — сигнал запроса прерывания, подаваемый на вход *RST* ×.5 МП 8085;

$PC_1 = \overline{BF_A}$ ($PC_4 = \overline{BF_B}$) — сигнал квитирования, поступающий на внешнее устройство;

$PC_2 = \overline{STB_A}$ ($PC_5 = \overline{STB_B}$) — сигнал, поступающий от внешнего устройства.

В режиме ввода флаг BF_A (BF_B) является флагом IBF_A (IBF_B), а в режиме вывода — флагом OBF_A (OBF_B). В режиме ввода сигнал управления $\overline{STB_A}$ ($\overline{STB_B}$) является сигналом записи в регистр ввода данных, поступающих от внешнего устройства, а в режиме вывода — сигналом подтверждения (*Acknowledge*) приема данных внешним устройством $\overline{ACK_A}$ ($\overline{ACK_B}$). Сигналы $\overline{STB_A}$ ($\overline{STB_B}$) и $\overline{ACK_A}$ ($\overline{ACK_B}$) управляют флагами BF_A (BF_B) и сигналами запроса прерывания $INTR_A$ ($INTR_B$) в соответствии с рис. 3.124.

Слово состояния БИС 8155*H*/8156*H*. Формат слова состояния *SW* показан на рис. 3.126. Регистр состояния *RGSW* содержит информацию о состоянии портов ввода-вывода *PA* и *PB* (разряды $AD_{5\text{-}0}$) и о состоянии таймера (разряд AD_6). В разрядах $AD_{5\text{-}0}$ дублируются значения выходных сигналов управления BF_A и BF_B, $INTR_A$ и $INTR_B$ и двух разрядов разрешения прерываний $INTE_A$ и $INTE_B$, значения которых задаются словом управления *CW*.

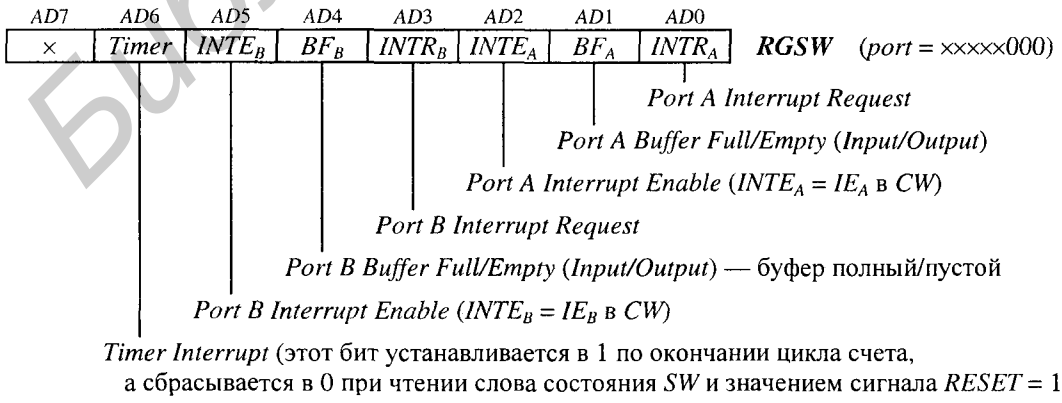


Рис. 3.126. *Формат слова состояния SW*

Задача 2. Запрограммировать работу порта *PA* для вывода с квитированием и порта *PB* для ввода без квитирования. Адрес регистра слова управления *RGCW* принять равным 08*h*. Вывести 16 байт данных из ячеек памяти, имеющих начальный адрес 00A0*h*, в порт *PA*. *Решение*:

```
      MVI   A, 05h    ; A ← CW = 05h = 0000 0101 (см. рис. 3.121 и 3.123)
      OUT   8         ; RGCW ← CW — запись слова управления
      MVI   B, 10h    ; B ← 10h = 16d — число выводимых байт данных
      LXI   H, 0A0h   ; rp H ← 00A0h — начальный адрес памяти
L1:   IN    8         ; A ← SW — чтение слова состояния
      ANI   2         ; 2 = 0000 0010 — маска для выделения флага OBF_A
      JZ    L1
      MOV   A, M      ; A ← M(rp H) — пересылка в аккумулятор байта данных из памяти
      OUT   9         ; PA_7-0 ← A, 09h = 0000 1001 — адрес порта PA_7-0
      INX   H         ;   (в общем виде адрес порта PA  равен ×××××001 — см. табл. 3.20)
      DCR   B
      JNZ   L1
```

Адресация БИС 8155*H*/8156*H*. При выполнении команд IN *port* и OUT *port* микропроцессор 8085*A* в машинном цикле передачи данных между аккумулятором *A* и внешним устройством *I/O* выдает адрес A_{15-0}, состоящий из двух одинаковых байтов (см. рис. 1.9):

$$A_7A_6A_5A_4A_3A_2A_1A_0 = A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8 = port \quad (A_i = A_{i+8},\ i = 0, 1, \ldots, 7).$$

Разряды адреса $A_2A_1A_0$ в БИС 8155*H*/8156*H* используются для адресации внутренних регистров, поэтому разряды адреса $A_{10}A_9A_8$ не могут участвовать в селектировании БИС по входу $\overline{CE}$. Дешифратор *DC* 5 × 32 пяти адресных сигналов $A_{15}A_{14}A_{13}A_{12}A_{11}$, вырабатывающий сигналы $\overline{CE}_{31} \ldots \overline{CE}_0$, позволяет подключить к МП не более 32 штук БИС 8155*H* (все адресное пространство памяти и ввода-вывода МП исчерпано). В *задаче* 2 для регистра слова управления *RGCW* задан адрес $08h = \overline{A}_{15}\overline{A}_{14}\overline{A}_{13}\overline{A}_{12}A_{11}\overline{A}_2A_1\overline{A}_0 = 00001000$, поэтому для используемой в этой задаче БИС сигнал $CE_1 = \overline{A}_{15}\overline{A}_{14}\overline{A}_{13}\overline{A}_{12}A_{11}$ (соотношение записано для прямого сигнала выбора кристалла).

БИС 8755*A* (ее писание приведено ниже) содержит *EPROM* 2048 × 8 бит, адресуемое разрядами A_{10-0} и предназначенное для хранения программного обеспечения МП 8085*A*. Использование этой БИС компенсирует потери адресного пространства памяти из-за невозможности использования разрядов адреса $A_{10}A_9A_8$ для селектирования БИС 8155*H*/8156*H*. Суммарное число БИС 8155*H*/8156*H* и 8755*A*, селектируемых сигналами $\overline{CE}_{31} \ldots \overline{CE}_0$ разрешения кристалла, не более 32. Потери адресного пространства памяти в принципе невелики, так как специализированная МП-система всегда содержит ПЗУ значительно большего объема, чем ОЗУ.

Выпускаются также БИС 8185/8185-2 (их писание приведено ниже), содержащие только статическое ОЗУ 1024 × 8 бит. Эти БИС позволяют минимизировать аппаратные затраты при необходимости значительного увеличения объема памяти ОЗУ.

Минимальную МП-систему можно построить всего на трех БИС: 8085*A*, 8156*H* и 8755*A*. На рис. 3.127 изображена такая МП-система с использованием отображения адресного пространства ввода-вывода на адресное пространство памяти (в программном обеспечении команды IN *port* и OUT *port* использовать нельзя, а значит, не нужен и выходной сигнал МП $IO/\overline{M}$).

Адресация памяти и устройств ввода-вывода для минимальной МП-системы приведена в табл. 3.23. Разряды адреса, значения которых подчеркнуты ($\underline{0}$), в адресации не участвуют, а значит, могут иметь значения и 0, и 1, но для вычисления адресов удобнее использовать значения 0.

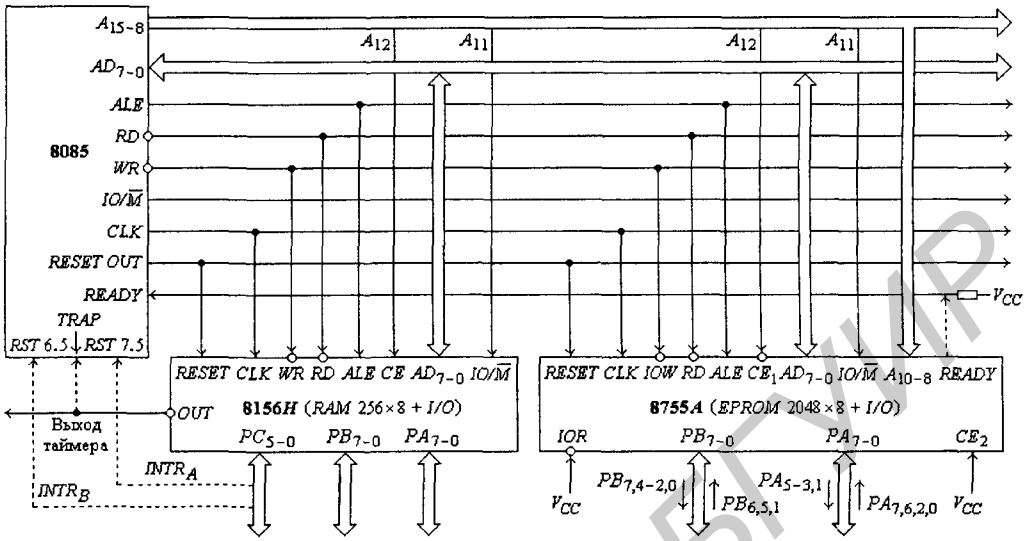


Рис. 3.127. Минимальная конфигурация МП-системы с отображением адресного пространства ввода-вывода на адресное пространство памяти

Таблица 3.23. **Адресация памяти и портов ввода-вывода**

Разряды адреса														Адреса		БИС
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	*Memory*	*I/O*	
0	0	0	0	0	×	×	×	×	×	×	×	×	×	×	×	0000 ÷ 07FF*h*	—	8755*A*
0	0	0	0	1	0	0	0	0	0	0	0	0	0	×	×	—	0800 ÷ 0803*h*	
0	0	0	1	0	0	0	0	×	×	×	×	×	×	×	×	1000 ÷ 10FF*h*	—	8156*H*
0	0	0	1	1	0	0	0	0	0	0	0	0	×	×	×	—	1800 ÷ 1805*h*	
Примечание: × — значения 0 и 1.																		

Если эти три БИС расположить на одном кристалле, то получится однокристальный микроконтроллер типа 8051, но с другой системой команд и несколько иными функциональными возможностями.

Таймер БИС 8155*H*/8156*H*. Таймер представляет собой 14-разрядный вычитающий счетчик с параллельной загрузкой данных, состояния которого изменяются тактовым сигналом *CLK*. Таймер имеет два 8-разрядных регистра для хранения 14-разрядного числа, задающего модуль пересчета, и 2-разрядного кода, задающего режимы работы таймера (рис. 3.128). Регистр хранения младшего байта модуля пересчета T_{7-0} имеет адрес ××××100, а регистр хранения 6 старших разрядов модуля пересчета T_{13-8} и двух бит M_2M_1 режима работы таймера — адрес ××××101.

В режимах *Mode* 0 и *Mode* 1 длительности значений 0 и 1 выходного сигнала таймера $\overline{OUT}$ одинаковы при задании четного модуля пересчета и длительность значения 1 на один такт больше длительности значения 0 при задании нечетного модуля пересчета. В режимах *Mode* 2 и *Mode* 3 длительность значения 0 выходного сигнала таймера $\overline{OUT}$ равна одному периоду его тактового сигнала *CLK*. Это значение сигнала $\overline{OUT}$ выдается в нулевом состоянии счетчика. В режимах *Mode* 1 и *Mode* 3 производится непрерывный счет с автозагрузкой заданного модуля пересчета, а в режимах *Mode* 0 и *Mode* 2 — одиночные циклы счета.

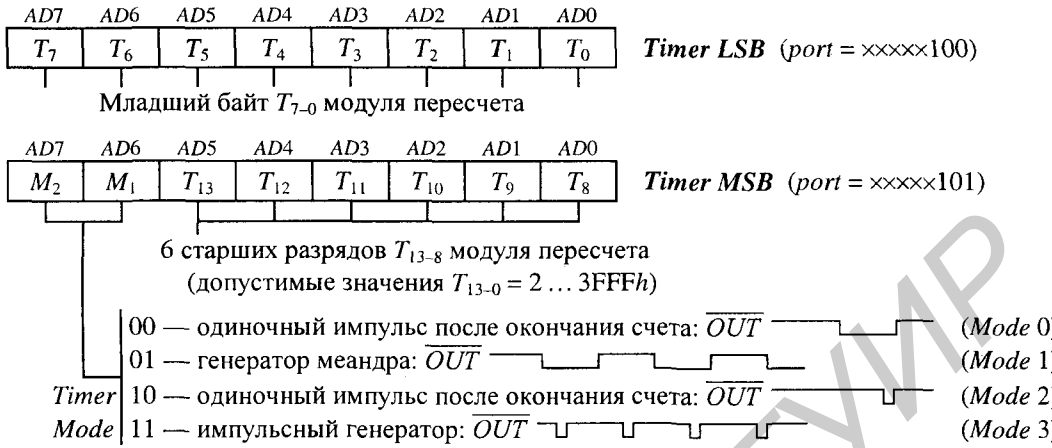


Рис. 3.128. Регистры памяти таймера

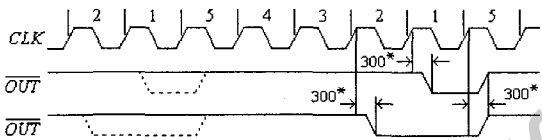


Рис. 3.129. Временны́е диаграммы таймера

Временны́е диаграммы таймера представлены на рис. 3.129 при задании модуля пересчета, равного пяти (без штриховых линий — для режимов *Mode* 2 и *Mode* 0, со штриховыми линиями — для режимов *Mode* 3 и *Mode* 1; * — не более нс).

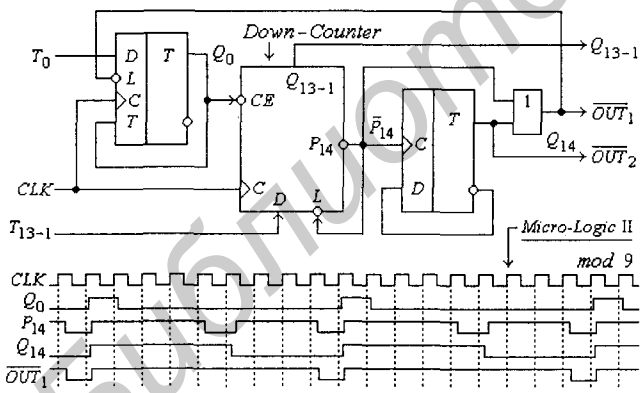


Рис. 3.130. Структурная схема таймера

Схема таймера была специально разработана для использования в качестве делителя частоты с равными при четном модуле пересчета M и примерно равными при нечетном модуле пересчета M полупериодами его выходного сигнала $\overline{OUT}$. Структурная схема таймера изображена на рис. 3.130 (описание принципа работы *D*–*T*–*L*-триггеров и программирование модулей пересчета вычитающих счетчиков см. в книге [5]). В зависимости от запрограммированного режима работы на выход таймера $\overline{OUT}$ выдается сигнал $\overline{OUT}_1$ или сигнал $Q_{14} = \overline{OUT}_2$.

Модуль M пересчета задается числом $M = T_{13}T_{12} \ldots T_2T_1T_0$. В каждом цикле счета по *mod M* загрузка модуля пересчета производится два раза: при значении сигнала $Q_{14} = 0$ загружаются значения T_{13-0}, а при $Q_{14} = 1$ — только значения T_{13-1}. При четном значении модуля пересчета M ($T_0 = 0$) триггер Q_0 исключен из счета, а при нечетном значении M ($T_0 = 1$) триггер Q_0 срабатывает только один раз в течение всего цикла пересчета, изменяя свое состояние с 1 на 0. Вход $\overline{CE}$ (*Count Enable*) вычитающего счетчика (*Down-Counter*) используется для запрета/разре-шения счета. Значение сигнала переполнения счетчика $\overline{P}_{14} = 0$, появляющееся в нулевом его состоянии, производит загрузку значений T_{13-0} или T_{13-1}.

Состояние таймера, передаваемое в МП по шине адреса-данных AD_{7-0}, представляется двумя байтами в виде:

$$AD_7AD_6AD_5AD_4AD_3AD_2AD_1AD_0 = Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_{14} \text{ — младший байт,}$$
$$AD_7AD_6AD_5AD_4AD_3AD_2AD_1AD_0 = \times\times Q_{13}Q_{12}Q_{11}Q_{10}Q_9Q_8 \text{ — старший байт}$$

(старший разряд счетчика Q_{14} перемещен на место младшего разряда Q_0).

При таком построении таймера счет изменений тактового сигнала *CLK* с 0 на 1 производится не в двоичной системе счисления. Если в некоторый момент времени произвести останов счета таймера и после этого прочитать его состояние (значения сигналов Q_{14-1}), то оно не будет равно двоичному коду числа импульсов *CLK*, оставшихся до окончания счета (до нулевого состояния). Это осложняет использование таймера в качестве счетчика событий, состояние которого должно давать двоичный код остатка счета.

Полный цикл счета таймера при задании модуля пересчета $M = 13$ выполняется в соответствии со схемой:

2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	← вес разряда Q_r
14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	← r — номер разряда счетчика
1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	= 13 — Загрузка модуля пересчета $M = 13$
1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	= 12 — $Q_0 \leftarrow 0$ (*CLK* ⌡ — см. рис. 3.130)
1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	= 10 — После вычитания 1 (*CLK* ⌡)
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	= 8 — После вычитания 2 (*CLK* ⌡)
1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	= 6 — После вычитания 3 (*CLK* ⌡) ↓
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	= 4 — После вычитания 4 (*CLK* ⌡) ↓
1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	= 2 — После вычитания 5 (*CLK* ⌡) ↓
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 0 — После вычитания 6 (*CLK* ⌡) ↓
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	= 12 — Загрузка модуля пересчета $M - 1$
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	= 10 — После вычитания 7 (*CLK* ⌡) ↓
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	= 8 — После вычитания 8 (*CLK* ⌡) ↓
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	= 6 — После вычитания 9 (*CLK* ⌡) ↓ ↓
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	= 4 — После вычитания 10 (*CLK* ⌡) ↓ ↓
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	= 2 — После вычитания 11 (*CLK* ⌡) ↓ ↓
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 0 — После вычитания 12 (*CLK* ⌡) ↓ ↓
1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	= 13 — Загрузка модуля пересчета $M = 13$

Для перевода кода состояния таймера в момент останова в двоичное число остатка счета следует произвести действия:

1) остановить счет;

2) прочитать 16-разрядное число из таймера;

3) обнулить старшие два бита M_2M_1, задающие режим работы таймера;

4) сбросить в 0 флаг переноса *CY* в регистре признаков и сдвинуть вправо полученное 16-разрядное число на один разряд с использованием разряда переноса;

5) если после сдвига разряд $CY = 1$, то следует добавить к полученному результату число $M/2$ или $(M - 1)/2$, где M — запрограммированный модуль пересчета.

Счет таймера может быть остановлен в любой момент времени командой $TM_2TM_1 = 01$ в слове управления *CW* (см. рис. 3.121). Если таймер остановлен в состоянии 6 при значении 1 разряда счетчика Q_{14}, то после чтения состояния его триггеров Q_{14-1} выполняются операции, определяющие остаток до конца счета:

2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	← вес разряда состояния счетчика
CY	13	12	11	10	9	8	7	6	5	4	3	2	1	14	← номер разряда Q_r
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	= 7
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	= 3 — после сдвига на один разряд вправо через *CY*
	+														
	0	0	0	0	0	0	0	0	0	0	0	1	1	0	= 6 = (*M* – 1)/2
	0	0	0	0	0	0	0	0	0	0	1	0	0	1	= 9 — остаток до конца счета

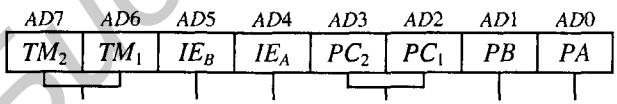
Если таймер остановлен в состоянии 6 при значении 0 разряда Q_{14}, то после чтения его состояния выполняется только операция сдвига, определяющая остаток до конца счета:

2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	← вес разряда состояния счетчика
CY	13	12	11	10	9	8	7	6	5	4	3	2	1	14	← номер разряда Q_r
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	= 6
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	= 3 — остаток до конца счета после сдвига вправо через *CY*

Разряды $AD_7AD_6 = TM_2TM_1$ слова управления *CW* (см. рис. 3.121) используются для задания начала и конца счета таймера. После сброса БИС (*RESET* = 1) для запуска таймера всегда нужно подавать команду *Start* ($TM_2TM_1 = 11$), предварительно задав режим работы и модуль его пересчета. После загрузки нового режима работы таймера M_2M_1 или его модуля пересчета T_{13-0} также следует подавать команду *Start*.

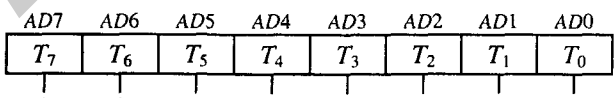
Независимое управление двумя разными устройствами (таймером и портами ввода-вывода) одним байтом слова управления *CW* требует хранения его копии в ОЗУ. Чтение из ОЗУ копии слова управления *CW* и модификация в нем только двух старших разрядов позволяет производить управление работой таймера без нарушения настройки режимов работы портов ввода-вывода. Сохранение слова управления *CW* в ОЗУ следует производить только при задании или изменении режимов работы портов ввода-вывода.

Задача 3 (файл 3#09_03.asm). Для МП-системы с отображением адресного пространства ввода-вывода на адресное пространство памяти, изображенной на рис. 3.127, запрограммировать работу портов *PA* и *PB* БИС 8156*H* на вывод с квитированием и по прерыванию. Установить режим *Mode* 2 таймера, задать модуль пересчета $M = T_{13-0} = 13d$ и запустить счет. Через некоторое время остановить счет таймера и вычислить число тактов, оставшихся до окончания текущего цикла счета. *Решение*:



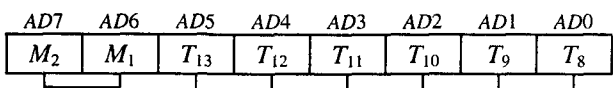
RGCW: *addr* = 1800*h* (см. табл. 3.20 и 3.23)

= 3B*h* = *CW*



Timer LSB: *addr* = 1804*h* (см. табл. 3.20 и 3.23)

= 0D*h* — младший байт T_{7-0} модуля пересчета



Timer MSB: *addr* = 1805*h* (см. табл. 3.20 и 3.23)

= 80*h* — *Mode* 2 и $T_{13-8} = 0$

```
        defseg  IO8156, start = 1800h, class = Data          ; Data Segment
        seg     IO8156
tim     ds      8           ; С адреса tim = 1800h резервируется место для 8 байт данных
        defseg  Ram8156, start = 1000h, class = Data
        seg     Ram8156
copy    ds      1           ; С адреса copy = 1000h резервируется место для одного байта
        defseg  Rom8755, start = 100h, class = Code          ; Code Segment
        seg     Rom8755
cw      equ     3Bh         ; cw = 3Bh — слово управления CW БИС 8156
mdl     equ     800Dh       ; mdl = M2M1T13–0 (Timer MSB и Timer LSB)
        MVI     A, low cw   ; A ← CW = 3Bh — слово управления БИС 8156
        STA     tim         ; 8156 ← CW (команда STA вместо команды OUT port)
        STA     copy        ; M(1000) ← CW — копия слова управления
        LXI     H, tim + 5  ; HL ← 1805h — адрес регистра Timer MSB БИС 8156
        MVI     M, high mdl ; 8156 ← 80h — MSB (команда MVI вместо команды OUT port)
        DCX     H           ; HL ← 1804h — адрес регистра Timer LSB БИС 8156
        MVI     M, low mdl  ; 8156 ← 0Dh — LSB (команда MVI вместо команды OUT port)
        LDA     copy        ; A ← M(1000) — копия слова управления CW
        ORI     0C0h        ; A ← CW ∨ 1100 0000 (TM2TM1 = 11 — см. рис. 3.121)
        STA     tim         ; 8156 ← CW — запуск счета (команда STA вместо OUT port)
          ∴                 ; Основная программа
        MVI     A, 7        ; Имитация изменения состояния счетчика таймера:
        STA     tim + 4     ;          Q7Q6Q5Q4Q3Q2Q1Q14 = 0000 0111
          ∴                 ; (эти две команды нужны только для отладки программы)
        LDA     copy        ; A ← M(1000) — копия слова управления CW
        ORI     40h         ; A ← CW ∨ 0100 0000 (TM2TM1 = 01 — см. рис. 3.121)
        STA     tim         ; 8156 ← CW — останов счета      (команда STA вместо OUT port)
        LDA     tim + 5     ; A ← ××Q13Q12Q11Q10Q9Q8 — разряды состояния счетчика
        ANI     3Fh         ; A ← 00Q13Q12Q11Q10Q9Q8, флаг CY ← 0      (LDA вместо IN port)
        RAR                 ; A ← 000Q13Q12Q11Q10Q9, CY ← Q8
        MOV     H, A        ; H ← 000Q13Q12Q11Q10Q9
        LDA     tim + 4     ; A ← Q7Q6Q5Q4Q3Q2Q1Q14 — разряды состояния счетчика
        RAR                 ; A ← Q8Q7Q6Q5Q4Q3Q2Q1, CY ← Q14          (LDA вместо IN port)
        MOV     L, A        ; L ← Q8Q7Q6Q5Q4Q3Q2Q1
        LXI     D, (mdl and 3FFFh) shr 1 ; DE ← 0006h = 00T13 ... T8 T7 ... T0
; DE = M/2 при четном M и DE = (M – 1)/2 при нечетном M
        JNC     L1
        DAD     D           ; HL ← HL + DE
L1:     end                 ; HL = число тактов, оставшихся до окончания текущего цикла счета
```

БИС 8755A/8355. Эти БИС содержат ПЗУ 2048 × 8 бит и два 8-разрядных порта ввода-вывода с индивидуально программируемыми на ввод и вывод разрядами. Различаются БИС 8755*A* и 8355 только типом ПЗУ: БИС 8355 выполнено с масочным ПЗУ (потребителем не программируется), а БИС 8755*A* содержит *EPROM* со стиранием ультрафиолетовыми лучами и программируемым потребителем. Поэтому ниже будет описана только БИС 8755*A*, условное графическое обозначение которой представлено на рис. 3.117. Эта БИС потребляет токи:

$I_{CC\,max}$ = 180 мА и $I_{DD\,max}$ = 30 мА при $V_{DD} = V_{CC}$ = +5 В (режим нормальной работы);
$I_{DD\,max}/I_{DD\,typ}$ = 30/15 мА при $V_{DD\,max}/V_{DD\,typ}$ = 26/25 В (режим программирования).

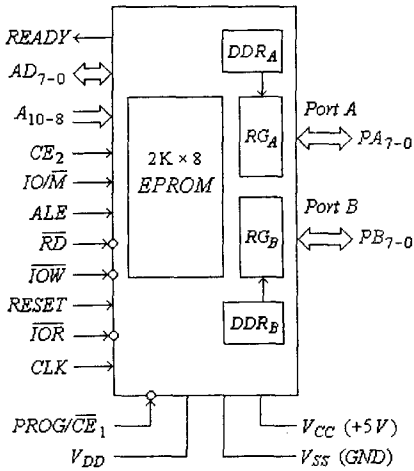


Рис. 3.131. Структурная схема БИС 8755*A*

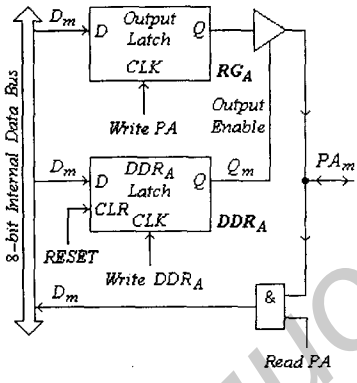


Рис. 3.132. Управление вводом и выводом

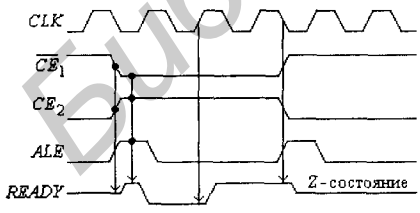


Рис. 3.133. Генерация сигнала готовности

Структурная схема БИС 8755*A* (1821РФ55) изображена на рис. 3.131:

EPROM 2K × 8 *bit* (*UV-Erasable and Electrically Reprogrammable ROM*) — стираемое ультрафиолетовыми лучами и электрически перепрограммируемое ПЗУ 2048 × 8 бит, адресуемое 11 разрядами шины адреса;

RG_A, RG_B — два одинаковых 8-разрядных буферных регистра вывода портов *PA* и *PB* (*Output Latch* на рис. 3.132) с обратным чтением. При вводе данные в регистрах не фиксируются. Сигналы квитирования в БИС не предусмотрены, поэтому можно использовать только программный метод ввода-вывода без квитирования. Каждый разряд портов *PA* и *PB* может независимо программироваться на ввод или на вывод;

DDR_A, DDR_B (*Data Direction Registers*) — два 8-разрядных регистра, задающие направление передачи данных по линиям портов *PA* и *PB* (DDR_A *Latch* на рис. 3.132 — $Q_m = 0$ — режим ввода, $Q_m = 1$ — режим вывода, $m = 0 \ldots 7$).

Сигналы БИС 8755*A* имеют назначение:

AD_{7-0} (*Address/Data*) — двунаправленная мультиплексная шина адреса/данных МП. Младший байт адреса A_{7-0} фиксируется во внутреннем регистре БИС сигналом *ALE* = 1 (адрес *EPROM* и внешних устройств *I/O* в зависимости от значения сигнала $IO/\overline{M}$);

A_{10-8} (*Address*) — 3 младших разряда старшего байта адреса для адресации *EPROM*;

$\overline{CE}_1/PROG$, CE_2 (*Chip Enable*) — сигналы разрешения кристалла, подаваемые с дешифратора адресных сигналов A_{15-11}. Активные уровни сигналов $\overline{CE}_1 = 0$ и $CE_2 = 1$ фиксируются в регистре по спадающему фронту сигнала *ALE*. Если хотя бы один из этих сигналов находится не в активном состоянии, выходы AD_{7-0} и *READY* переводятся в Z-состояние. Вход $\overline{CE}_1$ используется также для подачи сигнала программирования *PROG*;

$IO/\overline{M}$ (*I/O Memory*) — сигнал МП, указывающий на обращение к регистрам и портам ввода ($IO/\overline{M} = 1$) или к *EPROM* ($IO/\overline{M} = 0$). Значение сигнала $IO/\overline{M}$ фиксируется в регистре по спадающему фронту сигнала *ALE*;

CLK (*Clock*) — тактовый сигнал, используемый для перевода сигнала готовности *READY* в Z-состояние, после того, как он был установлен в 0 значениями сигналов $\overline{CE}_1 = 0$, $CE_2 = 1$ и *ALE* = 1 (рис. 3.133 — обычно используется выходной сигнал МП *CLK*);

READY — сигнал готовности БИС с Z-состоянием выхода, управляемый сигналами $\overline{CE}_1$, CE_2, *ALE* и *CLK* (рис. 3.133). Максимальное время доступа *EPROM* равно 450 нс, что обеспечивает использование БИС 8755*A* с МП 8085*A* и 8085*AH* без состояний ожидания, т. е. нет необходимости подключения выхода *READY* БИС 8755*A* к входу *READY* МП;

ALE (*Address Latch Enable*) — сигнал фиксации значений сигналов AD_{7-0} (A_{7-0}), $IO/\overline{M}$, A_{10-8}, $\overline{CE}_1$ и CE_2 в 14-разрядном регистре памяти (фиксация производится по спадающему фронту сигнала *ALE*);

PA_{7-0} и PB_{7-0} (*Port A* и *Port B*) — 8-разрядные порты ввода и вывода, направление передачи каждого разряда в которых программируется записью 8-разрядного кода в регистры DDR_A и DDR_B (см. рис. 3.132);

$\overline{RD}$ (*Read*) — сигнал чтения данных ($\overline{RD} = 0$) из портов ввода *PA* и *PB* при значении сигнала $IO/\overline{M} = 1$ и из ячеек памяти *EPROM* при значении сигнала $IO/\overline{M} = 0$;

$\overline{IOR}$ (*I/O Read*) — сигнал чтения выбранного порта ввода-вывода ($\overline{IOR} = 0$). Значение сигнала $\overline{IOR} = 0$ выполняет ту же самую функцию, что и комбинация значений сигналов $IO/\overline{M} = 1$ и $\overline{RD} = 0$. Если вход $\overline{IOR}$ в МП-системе не используется, то он должен быть подсоединен к источнику питания $V_{CC} = +5$ В ($\overline{IOR} \equiv 1$ — см. рис. 3.127);

$\overline{IOW}$ (*I/O Write*) — сигнал записи данных во внутренние регистры (значение сигнала $IO/\overline{M}$ при записи данных игнорируется);

RESET — значение сигнала *RESET* = 1 устанавливает все разряды портов *PA* и *PB* на ввод (сбрасывает регистры DDR_A и DDR_B в 0 — см. рис. 3.132).

Управление вводом-выводом БИС 8755*A* представлено в табл. 3.24. Содержимое регистров DDR_A и DDR_B для чтения недоступно. Данные могут быть записаны в регистры RG_A и RG_B даже в том случае, когда выходные буферы с *Z*-состоянием выхода заблокированы значениями сигналов $Q_m = 0$ (см. рис. 3.132). Это дает возможность линиям вывода порта быть инициализированными определенными значениями (0 или 1) до предоставления вывода (до установки значений $Q_m = 1$). Содержимое разрядов регистров RG_A и RG_B, установленных в режим вывода, можно читать (регистры с обратным чтением) вместе с разрядами портов PA_{7-0} и PB_{7-0}, запрограммированными на ввод.

Таблица 3.24. **Управление вводом-выводом БИС 8755*A***

$\overline{CE}$	AD_1	AD_0	$\overline{IOW}$	$\overline{RD}$	$\overline{IOR}$ *	Операция	Примечание
0	0	0	0	1	1	$D_{7-0} \rightarrow RG_A$	Запись данных в порт *PA*
0	0	1	0	1	1	$D_{7-0} \rightarrow RG_B$	Запись данных в порт *PB*
0	1	0	0	1	1	$D_{7-0} \rightarrow DDR_A$	Запись байта в регистр DDR_A
0	1	1	0	1	1	$D_{7-0} \rightarrow DDR_B$	Запись байта в регистр DDR_B
0	0	0	1	0	0	$D_{7-0} \leftarrow PA$	Чтение данных из порта *PA*
0	0	1	1	0	0	$D_{7-0} \leftarrow PB$	Чтение данных из порта *PB*
0	1	×	1	0	0	*Z*-состояние AD_{7-0}	Нет операций
0	×	×	1	1	1	*Z*-состояние AD_{7-0}	Нет операций
1	×	×	×	×	×	*Z*-состояние AD_{7-0}	Нет операций

Примечание: * $\overline{CE} = \overline{CE_1}CE_2$; $IO/\overline{M} = 1$, если используется сигнал $\overline{RD}$; $IO/\overline{M} = \times$, если используется сигнал $\overline{IOR}$.

Задача 4. В соответствии с приведенными на рис. 3.127 указаниями запрограммировать работу портов БИС 8755*A*. *Решение*:

AD7	*AD6*	*AD5*	*AD4*	*AD3*	*AD2*	*AD1*	*AD0*	
0	0	1	1	1	0	1	0	= 3A*h*
1	0	0	1	1	1	0	1	= 9D*h*

DDR_A: *addr* = 0802*h*
DDR_B: *addr* = 0803*h*

(см. табл. 3.23 и 3.24)

```
MVI   A, 3Ah    ; A ← 3Ah — код конфигурации порта PA БИС 8755A
STA   802h      ; DDR_A ← 3Ah (команда STA вместо команды OUT port)
MVI   A, 9Dh    ; A ← 9Dh — код конфигурации порта PB БИС 8755A
STA   803h      ; DDR_B ← 9Dh (команда STA вместо команды OUT port)
```

Стирание и программирование БИС 8755A. Стирание *EPROM* производится излучением с длиной волны короче, чем 4000 Å (Ангстрем). Необходимо отметить, что солнечный свет и некоторые типы флуоресцентных ламп имеют длину волны в диапазоне 3000 ÷ 4000 Å. Постоянное воздействие флуоресцентного освещения приведет к потере (стиранию) записанной информации за 3 года, прямое воздействие солнечного света — за 1 неделю. Поэтому для предотвращения неумышленного стирания рекомендуется использовать непрозрачные наклейки на кварцевое окно БИС 8755A.

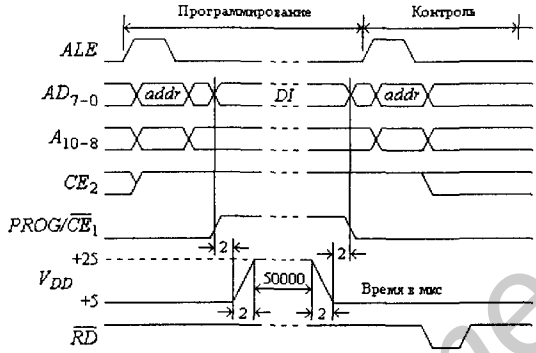


Рис. 3.134. Программирование БИС 8755A

Рекомендуемая процедура стирания для 8755A — экспозиция на короткой волне ультрафиолетового света, который имеет длину волны 2537 Å. Интегрированная доза (произведение интенсивности ультрафиолетового излучения на время экспозиции) для стирания должно быть минимум 15 Вт·с/см^2. Время стирания с этой дозировкой равно приблизительно 15 ÷ 20 минут при использовании ультрафиолетовой лампы мощностью 12 мВт/см^2 (БИС 8755A должна находиться на расстоянии 2,5 см от лампы).

Временны́е диаграммы для режима программирования изображены на рис. 3.134. В каждом цикле длительностью 50 мс программируется одна ячейка памяти по заданному адресу (один байт).

Первоначально, и после каждого стирания, все биты *EPROM* находятся в состоянии логической 1. Программируются только значения 0 разрядов байта для выбранных ячеек памяти. Значения 0 разрядов могут быть изменены на значение 1 только ультрафиолетовым стиранием. При программировании напряжение питания V_{DD} = +25 В, а при верификации (контроле правильности программирования байта) напряжение V_{DD} = +5 В.

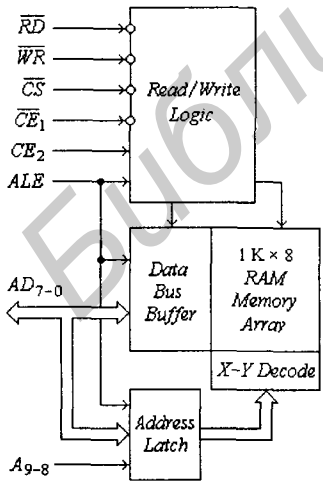


Рис. 3.135. Структурная схема БИС 8185

БИС 8185. Структурная схема этой БИС изображена на рис. 3.135 — БИС содержит только статическую память (*SRAM*) объемом 1024 × 8 бит, ячейки памяти которой селектируются 10 разрядами адреса A_{9-0}. Выбор кристалла (включение БИС) производится значением конъюнкции сигналов CE_2CE_1CS = 1, т. е. одновременны́м поступлением на БИС значений физических сигналов CE_2 = 1, $\overline{CE}_1$ = 0 и $\overline{CS}$ = 0.

Спадающим фронтом сигнала *ALE* ⎍ в 12-разрядном регистре фиксируются значения сигналов CE_2, $\overline{CE}_1$, A_9, A_8 и AD_{7-0} (A_{7-0}), но сигнал выбора кристалла $\overline{CS}$ (*Chip Select*) не фиксируется, что позволяет подавать его с задержкой (например, из-за адресного дешифратора), смещающей его активный уровень за пределы значения сигнала *ALE* = 1.

Если сигналы $\overline{CE}_1$ и CE_2 активны (имеют значения 0 и 1), то БИС 8185 готова к работе (режим *Powered Up* — режим с повышенным потреблением тока I_{CC} = 100 мА). Однако работа БИС (чтение или запись данных) будет запрещена до тех пор, пока сигнал $\overline{CS}$ не примет значения 0 и один из сигналов управления $\overline{RD}$ или $\overline{WR}$ не станет активным (0). Если хотя бы один из сигналов $\overline{CE}_1$ или CE_2 принимает неактивное значение (нет обращения к *SRAM*), то БИС 8185 переходит в режим *Powered Down* — режим с пониженным потреблением тока I_{CC} = 35 мА. В соответствии с этим вход $\overline{CE}_1$ обычно используется для подачи сигнала $IO/\overline{M}$ от МП 8085*A* (рис. 3.136). Шину AD_{7-0} переводит в *Z*-состояние неактивное значение любого из сигналов $\overline{CE}_1$, CE_2, $\overline{CS}$ или $\overline{RD}$ (*Read* — чтение). В табл. 3.25 приведена адресация памяти и устройств ввода-вывода для МП-системы, изображенной на рис. 3.136 (сравните с табл. 3.23).

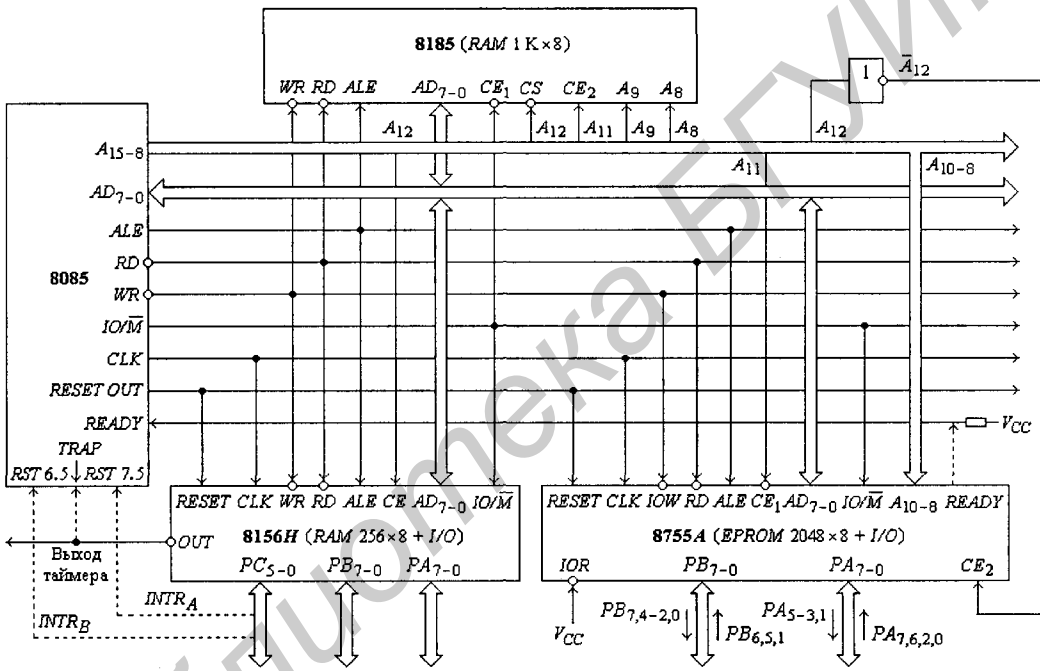


Рис. 3.136. Конфигурация МП-системы с раздельными адресными пространствами памяти и ввода-вывода

Таблица 3.25. **Адресация памяти и портов ввода-вывода**

Разряды адресных сигналов A_{15-0}																Адреса		БИС
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	*addr Memory*	*port I/O*	
0	0	0	0	0	×	×	×	×	×	×	×	×	×	×	×	0000 ÷ 07FF*h*	—	8755*A*
0	0	0	0	0	0	0	0	0	0	0	0	0	0	×	×	—	00 ÷ 03*h*	
0	0	0	0	1	0	×	×	×	×	×	×	×	×	×	×	0800 ÷ 0BFF*h*	—	8185
0	0	0	1	0	0	0	0	×	×	×	×	×	×	×	×	1000 ÷ 10FF*h*	—	8156*H*
0	0	0	1	0	0	0	0	0	0	0	1	0	×	×	×	—	10 ÷ 17*h*	

Примечание: × — значения 0 и 1; 0 — разряд в адресации не участвует.

3.10. Программируемый контроллер клавиатуры и дисплея 8279

Программируемый контроллер клавиатуры и дисплея (*Programmable Keyboard/Display Interface — PKDI*) фирмы *Intel* 8279/8279–5 (отечественный аналог КР580ВВ79/КР580ВВ79Д) предназначен для обслуживания 64-клавишной клавиатуры и одного 16-разрядного алфавитно-цифрового дисплея (*alphanumeric display* — рис. 3.137, *а*) или двух 16-разрядных цифровых (7-сегментных) дисплеев (*numeric display* — рис. 3.137, *б*). Для построения дисплеев обычно используются индикаторы, выполненные на светоизлучающих диодах (*LED — light-emitting diode* — светодиод, светоизлучающий диод, СИД). Контроллер 8279 освобождает МП 8080/8085 от рутинной работы, связанной с программным динамическим (мультиплексным) управлением многоразрядных дисплеев, рассмотренным в § 1.9 (см. рис. 1.38).

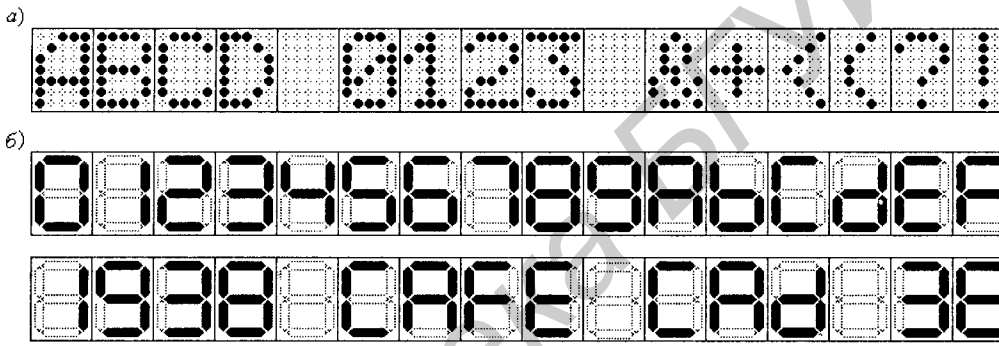


Рис. 3.137. Алфавитно-цифровой и два цифровых дисплея

Контроллеры 8279 и 8279–5 (рис. 3.138) изготовляются по *n*-МОП технологии (*NMOS*) и характеризуются параметрами (максимальная рассеиваемая мощность составляет 1 Вт):

V_{CC} = +5 В ± 5% для *PKDI* 8279 и V_{CC} = +5 В ± 10% для *PKDI* 8279–5, $I_{CC\,max}$ = 120 мА;

$V_{OL\,max}$ = 0,45 В при I_{OL} = 1,6 мА для *PKDI* 8279 и I_{OL} = 2,2 мА для *PKDI* 8279–5;

$V_{OH\,min}$ = 2,4 В при I_{OH} = –100 мкА для *PKDI* 8279 и I_{OH} = –400 мкА для *PKDI* 8279–5;

$F_{CLK} \le 2{,}0$ МГц для *PKDI* 8279 и $F_{CLK} \le 3{,}1$ МГц для *PKDI* 8279–5.

Структурная схема *PKDI*. На структурной схеме, приведенной на рис. 3.139, изображены цифровые узлы:

Data Buffer — буфер шины данных, представляющий собой двунаправленный приемопередатчик с *Z*-состоянием выхода. Буфер используется для приема от МП команд управления и данных для дисплея, а также для чтения данных и слова состояния из *PKDI*;

I/O Control — устройство управления вводом-выводом ($\overline{RD} = 0$ — ввод, $\overline{WR} = 0$ — вывод), обеспечивающее включение БИС ($\overline{CS} = 0$), адресацию данных ($A_0 = 0$) и команд управления CW_{7-0} или слова состояния ($A_0 = 1$), а также задающее направление передачи данных приемопередатчиком буфера шины данных (передача от *PKDI* к МП при значении $\overline{RD} \vee \overline{CS} = 0$ и передача от МП к *PKDI* при значении $\overline{WR} \vee \overline{CS} = 0$);

Control and Timing Registers — регистры управления и синхронизации, используемые для хранения режимов работы клавиатуры и дисплея, а также других параметров, программно устанавливаемых микропроцессором;

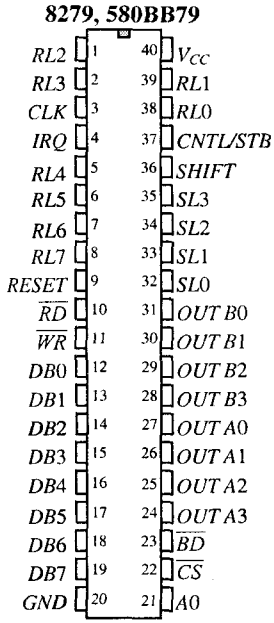


Рис. 3.138. БИС 8279

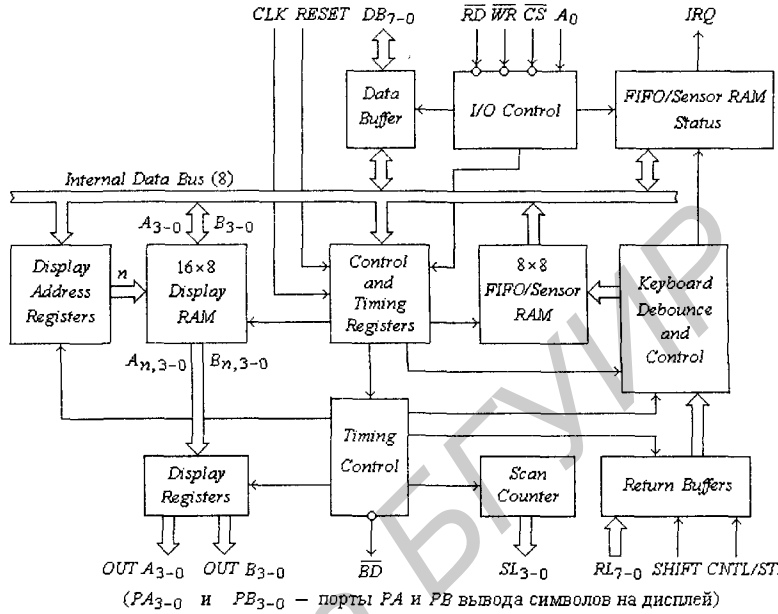


Рис. 3.139. Структурная схема контроллера 8279

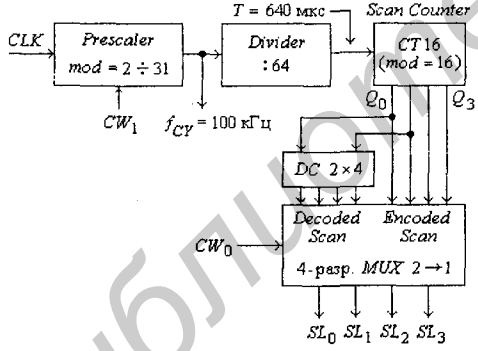


Рис. 3.140. Структурная схема устройства синхронизации и счетчика сканирования

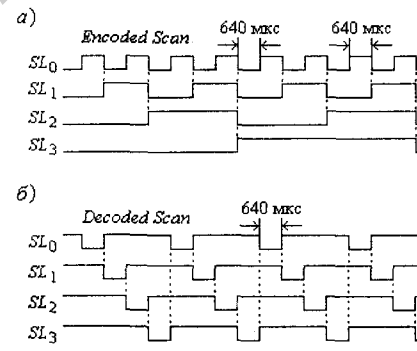


Рис. 3.141. Временны́е диаграммы сигналов счетчика сканирования

Timing Control — устройство синхронизации, содержащее прескалер с программируемым командой CW_1 (см. рис. 3.151) модулем пересчета от 2 до 31 и делитель частоты с коэффициентом деления, равным 64 (рис. 3.140; *Prescaler* — предварительный делитель частоты, используемый для масштабирования частоты входного тактового сигнала *CLK*, *Divider* — делитель частоты). Прескалер используется для получения базовой частоты f_{CY} = 100 кГц внутренней синхронизации всех узлов БИС (период T_{CY} = 10 мкс, *CY* — *Cycle*). Двоичный 6-разрядный делитель частоты *Divider* определяет временны́е параметры сканирования клавиатуры и дисплея: период сканирования клавиатуры равен 640 мкс × 8 = 5,12 мс, время устранения "дребезга" (*debounce*) клавиатуры равно 10,24 мс;

Scan Counter — счетчик сканирования клавиатуры и дисплея (4-разрядный двоичный счетчик — рис. 3.140). Тип четырех выходных сигналов счетчика SL_{3-0} (*Scan Lines* — линии сканирования клавиатуры и дисплея) программируется командой CW_0 (см. рис. 3.150) и задает два режима сканирования клавиатуры и дисплея: 1) кодированный режим (*Encoded Mode* — рис. 3.141, *а*), требующий применения внешних дешифраторов и используемый для построения 64-клавишной клавиатуры и одного 16-разрядного алфавитно-цифрового дисплея или двух 16-разрядных цифровых дисплеев; 2) декодированный режим (*Decoded Mode* — рис. 3.141, *б*), не требующий применения внешних дешифраторов и используемый для построения 32-клавишной клавиатуры и одного 4-разрядного алфавитно-цифрового дисплея или двух 4-разрядных цифровых дисплеев. В декодированном режиме (с внутренней дешифрацией) сигналы SL_{3-0} имеют низкий активный уровень (0);

Return Buffers — 8-разрядный буферный регистр памяти для сигналов возврата RL_{7-0} (*Return Line*), поступающих с клавиатуры. К входам RL_{7-0} вместо клавиатуры можно подключать матрицу датчиков (*Sensor Matrix*) или 8-разрядное внешнее устройство, записывающее в регистр памяти байт данных положительным фронтом сигнала *STB*, подаваемым на вход *CNTL/STB* (*Control/Strobe Input Mode* — клавиша *Cntl*/режим стробирования ввода). Режим работы *PKDI* 8279 (клавиатура, матрица датчиков, стробируемый ввод) программируется командой CW_0 (см. рис. 3.150). Стробируемый ввод обеспечивает программный ввод данных с квитированием и по прерыванию с использованием *FIFO* 8 × 8 бит в качестве буферной памяти (см. § 2.6);

Keyboard Debounce and Control — устройство управления и устранения "дребезга" клавиатуры. В режиме сканирования матричной клавиатуры данное устройство анализирует сигналы RL_{7-0} для обнаружения замыкания клавишного контакта и определяет номер строки, в которой контакт был замкнут (принцип работы матричного контроллера клавиатуры был рассмотрен в § 1.9 — см. рис. 1.37). При обнаружении замыкания контакта противодребезговая схема через 10,24 мс проверяет его состояние. Если контакт сохраняет замкнутое состояние, то его номер в клавишной матрице и состояния сигналов *CNTL* и *SHIFT* передаются в буферную память типа *FIFO*. В режиме сканирования матрицы датчиков значения сигналов RL_{7-0} непосредственно записываются в соответствующую ячейку памяти ОЗУ датчиков (*Sensor RAM*; *sensor* — датчик, чувствительный элемент, сенсор). В режиме стробируемого ввода значения сигналов на линиях RL_{7-0} записываются в *FIFO* по положительному фронту сигнала *CNTL/STB*;

8 × 8 *FIFO/Sensor RAM* (*FIFO*/ОЗУ датчиков) — память объемом 8 байт для хранения кодов нажатых клавиш или данных, поступающих от матрицы датчиков или внешнего устройства со стробируемым вводом. В режимах сканирования клавиатуры и стробируемого ввода память используется как *FIFO* (принцип работы *FIFO* был описан в § 2.6). В режиме сканирования матрицы датчиков ОЗУ используется для хранения их состояния. Каждая строка ОЗУ датчиков хранит текущее состояние соответствующей строки матрицы датчиков;

FIFO/Sensor RAM Status — 8-разрядный регистр состояния *FIFO/Sensor RAM*, в котором фиксируется число введенных символов (байт данных), ошибки записи в полное *FIFO* и чтения пустого *FIFO* и др. (см. рис. 3.158). Если *FIFO* не пусто, то выдается значение сигнала запроса прерывания *IRQ* = 1. В режиме сканирования матрицы датчиков сигнал *IRQ* переходит в состояние 1 при каждом обнаруженном при сканировании изменении состояния сенсоров. Регистр состояния может быть опрошен командой IN *port* для анализа ошибок и реализации программных методов ввода и вывода с квитированием;

Display Address Registers — два 4-разрядных регистра хранения адреса $n = 0 \ldots 15$ ОЗУ дисплея, позволяющие адресовать запись и чтение данных как байтами $D_{7-0} = A_{3-0}B_{3-0}$, так и независимыми тетрадами (*nibbles*) $D_{7-0} = A_{3-0}\times\times\times\times$ и $D_{7-0} = \times\times\times\times B_{3-0}$, что программируется коман-

дой CW_5 (см. рис. 3.155). Записанные в ОЗУ дисплея значения A_n,3–0 и B_n,3–0 (n — номер ячейки памяти ОЗУ дисплея) затем последовательно периодически выводятся на индикаторы дисплея с интервалом в 640 мкс — время вывода содержимого одной из шестнадцати ячеек памяти ОЗУ дисплея на выходы *OUT* A_{3-0} (PA_{3-0}) и *OUT* B_{3-0} (PB_{3-0}) без учета времени гашения индикаторов в течение 150 мкс (*Blank Code*), автоматически вводимого при переключении разрядов дисплея (рис. 3.142). Управление адресацией ОЗУ дисплея осуществляется командами CW_3 и CW_4 (см. рис. 3.153 и 3.154), предоставляющими возможность введения автоинкремента адреса при чтении и записи данных;

16 × 8 *Display RAM* — ОЗУ дисплея для хранения 16 байт данных, предназначенных для отображения как на алфавитно-цифровых, так и на цифровых (7-сегментных) дисплеях;

Display Registers — буферные регистры дисплея, периодически обновляемые значениями A_n,3–0B_n,3–0, хранящимися в ОЗУ дисплея.

Описание сигналов *PKDI*. Сигналы БИС 8279 имеют назначение:

DB_{7-0} (*Data Bidirectional*) — сигналы двунаправленной шины данных МП;

A_0 (*Address*) — младший разряд шины адреса МП, интерпретирующий назначение информации, передаваемой по шине данных D_{7-0} (табл. 3.26);

$\overline{CS}$ (*Chip Select*) — сигнал выбора кристалла, имеющий низкий активный уровень (подается от дешифратора адресных сигналов A_{7-1});

$\overline{RD}$ (*Read*), $\overline{WR}$ (*Write*) — сигналы чтения и записи информации в *PKDI*, являющегося для МП обычным внешним устройством ($\overline{RD} = \overline{I/OR}$, $\overline{WR} = \overline{I/OW}$ — для МП 8080/8085);

CLK — тактовый сигнал (φ_2 от генератора 8224 или *CLK* от МП 8085);

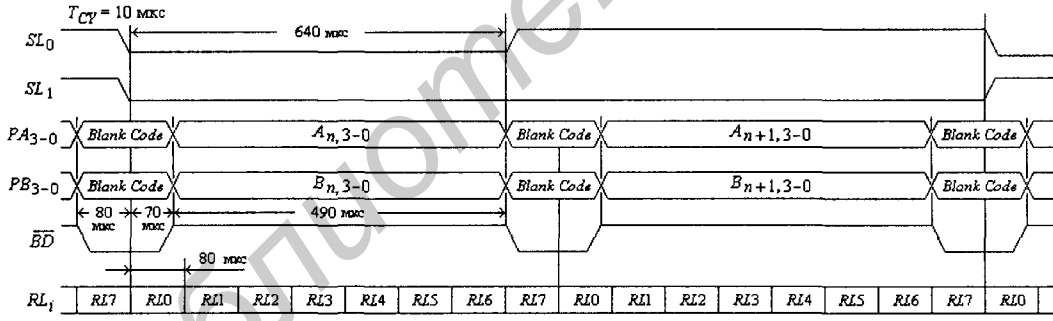


Рис. 3.142. Временны́е диаграммы сигналов управления дисплеем

Таблица 3.26. **Операции ввода-вывода**

$\overline{CS}$ A_0	$\overline{RD}$ $\overline{WR}$	Операция	Примечание
0 0	0 1	$D_{7-0} \leftarrow$ *FIFO/Sensor RAM, Display RAM*	Ввод в МП данных ОЗУ дисплея, данных и состояния *FIFO*
0 1	0 1	$D_{7-0} \leftarrow$ *FIFO Status*	
0 0	1 0	$D_{7-0} \rightarrow$ *Display RAM*	Вывод из МП данных для ОЗУ дисплея и команд управления
0 1	1 0	$D_{7-0} \rightarrow CW_0 \div CW_7$ (*Command Word*)	
0 ×	1 1	Нет операций	Шина D_{7-0} в *Z*-состоянии
1 ×	× ×	Нет операций	

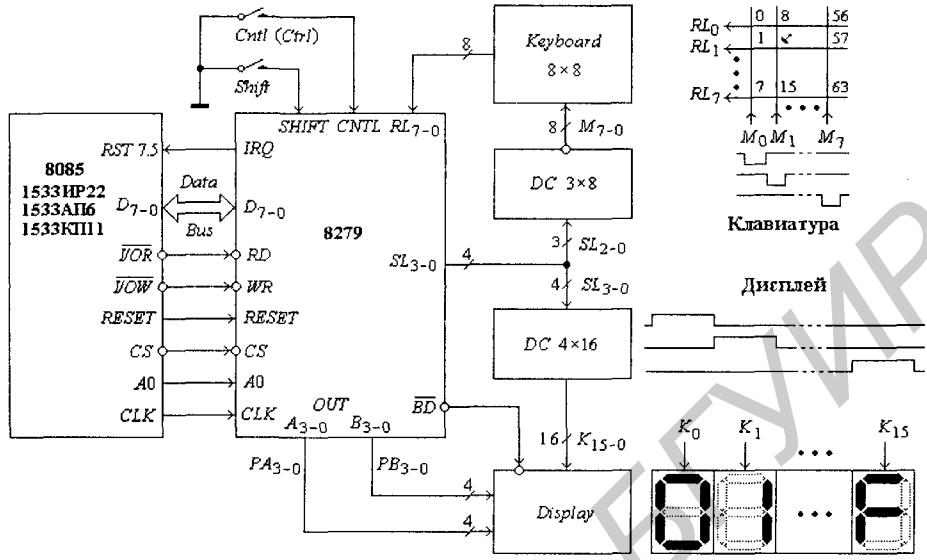


Рис. 3.143. Структурная схема управления клавиатурой и дисплеем

RESET — сигнал от генератора 8224 или от МП 8085 (с выхода *RESET OUT*), предназначенный для задания режимов работы (по умолчанию) БИС 8279: 16-разрядный дисплей с восьмиразрядным кодированием отображаемых символов и заданием ввода с левой стороны дисплея, сканирование клавиатуры с внешней дешифрацией и блокировкой нажатия двух клавиш, *mod* 31 — модуль пересчета прескалера. Задание этих режимов работы эквивалентно подаче команд $CW_0 = 08h$ (см. рис. 3.150) и $CW_1 = 3Fh$ (см. рис. 3.151);

SL_{3-0} (*Scan Lines*) — линии сканирования матричной клавиатуры, матрицы датчиков и индикаторов дисплея в кодированном и декодированном режимах (см. рис. 3.141). В кодированном режиме работы эти сигналы подаются на внешние дешифраторы *DC* 3 × 8 и *DC* 4 × 16 (рис. 3.143), выдающие сигналы сканирования клавиатуры M_{7-0} и дисплея K_{15-0};

RL_{7-0} (*Return Line*) — линии возврата, подключенные через внутренние резисторы к источнику питания V_{CC} = +5 В (*pullups*) для задания исходного логического уровня 1. В режиме сканирования клавиатуры линии RL_{7-0} при нажатии клавиш подключаются к сигналам сканирования M_{7-0}, имеющим низкий (0) активный уровень (рис. 3.143). В режиме стробируемого ввода на линии RL_{7-0} от внешнего устройства подаются 8-разрядные данные, записываемые в *FIFO* по положительному фронту сигнала *CNTL/STB*;

SHIFT, *CNTL/STB* (*Control/Strobe Input Mode*) — входы для подключения клавиш *Shift* и *Cntl* (*Control*) в режиме сканирования клавиатуры и подачи сигнала *STB* в режиме стробируемого ввода. Клавиша *Shift* обычно используется для управления верхним и нижним регистрами клавиатуры. Состояния клавиш *Shift* и *Cntl* вводятся в *FIFO* только при нажатии одной из клавиш основной 64-клавишной клавиатуры (см. рис. 3.159) — в итоге клавиатура выдает 256 различных кодов от 00*h* до FF*h*. Линии *SHIFT* и *CNTL/STB* имеют внутренние резисторы, подключенные к источнику питания V_{CC} = +5 В (*pullups*) для задания исходного логического уровня 1;

IRQ (*Interrupt Request*) — сигнал запроса прерывания. В режиме ввода с клавиатуры значение этого сигнала устанавливается в 1 при наличии в *FIFO* хотя бы одного символа и устанавливается в 0 при каждом чтении символа из *FIFO*, но после этого, если *FIFO* все еще не пусто, сигнал *IRQ* принимает значение 1. В режиме сканирования матрицы датчиков каждый

раз, когда изменяются выдаваемые ими данные, сигнал *IRQ* принимает значение 1. После чтения ОЗУ датчиков сигнал *IRQ* принимает значение 0;

OUT A_{3-0}, *OUT* B_{3-0} — порты PA_{3-0} и PB_{3-0} вывода символов на дисплей. Вывод символов синхронизирован с сигналами на линиях SL_{3-0} (см. рис. 3.142), что необходимо для построения мультиплексных (динамических) дисплеев;

$\overline{BD}$ (*Blank Display*) — значение сигнала $\overline{BD} = 0$ при гашении дисплея на время переходных процессов (см. рис. 3.142), возникающих при переключении разрядов индикаторов, и при выполнении команды CW_6 гашения дисплея (см. рис. 3.156). Этот сигнал можно не использовать, так как код гашения (*Blank Code*) автоматически вводится при переключении разрядов.

Алфавитно-цифровой дисплей. На рис. 3.143 была изображена структурная схема управления 64-клавишной клавиатурой и 16-разрядным 7-сегментным дисплеем, по которой не представляет труда разработать принципиальную схему с использованием как индикаторов с общим катодом, так и индикаторов с общим анодом. Параметры некоторых 7-сегментных индикаторов на светоизлучающих диодах (СИД) приведены в табл. 3.27 (расположение выводов см. на рис. 3.145, *в*). Так как индикаторы работают в динамическом режиме со скважностью $Q = 16$, то необходимо использовать драйверы (усилители тока), обеспечивающие импульсный ток, достаточный для полноценного и равномерного светоизлучения всех сегментов.

Принципиальная схема управления 64-клавишной клавиатурой и 16-разрядным алфавитно-цифровым дисплеем изображена на рис. 3.144 и 3.145, *а* (разработана для лабораторных работ).

Таблица 3.27. **Параметры светоизлучающих индикаторов**

Индикатор	Цвет свечения	Размер знака, мм	Сила света/$I_{пр}$, мккд/мА	$U_{пр}/I_{пр}$, В/мА	$I_{пр}$, мА	Импульсный ток $I_{имп}$, мА
АЛС321А (ОК) АЛС321Б (ОА)	Желто-зеленый	4,9 × 7,5	120/20	3,6/20	20	—
АЛС324А (ОК) АЛС324Б (ОА)	Красный	4,9 × 7,5	1500/20	2,5/20	20	300
АЛС333А (ОК) АЛС333Б (ОА)	Красный	6,2 × 12	200/20	2/20	20	200
АЛС333В (ОК) АЛС333Г (ОА)	Красный	6,2 × 12	150/20	2/20	20	200
АЛС334А (ОК) АЛС334Б (ОА)	Желтый	5,2 × 12	200/20	3,3/20	20	200
АЛС334В (ОК) АЛС334Г (ОА)	Желтый	5,2 × 12	150/20	3,3/20	20	200
АЛС335А (ОК) АЛС335Б (ОА)	Зеленый	5,2 × 12	250/20	3,5/20	20	200
АЛС335В (ОК) АЛС335Г (ОА)	Зеленый	5,2 × 12	150/20	3,5/20	20	200
АЛС340А (5×7)	Красный	8 × 10	125/10	2,5/10	10	200
АЛС357А (5×7)	Желтый	8 × 10	200/10	4/10	10	200
АЛС358А (5×7)	Зеленый	8 × 10	300/10	4/10	10	280
АЛС363А (5×7)	Зеленый	8 × 10	100/20	2,5/20	10	52,5

Примечание: ОК — общий катод, ОА — общий анод, 5 × 7 — матрица из 35 элементов (СИД).

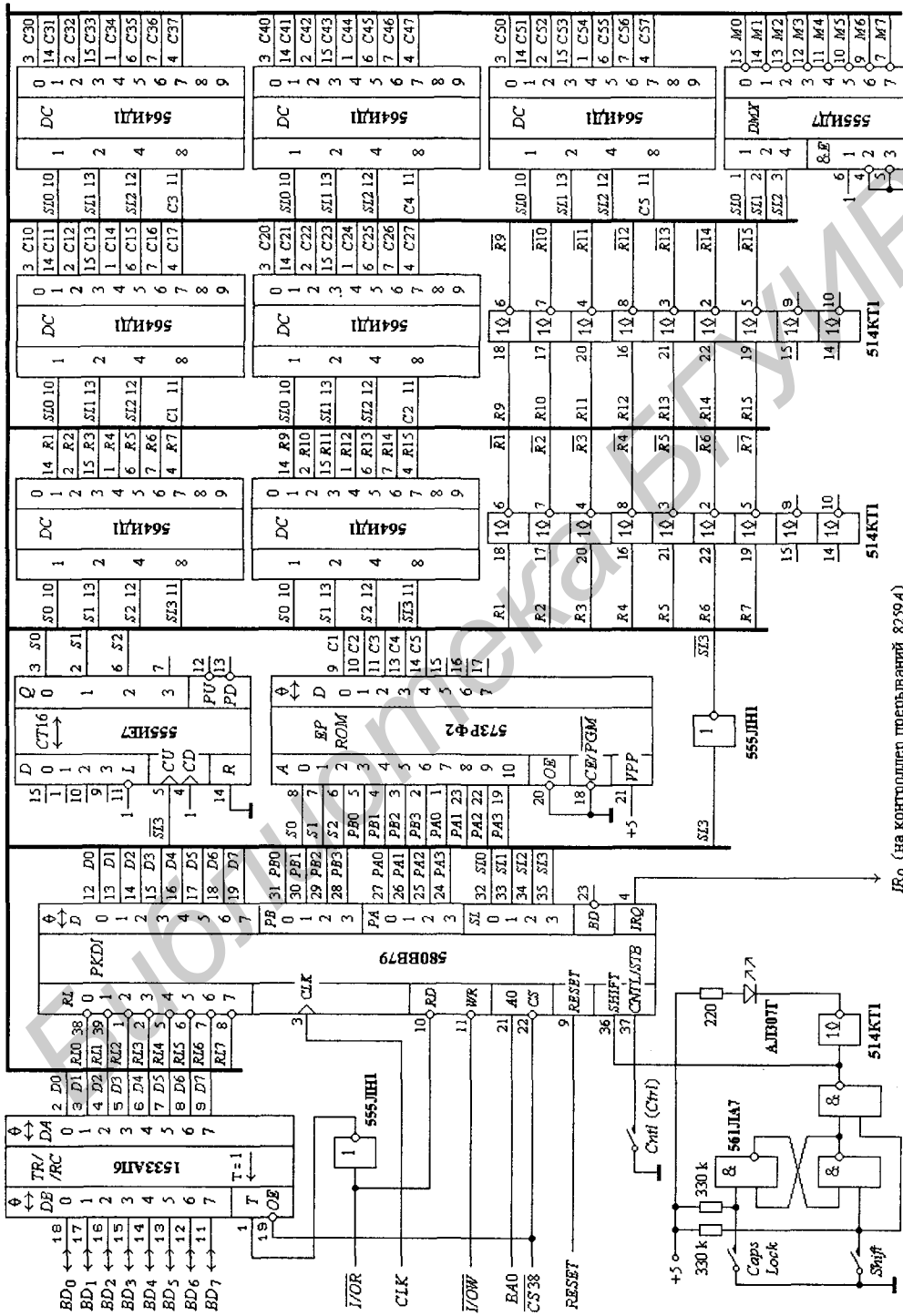


Рис. 3.144. Принципиальная схема контроллера клавиатуры и дисплея

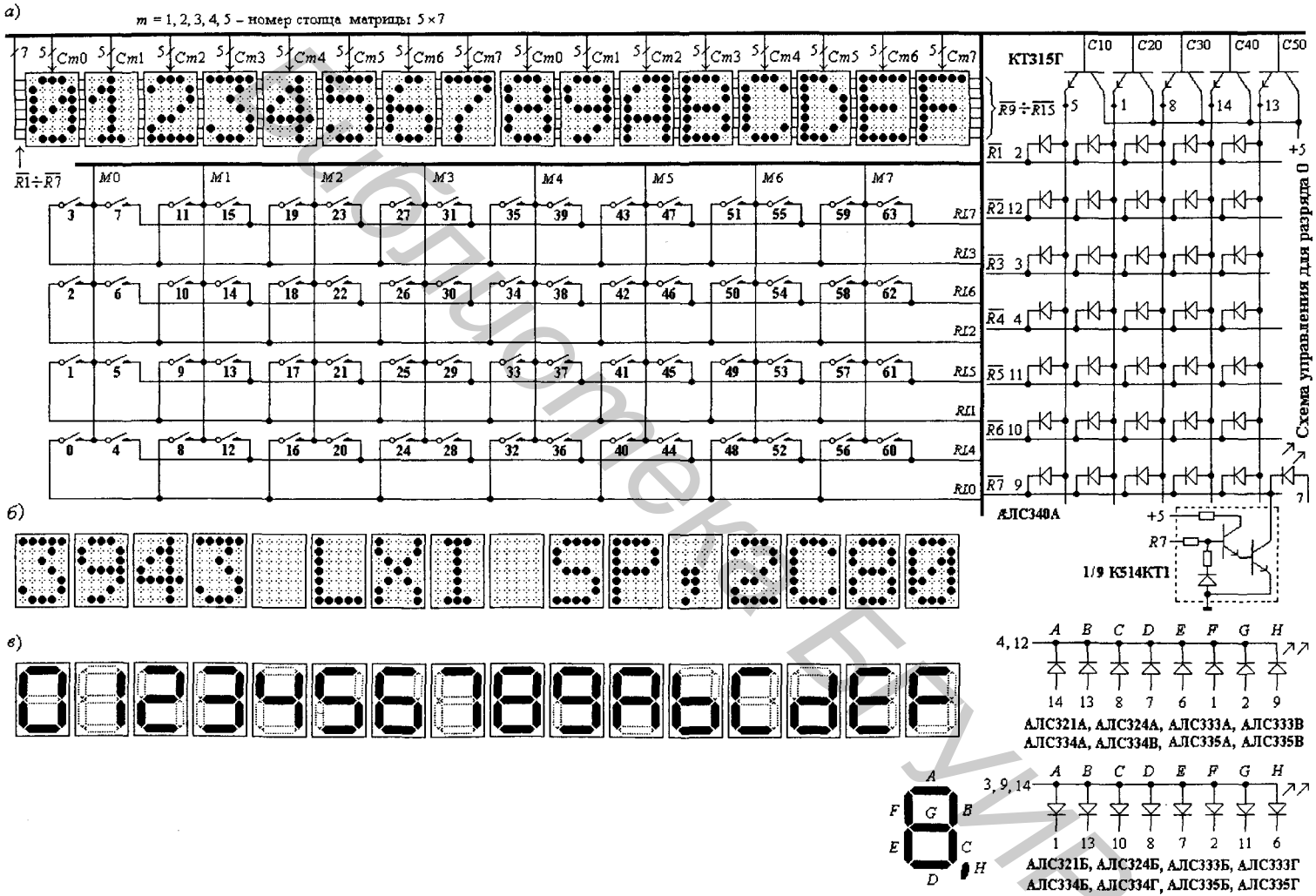


Рис. 3.145. Принципиальная схема 16-разрядного дисплея и 64-клавишной клавиатуры

Для построения 8-разрядного алфавитно-цифрового дисплея использованы дополнительные (внешние) цифровые схемы:

знакогенератор, выполненный на ИС 573РФ2 (ПЗУ для хранения построчных кодов отображаемых на дисплее символов, изображенных на рис. 3.146; C_{5-1} — построчный код столбцов символов) и ИС 555ИЕ7 (счетчик адресов строк S_{2-0} кодов символов для адресации ПЗУ);

устройство строчной развертки, выполненное на ИС 555ИЕ7 (та же самая ИС, что и в знакогенераторе), две ИС 564ИД1 (дешифратор строчной развертки *DC* 4×16, преобразующий адресные сигналы S_{2-0} в сигналы строчной развертки $R_{15-9,7-1}$) и ИС 514КТ1 (драйверы с открытым коллекторным выходом (рис. 3.145, *а*), обеспечивающие по выходам $\overline{R}_{15-9,7-1}$ импульсный ток достаточной величины);

демультиплексоры столбцов C_{5-1} знакогенератора, выполненные на пяти ИС 564ИД1 (выходы *Cmk*, где $m = 1 \ldots 5$ — номера столбцов, $k = 0 \ldots 7$ — номера разрядов дисплея, повторяющиеся два раза в двух частях дисплея; в качестве драйверов этих сигналов использованы эмиттерные повторители, реализованные на 40 транзисторах КТ315Г — рис. 3.145, *а*).

Поскольку для кодирования текстовой информации используются *ASCII*-коды, то они же и должны адресовать символы, выводимые на дисплей (на рис. 3.144 адресные сигналы *EPROM* $A_{10-3} = PA_{3-0}PB_{3-0} = OUT\ A_{3-0}B_{3-0}$ БИС 8279). Из этого следует, что адреса A_{10-0} ячеек памяти ПЗУ должны определяться значениями $a_7a_6a_5a_4a_3a_2a_1a_0S_2S_1S_0$, где $S_2S_1S_0$ — номер строки отображаемого символа, $a_7a_6a_5a_4a_3a_2a_1a_0$ — *ASCII*-код этого символа, выдаваемый контроллером 8279 на выходы PA_{3-0} и PB_{3-0}.

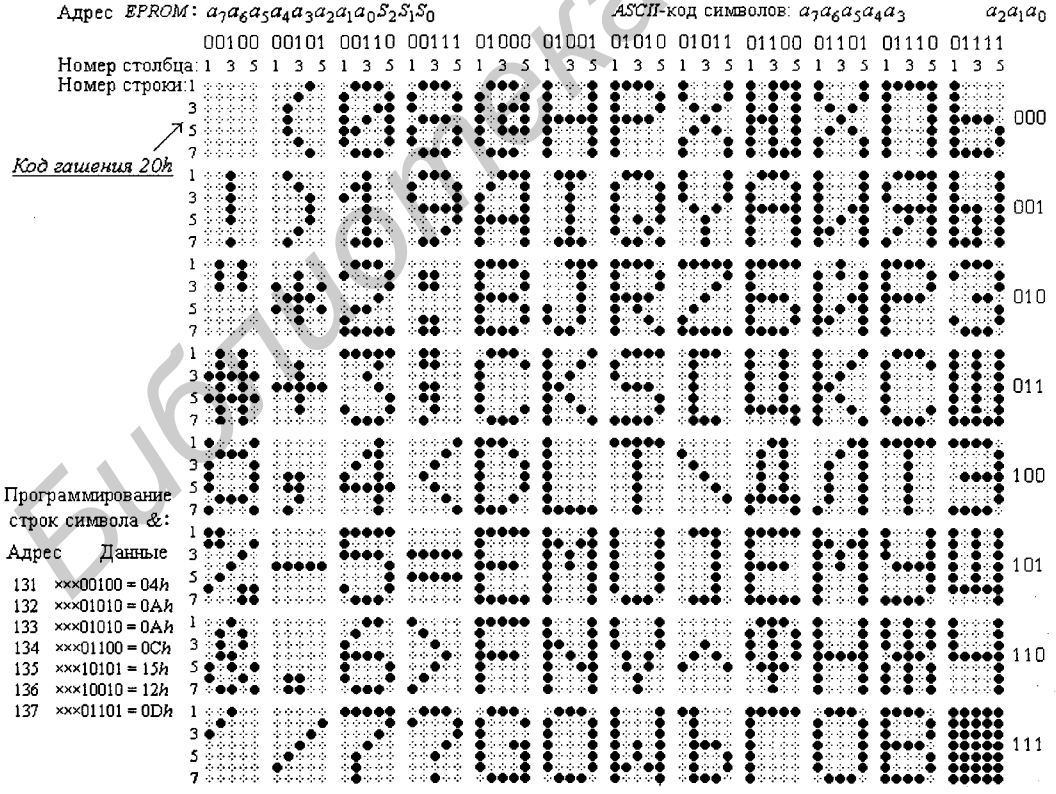


Рис. 3.146. Построчное кодирование отображаемых на дисплее символов

На рис. 3.146 изображена только половина символов для значения $a_7 = 0$. Строчные латинские и русские буквы задаются значением $a_7 = 1$, но часто достаточно использовать только первую (основную) часть символов. Структурные схемы управления матричными дисплеями приведены на рис. 3.147. Первая схема (рис. 3.147, *а*) требует больших затрат (10 ИС 564ИД1 для построения пяти демультиплексоров *DMX* 1 → 16 и 80 транзисторов КТ315Г для драйверов столбцов), чем вторая (рис. 3.147, *б* — только пять ИС 564ИД1 для построения пяти демультиплексоров *DMX* 1 → 8 и 40 транзисторов КТ315Г), поэтому принципиальная схема управления дисплеем, изображенная на рис. 3.144 и 3.145, *а* выполнена по второму варианту.

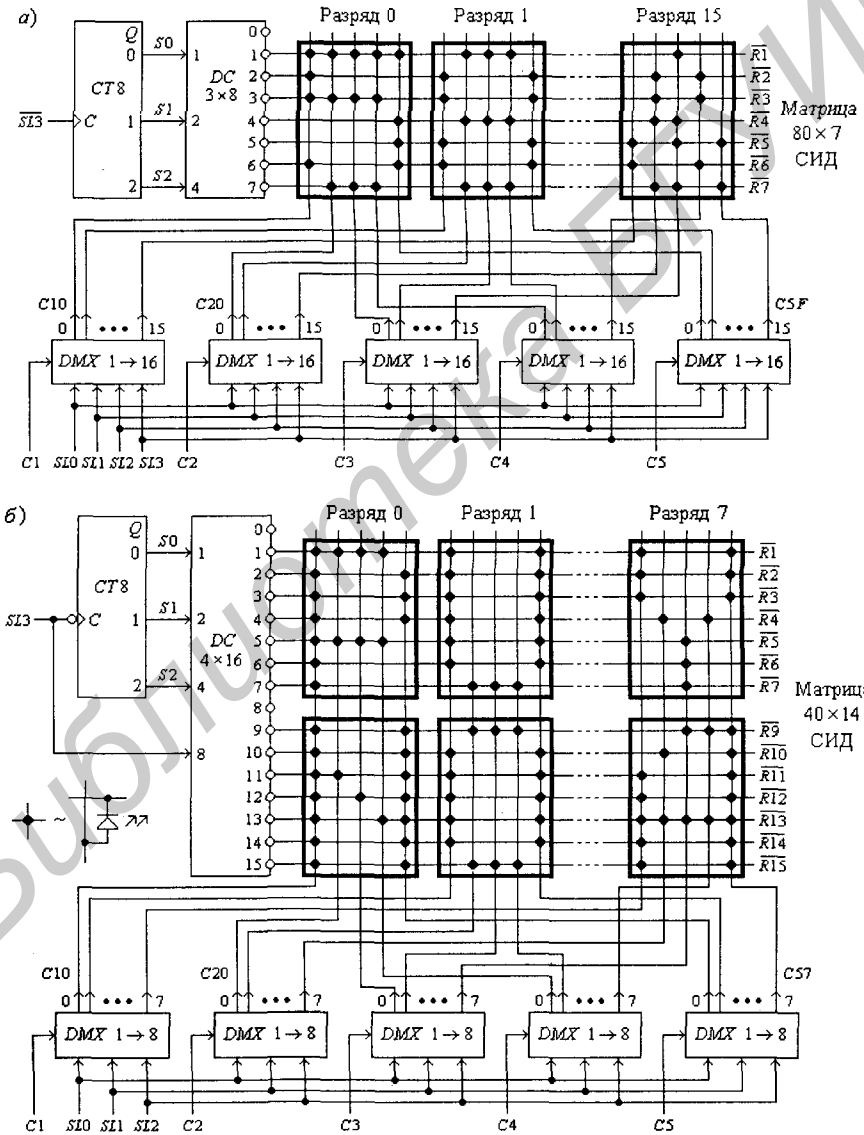


Рис. 3.147. Структурные схемы управления матричными дисплеями

Принцип работы знакогенератора и управление светоизлучающими диодами матричных индикаторов 5 × 7 поясняет рис. 3.148 (резисторы, ограничивающие ток, не нужны, а для уменьшения потребления тока специально использованы ИС 564ИД1, изготовляемые по КМОП технологии [5]).

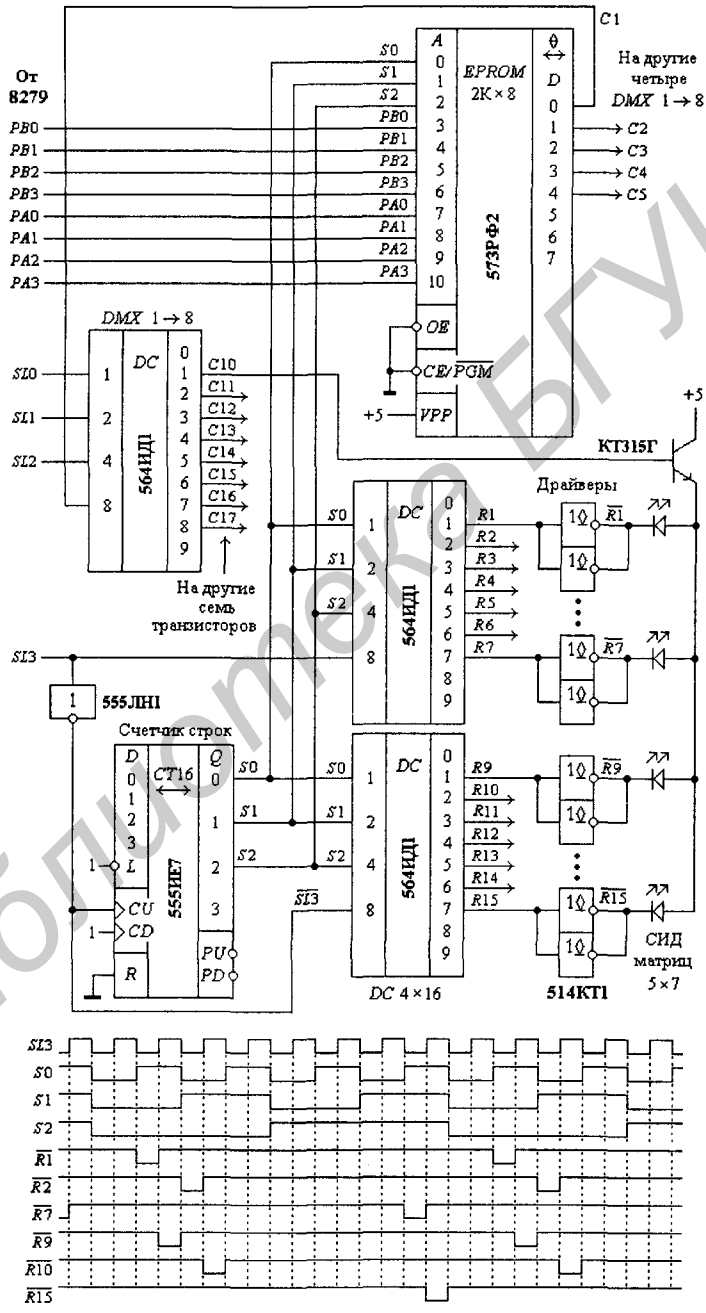


Рис. 3.148. Принципиальная схема знакогенератора

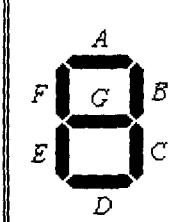
Строчная развертка символов производится так же, как и в телевизоре — сначала выводится первая строка R_1/R_9 всех 16 символов, затем вторая строка R_2/R_{10} и т. д. (строка R_0/R_8 пропускается), но вывод пяти значений столбцов производится одновременно. На рис. 3.145, *б* показан пример вывода на дисплей адреса и команды на языке ассемблера при запуске программы дисассемблера (*disassembler*), преобразующей машинные коды команд в их мнемонику (используется для просмотра программного обеспечения, хранящегося в *EPROM*).

Скважность динамической индикации алфавитно-цифрового дисплея $Q = 128$ (8 строк на 16 символов), поэтому драйверы должны обеспечивать значительно большие импульсные токи, чем в 7-сегментных дисплеях. Для полноценного и равномерного светоизлучения СИД всех столбцов недостаточно одного вентиля (драйвера) 514КТ1, поэтому необходимо использовать две ИС 514КТ1 с параллельной работой пар вентилей (у каждой пары объединяются входы и объединяются выходы — см. рис. 3.148). В одной ИС 514КТ1 (*DS*8872*N* фирмы *National Semiconductor Corp.*) имеется 9 вентилей с открытым коллекторным выходом, каждый из которых характеризуется импульсным током $I_{имп} \le 400$ мА при скважности $Q = 9$ и длительности импульса не более 500 мкс.

7-сегментные дисплеи. Контроллер клавиатуры и дисплея 8279 можно использовать для управления одним или двумя 7-сегментными 16-разрядными дисплеями. При управлении одним дисплеем преобразование 16-ричных чисел 0 ÷ 9, A ÷ F в 7-сегментный код целесообразно возложить на программное обеспечение МП-системы — на выходы $PA_{3-0}PB_{3-0}$ БИС 8279 будут выдаваться 7-сегментные коды чисел (семь из восьми разрядов байта $PA_{3-0}PB_{3-0}$). В этом случае для вывода на дисплей можно использовать и другие символы, заданные в исходном виде двоичными кодами (см. табл. 1.19 в § 1.9).

Четырехразрядные порты PA_{3-0} и PB_{3-0} БИС 8279 могут работать и независимо, но в этом случае выводимые данные могут быть представлены только в кодированной форме, непосредственно не отображаемой на дисплее (4-разрядными двоичными кодами). Режим независимой работы портов позволяет управлять двумя 7-сегментными 16-разрядными дисплеями при использовании двух внешних аппаратных преобразователей 4-разрядных двоичных чисел в 7-сегментные коды. В табл. 3.28 приведено преобразование двоично-десятичного кода, или кода 8–4–2–1, в 7-сегментный код, которое выполняют ИС, изображенные на рис. 3.149.

Таблица 3.28. **Таблица истинности преобразования *X*/7*S***

X	X_3	X_2	X_1	X_0	G	F	E	D	C	B	A	Символ	Сегменты
0	0	0	0	0	0	1	1	1	1	1	1	0	
1	0	0	0	1	0	0	0	0	1	1	0	1	
2	0	0	1	0	1	0	1	1	0	1	1	2	
3	0	0	1	1	1	0	0	1	1	1	1	3	
4	0	1	0	0	1	1	0	0	1	1	0	4	
5	0	1	0	1	1	1	0	1	1	0	1	5	
6	0	1	1	0	1	1	1	1	1	0	1	6	
7	0	1	1	1	0	0	0	0	1	1	1	7	
8	1	0	0	0	1	1	1	1	1	1	1	8	
9	1	0	0	1	1	1	0	1	1	1	1	9	

Сегменты
$S = A \div G$

Код 8–4–2–1
$X = X_3X_2X_1X_0$

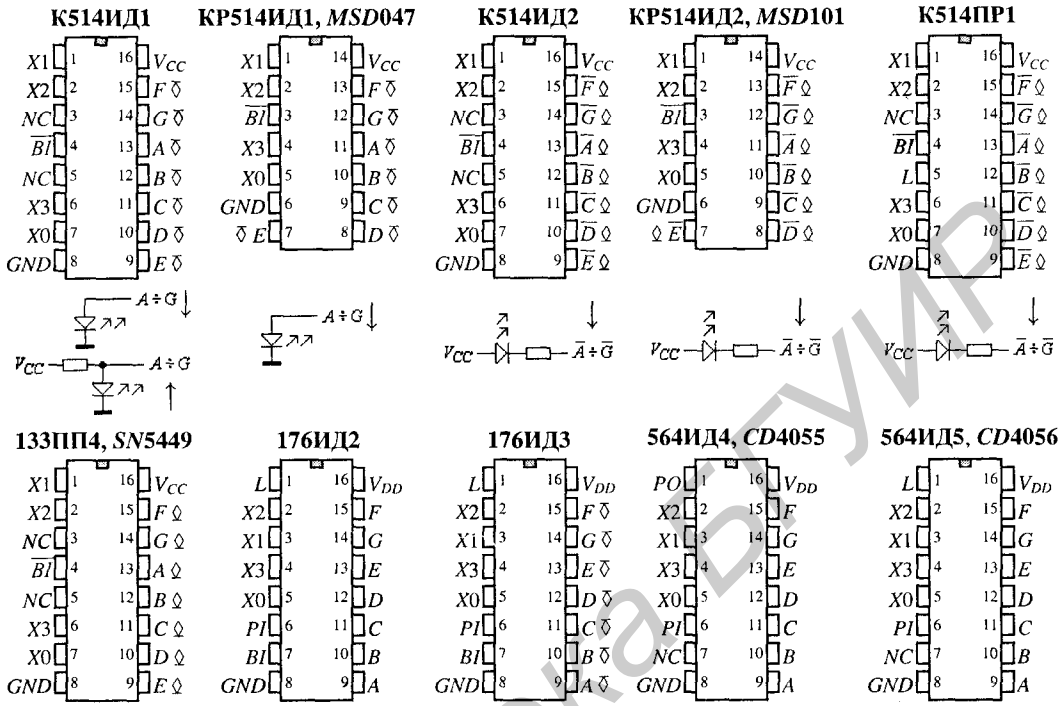


Рис. 3.149. Преобразователи *X*/7*S* кода 8–4–2–1 в 7-сегментный код

Интегральные схемы 176ИД2, 176ИД3, 564ИД4 и 564ИД5 изготовляются по КМОП технологии и предназначены для управления 7-сегментными жидкокристаллическими индикаторами (*LCD — liquid-crystal display*), а остальные ИС — по ТТЛ технологии и используются для управления 7-сегментными СИД-индикаторами. Эти ИС дополнительно имеют от одного до трех входов для подачи сигналов управления ($S = A \div G$ — сигналы управления сегментами):

$\overline{BI}$ (*Blanking Input*) — сигнал гашения 7-сегментного индикатора, который можно использовать для импульсного управления его яркостью ($S = \overline{BI} \cdot S'$, где $S = A \div G$ — выходные сигналы ИС, $S' = A' \div G'$ — внутренние сигналы ИС, описываемые табл. 3.28, т. е. $\overline{BI} = 0 \Rightarrow S = 0$);

L (*Load*) — сигнал загрузки кода $X_{3\text{-}0}$ во внутренний 4-разрядный асинхронный потенциальный регистр памяти, описываемый функциями $Q_r^+ = D_r L \vee Q_r \overline{L}$, $D_r = X_r$, $r = 0, 1, 2, 3$ (если задать $L \equiv 1$, то $Q_r^+ = X_r$ — "прозрачный" регистр памяти);

PI (*Polarity Input*) — сигнал коммутации полярности выходных сигналов $A \div G$, используемый для формирования переменного напряжения управления сегментами жидкокристаллических индикаторов ($S = \overline{BI} \cdot S' \oplus PI$ или $S = S' \oplus PI$, где $S = A \div G$ — выходные сигналы ИС, $S' = A' \div G'$ — внутренние сигналы ИС, описываемые табл. 3.28). Допустимая частота сигнала *PI* заключена в пределах 30 ... 1000 Гц (рабочая частота выбирается в соответствии с используемым типом жидкокристаллического индикатора).

Приведенные на рис. 3.149 ИС имеют назначение:

514ИД1 (*MSD*047 фирмы *Monsanto Commercial Products*) — преобразователь *X*/7*S* с прямыми открытыми эмиттерными выходами $A \div G$ для СИД-индикаторов с общим катодом;

514ИД2 (*MSD*101 фирмы *Monsanto Commercial Products*) — преобразователи *X*/7*S* с инверсными открытыми коллекторными выходами $\overline{A} \div \overline{G}$ для СИД-индикаторов с общим анодом;

514ПР1 — преобразователь *X/7S* с инверсными открытыми коллекторными выходами $\overline{A} \div \overline{G}$ для СИД-индикаторов с общим анодом;

133ПП4 (*SN*5449 фирмы *Texas Instruments*) — преобразователь *X/7S* с прямыми открытыми коллекторными выходами $A \div G$ для СИД-индикаторов с общим катодом;

176ИД2 — преобразователь *X/7S* для жидкокристаллических индикаторов (V_{DD} = +9 В);

176ИД3 — преобразователь *X/7S* с открытыми истоковыми выходами $A \div G$ для жидкокристаллических и люминесцентных индикаторов (V_{DD} = +9 В). Допустимое напряжение между контактами V_{DD} и $A \div G$ равно 30 В, что необходимо для управления люминесцентными индикаторами (на контакты $A \div G$ через резисторы подается высокое отрицательное напряжение);

564ИД4 (*CD*4055 фирмы *RCA Corp.*) — преобразователь *X/7S* для жидкокристаллических индикаторов (V_{DD} = +3 ... +15 В, *PO — Polarity Output* — сигнал *PI*, переданный на выход через повторитель);

564ИД5 (*CD*4056 фирмы *RCA Corp.*) — преобразователь *X/7S* для жидкокристаллических индикаторов (V_{DD} = +3 ... +15 В).

16-разрядный дисплей будет представлять собой матрицу 7 × 16 СИД. Выпускаются также ИС (например, ИС КР514ИД5), преобразующие 4-разрядные двоичные числа в 16-ричные символы 0 ÷ 9, A ÷ F в соответствии с рис. 3.145, *в*. Принципиальные схемы часов, построенных на жидкокристаллических, люминесцентных и СИД-индикаторах приведены в книге [8].

Программное управление БИС 8279. Для задания режимов работы и управления БИС в распоряжение программиста предоставлено 8 команд — CW_{7-0} (*Command Word* — разряды $D_7D_6D_5$ команд используются для адресации их получателя внутри БИС 8279):

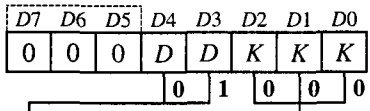
CW_0 — команда установки режима работы клавиатуры и дисплея (рис. 3.150). Если установлен режим сканирования клавиатуры с внутренней дешифрацией, то на дисплей выдаются только первые четыре символа ОЗУ дисплея независимо от значения поля *DD*;

CW_1 — команда программирования тактовой частоты (рис. 3.151). Для получения базовой частоты внутренней синхронизации f_{CY} = 100 кГц следует установить коэффициент деления прескалера $PPPPP = f_{CLK}/f_{CY}$ (см. рис. 3.140);

CW_2 — команда чтения буферной памяти клавиатуры или матрицы датчиков (рис. 3.152). Эта команда подается в контроллер 8279 перед чтением данных из *FIFO/Sensor RAM* последовательностью команд IN *port* при значении разряда адреса $A_0 = 0$ (см. табл. 3.26). Действие команды CW_2 остается в силе, пока не будет подана команда CW_3 (команды CW_2 и CW_3 определяют источник данных для чтения). В режиме сканирования клавиатуры флаг автоинкремента *AI* и адрес *AAA* игнорируются (*FIFO* — безадресная память и данные выдаются в том же порядке, в каком они были занесены в *FIFO* при нажатиях клавиш). В режиме сканирования матрицы датчиков адресные разряды *AAA* выбирают один из 8 байтов ОЗУ датчиков. Если флаг *AI* = 1, то при каждом чтении адрес автоматически увеличивается на 1;

CW_3 — команда чтения кода символа из ОЗУ дисплея (рис. 3.153). Эта команда подается в контроллер 8279 перед чтением данных из *Display RAM* последовательностью команд IN *port* при значении разряда адреса $A_0 = 0$ (см. табл. 3.26). Действие команды CW_3 остается в силе, пока не будет подана команда CW_2. Адресные разряды *AAAA* выбирают один из 16 байтов ОЗУ дисплея. Если флаг *AI* = 1, то при каждом обращении к ОЗУ дисплея (и при чтении, и при записи) адрес автоматически увеличивается на 1;

CW_4 — команда записи кода символа в ОЗУ дисплея (рис. 3.154). Эта команда задает значение флага *AI* и адреса *AAAA* для записи кодов символов в ОЗУ дисплея последовательностью команд OUT *port* при значении разряда адреса $A_0 = 0$ (см. табл. 3.26). Команда CW_4 не влияет на задание источника чтения данных (*FIFO/Sensor RAM* или *Display RAM*), но изменяет значение флага *AI* и адреса *AAAA* для чтения кодов символов из ОЗУ дисплея;



$D_7D_6D_5 = 000$ — ***Keyboard/Display Mode Set*** ***(CW_0)***

⇐ *RESET* = 1 (*Default after reset*)

000 — *Encoded Scan Keyboard, 2-Key Lockout*
(сканирование клавиатуры с внешней дешифрацией и блокировкой нажатия двух клавиш)

001 — *Decoded Scan Keyboard, 2-Key Lockout*
(сканирование клавиатуры с внутренней дешифрацией и блокировкой нажатия двух клавиш)

010 — *Encoded Scan Keyboard, N-Key Rollover*
(сканирование клавиатуры с внешней дешифрацией и одновременным нажатием *N* клавиш)

011 — *Decoded Scan Keyboard, N-Key Rollover*
(сканирование клавиатуры с внутренней дешифрацией и одновременным нажатием *N* клавиш)

100 — *Encoded Scan Sensor Matrix*
(сканирование матрицы датчиков с внешней дешифрацией)

101 — *Decoded Scan Sensor Matrix*
(сканирование матрицы датчиков с внутренней дешифрацией)

110 — *Strobed Input, Encoded Display Scan*
(стробируемый ввод при сканировании дисплея с внешней дешифрацией)

111 — *Strobed Input, Decoded Display Scan*
(стробируемый ввод при сканировании дисплея с внутренней дешифрацией)

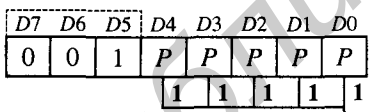
00 — 8 8-*bit character display, Left entry*
(8-разрядный дисплей с вводом 8-разрядных символов с левой стороны дисплея)

01 — 16 8-*bit character display, Left entry*
(16-разрядный дисплей с вводом 8-разрядных символов с левой стороны дисплея)

10 — 8 8-*bit character display, Right entry*
(8-разрядный дисплей с вводом 8-разрядных символов с правой стороны дисплея)

11 — 16 8-*bit character display, Right entry*
(16-разрядный дисплей с вводом 8-разрядных символов с правой стороны дисплея)

Рис. 3.150. Команда CW_0 установки режима работы клавиатуры и дисплея

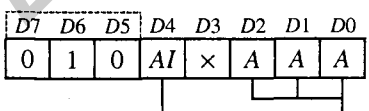


$D_7D_6D_5 = 001$ — ***Program Clock*** ***(CW_1)***

⇐ *RESET* = 1 (*Default after reset*)

PPPPP = 2 ÷ 31 — коэффициент деления частоты сигнала *CLK*

Рис. 3.151. Команда CW_1 программирования тактовой частоты



$D_7D_6D_5 = 010$ — ***Read FIFO/Sensor RAM*** ***(CW_2)***

AAA = 0 ÷ 7 — адрес буферной памяти ОЗУ датчиков

Auto-Increment flag — флаг автоинкремента адреса буферной памяти

Рис. 3.152. Команда CW_2 чтения буферной памяти клавиатуры и матрицы датчиков

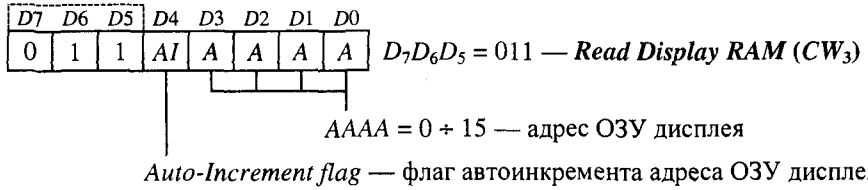


Рис. 3.153. Команда CW_3 чтения кода символа из ОЗУ дисплея

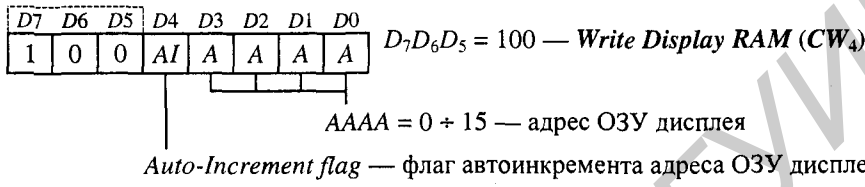


Рис. 3.154. Команда CW_4 записи кода символа в ОЗУ дисплея

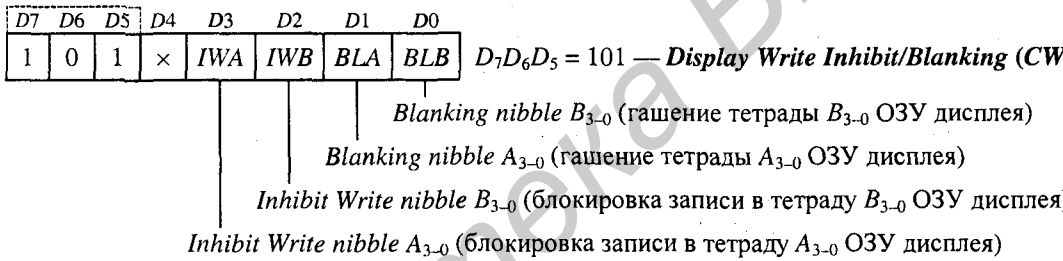


Рис. 3.155. Команда CW_5 блокировки записи данных и гашения ОЗУ дисплея

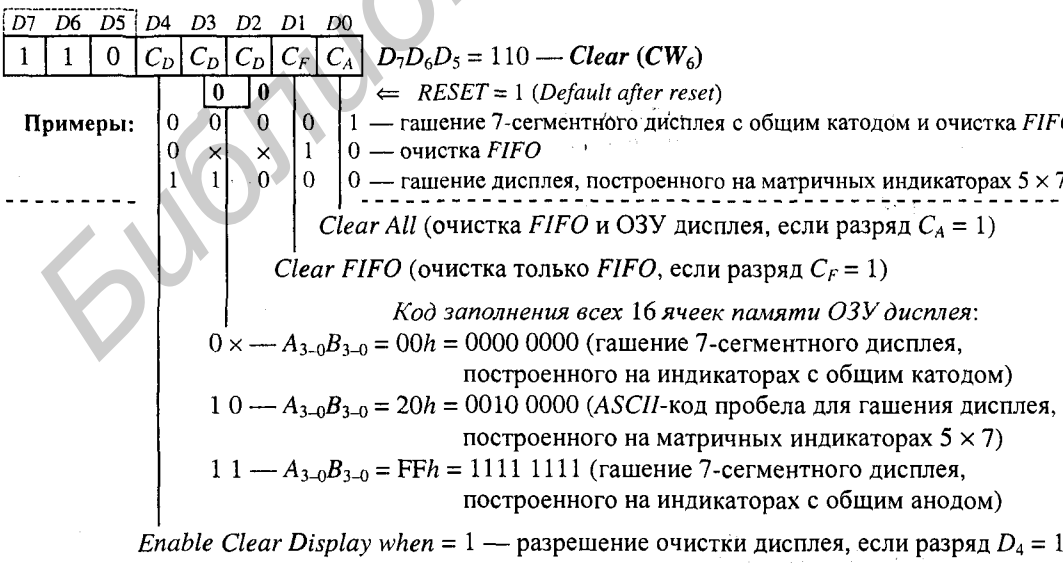


Рис. 3.156. Команда CW_6 очистки

CW_5 — команда блокировки записи данных и гашения ОЗУ дисплея (рис. 3.155). Разряды *IWA* и *IWB* используются для раздельного запрета записи в старшую A_{3-0} и младшую B_{3-0} тетрады ОЗУ дисплея, что позволяет производить независимую запись 4-разрядных кодов символов в ОЗУ дисплея при использовании двух 16-разрядных 7-сегментных дисплеев. Для раздельного гашения таких индикаторов предназначены флаги *BLA* и *BLB*. При использовании 8-разрядных кодов символов, а значит, и одного дисплея, для его гашения следует установить значения $BLA = 1$ и $BLB = 1$;

CW_6 — команда очистки (рис. 3.156). По этой команде при значении $D_4 \vee D_0 = 1$ во все 16 ячеек памяти ОЗУ дисплея записывается код гашения дисплея, заданный разрядами D_3D_2. Время записи в одну ячейку памяти равно 10 мкс, поэтому на время 160 мкс ОЗУ дисплея недоступно для записи данных, что фиксируется в слове состояния *FIFO* (см. рис. 3.158) значением разряда $D_7 = D_U = 1$. После окончания записи кода гашения разряд D_U автоматически устанавливается в 0. Этот разряд предназначен для квитирования вывода кодов символов в ОЗУ дисплея сразу после подачи команды очистки CW_6. Если задано значение разряда $D_1 = C_F = 1$, то выполняется очистка слова состояния *FIFO* (см. рис. 3.158), сбрасывается в 0 сигнал запроса прерывания *IRQ* и устанавливается указатель адреса ОЗУ датчиков на строку 0. Задание значения $D_0 = C_A = 1$ эквивалентно заданию значений разрядов $D_4 = D_1 = 1$ (при этом производится еще и перезапуск внутренних цепей синхронизации);

CW_7 — команда конца обработки прерывания и установки режима обнаружения ошибок (рис. 3.157). В режиме матрицы датчиков команда CW_7 при значении разряда $E = 1$ сбрасывает в 0 сигнал запроса прерывания *IRQ* и разрешает дальнейшую запись информации в ОЗУ датчиков (после обнаружения изменения состояния датчиков сигнал *IRQ* устанавливается в состояние 1 и запрещается запись в ОЗУ датчиков). В режиме независимого ввода кодов *N* одновременно нажатых клавиш команда CW_7 при значении разряда $E = 1$ включает специальный режим обнаружения ошибок (описание см. на с. 326).

Слово состояния *FIFO*. Слово состояния *FSW* (рис. 3.158) содержит информацию о состоянии *FIFO*, ошибках и запрещении записи в ОЗУ дисплея.

В режимах сканирования клавиатуры и стробируемого ввода слово состояния *FSW* используется для указания числа введенных в *FIFO* символов (разряды $D_2D_1D_0 = NNN = 0 \ldots 7$), полного *FIFO* (разряд $D_3 = 1$) и фиксации ошибок (разряд $D_4 = 1$ — чтение данных из пустого *FIFO*, разряд $D_5 = 1$ — запись данных в полное *FIFO*).

Разряд $D_7 = 1$ указывает на недоступность ОЗУ дисплея для записи данных до тех пор, пока команда CW_6 не завершит операции очистки *Clear Display* или *Clear All* (см. рис. 3.156).

В режиме сканирования матрицы датчиков разряд $D_6 = S/E = 1$ указывает на изменение показаний, по крайней мере, одного датчика в содержимом ОЗУ датчиков. В режиме обнаружения ошибок разряд $D_6 = S/E$ устанавливается в состояние 1 при одновременном нажатии (замыкании) нескольких клавиш.

Форматы данных. Формат байта данных в режиме сканирования клавиатуры, поступающий в *FIFO* при нажатии клавиши показан на рис. 3.159. Разряды $D_5D_4D_3$ принимают значение номера столбца, в котором обнаружена нажатая клавиша, а разряды $D_2D_1D_0$ — значение номера строки (линии возврата) RL_{7-0} этой клавиши (см. рис. 3.143). Номера столбцов определяются номерами сигналов M_{7-0}, получаемых декодированием линий сканирования SL_{2-0}. Например, клавиша, подключенная к столбцу M_5 и строке RL_2, будет иметь номер 101 010 = 2A*h* = 42*d* (см. рис. 3.145, *а*). Номера клавиш необходимо знать для дальнейшего их преобразования программным способом в *ASCII*-коды с учетом значений разрядов $D_7 = CNTL$ и $D_6 = SHIFT$, так как клавиши маркируются алфавитно-цифровыми символами для ввода в память МП-системы текстовой информации.

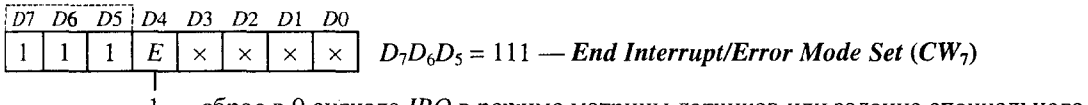


Рис. 3.157. Команда CW_7 конца обработки прерывания и установки режима обнаружения ошибок

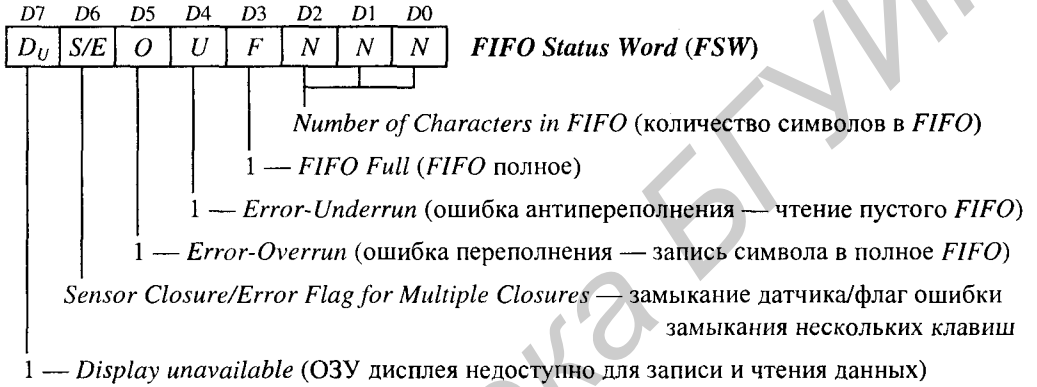


Рис. 3.158. Слово состояния *FIFO*

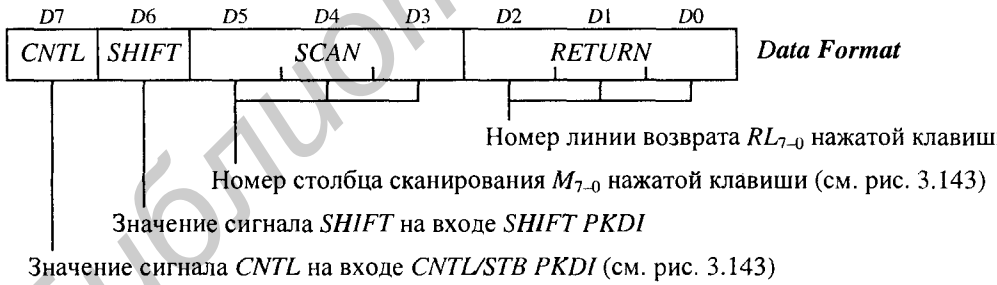


Рис. 3.159. Формат данных в режимах сканирования клавиатуры

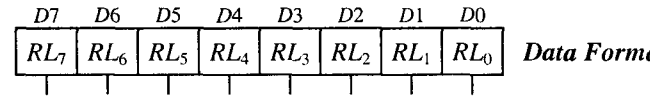


Рис. 3.160. Формат данных в режимах сканирования матрицы датчиков и стробируемого ввода

Формат данных в режимах сканирования матрицы датчиков и стробируемого ввода показан на рис. 3.160:

RL_{7-0} — байт данных опрашиваемого в данный момент столбца матрицы бинарных датчиков, записываемый в ОЗУ датчиков по адресу, равному номеру этого столбца (сигналы *CNTL* и *SHIFT* в режиме сканирования матрицы датчиков игнорируются, а в качестве матрицы датчиков к линиям возврата RL_{7-0} можно подключить восемь 8-канальных мультиплексоров, на адресные входы A_{2-0} которых должны быть поданы сигналы SL_{2-0} для сканирования каналов);

RL_{7-0} — байт данных на линиях RL_{7-0}, поступающий от внешнего устройства и записываемый в *FIFO* положительным фронтом сигнала *STB*, поданного на вход *CNTL/STB*.

Режим сканирования клавиатуры с блокировкой нажатия двух клавиш (*Scanned Keyboard Mode*, *2-Key Lockout*). При обнаружении нажатой клавиши включается схема подавления "дребезга" контактов (*debounce*) и производится обнаружение других нажатых клавиш в течение двух следующих циклов сканирования (время выполнения одного цикла сканирования клавиатуры равно 5,12 мс при базовой частоте f_{CY} = 100 кГц, а время выполнения цикла *debounce* равно 10,24 мс). Если других нажатых клавиш не обнаружено, то код нажатой клавиши (см. рис. 3.159) вводится в *FIFO* при ее отпускании. Если *FIFO* было пусто, то устанавливается значение сигнала запроса прерывания *IRQ* = 1, сообщающее МП о наличии в *FIFO* данных для ввода. Если *FIFO* было полно, то код нажатой клавиши в *FIFO* не вводится и в слове состояния *FSW* устанавливается в 1 значение флага переполнения *O* (см. рис. 3.158). Если будет нажато несколько клавиш с любым интервалом времени, то в *FIFO* вводится только код первой нажатой клавиши при условии, что она будет отпущена последней.

Если две клавиши нажаты в пределах цикла *debounce*, то это считается одновременным нажатием. В этом случае в *FIFO* вводится код последней отпущенной клавиши.

Режим сканирования клавиатуры с одновременным нажатием *N* клавиш (*Scanned Keyboard Mode*, *N-Key Rollover*). Каждая клавиша в этом режиме рассматривается независимо от всех остальных. При обнаружении нажатия клавиши ее состояние проверяется по истечении цикла *debounce*. Если клавиша все еще нажата, ее код вводится в *FIFO*.

Если одновременно нажаты несколько клавиш, то их коды будут введены в *FIFO* в последовательности, в которой они были обнаружены при сканировании.

Сканирование клавиатуры с обнаружением ошибок (*Scanned Keyboard — Special Error Modes*). Для предыдущего режима командой CW_7 можно задать специальный режим обнаружения ошибок. В этом случае при обнаружении в одном цикле *debounce* двух нажатых клавиш в слове состояния *FSW* устанавливается в 1 значение флага ошибки *S/E* (см. рис. 3.158), что блокирует запись в *FIFO* кодов нажатых клавиш и задает значение сигнала запроса прерывания *IRQ* = 1 (если оно еще не было установлено). Сброс флага ошибки *S/E* в 0 производится командой очистки CW_6 при значении разряда C_F = 1 (см. рис. 3.156).

Режим матрицы датчиков (*Sensor Matrix Mode*). В этом режиме схема *debounce* отключена. Состояние ключей датчиков (0 и 1) сенсорной матрицы 8×8 при сканировании непосредственно вводится в ОЗУ датчиков размера 8×8 (периодически байт за байтом, появляющимися при сканировании на линиях возврата RL_{7-0} — см. рис. 3.160). Таким образом, ОЗУ датчиков хранит образ состояний ключей датчиков сенсорной матрицы и непрерывно отслеживает его изменения (например, снабдив шахматную доску герконовыми магнитными датчиками, можно отслеживать перемещение всех фигур).

При любом изменении состояния датчиков, обнаруженном в течение цикла сканирования клавиатуры, устанавливается значение сигнала запроса прерывания *IRQ* = 1. Если флаг автоинкремента *AI* = 0, то первая же операция чтения данных из ОЗУ датчиков сбросит сигнал *IRQ* в 0. Если флаг автоинкремента *AI* = 1, то сигнал *IRQ* сбрасывается в 0 только командой CW_7

конца обработки прерывания (см. рис. 3.157). Изменение состояния нескольких датчиков в матрице при значениях сигналов $SL_2SL_1SL_0 = 000$ ($SL_0 = 0$ в декодированном режиме), может вызвать несколько прерываний (это же может вызвать и сброс БИС 8279 значением сигнала *RESET* = 1).

Режим работы дисплея с вводом слева (*Left Entry*). Это самый простой режим работы дисплея — каждый разряд дисплея непосредственно соответствует байту (или тетраде) в ОЗУ дисплея. Адрес 0 в ОЗУ дисплея соответствует крайнему левому разряду дисплея, а адрес 15 (или адрес 7 в 8-разрядном дисплее) — крайнему правому разряду дисплея (рис. 3.161). Ввод символов слева с автоинкрементом, начиная с адреса 0, последовательно заполняет слева направо всю строку дисплея (следует помнить, что автоинкремент адреса 15 дает адрес 0).

При выводе коротких сообщений по центру дисплея целесообразно предварительно гасить его (стирать предыдущее сообщение) командой очистки CW_6, а затем командой CW_4 задавать режим записи с автоинкрементом и необходимым начальным адресом.

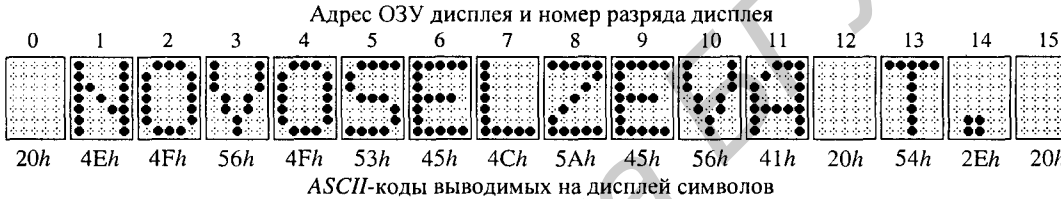


Рис. 3.161. Вывод сообщения на дисплей

Задача 1 (файл 3#10_01.asm). Для контроллера клавиатуры и дисплея, изображенного на рис. 3.144 и 3.145, *а*, произвести задание режимов работа клавиатуры (*Encoded Scan Keyboard, 2-Key Lockout*) и дисплея (16 *8-bit character display, Left entry*) с установкой модуля пересчета прескалера равным 26. Вывести на дисплей сообщение, приведенное на рис. 3.161, с предварительным гашением дисплея. *Решение*:

```
        defseg  d8279, start = 720h, class = Data          ; Data Segment
        seg     d8279
nov     db      ' NOVOSELZEVA T.', 0 ; 4E 4F 56 4F 53 45 4C 5A 45 56 41 20 54 2E — ASCII
                                     ; 0 — задает конец строки (на дисплей не выводится)
        defseg  IO_seg, start = 38h, class = IOspace       ; I/O Segment
        seg     IO_seg
dat     ds      1           ; dat — порт данных (38h)
com     ds      1           ; com — порт команд и слова состояния FIFO (39h)
        defseg  c_seg, start = 100h, class = Code          ; Code Segment
        seg     c_seg
        MVI     A, 8        ; A ← CW0 = 8 = 00001000 — режимы работы (см. рис. 3.150)
        OUT     com         ; 8279 ← CW0
        MVI     A, 26       ; A ← CW1 = 26d — модуля пересчета прескалера (см. рис. 3.151)
        OUT     com         ; 8279 ← CW1
        MVI     A, 0D8h     ; A ← CW6 = D9h = 11011000 — гашение дисплея (см. рис. 3.156)
        OUT     com         ; 8279 ← CW6
        MVI     A, 91h      ; A ← CW4 = 91h = 10010001 — AI = 1 (см. рис. 3.154)
        OUT     com         ; 8279 ← CW4  (запись с автоинкрементом с адреса 1)
```

```
        LXI   B, nov      ; BC ← 0720h — адрес строки выводимых на дисплей символов
L1:     IN    com         ; A ← FSW — слово состояния FIFO (см. рис. 3.158)
        ANI   80h         ; A ← D_U000 0000 — выделение флага D_U
        JNZ   L1          ; Переход, пока D_U = 1 — квитирование вывода
L3:     LDAX  B           ; A ← ASCII-код выводимого символа
        ANA   A           ; A ← A & A — обнаружение конца строки символов
        JZ    L2          ; Переход, если A = 0
        OUT   dat         ; 8279 ← символ строки NOVOSELZEVA T.
        INX   B
        JMP   L3
L2:     end               ; End Program
```

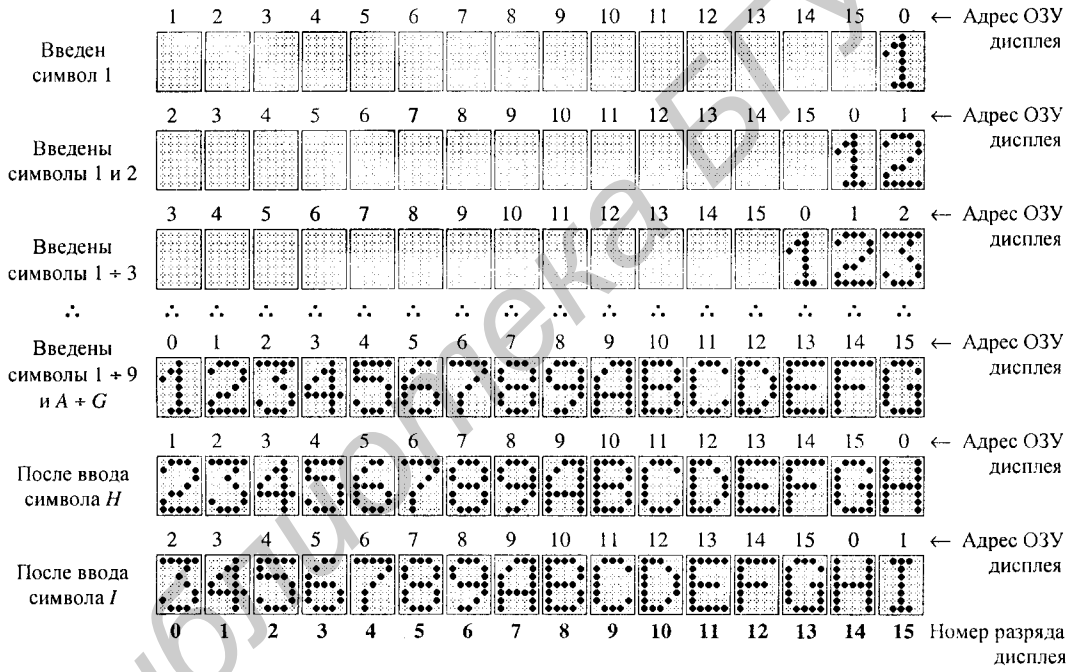


Рис. 3.162. Ввод символов справа

Режим работы дисплея с вводом справа (*Right Entry*). Этот режим используется в большинстве электронных калькуляторов. Каждый новый символ вводится в крайний правый разряд дисплея. Затем он сдвигается на один разряд влево следующим введенным символом и т. д. (рис. 3.162). Таким образом, при вводе каждого нового символа все содержимое строки сдвигается на один разряд влево, а крайний левый символ теряется.

В режиме ввода справа адрес ОЗУ дисплея и номер разряда дисплея не совпадают. Следовательно, запись байта в ОЗУ дисплея по некоторому заданному адресу в режиме автоинкремента может дать самый непредсказуемый результат. Для ввода справа последовательности символов рекомендуется использовать стартовый адрес 0.

Если запись символов в ОЗУ дисплея производить с низкой частотой (порядка 2 ÷ 4 Гц), то сообщения на дисплей будут выводиться в режиме "бегущей" строки.

Глава 4

МИКРОПРОЦЕССОРЫ 8086/8088 И СОПРОЦЕССОР 8087

4.1. Структурная схема МП 8086

Микропроцессор 8086 фирмы *Intel* (отечественный аналог 1810ВМ86) имеет 16-разрядную шину данных, содержит на кристалле около 29000 транзисторов и изготовляется по высококачественной *n*-канальной *HMOS*-технологии. В МП используется одно напряжение питания +5 В и однофазная синхронизация (сигнал *CLK*): 5 МГц в модели 8086, 8 МГц в модели 8086-2 и 10 МГц в модели 8086-1 (рис. 4.1). Выпущены также версии **80*C*86** (5 МГц) **и 80*C*86-2** (8 МГц), изготовляемые по *CHMOS* технологии. Производительность этих МП значительно выше производительности 8-разрядных МП 8080/8085. Хотя и имеется определенная совместимость архитектуры МП 8086 с архитектурой МП 8085, разработчики не ставили перед собой цели достичь ее полностью. Число линий адреса увеличено с 16 до 20, что позволяет адресовать 1 Мбайт памяти (2^{20} = 1 М) вместо 64 Кбайт. Увеличение объема памяти обеспечивает переход к мультипрограммированию, поэтому в МП 8086 предусмотрено несколько мультипрограммных возможностей. Кроме того, в МП 8086 встроены некоторые средства, упрощающие реализацию мультипроцессорных систем, что позволяет применять его с другими процессорами, например с арифметическим сопроцессором 8087 — *NDCP* (*Numeric Data CoProcessor*).

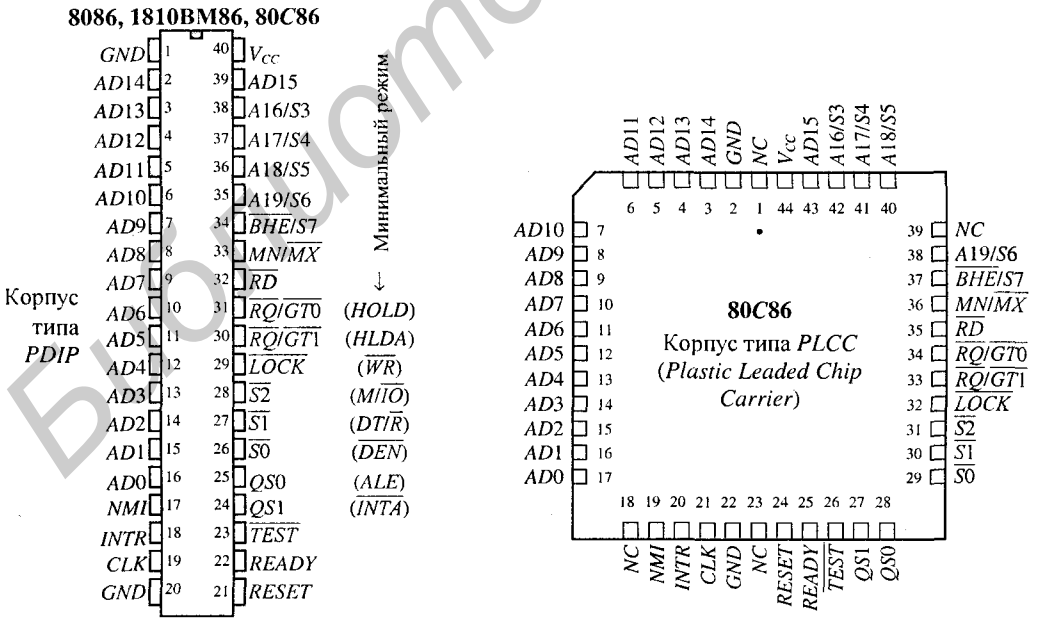


Рис. 4.1. Микропроцессоры семейства 8086

Максимальная рассеиваемая мощность составляет 2,5 Вт для МП 8086/8086-2/8086-1 и 1,0 Вт для МП 80*C*86/80*C*86-2. Токи потребления и параметры выходных сигналов характеризуются значения:

$I_{CC\,max}$ = 340/350/360 мА — 8086/8086-2/8086-1; I_{CC} = 10 мА/МГц — 80*C*86/80*C*86-2;

$V_{OL\,max}$ = 0,45 В при I_{OL} = 2,5 мА, $V_{OH\,min}$ = 2,4 В при I_{OH} = –0,4 мА — 8086/8086-2/8086-1;

$V_{OL\,max}$ = 0,40 В при I_{OL} = 2,5 мА и $V_{OH\,min}$ = 3,0 В при I_{OH} = –2,5 мА — 80*C*86/80*C*86-2.

Микропроцессор 8088 имеет 8-разрядную внешнюю шину данных при 16-разрядной внутренней шине, и программно совместим с МП 8086. Микропроцессор 8088 удобен для модернизации аппаратных конфигураций на базе МП 8080/8085 при переходе на более мощную систему команд МП 8086 [12, 14].

Структурная схема МП 8086. На рис. 4.2, *а* изображена структурная схема МП, состоящая из двух относительно независимых устройств: исполнительного устройства (*Execution Unit*) и шинного интерфейса (*Bus Interface Unit*). В исполнительном устройстве реализуются основные функции по обработке данных. Оно состоит из регистров данных, указательных и индексных регистров (*Data, Pointer and Index Regs*), арифметическо-логического устройства (*16-Bit ALU — Arithmetic and Logic Unit*) и регистра признаков (*Flags*). Исполнительное устройство принимает из шинного интерфейса предварительно извлеченные команды и возвращает в него несмещенные адреса операндов. Затем оно получает через шинный интерфейс находящиеся в памяти операнды и возвращает полученный результат для записи в память. Функция шинного интерфейса заключается в обеспечении максимально возможной пропускной способности шины: производится упреждающая выборка команд с размещением их в 6-байтном регистре очереди команд (*6-Byte Instruction Queue*) и реализуются функции, связанные с чтением и записью операндов в память, модификацией (перемещением) адресов и управлением шинами. Все интерфейсные операции выполняются параллельно с процессом обработки данных.

Основные узлы, входящие в состав МП 8086, изображены на рис. 4.2, *б*:

AX, *BX*, *CX*, *DX* — 16-разрядные *регистры общего назначения* (РОНы), которые могут использоваться и как 8-разрядные регистры данных *AH* и *AL*, *BH* и *BL*, *CH* и *CL*, *DH* и *DL* (*H* — *High* — старший байт и *L* — *Low* — младший байт 16-разрядных РОНов *AX*, *BX*, *CX*, *DX*);

IP, *SP*, *BP*, *SI*, *DI* — *указательные и индексные регистры*, используемые для адресации памяти. За исключением регистра *IP*, любой из этих регистров можно использовать для хранения 16-разрядных операндов;

CS, *SS*, *DS*, *ES* — *сегментные регистры*, используемые только для адресации памяти;

ALU — 16-разрядное *арифметическо-логическое устройство* (АЛУ);

Flags (*PSW — Program Status Word*) — *регистр признаков* (слово состояния программы);

Control Logic — схема управления;

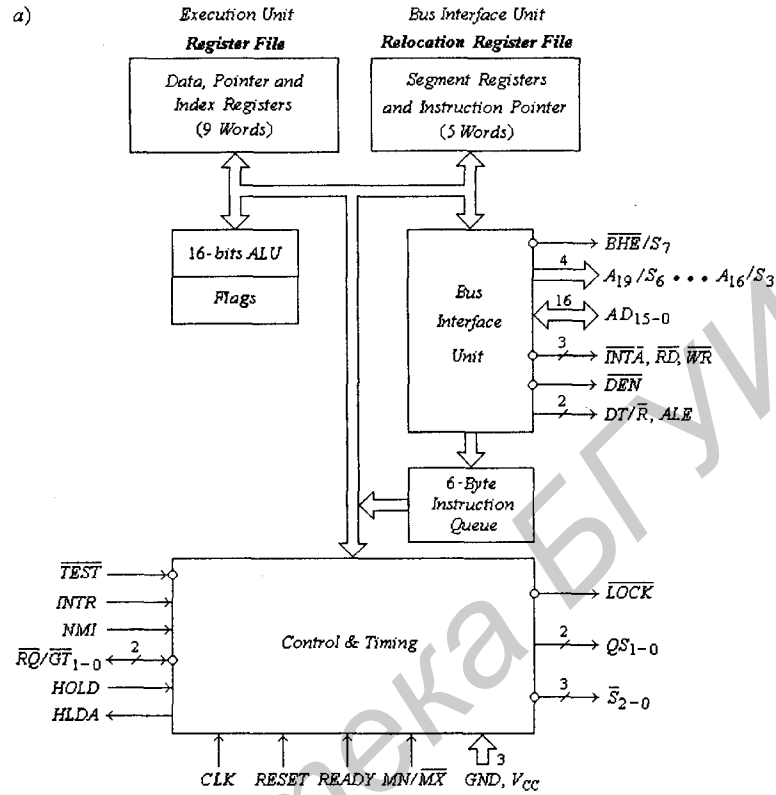
6 *Byte Instruction Queue* — 6-байтная *очередь команд*, выбираемых из памяти с опережением по отношению к их декодированию и исполнению. Очередь команд представляет собой память типа *FIFO* 6 × 8 бит (*first-in, first-out* — первым вошел, первым вышел; см. § 2.6).

Слово состояния программы *PSW* (регистр признаков) содержит 16 разрядов, но используется только 9 флагов (рис. 4.2, *в*). Флаги МП 8086 разделяются на *условные* (или *флаги условий*), отражающие результат предыдущей операции АЛУ, и *управляющие* (или *флаги управления*), от которых зависит выполнение специальных функций:

SF (*Sign Flag*) — флаг знака (равен старшему разряду результата операции, представленного в дополнительном коде: *SF* = 0 — число положительное, *SF* = 1 — число отрицательное);

ZF (*Zero Flag*) — флаг нуля (*ZF* = 1 при получении нулевого результата и *ZF* = 0, если результат отличается от нуля);

PF (*Parity Flag*) — флаг паритета (*PF* = 1, если младшие 8 бит результата содержат четное число единиц и *PF* = 0 в противном случае);



в)

Flags (PSW — Program Status Word) — регистр признаков

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	1	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF
								← Флаги МП 8080/8085 →							
–	–	–	–	–	–	INTE/IE	–	S	Z	0	AC	0	P	1	CY

Рис. 4.2. Структурная схема и программная модель МП 8086

CF (*Carry Flag*) — флаг переноса (*CF* = 1, если при сложении или вычитании возникает перенос или заем, и *CF* = 0 в противном случае; другие команды также воздействуют на этот флаг);

AF (*Auxiliary Carry Flag*) — флаг вспомогательного переноса (*AF* = 1, если при сложении или вычитании возникает перенос или заем из разряда 3, и *AF* = 0 в противном случае; флаг предназначен только для двоично-десятичной арифметики);

OF (*Overflow Flag*) — флаг переполнения. Флаг *OF* устанавливается в 1, если возникло переполнение, т. е. полученный результат вышел из допустимого диапазона представления чисел в дополнительном коде. При сложении флаг *OF* устанавливается в 1, если имеется перенос в старший бит и нет переноса из старшего бита, или наоборот. При вычитании флаг *OF* устанавливается в 1, когда возникает заем из старшего бита, но заем в старший бит отсутствует, или наоборот;

DF (*Direction Flag*) — флаг направления (применяется в командах манипуляций цепочками данных; если флаг *DF* = 0, цепочка обрабатывается с первого элемента, имеющего наименьший адрес; если же флаг *DF* = 1, цепочка обрабатывается от наибольшего адреса к наименьшему);

IF (*Interrupt Enable Flag*) — флаг разрешения прерываний, управляемый командами разрешения STI и запрета CLI прерываний (*IF* = 1 — прерывания разрешены; *IF* = 0 — прерывания запрещены; этот флаг по назначению эквивалентен сигналу *INTE* МП 8080 и флагу *IE* МП 8085);

TF (*Trap Flag*) — флаг трассировки (если *TF* = 1, то после выполнения каждой команды генерируется внутреннее прерывание для выполнения программы в пошаговом режиме).

Последние три из описанных флагов принадлежат к группе флагов управления. Младший байт в *PSW* соответствует 8-разрядному регистру признаков МП 8080/8085 и содержит все флаги условий, кроме флага переполнения *OF*.

В табл. 4.1 показано соответствие регистров МП 8086 и 8080/8085 с точки зрения их программной совместимости. Регистры *AX*, *BX*, *CX* и *DX* предназначены для хранения операндов и результатов операций и допускают адресацию не только целых регистров, но и их младшей и старшей половин. Регистры *BX*, *CX* и *DX* имеют и специальные назначения: *BX* служит базовым регистром в вычислениях адреса, *CX* в некоторых командах выступает неявным счетчиком, *DX* в некоторых операциях ввода-вывода содержит адрес порта ввода-вывода.

Таблица 4.1. **Соответствие регистров МП 8086 и 8080/8085**

Регистр МП 8086	*AL*	*BH*	*BL*	*CH*	*CL*	*DH*	*DL*	*SP*	*IP*	*PSW*
Регистр МП 8080	*A*	*H*	*L*	*B*	*C*	*D*	*E*	*SP*	*PC*	*F*

Указатель команды (инструкции) *IP* и регистр *SP* являются 16-разрядными программным счетчиком и указателем стека, но полные 20-разрядные *физические адреса* команды и стека образуются суммированием содержимого этих регистров и содержимого сегментных регистров *CS* и *SS*, сдвинутого на 4 разряда в сторону старших разрядов:

$$\text{физический адрес команды} = 16 \cdot CS + IP, \quad \text{физический адрес стека} = 16 \cdot SS + SP.$$

Пусть, например, содержимое сегментного регистра *CS* = 38BC*h*, а содержимое указателя команды *IP* = 7A53*h*. Тогда физический адрес команды будет равен $16 \cdot CS + IP$ = 38BC0 + + 7A53 = 40613*h*. Аналогичным образом используются и все остальные сегментные регистры.

Регистр *BP* является базовым при обращении к стеку и может использоваться с другими регистрами и (или) смещением, которое является частью команды. Регистры *SI* и *DI* предназначены для индексирования. Хотя их можно использовать сами по себе, они часто комбини-

руются с регистрами *BX* и *BP* и (или) смещением. Вычисление физических адресов памяти данных при выполнении команд может быть произведено, например, таким образом:

физический адрес = 16 · *SS* + *BP* + *SI* + *disp*8, физический адрес = 16 · *DS* + *BX* + *DI* + *disp*16,

где *disp*8 и *disp*16 — 8- и 16-разрядные смещения (*disp* — *displacement* — смещение), а вычисляемые 16-разрядные адреса *EA* = *BP* + *SI* + *disp*8 и *EA* = *BX* + *DI* + *disp*16 называются *эффективными адресами* (подробнее адресацию данных см. в § 4.2).

Применение сегментных регистров разделяет пространство памяти на сегменты, каждый из которых имеет размер 64 Кбайта и начинается на 16-байтной границе, называемой *границей параграфа*, т. е. любой сегмент всегда начинается с адреса, кратного 16. Содержимое сегментного регистра называется *сегментным адресом*, а сегментный адрес, умноженный на 16, — *начальным физическим сегментным адресом*, или просто *начальным* (*базовым*) *сегментным адресом*. Сегменты памяти данных, стека и кода могут как перекрываться, так и не перекрываться. Более того, все сегменты могут совпадать — иметь один и тот же базовый сегментный адрес. На рис. 4.3 изображено разделение памяти на не перекрывающиеся сегменты. Наличие сегментных регистров обеспечивает следующие преимущества:

максимальный объем адресуемой памяти равен 1 Мбайт, хотя команды оперируют 16-разрядными адресами;

секции кода (команд программы), данных и стека могут иметь длину более 64К байт благодаря использованию нескольких сегментов кода, данных и стека;

упрощается использование отдельных областей памяти для программы, ее данных и стека;

при каждой загрузке программы в память для ее выполнения она сама и (или) ее данные могут размещаться в различных областях памяти, т. е. сегментная структура памяти обеспечивает создание *позиционно-независимых* или *динамически перемещаемых программ*; перемещение программы в адресном пространстве памяти производится изменением содержимого только сегментных регистров без изменения программного кода команд переходов и вызова подпрограмм.

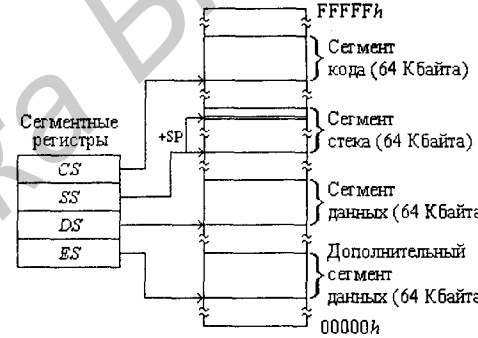


Рис. 4.3. Сегментная структура памяти

Общие сигналы для максимального и минимального режимов работы МП. Условное графическое обозначение МП 8086 с указанием обозначений сигналов при работе в *максимальном и минимальном режимах* приведено на рис. 4.4. Режим работы задает значение сигнала $MN/\overline{MX}$ (*Minimum*/*Maximum*): $MN/\overline{MX}$ = 0 — максимальный режим (вывод подключен к земле) и $MN/\overline{MX}$ = 1 — минимальный режим (вывод подключен к V_{CC} = +5 В). В зависимости от установленного режима изменяются функции восьми внешних выводов МП (выводов 24 ÷ 31). Под мультиплексную шину адреса-данных и адреса-состояния заняты 21 линия, а для внешнего управления микропроцессором и сигналов квитирования передач данных — 16 линий.

Сигналы, общие для максимального и минимального режимов работы МП 8086, имеют назначение:

AD_{15-0} (*Address Data Bus*) — мультиплексная шина адреса-данных. По этим двунаправленным линиям с разделением во времени передаются младшие 16 разрядов адреса памяти или полные адреса устройств ввода-вывода A_{15-0} и данные D_{15-0}. В первом такте (T_1) каждого цикла шины выдается адрес A_{15-0}, который необходимо зафиксировать во внешнем регистре сигналом *ALE*, а затем (такты T_2, T_3, T_W, T_4) принимаются или передаются данные D_{15-0} (см. рис. 4.5). Мультиплексирование адресов и данных во времени сокращает число контактов корпуса, но и

замедляет скорость передачи данных (благодаря тщательно разработанной временной диаграмме работы скорость передачи уменьшается не столь значительно, как этого можно было бы ожидать). При подтверждении прерывания (сигнал $\overline{INTA} = 0$) и захвата шин (сигнал $HLDA = 1$) линии AD_{15-0} переводятся в Z-состояние;

$A_{19}/S_6 \div A_{16}/S_3$ (*Address/Status*) — мультиплексные выходные сигналы адреса-состояния. В первом такте (T_1) на эти линии выводятся значения старших четырех разрядов адреса памяти A_{19-16}, а при адресации устройств ввода-вывода выдаются сигналы низкого уровня. В остальных тактах цикла шины (T_2, T_3, T_W, T_4) по этим линиям выдаются сигналы состояния S_{6-3}. Значения сигналов S_4 и S_3 идентифицируют сегментный регистр, участвующий в формировании физического адреса памяти в текущем машинном цикле (табл. 4.2). Сигнал S_5 соответствует состоянию флага прерывания *IF* в *PSW* (см. рис. 4.2, *в*). Сигнал S_5 обновляется в начале каждого цикла *CLK*. Сигнал S_6 всегда имеет низкий уровень. При подтверждении запроса *DMA* (захвата шин) линии $A_{19}/S_6 \div A_{16}/S_3$ переводятся в Z-состояние;

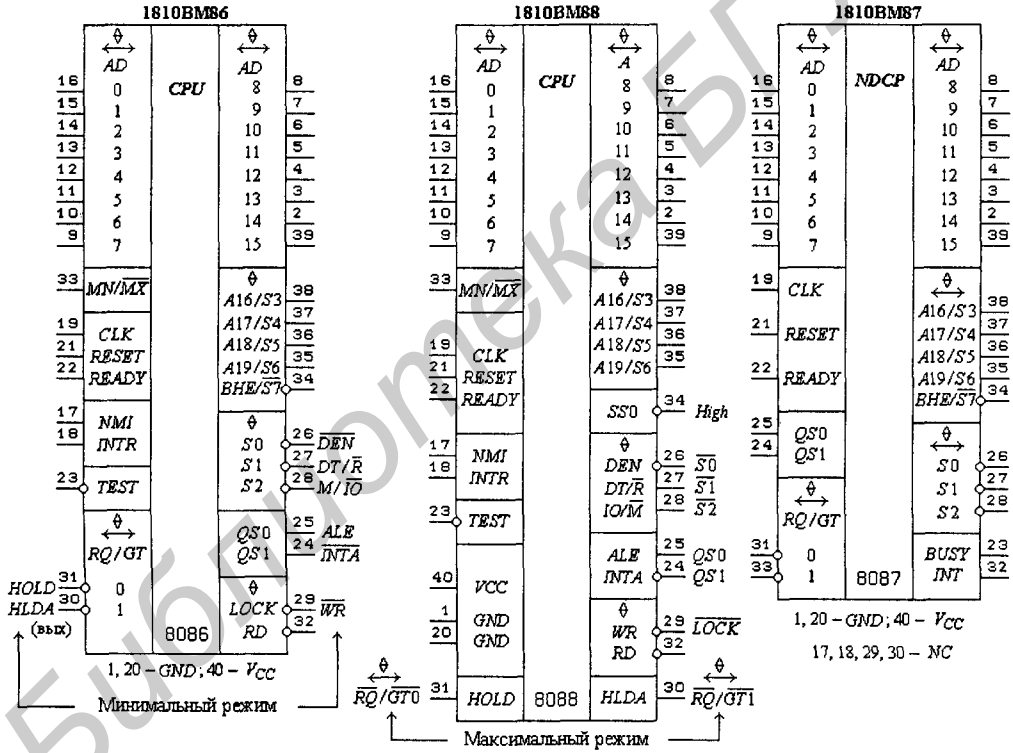


Рис. 4.4. Условные графические обозначения микропроцессоров серии 1810

Таблица 4.2. **Указатели сегментных регистров**

S_4	S_3	Сегмент
0	0	*ES*
0	1	*SS*
1	0	*CS* или никакой
1	1	*DS*

$\overline{BHE}/S_7$ (*Bus High Enable/Status*) — разрешение старшей части шины/состояние. В первом такте на эту линию выводится значение сигнала $\overline{BHE}$, а в остальных тактах цикла шины — сигнал состояния S_7. Сигнал $\overline{BHE}$ совместно с сигналом A_0 позволяет подключать одну часть памяти с байтной организацией к младшей половине шины AD_{15-0}, а другую часть — к старшей половине (табл. 4.3). Чтение слова, расположенного по четному адресу, производится за

одно обращение к памяти, а по нечетному адресу — за два обращения (сначала читается младший байт по верхней части шины, а затем старший байт по нижней части шины). Значения сигналов $\overline{BHE}$ и A_{19-16}, как и сигналов A_{15-0}, необходимо запоминать (фиксировать сигналом *ALE*) во внешнем регистре. При подтверждении запроса захвата шин линия $\overline{BHE}/S_7$ переводится в *Z*-состояние и $\overline{BHE} = 0$ в течение такта T_1 в первом цикле подтверждения прерывания;

Таблица 4.3. **Управление шиной данных**

$\overline{BHE}$	A_0	Операции запись/чтение
0	0	Целое слово (два байта) AD_{15-0}
0	1	Байт по верхней части шины *AD* (AD_{15-8}) / по нечетному адресу
1	0	Байт по нижней части шины *AD* (AD_{7-0}) / по четному адресу
1	1	Нет операций

$\overline{RD}$ (*Read*) — сигнал чтения устройств, подключенных к локальной шине МП. Сигнал $\overline{RD} = 0$ в течение тактов T_2, T_3 и T_W (данные из памяти или внешних устройств передаются по шине AD_{15-0} в МП). При подтверждении запроса захвата шин выход $\overline{RD}$ переводится в *Z*-состояние;

CLK (*Clock*) — тактовый сигнал от внешнего генератора для синхронизации работы МП (частота сигнала 5, 8 и 10 МГц в зависимости от модели МП). Оптимальная скважность сигнала *CLK* равна 3;

RESET — сигнал системного сброса, переводящий МП в определенное начальное состояние. Минимальная длительность значения сигнала *RESET* = 1 должна быть при первом включении МП не менее 50 мкс, а при повторном запуске — не менее четырех тактов синхронизации. При этом производится сброс (установка в 0) регистров *PSW* (флаг *IF* = 0 — прерывания запрещены), *IP*, *SS*, *DS*, *ES*, очереди команд и загрузка в сегментный регистр *CS* числа FFFF*h*, т. е. после сброса МП стартует с адреса памяти FFFF0*h* = 16 · *CS* + *IP*;

READY — сигнал готовности, приостанавливающий работу МП при значении *READY* = 0. Этот сигнал используется в интерфейсах устройств ввода-вывода и памяти, быстродействия которых недостаточно для синхронной работы с МП. Значение сигнала *READY* = 1 устанавливает адресуемое устройство после окончания им передачи или приема данных;

$\overline{TEST}$ — сигнал проверки освобождения шины данных другим процессором в мультипроцессорных системах. Этот сигнал используется вместе с командой ожидания WAIT. Выполнив команду WAIT, МП проверяет уровень сигнала $\overline{TEST}$: если значение $\overline{TEST} = 1$, то МП вводит холостые состояния и периодически проверяет уровень сигнала $\overline{TEST}$ с интервалом в пять тактов синхронизации, а если значение сигнала $\overline{TEST} = 0$, МП переходит к выполнению следующей по порядку команды. Таким образом, команда WAIT и сигнал $\overline{TEST}$ обеспечивают синхронизацию действий МП с внешними событиями;

NMI (*Non-maskable Interrupt*) — сигнал запроса по положительному фронту немаскируемого прерывания типа 2 (*type* = 2 — см. рис. 4.26, с. 398). Запрос прерывания всегда удовлетворяется по завершении выполнения текущей команды независимо от того, разрешены прерывания или нет (независимо от значения флага *IF*). Для прерывания типа 2 адрес (вектор) подпрограммы обслуживания прерывания (CS:IP) должен быть записан в четыре ячейки памяти, начиная с адреса 4 × *type* = 8. Немаскируемые прерывания предназначены для сигнализации о некоторых катастрофических событиях, например аварийном отключении сетевого питания, ошибке в памяти и др.;

INTR (*Interrupt Request*) — сигнал запроса прерывания высоким уровнем с фиксацией во внутреннем триггере. МП проверяет состояние этого триггера в последнем такте выполнения

каждой команды. При наличии запроса прерывания (*INTR* = 1) МП формирует цикл подтверждения прерывания, в результате которого он по шине данных получает байт типа (*type*) прерывания, идентифицирующий устройство, запросившее обслуживание. Адрес подпрограммы обслуживания прерывания (CS:IP) должен быть записан в четыре ячейки памяти, начиная с адреса 4 × *type*. Программной установкой значений 0 и 1 флага *IF* можно запрещать или разрешать обслуживание прерываний. Обычно на вход *INTR* подается сигнал *INT* с выхода контроллера прерываний 1810ВН59А (8259*A* фирмы *Intel*).

Циклом шины называется одно обращение МП 8086 к шине данных. В этом смысле цикл шины эквивалентен машинному циклу в МП 8080/8085. Цикл шины состоит из тактов T_1, T_2, T_3, T_W и T_4. Такты ожидания T_W вводятся между тактами T_3 и T_4 только в том случае, если в такте T_3 значение сигнала готовности *READY* = 0. Число введенных тактов ожидания T_W определяется длительностью значения сигнала *READY* = 0.

Сигналы МП 8086 в минимальном режиме работы. Сигналы на 8 выводах МП при работе в минимальном режиме имеют назначение:

M/IO (*Status line*) — сигнал управления обращения к памяти (*M/IO* = 1) и внешним устройствам (*M/IO* = 0). Сигнал *M/IO* логически эквивалентен сигналу состояния $\overline{S}_2$ в максимальном режиме работы МП. Истинное значение сигнала *M/IO* устанавливается в конце такта T_4 предыдущего цикла шины и удерживается до конца такта T_4 текущего цикла шины (рис. 4.5). Для увеличения нагрузочной способности выхода *M/IO* значение сигнала *M/IO* можно фиксировать во внешнем регистре одновременно с фиксацией значений сигналов $\overline{BHE}$ и $A_{19\ 0}$. При подтверждении запроса захвата шин линия *M/IO* переводится в Z-состояние;

$\overline{WR}$ (*Write*) — сигнал записи данных в память при значении сигнала *M/IO* = 1 или внешнее устройство при *M/IO* = 0. Значение сигнала $\overline{WR}$ = 0 указывает, что шина $AD_{15\text{-}0}$ содержит записываемые в память или внешнее устройство данные (выводимые из МП данные — *Data Out* на рис. 4.5). Активный уровень сигнала $\overline{WR}$ = 0 микропроцессор выдает в тактах T_2, T_3 и T_W. При подтверждении запроса захвата шин выход $\overline{WR}$ переводится в Z-состояние;

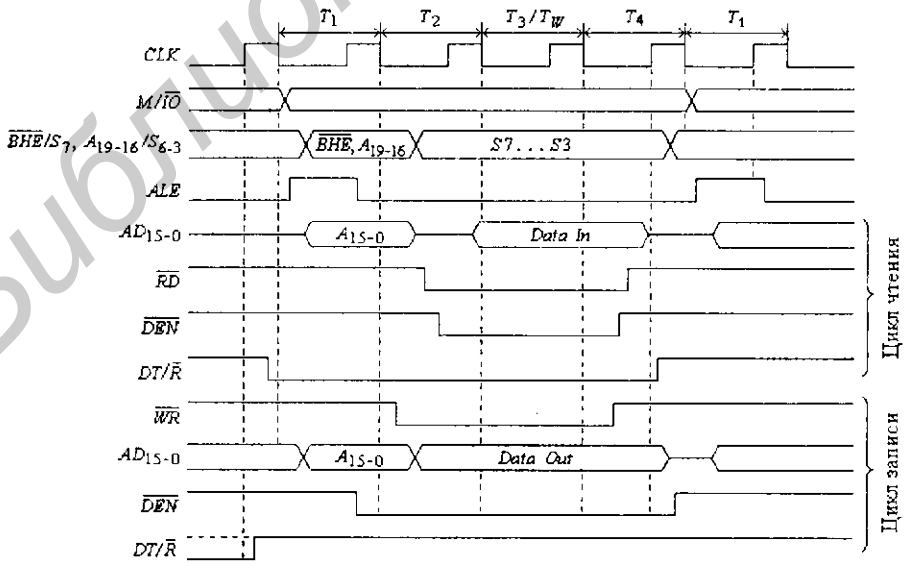


Рис. 4.5. Временны́е диаграммы для минимального режима работы МП 8086

$\overline{INTA}$ (*Interrupt Acknowledge*) — сигнал подтверждения прерывания. МП выдает активный уровень сигнала $\overline{INTA} = 0$ в тактах T_2, T_3 и T_W двух циклов подтверждения прерывания. Во втором цикле подтверждения прерывания из контроллера прерываний 8259*A* производится чтение байта *type* типа прерывания;

ALE (*Address Latch Enable*) — сигнал фиксации адресных сигналов $A_{19\text{-}0}$ и $\overline{BHE}$ во внешнем регистре (рис. 4.6; при необходимости можно фиксировать и значение сигнала $M/\overline{IO}$). Значение сигнала *ALE* = 1 в первом такте каждого цикла шины и он никогда не переводится в *Z*-состояние;

$DT/\overline{R}$ (*Data Transmit/Receive*) — сигнал управления направлением передачи данных ($DT/\overline{R} = 1$ — передача данных от МП в память и внешние устройства, $DT/\overline{R} = 0$ — прием данных микропроцессором от памяти и внешних устройств). Этот сигнал предназначен для управления шинными приемопередатчиками (рис. 4.6). Сигнал $DT/\overline{R}$ логически эквивалентен сигналу состояния $\overline{S}_1$ в максимальном режиме работы МП. Истинное значение сигнала $DT/\overline{R}$ устанавливается в конце такта T_4 предыдущего цикла шины и удерживается до конца такта T_4 текущего цикла шины (рис. 4.5). При подтверждении запроса захвата шин выход $DT/\overline{R}$ переводится в *Z*-состояние;

$\overline{DEN}$ (*Data Enable*) — *сигнал разрешения данных, предназначенный для включения шинных* приемопередатчиков (рис. 4.6). Значение сигнала $\overline{DEN} = 0$ при обращении к памяти и внешним устройствам, а также при выполнении циклов $\overline{INTA}$. При выполнении циклов чтения данных из памяти или внешних устройств и циклов $\overline{INTA}$ сигнал $\overline{DEN} = 0$ от середины такта T_2 до середины такта T_4 (рис. 4.5). При выполнении циклов записи сигнал $\overline{DEN} = 0$ с начала такта T_2 до середины такта T_4. При подтверждении запроса захвата шин выход $\overline{DEN}$ переводится в *Z*-состояние;

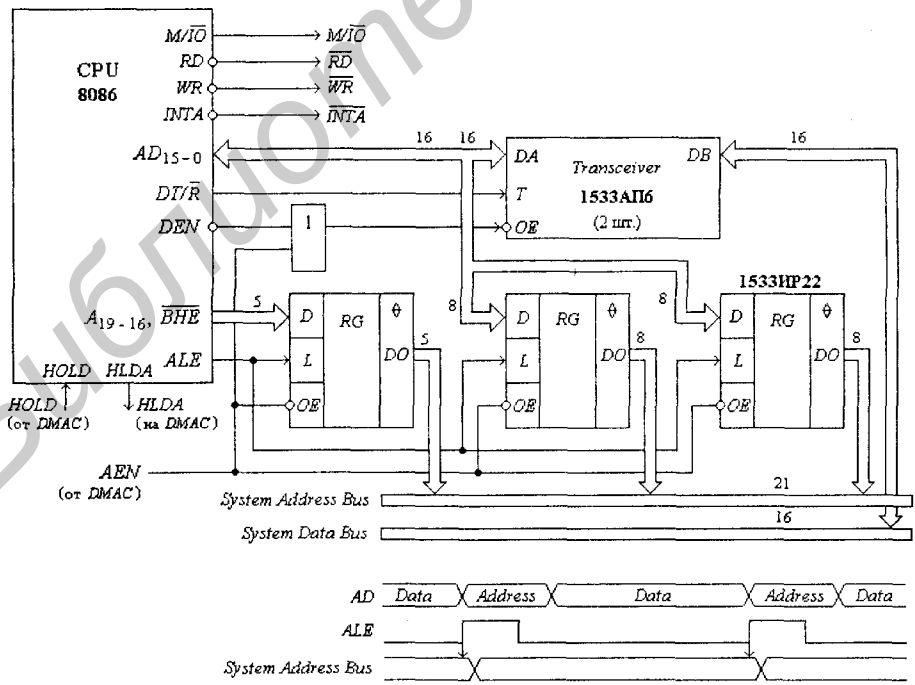


Рис. 4.6. Демультиплексирование и буферизация адреса и данных МП 8086

HOLD — сигнал запроса захвата шин, поступающий от другого устройства, способного управлять шинами МП-системы (например, от контроллера прямого доступа к памяти 8237). Значение *HOLD* = 1 должно удерживаться до тех пор, пока контроллеру требуется управление шинами;

HLDA (*Hold Acknowledge*) — сигнал подтверждения захвата шин, выдаваемый МП в ответ на значение сигнала *HOLD* = 1 с одновременным остановом выполнения программы и переводом своих шин в *Z*-состояние. Значение *HLDA* = 1 выдается в середине такта T_1. При переходе значения сигнала *HOLD* с 1 на 0 (при снятии запроса захвата шин) МП устанавливает значение сигнала *HLDA* = 0 и начинает снова управлять шинами (продолжает выполнение приостановленной программы).

На рис. 4.6 показана структурная схема МП-системы при работе МП в минимальном режиме с буферированными шинами адреса и данных и сигналами управления *IO*/$\overline{M}$, $\overline{RD}$, $\overline{WR}$ и $\overline{INTA}$. Фиксацию адресных сигналов в регистрах 1533ИР22 можно и не производить, если использовать память и внешние устройства, содержащие адресные регистры (см. БИС, описанные в § 3.9). В этом случае в небольших МП-системах можно отказаться и от приемопередатчика 1533АП6. Если ввод-вывод по прямому доступу к памяти не используется, то следует задать значения входных сигналов *HOLD* ≡ 0 и *AEN* ≡ 0.

Сигналы управления *IO*/$\overline{M}$, $\overline{RD}$ и $\overline{WR}$ можно преобразовать в четыре сигнала независимого управления чтением и записью данных в память и внешние устройства:

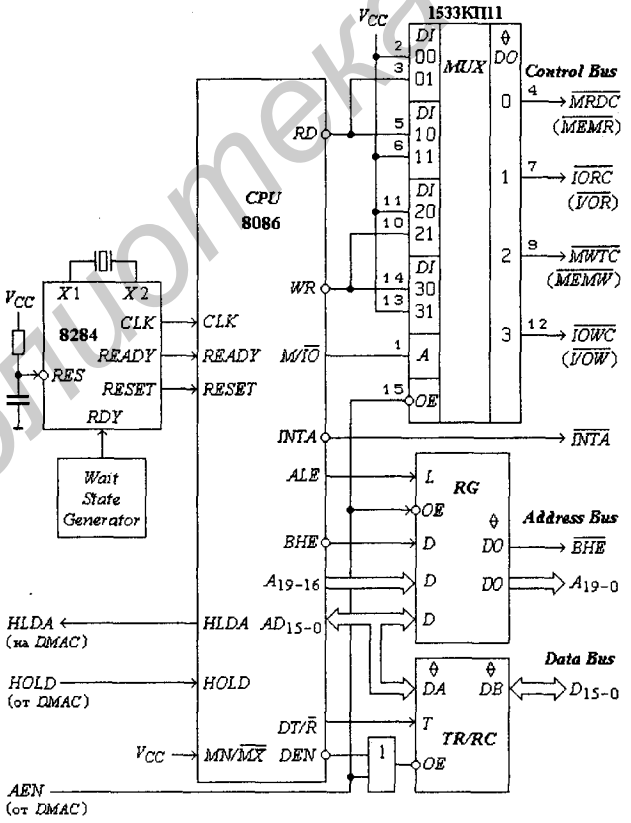


Рис. 4.7. Типовая структурная схема для минимального режима

$$\overline{MRDC} = \overline{RD \cdot M / \overline{IO}},\quad \overline{MWTC} = \overline{WR \cdot M / \overline{IO}},\quad \overline{IORC} = \overline{RD \cdot \overline{M / \overline{IO}}},\quad \overline{IOWC} = \overline{WR \cdot \overline{M / \overline{IO}}}$$

(эти сигналы аналогичны сигналам управления $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$ и $\overline{I/OW}$ МП 8080/8085). Данное преобразование легко выполнить с помощью четырехканального двухразрядного мультиплексора 1533КП11, как это показано на рис. 4.7. Например:

$$\overline{MRDC} = DO_0 = DI_{00} \cdot \overline{A} \vee DI_{01} \cdot A = 1 \cdot \overline{M / \overline{IO}} \vee \overline{RD} \cdot M / \overline{IO} = \overline{M / \overline{IO}} \vee \overline{RD} = \overline{RD \cdot M / \overline{IO}}.$$

Сигналы МП 8086 в максимальном режиме работы. Сигналы на 8 выводах МП при работе в максимальном режиме имеют назначение:

S_{2-0} (*Status*) — сигналы состояния, которые активны в тактах T_4, T_1 и T_2 каждого цикла шины (см. рис. 4.14) и переходят в пассивное состояние (111) в течение такта T_3 или T_W при значении сигнала *READY* = 1. Эти сигналы содержат информацию, определяющую тип цикла шины (табл. 4.4). Сигналы состояния S_{2-0} подаются на контроллер шин 8288 (1810ВГ88), который формирует полный набор системных сигналов управления памятью и внешними устройствами (рис. 4.8). Любое изменение сигналов S_{2-0} в такте T_4 определяет начало следующего цикла шины, а переход их в пассивное состояние (111) указывает конец цикла шины. При подтверждении запроса захвата шин выходы S_{2-0} переводятся в *Z*-состояние;

Таблица 4.4. **Системные сигналы управления**

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Состояние МП	Сигналы 8288	Назначение сигнала	8085
0	0	0	*Interrupt Acknowledge*	$\overline{INTA}$	*Interrupt Acknowledge* $\overline{INTA}$	$\overline{INTA}$
0	0	1	*Read I/O Port*	$\overline{IORC}$	*I/O Read Command*	$\overline{I/OR}$
0	1	0	*Write I/O Port*	$\overline{AIOWC}$, $\overline{IOWC}$	*I/O Write Command*	$\overline{I/OW}$
0	1	1	*Halt*	*None*		
1	0	0	*Code Access*	$\overline{MRDC}$	*Memory Read Command*	$\overline{MEMR}$
1	0	1	*Read Memory*	$\overline{MRDC}$	*Memory Read Command*	$\overline{MEMR}$
1	1	0	*Write Memory*	$\overline{AMWC}$, $\overline{MWTC}$	*Memory Write Command*	$\overline{MEMW}$
1	1	1	*Passive*	*None*		

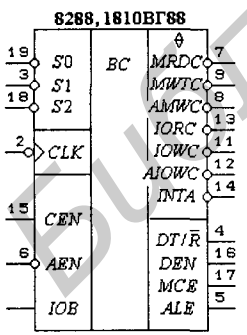


Рис. 4.8. Контроллер шин

Рис. 4.9. Временны́е диаграммы запроса и предоставления шины

$\overline{RQ}/\overline{GT}_{1-0}$ (*Request/Grant*) — двунаправленные линии запроса/предоставления шин, обеспечивающие бесконфликтное управление шинами несколькими ведущими устройствами, например, микропроцессором 8086, арифметическим сопроцессором 8087 и контроллером прямого доступа к памяти 8237. Линия $\overline{RQ}/\overline{GT}_0$ имеет более высокий приоритет, чем линия

$\overline{RQ}/\overline{GT_1}$, но если на линии $\overline{RQ}/\overline{GT_0}$ появляется запрос в то время, когда МП обрабатывает предшествующий запрос от $\overline{RQ}/\overline{GT_1}$, то запрос $\overline{RQ}/\overline{GT_0}$ не подтверждается до освобождения шины по линии $\overline{RQ}/\overline{GT_1}$. Управление запросом и предоставлением шин производится последовательностью из трех импульсов низкого уровня, длительность которого равна одному периоду тактового сигнала *CLK* (рис. 4.9). МП предоставляет управление шинами другому ведущему шины в конце текущего цикла шины, переводя свои шины в *Z*-состояние. Линии $\overline{RQ}/\overline{GT}_{1-0}$ независимы и подключены с помощью внутренних нагрузочных резисторов к напряжению питания для задания исходного высокого уровня (уровня логической 1);

$\overline{LOCK}$ — сигнал блокировки шины, используемый для запрета предоставления шин другим МП в мультипроцессорных системах. Значение $\overline{LOCK} = 0$ устанавливается однобайтной командой LOCK, называемой префиксом блокировки шины, и поддерживается до завершения выполнения следующей за префиксом команды. Значение $\overline{LOCK} = 0$ устанавливается автоматически во время и между импульсами $\overline{INTA} = 0$ (до завершения чтения байта *type* из контроллера прерываний 8259*A*). При подтверждении захвата шин сигнал $\overline{LOCK}$ переводится в *Z*-состояние. Если в состоянии блокировки поступает запрос шин по линиям $\overline{RQ}/\overline{GT}_{1-0}$, то МП фиксирует запрос, но не подтверждает его до завершения выполнения команды, следующей за префиксом LOCK. Префикс LOCK можно использовать и в минимальном режиме, в котором выход микропроцессора $\overline{LOCK}$ отсутствует — в этом случае префикс LOCK задерживает генерацию значения сигнала подтверждения захвата шин *HLDA* = 1 на запрос захвата *HOLD* = 1 до завершения выполнения команды, следующей за префиксом LOCK;

QS_{1-0} (*Queue Status*) — состояние 6-байтной очереди команд (табл. 4.5), отражающее операции над очередью в предыдущем такте синхронизации. Сигналы QS_{1-0} позволяют арифметическому сопроцессору 8087 производить отслеживание состояния очереди команд МП 8086 (синхронизировать свою очередь команд с очередью команд МП 8086).

Для подключения к МП контроллера прямого доступа к памяти 8237A (см. § 3.6) следует синтезировать схему взаимного преобразования однонаправленных сигналов *HOLD/HLDA* в двунаправленный сигнал $\overline{RQ}/\overline{GT_0}$ или $\overline{RQ}/\overline{GT_1}$ (конвертор сигналов — *Control Logic* на рис. 4.13). По запросу захвата шин *HOLD* = $\sqcap$, поступающему от контроллера прямого доступа к памяти 8237, необходимо сформировать первый импульс последовательности $\overline{RQ}/\overline{GT}$, а по снятию запроса *HOLD* = $\neg\sqcup$ — третий импульс этой последовательности (см. рис. 4.9). Конвертор сигналов легко реализовать в виде цифрового автомата, задаваемого временны́ми диаграммами, изображенными на рис. 4.10 (необходимо использовать два синхронных триггера Q_1 и Q_2, например, два *T*-триггера со счетными входами T_1 и T_2). В ответ на первый импульс последовательности $\overline{RQ}/\overline{GT} = \overline{RQ}$ микропроцессор выдаст второй импульс последовательности $\overline{RQ}/\overline{GT} = \overline{GT}$ и конвертор должен сформировать значение сигнала подтверждения захвата шин *HLDA* = 1. При генерации третьего импульса последовательности $\overline{RQ}/\overline{GT} = \overline{RQ}$ (уведомление МП об освобождении шин) конвертор должен выдать значение сигнала *HLDA* = 0.

На основании временны́х диаграмм (рис. 4.10) легко составить таблицу истинности (табл. 4.6), диаграммы Вейча (рис. 4.11) и методом, изложенным в [5], найти, что работа конвертора при использовании *J–K*-триггеров описывается функциями:

Таблица 4.5. **Идентификация состояния очереди команд**

QS_1	QS_0	Операции с очередью команд
0	0	Операции не производились — в предыдущем такте байты из очереди не выбирались
0	1	Из очереди выбран первый байт (КОП) команды
1	0	Очередь сброшена (пустая) из-за команды передачи управления
1	1	Из очереди выбран байт, отличающийся от первого байта команды

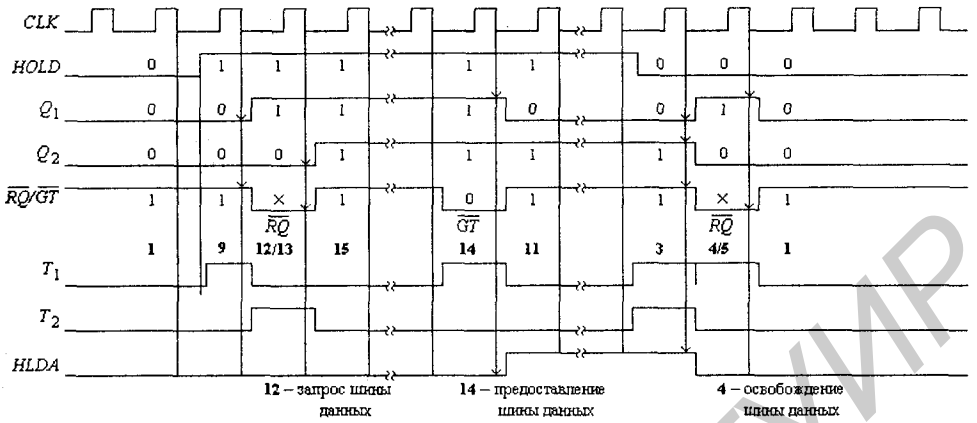


Рис. 4.10. Временны́е диаграммы преобразователя сигналов *HOLD/HLDA* ⇔ $\overline{RQ}/\overline{GT}$

Таблица 4.6. **Состояния преобразователя сигналов *HOLD/HLDA* ⇔ $\overline{RQ}/\overline{GT}$**

HOLD	Q_1	Q_2	*RT* (вход)	Q_1^+	Q_2^+	*HLDA*	*OE*	*RT* (выход)	Примечание
0	0	0	1	0	0	0	0	Z	Исходное состояние автомата
1	0	0	1	1	0	0	0	Z	Запрос прямого доступа к памяти от *DMAC*
1	1	0	×	1	1	0	1	0	$RT = \overline{RQ} = 0$
1	1	1	1	1	1	0	0	Z	$RT = \overline{GT} = 1$ — пассивное состояние сигнала
1	1	1	0	0	1	0	0	Z	$RT = \overline{GT} = 0$ — ответ МП на запрос $\overline{RQ} = 0$
1	0	1	1	0	1	1	0	Z	Выполнение *DMA*-пересылок пока *HOLD* = 1
0	0	1	1	1	0	1	0	Z	Снятие запроса прямого доступа к памяти
0	1	0	×	0	0	0	1	0	$RT = \overline{RQ} = 0$
0	0	0	1	0	0	0	0	Z	Исходное состояние автомата

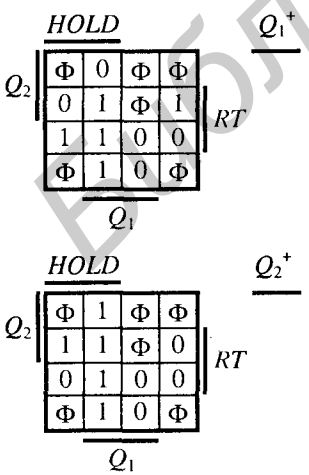


Рис. 4.11. Диаграммы Вейча

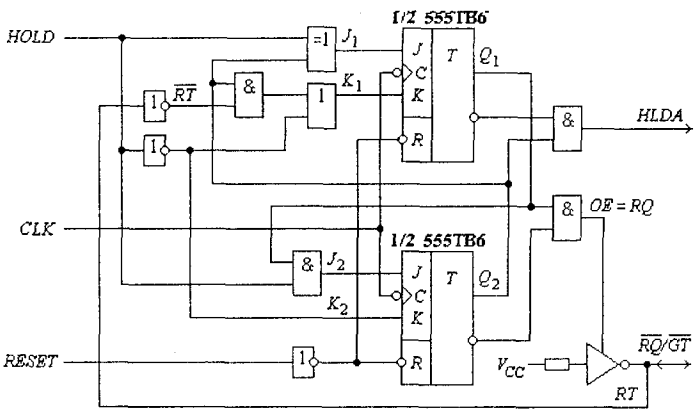


Рис. 4.12. Преобразователь сигналов *HOLD/HLDA* ⇔ $\overline{RQ}/\overline{GT}$

$$J_1 = Q_2 \oplus \mathit{HOLD},\ K_1 = \overline{HOLD} \vee Q_2\,\overline{RT},\quad J_2 = Q_1\,HOLD,\ K_2 = \overline{HOLD},$$

$$HLDA = \overline{Q}_1 Q_2,\ OE = RQ = Q_1\overline{Q}_2,$$

где $RT = \overline{RQ}/\overline{GT}$ (вход $RT = \overline{GT}$, выход $RT = \overline{RQ}$ или находится в Z-состоянии). Этим функциям соответствует схема цифрового автомата, изображенная на рис. 4.12. Исходное внутреннее состояние автомата $Q_2Q_1 = 00$ задается значением асинхронного потенциального сигнала сброса $RESET = 1$.

В максимальном режиме работы микропроцессора контроллер шин 8288 интерпретирует информацию состояния S_{2-0} для генерации сигналов синхронизации и управления шинами, совместимых с *архитектурой MultiBus*, разработанной фирмой *Intel* для проектирования *мультипроцессорных систем*. На рис. 4.13 изображена типовая структурная схема МП-системы для максимального режима. Сигналы *ALE*, *DT/R̄* и *DEN* (без инверсии), а также системные сигналы управления $\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$, $\overline{IOWC}$ и $\overline{INTA}$ формируются контроллером шин 8288 по сигналам состояния S_{2-0}, поступающим от МП. Детальное описание контроллера шин 8288 и назначение остальных сигналов управления дано в § 4.8.

Временны́е диаграммы основных сигналов в максимальном режиме работы МП 8086 изображены на рис. 4.14. Циклы чтения и записи выполняются за 4 такта, если не вводятся такты ожидания T_W по значениям сигнала готовности $RDY = 0$, поступающего от медленных памяти и внешних устройств (*Wait State Generator* — генератор состояний ожидания на рис. 4.13). Сигналы $\overline{MWTC}$ и $\overline{IOWC}$ соответствуют сигналам $\overline{MEMW}$ и $\overline{I/OW}$, генерируемым системным контроллером 8228, а сигналы $\overline{AMWC}$ и $\overline{AIOWC}$ — сигналам $\overline{MEMW}$ и $\overline{I/OW}$, генерируемым системным контроллером 8238 для МП 8080 (см. рис. 1.31).

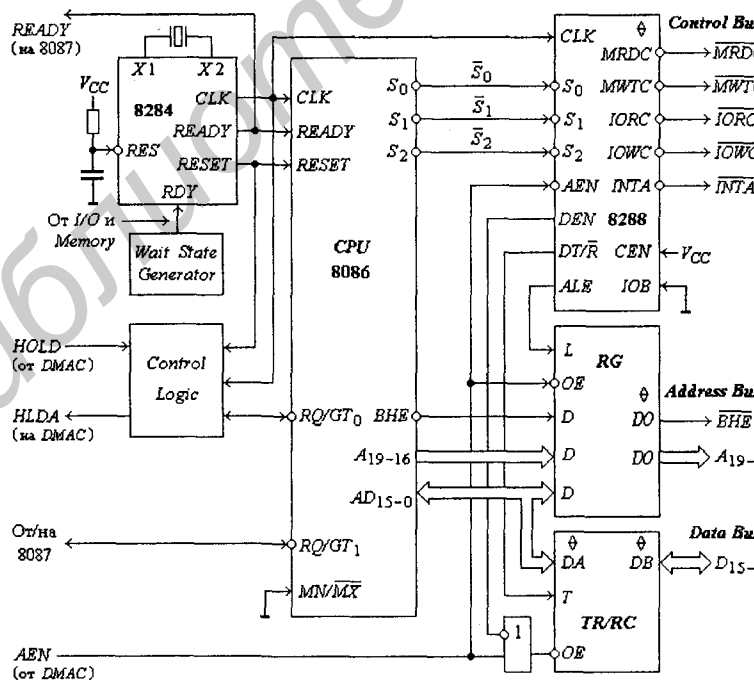


Рис. 4.13. Типовая структурная схема для максимального режима

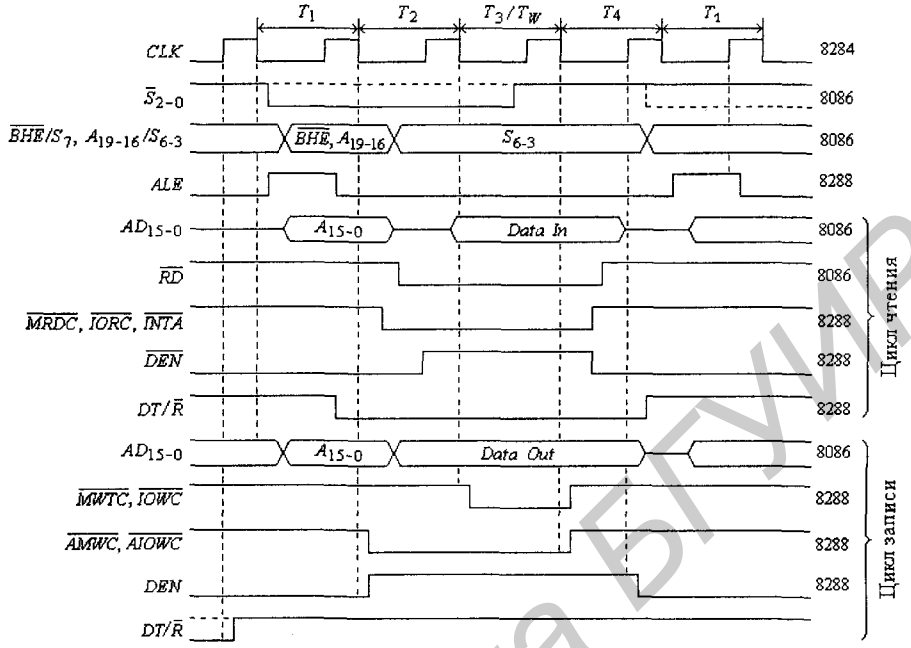


Рис. 4.14. Временны́е диаграммы для максимального режима работы МП 8086

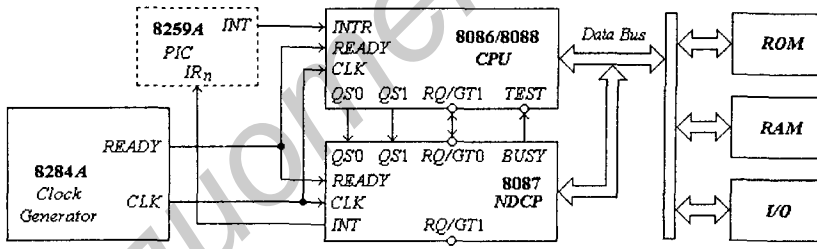


Рис. 4.15. Сопроцессорная конфигурация МП-системы

Сопроцессорные конфигурации МП-систем. Хотя МП 8086/8088 являются мощными однокристальными процессорами, их системы команд недостаточно для эффективного решения сложных вычислительных задач, так как они не имеют команд арифметики с плавающей точкой. В этих случаях МП 8086/8088 необходимо дополнить сопроцессорами, которые расширяют систему команд для более эффективной реализации специальных вычислений. Для таких применений фирма *Intel* специально разработала арифметический сопроцессор 8087 (*NDCP — Numeric Data Coprocessor*; см. рис. 4.4), который снабжен мощной системой команд арифметики с плавающей точкой, позволяющей производить вычисления с высочайшей точностью (диапазон представимых вещественных чисел равен $\pm 10^{\pm 4932}$).

Для включения в микропроцессорную систему *NDCP* не требуется дополнительной внешней логики (рис. 4.15 — остальные одноименные выводы МП 8086/8088 и *NDCP* 8087 должны быть соединены). Это достигнуто тем, что выборку всех команд из памяти производят только МП 8086/8088. Все команды *NDCP* (группа из восьми команд, обозначаемая общей мнемоникой ESC — *Escape* — переход, переключение на сопроцессор), характеризуются тем, что их

первый байт равен 11011×××= D8*h* ÷ DF*h*, а в системе команд МП такие команды отсутствуют. Если выбранная команда принадлежит *NDCP*, то МП 8086/8088 не предпринимают более никаких действий кроме возможного считывания операнда для сопроцессора и передачи ему физического адреса этого операнда (см. § 4.6).

Предназначенная для *NDCP* команда определяется появлением в программе команды ESC. Хотя выбирать команды могут только (главные) МП, сопроцессор тоже получает все команды и контролирует выполнение команд МП, используя сигналы на линиях состояния очереди QS_{1-0}. Команда ESC содержит код внешней операции *NDCP* (три младших разряда первого байта и второй байт), и одновременно дешифрируется сопроцессором и МП. В этой точке МП может просто перейти к выборке следующей команды или считать первое слово из памяти как операнд для *NDCP*, а затем перейти к выборке следующей команды. Если МП считывает первое слово операнда, *NDCP* перехватывает это слово данных и его 20-разрядный физический адрес. Когда операнд-источник длиннее одного слова, *NDCP* получает остальные слова посредством запросов циклов шины по линии $\overline{RQ}/\overline{GT}_0 \leftrightarrow \overline{RQ}/\overline{GT}_1$. Если же определенный в команде ESC операнд является получателем, *NDCP* игнорирует считанное микропроцессором слово данных, а позднее записывает результат вычислений по перехваченному адресу. В любом случае *NDCP* выдает значение сигнала занятости *BUSY* = 1 на вход $\overline{TEST}$ МП и пока МП продолжает выполнение программы *NDCP* производит вычисления, указанные в команде ESC. Такая параллельная работа продолжается до тех пор, пока микропроцессору не понадобится *NDCP* для выполнения другой операции или микропроцессору не потребуется получить результат текущей операции из *NDCP*. Для этого МП должен выполнить команду WAIT и ожидать, пока *NDCP* не выдаст значение сигнала занятости *BUSY* = 0 на вход $\overline{TEST}$. Команда WAIT периодически проверяет значение сигнала $\overline{TEST}$ и, когда он становится активным, осуществляет передачу управления находящейся за ней команде.

Чтобы *NDCP* мог определить, когда МП выполняет команду ESC, он должен контролировать состояние МП по линиям S_{2-0} и выборку команды по линиям AD_{15-0}. Так как МП выбирает команды с опережением, выборка команды ESC не означает ее немедленного выполнения, а при наличии перед ней команды перехода она может не выполняться совсем. Сопроцессор должен следить за командным потоком, контролируя биты состояния очереди QS_{1-0} и управляя своей очередью команд, идентичной очереди МП. Если состояние очереди QS_1QS_0 = 00 (см. табл. 4.5), *NDCP* ничего не делает, а если QS_1QS_0 = 01, он должен сравнить пять старших разрядов первого байта в очереди с кодом 11011 для обнаружения команды ESC. При обнаружении команды ESC сопроцессор выполняет ее. В противном случае байт игнорируется и удаляется из очереди. Состояние очереди QS_1QS_0 = 10 означает, что очередь в МП сброшена и *NDCP* также должен очистить свою очередь. Состояние QS_1QS_0 = 11 указывает, что первый байт в очереди не является первым байтом команды и этот байт *NDCP* анализирует, если только он является частью команды ESC. Когда же он не является частью команды ESC, этот байт игнорируется.

При возникновении ошибок при декодировании и выполнении команд ESC сопроцессор выдает запрос прерывания *INT* = 1 на контроллер прерываний 8259*A*. При необходимости чтения операндов из памяти или записи результата вычислений в память *NDCP* посылает запросы циклов шины по линии $\overline{RQ}/\overline{GT}_0 \leftrightarrow \overline{RQ}/\overline{GT}_1$.

Очередь команд. Для упреждающей выборки из памяти и временного хранения команд в МП 8086 использован 6-байтный регистр очереди команд (см. рис. 4.2), представляющий собой память типа *FIFO* 6 × 8 бит (*first-in, first-out* — первый вошел, первый вышел). Очередь команд непрерывно заполняется в те промежутки времени, когда системная шина не требуется для других операций. Такое опережение выборки команд по отношению к их исполнению значительно увеличивает пропускную способность шины, так как к моменту завершения текущей команды следующая команда чаще всего уже находится в МП. В случае выполнения команды перехода или вызова подпрограммы очередь должна сбрасываться, что уменьшает

перехода или вызова подпрограммы очередь должна сбрасываться, что уменьшает эффективность ее использования, но в среднем это происходит нечасто.

Хотя МП 8086 может обращаться к словам, младший байт которых расположен как по четному, так и нечетному адресу, но при нечетном адресе для чтения слова требуются два обращения к памяти: сначала читается младший байт по верхней части шины ($\overline{BHE} = 0$, $A_0 = 1$ — см. табл. 4.3), а затем старший байт по нижней части шины ($\overline{BHE} = 1$, $A_0 = 0$). Следовательно, возможна некоторая экономия времени, если хранить слова только по четным адресам. Длина команд составляет от одного до шести байт и их можно читать словами по четным адресам, хотя младший байт слова может принадлежать одной команде, а старший байт — другой команде. Поэтому *FIFO* организовано так, что позволяет читать команды из памяти словами по четным адресам (рис. 4.16, *а*). Имеется только одно исключение, связанное с переходом по нечетному адресу. В этом случае МП производит выборку сначала одного байта по нечетному адресу, а затем продолжает читать слова по четным адресам (рис. 4.16, *б*).

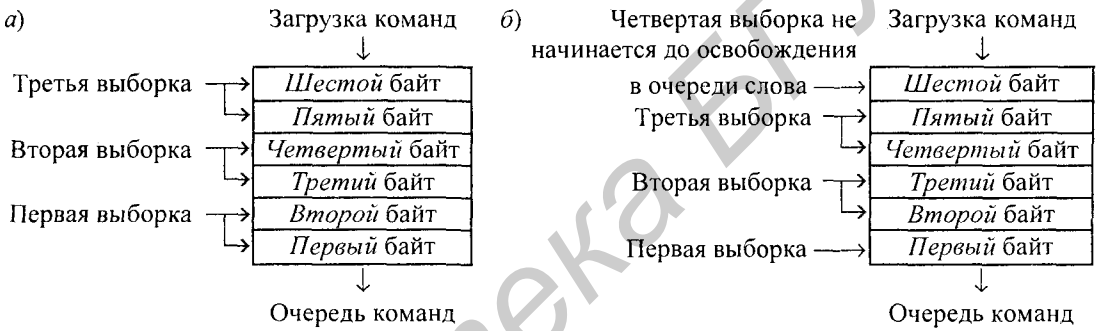


Рис. 4.16. Заполнение очереди команд после перехода на четный (*а*) и нечетный (*б*) адреса

Микропроцессор 8088. Микропроцессор 8088 имеет 8-разрядную внешнюю шину данных, как и МП 8080/8085, но его архитектура аналогична архитектуре МП 8086 и разводка контактов корпуса такая же, как и у МП 8086 (см. рис. 4.4). Системы команд у МП 8088 и 8086 также одинаковы.

Микропроцессор 8088 можно применять в аппаратуре, реализованной на базе МП 8080 или 8085. Поскольку линии управления МП 8088 и 8080/8085 различаются, при введении МП 8088 в систему на базе МП 8080/8085 требуется значительно изменить логику управления шиной. Особенно важно учесть тот факт, что в МП 8088, как и в МП 8085, адреса и данные мультиплексируются. В устройствах памяти и ввода-вывода потребуются небольшие изменения, если не увеличивать объем памяти благодаря 20 линиям адреса.

Таблица 4.7. **Системные сигналы управления**

$IO/\overline{M}$	$DT/\overline{R}$	$\overline{SS_0}$	Состояние МП
1	0	0	*Interrupt Acknowledge*
1	0	1	*Read I/O Port*
1	1	0	*Write I/O Port*
1	1	1	*Halt*
0	0	0	*Code Access*
0	0	1	*Read Memory*
0	1	0	*Write Memory*
0	1	1	*Passive*

В отличие от 6-байтной очереди команд МП 8086 в МП 8088 длина очереди команд равна 4 байтам. Это связано с тем, что МП 8088 может читать из памяти только байты, что увеличивает время их выборки — 6-байтная очередь использовалась бы не полностью. Алгоритм опережающей выборки отличается тем, что МП 8088 инициирует выборку команды из памяти, когда в очереди оказывается один свободный байт, а не два, как в МП 8086.

Большая часть сигналов МП 8088 имеет то же назначение, что и соответствующие сигналы МП 8086 (см. рис. 4.4). Поскольку МП 8088 имеет 8-разрядную внешнюю шину данных, то вместо мультиплексных сигналов адреса-данных AD_{15-8} используются адресные сигналы A_{15-8} и не требуется сигнал $\overline{BHE}$, который заменен сигналом $\overline{SS_0}$ (*Status line*), который эквивалентен сигналу $\overline{S}_0$ максимального режима. В максимальном режиме значение сигнала $\overline{SS_0} \equiv 1$ (*High* — высокий уровень напряжения). Понятно, что фиксировать адресные сигналы A_{15-8} во внешнем регистре памяти не требуется.

В минимальном режиме работы вырабатывается сигнал $IO/\overline{M}$, инверсный по отношению к сигналу $M/\overline{IO}$ МП 8086. Сигнал $IO/\overline{M}$ логически эквивалентен инвертированному сигналу $\overline{S}_2$ максимального режима. Значения сигналов $IO/\overline{M}$, $DT/\overline{R}$ и $\overline{SS_0}$ идентифицируют тип передачи в текущем цикле шины (табл. 4.7).

Управление памятью при использовании МП 8086. На рис. 4.17 изображена структурная схема памяти (ПЗУ — *EPROM* 27256/573РФ8 и статическое ОЗУ — *SRAM MT5C2568*), общий объем которой равен 128 Кбайт:

$\overline{DEN}$ (*Data Enable*) — сигнал включения приемопередатчиков,

$DT/\overline{R}$ (*Data Transmit/Receive*) — сигнал управления направлением передачи данных,

$\overline{MRDC}$ (*Memory Read Command*) — сигнал чтения данных из памяти,

$\overline{MWTC}$ (*Memory Write Command*) — сигнал записи данных в память,

AD_{7-0} — младшая часть мультиплексной шины адреса-данных (от/на МП),

AD_{15-8} — старшая часть мультиплексной шины адреса-данных (от/на МП),

A_{16-0} — разряды шины адреса (от адресного регистра),

$\overline{BHE}$ — сигнал управления старшей частью шины данных (от адресного регистра).

Сигналы управления приемопередатчиками, чтением и записью данных в память поступают от контроллера шин 8288. Для селекции БИС *SRAM* и *EPROM* используется разряд адреса A_{16} (ЛЭ НЕ в данном устройстве является дешифратором *DC* 1 × 2), т. е. не предполагается увеличивать объем памяти за счет использования адресных сигналов A_{19-17}.

Статическим оперативным запоминающим устройством управляют сигналы:

$\overline{CE}$ (*Chip Enable*) — выбор кристалла,

$\overline{WE}$ (*Write Enable*) — разрешение записи,

$\overline{OE}$ (*Output Enable*) — разрешение выхода (запись при $\overline{WE} = \overline{MWTC} = 0$ и чтение при $\overline{OE} = \overline{MRDC} = 0$, если $\overline{CE}_1 = \overline{A}_{16} = 0$).

Из рис.4.17 следует, что адреса *EPROM* заключены в диапазоне 00000*h* … 0FFFF*h* (64 Кбайта), а адреса *SRAM* (*Static RAM*) в диапазоне — 10000*h* … 1FFFF*h* (64 Кбайта).

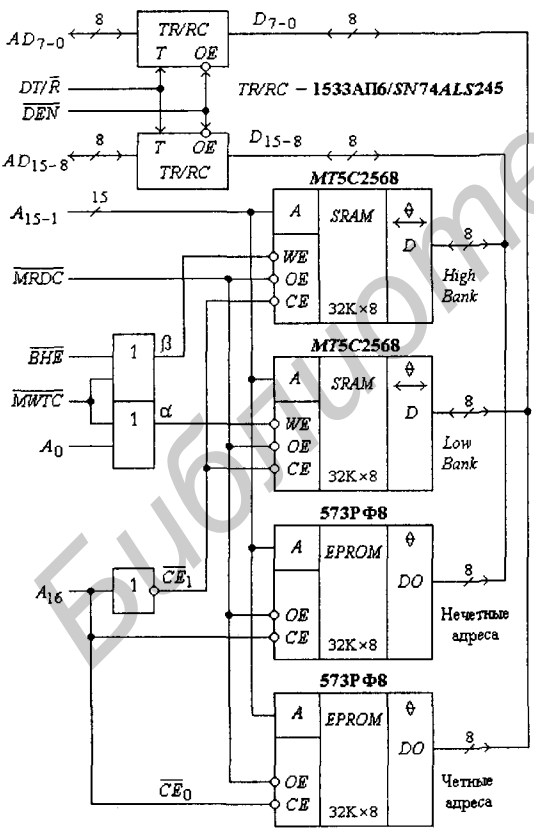


Рис. 4.17. Структурная схема управления памятью

Из-за неполной дешифрации адреса (разряды A_{19}, A_{18} и A_{17} могут иметь произвольные значения — от 000 до 111) памяти могут быть приписаны и другие адреса.

Чтение памяти всегда производится словами, но МП будет принимать слово или только один из двух байт в зависимости от значений сигналов A_0 и $\overline{BHE}$ в соответствии с табл. 4.3. Запись же данных следует производить строго в соответствии со значениями сигналов A_0 и $\overline{BHE}$. Для этого использованы ЛЭ ИЛИ, разделяющие ОЗУ на два банка: *младший банк* (*Low Bank*), селектируемый значением сигнала $A_0 = 0$, и *старший банк* (*High Bank*), выбираемый значением сигнала $\overline{BHE} = 0$:

$$\alpha = \overline{\overline{A_0} \cdot MWTC} = \begin{cases} 0 - \text{запись при } A_0 = 0 \text{ и } MWTC = 1, \\ 1 - \text{нет записи данных в младший банк;} \end{cases}$$

$$\beta = \overline{BHE \cdot MWTC} = \begin{cases} 0 - \text{запись при } BHE = 1 \text{ и } MWTC = 1, \\ 1 - \text{нет записи данных в старший банк} \end{cases}$$

(запись, конечно, возможна только при значении сигнала $\overline{CE_1} = 0$).

Объем памяти можно увеличить, добавив еще необходимое число блоков памяти по 64 Кбайта, селектируемых выходными сигналами адресного дешифратора сигналов A_{19-16}, построенного, например, на ИС 1533ИД7.

4.2. Режимы адресации данных и переходов

Команды МП 8086/8088 могут содержать от одного до шести байт. В первом байте команды всегда находится *код операции* (*Opcode* — КОП). Во многих командах второй байт команды (*постбайт*) имеет специальное назначение — задает *режим адресации данных*, а также может содержать три дополнительных разряда КОП. Это позволило количественно расширить систему команд и разработать гибкую систему адресации операндов. В однобайтных командах может указываться неявный операнд или регистр. Команды могут иметь не более двух операндов, причем хотя бы один из операндов должен быть регистром. Исключение составляют цепочечные команды, оба операнда которых находятся в памяти. Операнд может содержать данное, часть адреса данного, косвенный указатель данного или другую информацию, относящуюся к обрабатываемым командой данным. Наименьшую длину имеют команды, в которых используются только операнды, находящиеся в регистрах МП.

Форматы команд. На рис. 4.18 изображены форматы первого байта команд с указанием специальных бит (*индикаторов*) и групп бит (полей), используемых для адресации операндов:

w (*word*) — индикатор, определяющий операцию с байтом ($w = 0$) или словом ($w = 1$), если команда может оперировать байтом и словом;

d (*direction*) — индикатор, указывающий направление передачи данных в двухоперандных командах за исключением команд с непосредственным операндом и цепочечных команд. Одним из операндов должен быть регистр, определяемый полем *reg* во втором байте команды ($d = 0$ — регистр является *операндом-источником*, $d = 1$ — регистр является *операндом-получателем*);

s (*sign*) — индикатор, используемый только совместно с индикатором *w* для *расширения знака* непосредственных 8-разрядных операндов, представленных в дополнительном коде, до 16-разрядных операндов (*s*:*w* = 00 — 8-разрядная операция, *s*:*w* = 01 — 16-разрядная операция с 16-разрядным непосредственным операндом без знака, *s*:*w* = 11 — 16-разрядная операция с 8-разрядным непосредственным операндом, который расширяется со знаком до 16 разрядов).

Последний вариант допускает использование однобайтного операнда для чисел –128 ... +127 в 16-разрядных операциях (экономится один байт). Индикатор *s* имеется только в командах сложения, вычитания и сравнения операндов;

v (*value*) — индикатор, определяющий число сдвигов в командах сдвигов (*v* = 0 — число сдвигов равно 1, *v* = 1 — число сдвигов определяется регистром *CL*);

z (*zero*) — индикатор, используемый только в команде REP;

reg (*register*) — 2- или 3-разрядное поле, адресующее сегментные регистры или регистры общего назначения в соответствии с табл. 4.8. Неоднозначности использования поля *reg* для указания сегментных регистров и регистров общего назначения не возникает, так как они распознаются с помощью КОП.

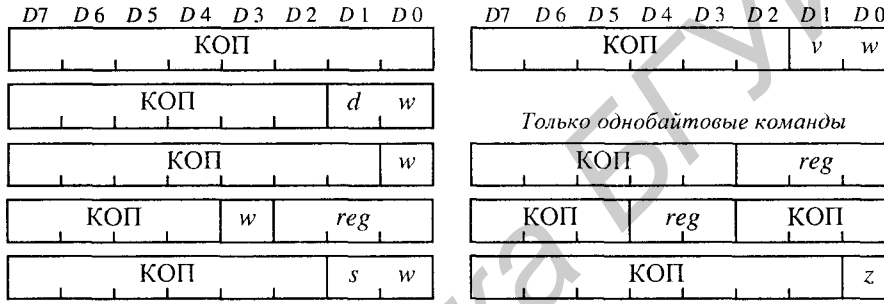


Рис. 4.18. Форматы первого байта команд

Таблица 4.8. **Кодирование регистров**

Адрес регистра (поле *reg*)	Регистры		Адрес регистра (поле *reg*) *	Сегментный регистр
	w = 1	*w* = 0		
0 0 0	AX	AL	0 0	ES
0 0 1	CX	CL	0 1	CS
0 1 0	DX	DL	1 0	SS
0 1 1	BX	BL	1 1	DS
1 0 0	SP	AH		
1 0 1	BP	CH		
1 1 0	SI	DH		
1 1 1	DI	BH		

Примечание: * старший разряд поля *reg* равен 0, если поле *reg* трехразрядное.

На рис. 4.19 показаны форматы всех команд сложения без переноса ADD, содержащие от двух до шести байт (штриховыми линиями изображены операнды, наличие которых определяется режимом адресации операндов и индикаторами *s* и *w*). Один байт *disp L* задает 8-разрядное смещение *disp*8 (расширяется со знаком), а два байта *disp H* и *disp L* — 16-разрядное смещение *disp*16, участвующие в вычислении эффективного адреса памяти *EA*. Байт данных *L* представляет 8-разрядный непосредственный операнд (операнд, содержащийся в самой команде), а байты данных *data H* и *data L* — 16-разрядный непосредственный операнд. Второй байт в первых двух командах сложения используется для задания режима адресации операндов. Второй байт второй команды содержит еще 3-разрядный код операции КОП = 000.

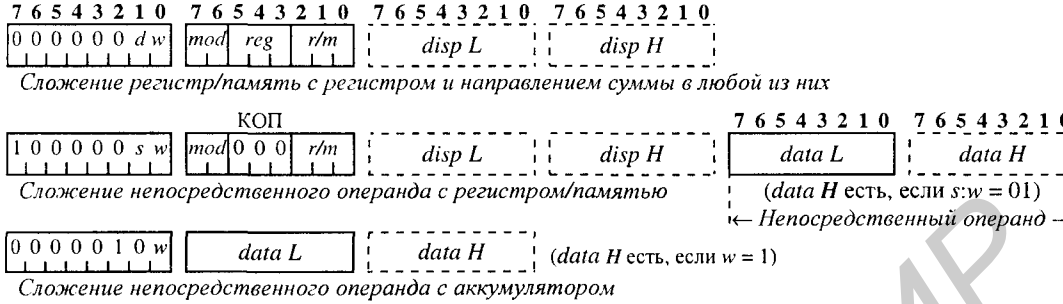


Рис. 4.19. Форматы команд сложения ADD

Режимы адресации данных. Способ определения операнда называется *режимом адресации*. Если на код операции и режим адресации отводятся два байта, то второй байт, называемый *постбайтом*, имеет один из двух форматов, изображенных на рис. 4.20. Первый формат используется в однооперандных командах и таких двухоперандных, в которых один из операндов неявно определяется кодом операции. Второй формат необходим для двухоперандных команд — поле *reg* определяет регистр, который в зависимости от значения индикатора *d* является операндом-источником или операндом-получателем. Операнд, указываемый полями *mod* и *r/m*, определяется в соответствии с табл. 4.9. Эта таблица определяет шесть режимов адресации данных, названия и определения которых приведены в примечании.

Рис. 4.20. Форматы постбайтов

Таблица 4.9. **Режимы адресации данных и сегментные регистры по умолчанию**

r/m	*mod* = 00	*mod* = 01	*mod* = 10	*mod* = 11	
				w = 0	*w* = 1
000	**DS :** BX + SI [5]	**DS :** BX + SI + *disp*8 [6]	**DS :** BX + SI + *disp*16 [6]	AL [2]	AX [2]
001	**DS :** BX + DI [5]	**DS :** BX + DI + *disp*8 [6]	**DS :** BX + DI + *disp*16 [6]	CL [2]	CX [2]
010	**SS :** BP + SI [5]	**SS :** BP + SI + *disp*8 [6]	**SS :** BP + SI + *disp*16 [6]	DL [2]	DX [2]
011	**SS :** BP + DI [5]	**SS :** BP + DI + *disp*8 [6]	**SS :** BP + DI + *disp*16 [6]	BL [2]	BX [2]
100	**DS :** SI [3]	**DS :** SI + *disp*8 [4]	**DS :** SI + *disp*16 [4]	AH [2]	SP [2]
101	**DS :** DI [3]	**DS :** DI + *disp*8 [4]	**DS :** DI + *disp*16 [4]	CH [2]	BP [2]
110	**DS :** *disp*16 [1]	**SS :** BP + *disp*8 [4]	**SS :** BP + *disp*16 [4]	DH [2]	SI [2]
111	**DS :** BX [3]	**DS :** BX + *disp*8 [4]	**DS :** BX + *disp*16 [4]	BH [2]	DI [2]

Примечание:
[1] *Прямая адресация*: эффективный адрес *EA* данного является частью команды.
[2] *Регистровая адресация*: данное находится в регистре *AX*, *BX*, *CX*, *DX*, *SI*, *DI*, *SP*, *AL*, *AH*, *BL*, *BH*, *CL*, *CH*, *DL*, *DH*.
[3] *Косвенная регистровая адресация*: эффективный адрес *EA* = BX, SI или DI.
[4] *Косвенная регистровая относительная адресация*: *EA* = {BX, BP, SI или DI} + {*disp*8 или *disp*16}.
[5] *Базовая индексная адресация*: эффективный адрес *EA* = {BX или BP} + {SI или **DI**}.
[6] *Относительная базовая индексная адресация*: *EA* = {BX или BP} + {SI или DI} + {*disp*8 или *disp*16}.

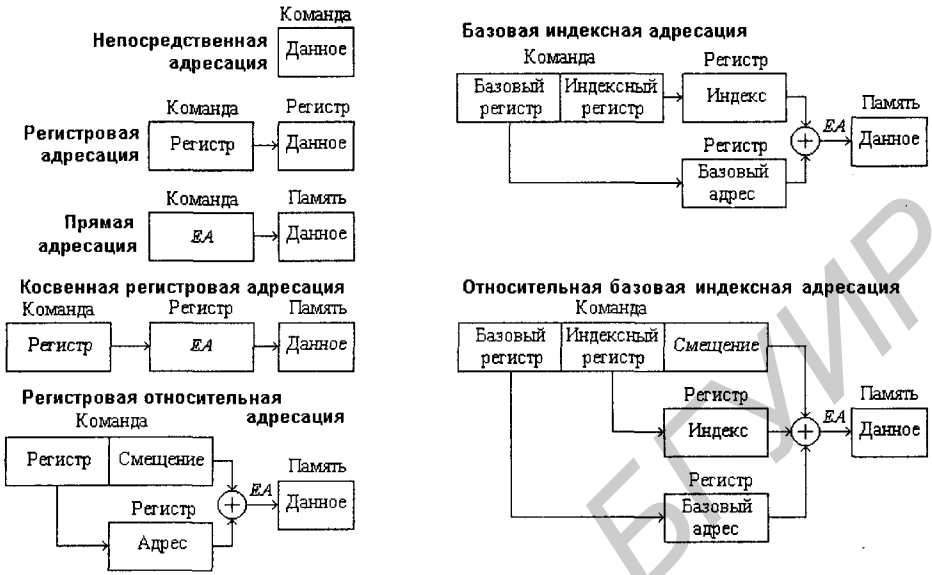


Рис. 4.21. Режимы адресации данных

Если поле *mod* ≠ 11, один из операндов находится в памяти и ее эффективный адрес *EA* вычисляется согласно табл. 4.9. Значение поля *mod* = 00 указывает на отсутствие смещения, за исключением значения поля *r/m* = 110, задающего прямую адресацию. Если значение поля *mod* = 01, то третий байт команды содержит 8-разрядное смещение *disp*8, которое до вычисления эффективного адреса *EA* автоматически расширяется со знаком до 16 разрядов. Значение поля *mod* = 10 означает, что третий и четвертый байты команды содержат 16-разрядное смещение *disp*16. Если же значение поля *mod* = 11, то операндом является регистр, адрес которого определяется полем *r/m*. Непосредственный режим адресации в табл. 4.9 отсутствует, так как 8- или 16-разрядный непосредственный операнд является частью команды.

На рис. 4.21 графически изображены определения операндов для всех допустимых режимов адресации данных. Не следует забывать, что физический адрес памяти данных определяется эффективным адресом *EA* и соответствующим сегментным регистром — *DS*, *ES* или *SS*. Многочисленные примеры команд, иллюстрирующих все семь методов адресации данных, приведены в листинге, находящемся в самом конце § 4.4 (с. 442 – 444).

В табл. 4.9 указаны также сегментные регистры, используемые по умолчанию при вычислении 20-разрядного физического адреса памяти для различных комбинаций полей *mod* и *r/m*. Сегментный регистр стека *SS* используется по умолчанию при адресации операндов с привлечением указателя базы *BP*, а в остальных случаях в вычислении физического адреса памяти участвует сегментный регистр данных *DS*. Для изменения используемых по умолчанию сегментных регистров, в систему команд введена специальная однобайтная команда, называемая *префиксом замены сегмента*. Формат этой команды представлен на рис. 4.22. Если команде предшествует префикс замены сегмента, то при обращении к данным в процессе ее выполнения участвует сегментный регистр, определяемый полем *reg* в соответствии с табл. 4.8.

Сегментный регистр *DS* допускается заменять сегментными регистры *CS*, *SS* или *ES*, а сегментный регистр *SS* при участии в адресации регистра *BP* — на *DS*, *CS* или *ES*. Замену сегментного регистра нельзя производить в следующих специальных случаях:

при вычислении адреса очередной выполняемой команды в качестве сегментного регистра всегда используется сегментный регистр *CS*;

при участии в адресации указателя стека *SP* сегментным регистром может быть только регистр *SS*;

в цепочечных командах в качестве сегментного регистра операнда-получателя всегда используется регистр *ES*.

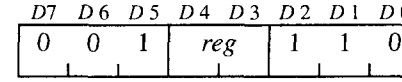


Рис. 4.22. Префикс замены сегмента

Режимы адресации переходов. Адреса переходов могут являться частью команды или находиться в регистрах и ячейках памяти. Всего имеется четыре режима адресации переходов:

внутрисегментный прямой (*near* — близкий) — эффективный адрес перехода *EA* равен сумме 8- или 16-разрядного смещения и текущего содержимого указателя инструкции *IP*. Когда смещение имеет длину 8 бит, этот режим называется *коротким переходом* (*short*). Внутрисегментную прямую адресацию называют также *относительной адресацией*, так как смещение вычисляется "относительно" указателя инструкции *IP*. Этот режим допустим в условных и безусловных переходах, но в команде условного перехода может быть только смещение длиной 8 бит (один байт). Однобайтовые и двухбайтовые смещения воспринимаются как целые числа, представленные в дополнительном коде. Поэтому однобайтовые смещения задают переход в пределах –128 … +127 байт относительно адреса следующей команды. При суммировании содержимого указателя инструкции *IP* с двухбайтовым смещением перенос игнорируется, поэтому текущий сегмент интерпретируется как кольцо — переход осуществляется по любому адресу текущего сегмента размером 64 Кбайта;

внутрисегментный косвенный (*near*) — эффективный адрес перехода *EA* есть содержимое регистра или ячейки памяти, которые указываются в любом режиме адресации данных, кроме непосредственного. Содержимое указателя инструкции *IP* заменяется эффективным адресом перехода *EA*. Данный режим допустим только в командах безусловного перехода;

межсегментный прямой (*far* — далекий) — заменяет содержимое указателя инструкции *IP* одной частью команды, а содержимое сегментного регистра кода *CS* — другой частью команды. Назначение данного режима адресации — обеспечить переход из одного сегмента кода в другой;

межсегментный косвенный (*far*) — заменяет содержимое указателя инструкции *IP* и сегментного регистра кода *CS* содержимым двух смежных слов из памяти, которые определяются в любом режиме адресации данных, кроме непосредственного и регистрового.

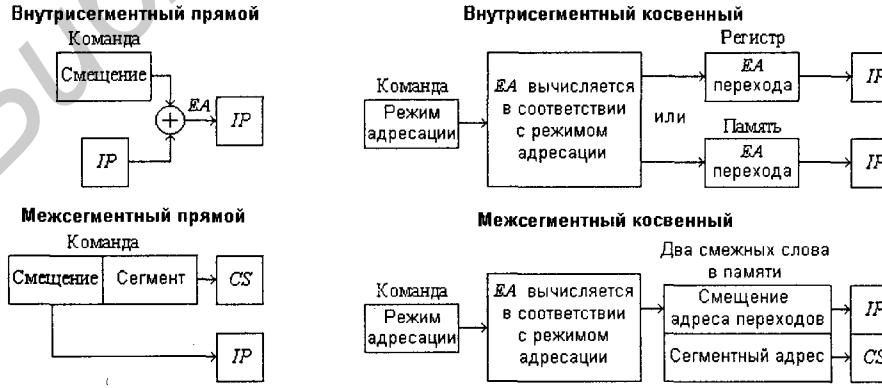


Рис. 4.23. Режимы адресации переходов

Отметим, что физический адрес перехода равен сумме нового содержимого указателя инструкции *IP* и содержимого сегментного регистра кода *CS*, умноженного на 16_{10}. Межсегментный переход может быть только безусловным. Графически определения адресов переходов изображены на рис. 4.23.

Время выполнения команд. *Базовым временем* выполнения команды называется время, затрачиваемое на выполнение команды уже находящейся в очереди команд. В противном случае необходимо учитывать дополнительные такты синхронизации *CLK*, необходимые для выборки команды — на каждое обращение к памяти при чтении команд затрачивается 4 такта синхронизации. Если операнд находится в памяти, то затрачивается дополнительное время *ea* на вычисление эффективного адреса *EA*. Это время зависит от режима адресации (см. табл. 4.9):

прямая — 6 тактов,
косвенная регистровая — 5 тактов,
регистровая относительная — 9 тактов,
базовая индексная — 7 тактов при *EA* = BP + DI или *EA* = BX + SI,
базовая индексная — 8 тактов при *EA* = BP + SI или *EA* = BX + DI,
относительная базовая индексная — 11 тактов при
EA = BP + DI + *disp* и *EA* = BX + SI + *disp*,
относительная базовая индексная — 12 тактов при
EA = BP + SI + *disp* и *EA* = BX + DI + *disp*.

Например, на выполнение команд сложения ADD, приведенных на рис. 4.19, требуется число тактов синхронизации:

регистр – регистр 3,
память – регистр . . . . . 9 + *ea*,
регистр – память . . . . . 16 + *ea*,
непосредственный операнд – регистр 4,
непосредственный операнд – память . . . . . 17 + *ea*,
непосредственный операнд – аккумулятор . . . . . 4.

Время выполнения команды можно определить умножением числа тактов синхронизации, необходимых для выполнения команды, на период тактового сигнала *CLK*. Время выполнения команд условных переходов зависит от выполнения или невыполнения условия перехода. Базовые времена выполнения команд приведены при их описании в § 4.3.

4.3. Система команд МП 8086/8088

По назначению команды МП 8086/8088 разделяются на 6 групп (табл. 4.10). Команды на языке ассемблера имеют вид:

COP; COP *dst*; COP *src*; COP *dst*, *src*,

где COP — мнемоника кода операции, *dst* — *операнд-получатель*, *src* — *операнд-источник* (*dst* — *Destination*, *src* — *Source*). В однобайтных командах операнды в явном виде отсутствуют — указание на используемые операнды заключено в коде операции COP (неявная адресация операндов). Например, при выполнении команды XLAT используются операнды, находящиеся в регистрах *AL* и *BX*. В двухоперандных командах операнды разделяются запятой, и операнд-получатель указывается слева от запятой.

Таблица 4.10. **Группы команд МП 8086/8088**

Команда	Стр.	Команда	Стр.	Команда	Стр.	Команда	Стр.
1. *Команды передачи данных* (14 команд)							
IN *dst*, *src*	365	LES *dst*, *src*	367	POPF	392	XCHG *dst*, *src*	365
LAHF	406	**MOV** *dst*, *src*	364	PUSH *src*	392	XLAT	366
LDS *dst*, *src*	367	OUT *dst*, *src*	366	PUSHF	392		
LEA *dst*, *src*	367	POP *dst*	392	SAHF	406		
2. *Арифметические команды* (20 команд)							
AAA	368	**ADD** *dst*, *src*	368	DAS	372	INC *dst*	360
AAD	378	CBW	375	DEC *dst*	360	MUL *src*	376
AAM	376	**CMP** src_1, src_2	374	DIV *src*	378	NEG *dst*	367
AAS	372	CWD	375	IDIV *src*	379	**SBB** *dst*, *src*	372
ADC *dst*, *src*	368	DAA	368	IMUL *src*	377	**SUB** *dst*, *src*	372
3. *Логические команды* (12 команд)							
AND *dst*, *src*	380	RCL *dst*, *cnt*	383	ROR *dst*, *cnt*	383	SHR *dst*, *cnt*	381
NOT *dst*	380	RCR *dst*, *cnt*	383	SAR *dst*, *cnt*	381	**TEST** src_1, src_2	381
OR *dst*, *src*	380	ROL *dst*, *cnt*	383	SHL/SAL *dst*, *cnt*	381	**XOR** *dst*, *src*	380
4. *Команды манипуляции цепочками* (6 команд)							
CMPS	390	MOVS	388	SCAS	390		
LODS	391	REP	388	STOS	391		
5. *Команды передачи управления* (26 команд)							
CALL *target*	393	JE/JZ *sL*	404	JNL/JGE *sL*	404	JS *sL*	404
INT *type*	397	JL/JNGE *sL*	404	JNLE/JG *sL*	404	LOOP *sL*	405
INTO	399	JLE/JNG *sL*	404	JNO *sL*	404	LOOPNZ/ LOOPNE *sL*	405
IRET	397	JMP *target*	403	JNP/JPO *sL*	404		
JB/JNAE/JC *sL*	404	JNB/JAE/JNC *sL*	404	JNS *sL*	404	LOOPZ/ LOOPE *sL*	405
JBE/JNA *sL*	404	JNBE/JA *sL*	404	JO *sL*	404		
JCXZ *sL*	405	JNE/JNZ *sL*	404	JP/JPE *sL*	404	**RET**	393
6. *Команды управления процессором* (12 команд)							
CLC	406	CMC	406	LOCK	406	STD	406
CLD	406	ESC	406	NOP	365	STI	406
CLI	406	HLT	406	STC	406	WAIT	406

В табл. 4.10 использованы также обозначения: *cnt* (*counter*) — число 1 или регистр *CL* (число сдвигов), *target* — адрес, *sL* (*short Label*) — короткий переход по метке (переход назад и вперед в интервале адресов –128 ... +127 относительно адреса команды), *type* — тип прерывания (*type* = 00 ... FF*h*). Полужирным шрифтом выделены команды, которые могут иметь непосредственные операнды.

Команды в табл. 4.10 в пределах групп расположены в алфавитном порядке и снабжены указателем страниц, на которых приведено основное их описание. Некоторые команды имеют один или два синонима, для которых алфавитный порядок расположения нарушается (например, команды условных переходов JBE/JNA *sL* и JB/JNAE/JC *sL* имеют две и три мнемоники, транслируемые ассемблером в одни и те же машинные коды первого байта команды 76*h* и 72*h*).

Машинные коды команд. В табл. 4.11 приведены машинные коды (МК) команд, где символом "*" отмечены команды, которые могут выполняться только микропроцессором 80286 и выше, а символом "#" помечены команды, которые могут выполняться только микропроцессорами 80386 или 80486 и выше. Для конкретизации операндов *src* и *dst* в табл. 4.11 и 4.12 использованы обозначения:

*r*8 и *r*16 — 8- и 16-разрядные регистры, *seg* — сегментный регистр,

r/m — регистр или память, *mem*8 и *mem*16 — 8- и 16-разрядные операнды в памяти,

*im*8 и *im*16 — 8- и 16-разрядные непосредственные операнды,

imm — *im*8 и *im*16 (непосредственным операндом может быть только операнд-источник),

short — короткий переход, *near* — внутрисегментный (близкий) переход, *far* — межсегментный (далекий) переход.

Таблица 4.11. **Машинные коды МП 8086/8088**

МК	×0	×1	×2	×3	×4	×5	×6	×7	МК
0×	ADD *r/m*, *r*8	ADD *r/m*, *r*16	ADD *r*8, *r/m*	ADD *r*16, *r/m*	ADD AL, *im*8	ADD AX, *im*16	PUSH ES	POP ES	0×
1×	ADC *r/m*, *r*8	ADC *r/m*, *r*16	ADC *r*8, *r/m*	ADC *r*16, *r/m*	ADC AL, *im*8	ADC AX, *im*16	PUSH SS	POP SS	1×
2×	AND *r/m*, *r*8	AND *r/m*, *r*16	AND *r*8, *r/m*	AND *r*16, *r/m*	AND AL, *im*8	AND AX, *im*16	*seg*ES	DAA	2×
3×	XOR *r/m*, *r*8	XOR *r/m*, *r*16	XOR *r*8, *r/m*	XOR *r*16, *r/m*	XOR AL, *im*8	XOR AX, *im*16	*seg*SS	AAA	3×
4×	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI	4×
5×	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI	5×
6×	* PUSHA	* POPA	* BOUND	ARPL	# SEG FS	# SEG GS	# opSize *prefix*	# addrSize *prefix*	6×
7×	JO	JNO	JB/JNAE	JNB/JAE	JE/JZ	JNE/JNZ	JBE/JNA	JNBE/JA	7×
8×	**ArOp1** ***r/m, im8***	**ArOp1** ***r/m, im16***	**ArOp2** ***r/m8, im8***	**ArOp2** ***r/m16, im8***	TEST *r/m*, *r*8	TEST *r/m*, *r*16	XCHG *r*8, *r/m*	XCHG *r*16, *r/m*	8×
9×	NOP	XCHG AX, CX	XCHG AX, DX	XCHG AX, BX	XCHG AX, SP	XCHG AX, BP	XCHG AX, SI	XCHG AX, DI	9×
A×	MOV AL, *mem*8	MOV AX, *mem*16	MOV *mem*8, AL	MOV *mem*16, AX	MOVSB	MOVSW	CMPSB	CMPSW	A×
B×	MOV AL, *im*8	MOV CL, *im*8	MOV DL, *im*8	MOV BL, *im*8	MOV AH, *im*8	MOV CH, *im*8	MOV DH, *im*8	MOV BH, *im*8	B×
C×	* ShfOp *r/m*8, *im*	* ShfOp *r/m*16, *im*	RET *near* *im*16	RET *near*	LES *r*16, *mem*	LDS *r*16, *mem*	MOV *mem*, *im*8	MOV *mem*, *im*16	C×
D×	**ShfOp** ***r/m8, 1***	**ShfOp** ***r/m16, 1***	**ShfOp** ***r/m8, CL***	**ShfOp** ***r/m16, CL***	AAM	AAD		XLAT	D×
E×	LOOPNE/ LOOPNZ	LOOPE/ LOOPZ	LOOP	JCXZ #JECXZ	IN AL, *port*8	IN AX, *port*8	OUT *port*8, AL	OUT *port*8, AX	E×
F×	LOCK		REP/ REPNE	REPZ/ REPE	HALT	CMC	**Grp1** ***r/m8***	**Grp1** ***r/m16***	F×
МК	×0	×1	×2	×3	×4	×5	×6	×7	МК

По табл. 4.11 легко найти машинный код первого байта любой команды. Например, машинный код команды MOV CH, *im*8 равен B5*h*. Эта команда выполняет операцию передачи непосредственного операнда *im*8, содержащегося во втором байте команды, в регистр *CH* (CH ← *im*8). Префиксы замены сегмента (см. рис. 4.22) выделены в табл. 4.11 строчными буквами *seg* (*seg*ES, *seg*SS, *seg*CS, *seg*DS — машинные коды 26*h*, 36*h*, 2E*h*, 3E*h*).

Восемь машинных кодов D8*h* ÷ DF*h* задают первый байт команд арифметического сопроцессора *NDCP* 8087 (команд ESC), который имеет достаточно разветвленную систему команд за счет дополнительного задания кода операции во втором байте команд (см. § 4.7).

В машинных кодах 80*h* ÷ 83*h*, D0*h* ÷ D3*h*, F6*h*, F7*h*, FE*h* и FE*h* содержится только часть кода операции — дополнительные три разряда КОП (от 000 до 111) содержатся во втором байте (постбайте; табл. 4.12). Эти команды разбиты на шесть групп и выделены в табл. 4.11 полужирным подчеркнутым шрифтом.

МК	×8	×9	×A	×B	×C	×D	×E	×F	МК
0×	OR *r/m*, *r*8	OR *r/m*, *r*16	OR *r*8, *r/m*	OR *r*16, *r/m*	OR AL, *im*8	OR AX, *im*16	PUSH CS	*# Extnsn *OpCode*	0×
1×	SBB *r/m*, *r*8	SBB *r/m*, *r*16	SBB *r*8, *r/m*	SBB *r*16, *r/m*	SBB AL, *im*8	SBB AX, *im*16	PUSH DS	POP DS	1×
2×	SUB *r/m*, *r*8	SUB *r/m*, *r*16	SUB *r*8, *r/m*	SUB *r*16, *r/m*	SUB AL, *im*8	SUB AX, *im*16	*seg*CS	DAS	2×
3×	CMP *r/m*, *r*8	CMP *r/m*, *r*16	CMP *r*8, *r/m*	CMP *r*16, *r/m*	CMP AL, *im*8	CMP AX, *im*16	*seg*DS	AAS	3×
4×	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI	4×
5×	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI	5×
6×	* PUSH *im*16	* IMUL *r/m*, *im*16	PUSH *im*8	* IMUL *r/m*, *im*8	* INSB	* INSW	* OUTSB	* OUTSW	6×
7×	JS	JNS	JP/JPE	JNP/JPO	JL/JNGE	JNL/JGE	JLE/JNG	JNLE/JG	7×
8×	MOV *r/m*, *r*8	MOV *r/m*, *r*16	MOV *r*8, *r/m*	MOV *r*16, *r/m*	MOV *r/m*, *seg*	LEA *r*16, *mem*	MOV *seg*, *r/m*	POP *r/m*	8×
9×	CBW	CWD	CALL *far*	WAIT	PUSHF	POPF	SAHF	LAHF	9×
A×	TEST AL, *mem*8	TEST AX, *mem*16	STOSB	STOSW	LODSB	LODSW	SCASB	SCASW	A×
B×	MOV AX, *im*16	MOV CX, *im*16	MOV DX, *im*16	MOV BX, *im*16	MOV SP, *im*16	MOV BP, *im*16	MOV SI, *im*16	MOV DI, *im*16	B×
C×	* ENTER *im*16, *im*8	* LEAVE	RET *far* *im*16	RET *far*	INT 3	INT *im*8	INTO	IRET	C×
D×	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7	D×
E×	CALL *near*	JMP *near*	JMP *far*	JMP *short*	IN AL, DX	IN AX, DX	OUT DX, AL	OUT DX, AX	E×
F×	CLC	STC	CLI	STI	CLD	STD	Grp2 r/m8	Grp3 r/m16	F×
МК	×8	×9	×A	×B	×C	×D	×E	×F	МК

Таблица 4.12. **Команды, определяемые кодом операции постбайта**

Группа КОП (МК)	Трехразрядный код операции ××× постбайта *mod* ××× *r/m*							
	000	001	010	011	100	101	110	111
ArOp1 (80, 81)	ADD *r/m, imm*	OR *r/m, imm*	ADC *r/m, imm*	SBB *r/m, imm*	AND *r/m, imm*	SUB *r/m, imm*	XOR *r/m, imm*	CMP *r/m, imm*
ArOp2 (82, 83)	ADD *r/m, imm*		ADC *r/m, imm*	SBB *r/m, imm*		SUB *r/m, imm*		CMP *r/m, imm*
ShfOp (D0 ÷ D3)	ROL *dst, cnt*	ROR *dst, cnt*	RCL *dst, cnt*	RCR *dst, cnt*	SHL/SAL *dst, cnt*	SHR *dst, cnt*		RAR *dst, cnt*
Grp1 (F6, F7)	TEST *r/m, imm*		NOT *dst*	NEG *dst*	MUL *src*	IMUL *src*	DIV *src*	IDIV *src*
Grp2 (FE)	INC *r/m*8	DEC *r/m*8						
Grp3 (FF)	INC *r/m*16	DEC *r/m*16	CALL *near*	CALL *far*	JMP *near*	JMP *far*	PUSH *r/m*	

Представление целых чисел без знака и со знаком. МП 8086/8088 могут выполнять арифметические операции сложения, вычитания, умножения и деления над операндами *oper*, представляющими собой как целые числа без знака, так и целые числа со знаком. Для представления величины целого числа без знака используются все разряды операнда *oper*. Поскольку $2^8 = 256d$ и $2^{16} = 65536d$, то диапазон представимых чисел *oper* без знака равен

$$0 \le oper8 \le 255d = \text{FF}h \text{ и } 0 \le oper16 \le 65535d = \text{FFFF}h$$

(*oper*8 — байт, *oper*16 — слово).

При сложении целых чисел без знака проблема переполнения разрядной сетки отсутствует, так как перенос (0 или 1) из старшего разряда суммы фиксируется во флаге переноса *CF*. Полученное значение переноса CF всегда может быть использовано для получения правильного результата вычислений.

Для представления чисел со знаком используется их дополнительный код, в котором старший разряд операнда *oper* определяет знак числа:

0 — число положительное, 1 — число отрицательное,

а значит диапазон представимых абсолютных значений (модулей) чисел будет меньше. Так как число 0 отнесено к положительным числам и одна половина значений операнда *oper* представляет положительные числа, а другая — отрицательные, то диапазон представимых чисел со знаком равен

$$-128d \le oper8 \le +127d \text{ и } -32768d \le oper16 \le +32767d,$$

т. е. минимальные и максимальные значения определяются соотношениями:

$-2^7 = -128$, $+(2^7 - 1) = +127$ (для *oper*8) и $-2^{15} = -32768d$, $+(2^{15} - 1) = +32767d$ (для *oper*16).

Дополнительный код $[oper]_д$ числа *oper* определяется двумя равносильными соотношениями:

$[oper]_д = oper$, если $oper \geq 0$,
$[oper]_д = 0 - |\, oper\,|$, если $oper < 0$,

и

$[oper]_д = oper$, если $oper \geq 0$,
$[oper]_д = \overline{|\, oper\,|} + 1$, если $oper < 0$,

где $|\, oper\,|$ — модуль отрицательного числа *oper*, $0 - |\, oper\,|$ — двоичное дополнение модуля отрицательного числа (*Two's Complement*), $\overline{|\, oper\,|}$ — поразрядно инвертированный модуль отрицательного числа. Дополнительный код положительного числа совпадает с самим числом, поэтому требуется вычислять дополнительный код только отрицательных чисел. Возникающий при вычитании из нуля модуля отрицательного числа *oper* заем игнорируется, хотя флаг переноса/заема *CF* устанавливается в 1.

Так как дополнительный код отрицательного числа $[oper]_д = 0 - |\, oper\,|$, то модуль отрицательного числа $|\, oper\,| = 0 - [oper]_д$, а значит справедливо и соотношение

$$|\, oper\,| = \overline{[oper]}_д + 1.$$

Эти соотношения позволяют вычислять модуль отрицательных чисел по их дополнительному коду двумя способами, идентичным двум способам вычисления дополнительных кодов по модулям отрицательных чисел. Первый способ удобно использовать для контрольных вычислений с помощью калькулятора, имеющегося в графической операционной системе *Windows* 95/98 персональных компьютеров *IBM PC*, а второй — для вычислений вручную с помощью карандаша и бумаги.

Примеры десятичных чисел, представимых одним байтом в дополнительном коде:

$$\begin{array}{rcl@{\qquad}rcl}
[+0]_д & = & 0000\ 0000, & [-0]_д & = & 0000\ 0000, \\
[+1]_д & = & 0000\ 0001, & [-1]_д & = & 1111\ 1111, \\
 & \therefore & & & \therefore & \\
[+126]_д & = & 0111\ 1110, & [-127]_д & = & 1000\ 0001, \\
[+127]_д & = & 0111\ 1111, & [-128]_д & = & 1000\ 0000.
\end{array}$$

Сложение чисел в дополнительном коде основывается на *теореме* [5]:

если при сложении разрядная сетка не переполняется, то дополнительный код арифметической суммы $[oper_1 + oper_2]_д$ двух двоичных чисел $oper_1$ и $oper_2$ любых знаков равен арифметической сумме дополнительных кодов этих чисел

$$[oper_1 + oper_2]_д = [oper_1]_д + [oper_2]_д$$ — возможный перенос игнорируется;

если числа $oper_1 > 0$, $oper_2 > 0$ и $oper_1 + oper_2 \geq 2^n$ (n — число разрядов чисел без учета знакового разряда), то происходит *переполнение разрядной сетки* (флаг OF = 1) с потерей значения $+2^n$ и изменением знака остатка суммы на противоположный, что фиксируется во флаге *SF*;

если $oper_1 < 0$ и $oper_2 < 0$ и $|\, oper_1\,| + |\, oper_2\,| \geq 2^n + 1$, то происходит *переполнение разрядной сетки* (флаг OF = 1) с потерей значения -2^n и изменением знака остатка суммы на противоположный, что фиксируется во флаге *SF*.

Пример 1. Десятичные числа $oper_1 = -98d$ и $oper_2 = +75d$ (байты) в дополнительном коде имеют представление: $[oper_1]_д = 0 - |\, oper_1\,| = 9Eh$ и $[oper_2]_д = oper_2 = 4Bh$. Сумма десятичных чисел $oper_1 + oper_2 = -23d$, а дополнительный код суммы определяется арифметическим сложением дополнительных кодов чисел:

$$\begin{array}{rr}
\text{дополнительный код } oper_1 \rightarrow & 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0 = 9Eh \quad \text{— число отрицательное} \\
\text{дополнительный код } oper_2 \rightarrow & +\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1 = 4Bh \quad \text{— число положительное} \\
\hline
\text{дополнительный код суммы} \rightarrow & 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1 = E9h \quad \text{— число отрицательное}
\end{array}$$

Модуль $|oper_1 + oper_2| = 0 - E9h = \overline{1}\,\overline{1}\,\overline{1}\,\overline{0}\,\overline{1}\,\overline{0}\,\overline{0}\,\overline{1} + 1 = 17h = 23d$.

Пример 2. Положительные десятичные числа $oper_1 = +98d$ и $oper_2 = +75d$ (байты) в дополнительном коде имеют представление: $[oper_1]_д = oper_1 = 62h$ и $[oper_2]_д = oper_2 = 4Bh$. Сумма $oper_1 + oper_2 = 173d > 2^7 = 128d$, т. е. при их сложении произойдет переполнение разрядной сетки. Проверка:

	0 1 1 0 0 0 1 0 = 62*h*	— число положительное
Флаги:	+ 0 1 0 0 1 0 1 1 = 4B*h*	— число положительное
OF ← 1, CF ← 0	1 0 1 0 1 1 0 1 = AD*h*	— число отрицательное

Пример 3. Отрицательные десятичные числа $oper_1 = -98d$ и $oper_2 = -75d$ в дополнительном коде имеют представление: $[oper_1]_д = 0 - |\,oper_1\,| = 9Eh$ и $[oper_2]_д = 0 - |\,oper_2\,| = B5h$. Сумма модулей этих чисел $|\,oper_1\,| + |\,oper_2\,| = 173d > 2^7 + 1 = 129d$, т. е. при их сложении произойдет переполнение разрядной сетки. Проверка:

	1 0 0 1 1 1 1 0 = 9E*h*	— число отрицательное
Флаги:	+ 1 0 1 1 0 1 0 1 = B5*h*	— число отрицательное
OF ← 1, CF ← 1	0 1 0 1 0 0 1 1 = 53*h*	— число положительное

Логика обнаружения переполнения весьма проста: переполнение разрядной сетки при сложении чисел в дополнительном коде происходит только в том случае, если функция

$$C_{S+1} \oplus C_S = 1,$$

где C_S — перенос в знаковый разряд, C_{S+1} — перенос из знакового разряда, фиксируемый во флаге переноса *CF* [5]. В микропроцессорах 8086/8088 переполнение разрядной сетки при сложении чисел в дополнительном коде фиксируется во флаге переполнения *OF*. Этот флаг устанавливается в 1 также и при выполнении других арифметических и логических команд (см. табл. 4.13 на с. 384 – 387).

Программист должен корректно использовать флаги, фиксирующие результат выполнения команд, для выполнения условных переходов. Ярким примером служит некорректное использование флага знака *SF*, в котором при выполнении арифметических операций фиксируется значение старшего разряда байта/слова. Команды условных переходов JS и JNS используют этот флаг, однако, при сложении чисел без знака значение флага $SF = 1$ не означает, что было получено отрицательное число (при выполнении многих команд МП в принципе не может "узнать", с какими типами чисел он производит операции).

Форматы команд МП 8086/8088. Ниже приведены форматы всех команд МП 8086/8088. Если команда способна адресовать операнд в памяти (в команде есть поле *r/m*), то она может содержать еще один или два байта смещения (*disp*8 или *disp*16).

Для непосредственных операндов будет указываться только один байт данных с общим обозначением *data* 8/16 (*data* 8 — один байт при значении индикатора $w = 0$, *data* 16 — два байта при значении индикатора $w = 1$, причем сначала идет младший, а затем старший байт данных).

Максимальная длина команд равна шести байтам. Далее при описании команд будет использоваться не более трех байт, но для любой команды с сокращенной длиной очень легко установить ее полный формат, так как исключаются только старший байт у двухбайтового непосредственного операнда (если он есть) и один или два байта смещений *disp*8 и *disp*16 у команд, имеющих поле *r/m*. Только у двух команд прямой межсегментной передачи управления (команды CALL и JMP) приведены полные пятибайтовые форматы.

Воздействие команд на флаги будет приведено в табл. 4.13 (с. 384 – 387).

Форматы команд МП 8086/8088 с кратким описанием простых операций:

Команды передачи данных	7 *Байт* 1 0	7 *Байт* 2 0	7 *Байт* 3 0
MOV — передать (*move*); ***dst* ← *src***			
Регистр/память в/из регистра	1 0 0 0 1 0 0 *w*	*mod reg r/m*	
Непосредственный операнд в регистр/память	1 1 0 0 0 1 1 *w*	*mod* 0 0 0 *r/m*	*data* 8/16
Непосредственный операнд в регистр	1 0 1 1 *w reg*	*data* 8/16	
Память в аккумулятор	1 0 1 0 0 0 0 *w*	*addr low*	*addr high*
Аккумулятор в память	1 0 1 0 0 0 1 *w*	*addr low*	*addr high*
Регистр/память в сегментный регистр	1 0 0 0 1 1 1 0	*mod* 0 *reg r/m*	
Сегментный регистр в регистр/память	1 0 0 0 1 1 0 0	*mod* 0 *reg r/m*	
PUSH — включить в стек; **SP ← SP – 2, *M*(SP) ← *src*16**			
Регистр/память	1 1 1 1 1 1 1 1	*mod* 1 1 0 *r/m*	
Регистр	0 1 0 1 0 *reg*		
Сегментный регистр	0 0 0 *reg* 1 1 0		
POP — извлечь из стека; ***dst*16 ← *M*(SP), SP ← SP + 2**			
Регистр/память	1 0 0 0 1 1 1 1	*mod* 0 0 0 *r/m*	
Регистр	0 1 0 1 1 *reg*		
Сегментный регистр	0 0 0 *reg* 1 1 1		
XCHG — обменять (*exchange*); ***dst* ↔ *src***			
Регистр/память с регистром	1 0 0 0 0 1 1 *w*	*mod reg r/m*	
Регистр с аккумулятором	1 0 0 1 0 *reg*		
IN — ввести из (*input from*)			
фиксированного порта	1 1 1 0 0 1 0 *w*	*port*	
переменного порта	1 1 1 0 1 1 0 *w*		
OUT — ввести в (*output to*)			
фиксированный порт	1 1 1 0 0 1 1 *w*	*port*	
переменный порт	1 1 1 0 1 1 1 *w*		
XLAT — преобразовать байт из AL (*translate byte to AL*)	1 1 0 1 0 1 1 1		
LEA — загрузить *EA* в регистр (*load EA to register*)	1 0 0 0 1 1 0 1	*mod reg r/m*	
LDS — загрузить полный указатель (4 байта) в регистр и DS (*load pointer to DS*)	1 1 0 0 0 1 0 1	*mod reg r/m*	
LES — загрузить полный указатель (4 байта) в регистр и ES (*load pointer to ES*)	1 1 0 0 0 1 0 0	*mod reg r/m*	
LAHF — загрузить младший байт флагов в AH (*load AH with flags*) **AH ← *low* byte PSW** (см. рис. 4.2, *в*)	* 1 0 0 1 1 1 1 1		
SAHF — запомнить AH в младшем байте флагов (*store AH into flags*) ***Low byte* PSW ← AH** (см. рис. 4.2, *в*)	* 1 0 0 1 1 1 1 0		
PUSHF — включить флаги (PSW) в стек **SP ← SP – 2, *M*(SP) ← PSW**	1 0 0 1 1 1 0 0		
POPF — извлечь флаги (PSW) из стека **PSW ← *M*(SP), SP ← SP + 2**	1 0 0 1 1 1 0 1		

Арифметические команды	7 *Байт* 1 0	7 *Байт* 2 0	7 *Байт* 3 0
ADD — сложить (*add*)			
Регистр/память с регистром	0 0 0 0 0 0 *d w*	*mod reg r/m*	
Непосредственный операнд с регистром/памятью	1 0 0 0 0 0 *s w*	*mod* 0 0 0 *r/m*	*data* 8/16
Непосредственный операнд с аккумулятором	0 0 0 0 0 1 0 *w*	*data* 8/16	
ADC — сложить с переносом (*add with carry*)			
Регистр/память с регистром	0 0 0 1 0 0 *d w*	*mod reg r/m*	
Непосредственный операнд с регистром/памятью	1 0 0 0 0 0 *s w*	*mod* 0 1 0 *r/m*	*data* 8/16
Непосредственный операнд с аккумулятором	0 0 0 1 0 1 0 *w*	*data* 8/16	
INC — инкремент (*increment*); ***dst* ← *dst* + 1**			
Регистра/памяти *	1 1 1 1 1 1 1 *w*	*mod* 0 0 0 *r/m*	
Регистра *	0 1 0 0 0 *reg*		
AAA — *ASCII*-коррекция для сложения (*ASCII adjust for add*)	0 0 1 1 0 1 1 1		
DAA — десятичная коррекция для сложения (*decimal adjust for add*)	0 0 1 0 0 1 1 1		
SUB — вычесть (*subtract*)			
Регистр/память из регистра	0 0 1 0 1 0 *d w*	*mod reg r/m*	
Непосредственный операнд из регистра/памяти	1 0 0 0 0 0 *s w*	*mod* 1 0 1 *r/m*	*data* 8/16
Непосредственный операнд из аккумулятора	0 0 1 0 1 1 0 *w*	*data* 8/16	
SBB — вычесть с заемом (*subtract with borrow*)			
Регистр/память из регистра	0 0 0 1 1 0 *d w*	*mod reg r/m*	
Непосредствен. операнд из регистра/памяти	1 0 0 0 0 0 *s w*	*mod* 0 1 1 *r/m*	*data* 8/16
Непосредственный операнд из аккумулятора	0 0 0 1 1 1 0 *w*	*data* 8/16	
DEC — декремент (*decrement*); ***dst* ← *dst* – 1**			
Регистра/памяти *	1 1 1 1 1 1 1 *w*	*mod* 0 0 1 *r/m*	
Регистра *	0 1 0 0 1 *reg*		
NEG — изменить знак числа (*change sign*)	1 1 1 1 0 1 1 *w*	*mod* 0 1 1 *r/m*	
CMP — сравнить (*compare*)			
Регистр/память и регистр	0 0 1 1 1 0 *d w*	*mod reg r/m*	
Непосредственный операнд и регистр/память	1 0 0 0 0 0 *s w*	*mod* 1 1 1 *r/m*	*data* 8/16
Непосредственный операнд и аккумулятор	0 0 1 1 1 1 0 *w*	*data* 8/16	
AAS — *ASCII*-коррекция для вычитания (*ASCII adjust for subtract*)	0 0 1 1 1 1 1 1		
DAS — десятичная коррекция для вычитания (*decimal adjust for subtract*)	0 0 1 0 1 1 1 1		
MUL — умножить целые числа без знака (*multiply unsigned*)	1 1 1 1 0 1 1 *w*	*mod* 1 0 0 *r/m*	
IMUL — умножить целые числа со знаком (*integer multiply signed*)	1 1 1 1 0 1 1 *w*	*mod* 1 0 1 *r/m*	
AAM — *ASCII*-коррекция для умножения (*ASCII adjust for multiply*)	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0	

DIV — разделить целые числа без знака (*divide unsigned*)

1 1 1 1 0 1 1 *w*	*mod* 1 1 0	*r/m*

IDIV — разделить целые числа со знаком (*integer divide signed*)

1 1 1 1 0 1 1 *w*	*mod* 1 1 1	*r/m*

AAD — *ASCII*-коррекция для деления (*ASCII adjust for divide*)

1 1 0 1 0 1 0 1	0 0 0 0 1 0	1	0

CBW — преобразовать байт в слово (*convert byte to word*)

1 0 0 1 1 0 0 0

CWD — преобразовать слово в двойное слово (*convert word to double word*)

1 0 0 1 1 0 0 1

Логические команды	**7 *Байт* 1 0**	**7 *Байт* 2 0**	**7 *Байт* 3 0**

NOT — инвертировать (*invert*)

dst ← $\overline{dst}$

1 1 1 1 0 1 1 *w*	*mod* 0 1 0	*r/m*

SHL/SAL — сдвинуть логически/арифметически влево (*shift logical/arithmetic left*)

1 1 0 1 0 0 *v w*	*mod* 1 0 0	*r/m*

SHR — сдвинуть логически вправо (*shift logical right*)

1 1 0 1 0 0 *v w*	*mod* 1 0 1	*r/m*

SAR— сдвинуть арифметически вправо (*shift arithmetic right*)

1 1 0 1 0 0 *v w*	*mod* 1 1 1	*r/m*

ROL — сдвинуть циклически влево (*rotate left*)

1 1 0 1 0 0 *v w*	*mod* 0 0 0	*r/m*

ROR — сдвинуть циклически вправо (*rotate right*)

1 1 0 1 0 0 *v w*	*mod* 0 0 1	*r/m*

RCL — сдвинуть циклически влево через перенос (*rotate through CF left*)

1 1 0 1 0 0 *v w*	*mod* 0 1 0	*r/m*

RCR — сдвинуть циклически вправо через перенос (*rotate through CF right*)

1 1 0 1 0 0 *v w*	*mod* 0 1 1	*r/m*

AND — поразрядная конъюнкция операндов; ***dst*** ← ***dst*** **&** ***src***

Регистр/память с регистром	0 0 1 0 0 0 *d w*	*mod*	*reg*	*r/m*
Непосредственный операнд с регистром/памятью	1 0 0 0 0 0 0 *w*	*mod* 1 0 0	*r/m*	*data* 8/16
Непосредственный операнд с аккумулятором	0 0 1 0 0 1 0 *w*	*data* 8/16		

TEST — поразрядное И операндов без их изменения; src_1 **&** src_2

Регистр/память и регистр	1 0 0 0 0 1 0 *w*	*mod*	*reg*	*r/m*
Непосредственный операнд и регистр/память	1 1 1 1 0 1 1 *w*	*mod* 0 0 0	*r/m*	*data* 8/16
Непосредственный операнд и аккумулятор	1 0 1 0 1 0 0 *w*	*data* 8/16		

OR — поразрядное ИЛИ операндов; ***dst*** ← ***dst*** ∨ ***src***

Регистр/память с регистром	0 0 0 0 1 0 *d w*	*mod*	*reg*	*r/m*
Непосредственный операнд с регистром/памятью	1 0 0 0 0 0 0 *w*	*mod* 0 0 1	*r/m*	*data* 8/16
Непосредственный операнд с аккумулятором	0 0 0 0 1 1 0 *w*	*data* 8/16		

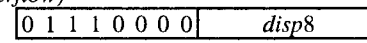
XOR — поразрядное исключающее ИЛИ (*exclusive OR*); ***dst*** ← ***dst*** ⊕ ***src***

Регистр/память с регистром	0 0 1 1 0 0 *d w*	*mod*	*reg*	*r/m*
Непосредственный операнд с регистром/памятью	1 0 0 0 0 0 0 *w*	*mod* 1 1 0	*r/m*	*data* 8/16
Непосредственный операнд с аккумулятором	0 0 1 1 0 1 0 *w*	*data* 8/16		

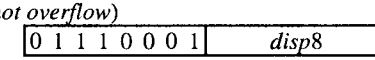
Команды манипуляции цепочками	7 *Байт 1* 0
REP — повторять операцию до CX = 0 (*repeat*)	
REP/REPZ/REPE (z = 1), REPNZ/REPNE (z = 0)	1 1 1 1 0 0 1 *z*
MOVS — переслать байт/слово (*move byte/word*)	
MOVS/MOVSB/MOVSW	1 0 1 0 0 1 0 *w*
CMPS — сравнить байт/слово (*compare byte/word*)	
CMPS/CMPSB/CMPSW	1 0 1 0 0 1 1 *w*
SCAS — сканировать байт/слово (*scan byte/word*)	
SCAS/SCASB/SCASW	1 0 1 0 1 1 1 *w*
LODS — загрузить байт/слово в AL/AX (*load byte/word to AL/AX*)	
LODS/LODSB/LODSW	1 0 1 0 1 1 0 *w*
STOS — запомнить байт/слово из AL/AX (*store byte/word from AL/AX*)	
STOS/STOSB/STOSW	1 0 1 0 1 0 1 *w*

Команды передачи управления	7 *Байт 1* 0	7 *Байт 2* 0	7 *Байт 3* 0
CALL — вызов процедуры (подпрограммы)			
Прямой внутрисегментный (*near*)	1 1 1 0 1 0 0 0	*disp low*	*disp high*
Косвенный внутрисегментный (*near*)	1 1 1 1 1 1 1 1	*mod* 0 1 0 *r/m*	
Прямой межсегментный (*far*)	1 0 0 1 1 0 1 0	*offset low*	*offset high*
		seg low	*seg high*
Косвенный межсегментный (*far*)	1 1 1 1 1 1 1 1	*mod* 0 1 1 *r/m*	
JMP — безусловный переход (*unconditional jump*)			
Прямой внутрисегментный (*near*)	1 1 1 0 1 0 0 1	*disp low*	*disp high*
Прямой внутрисегментный короткий (*short*)	1 1 1 0 1 0 1 1	*disp*8	
Косвенный внутрисегментный (*near*)	1 1 1 1 1 1 1 1	*mod* 1 0 0 *r/m*	
Прямой межсегментный (*far*)	1 1 1 0 1 0 1 0	*offset low*	*offset high*
		seg low	*seg high*
Косвенный межсегментный (*far*)	1 1 1 1 1 1 1 1	*mod* 1 0 1 *r/m*	
RET — возврат из процедуры (*return from CALL*)			
Внутрисегментный (*near*)	1 1 0 0 0 0 1 1		
Внутрисегментный и SP + *im*16 (*near*)	1 1 0 0 0 0 1 0	*data low*	*data high*
Межсегментный (*far*)	1 1 0 0 1 0 1 1		
Межсегментный и SP + *im*16 (*far*)	1 1 0 0 1 0 1 0	*data low*	*data high*
JE/JZ — перейти, если нуль/если равно (*jump on zero/on equal*)			
Флаг *ZF* = 1	0 1 1 1 0 1 0 0	*disp*8	
JNE/JNZ — перейти, если не нуль/если не равно (*jump on not zero/on not equal*)			
Флаг *ZF* = 0	0 1 1 1 0 1 0 1	*disp*8	
JS — перейти, если знак установлен (*jump on sign*)			
Флаг *SF* = 1	0 1 1 1 1 0 0 0	*disp*8	
JNS — перейти, если знак сброшен (*jump on not sign*)			
Флаг *SF* = 0	0 1 1 1 1 0 0 1	*disp*8	

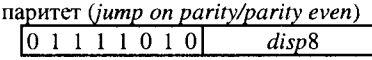
JO — перейти, если есть переполнение (*jump on overflow*)
Флаг *OF* = 1



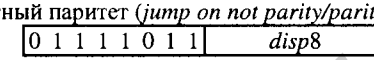
JNO — перейти, если нет переполнения (*jump on not overflow*)
Флаг *OF* = 0



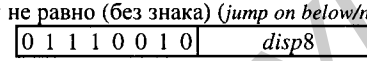
JP/JPE — перейти, если есть паритет/если четный паритет (*jump on parity/parity even*)
Флаг *PF* = 1



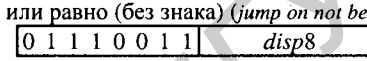
JNP/JPO — перейти, если нет паритета/если нечетный паритет (*jump on not parity/parity odd*)
Флаг *PF* = 0



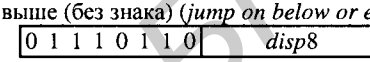
JB/JNAE/JC — перейти, если ниже/если не выше и не равно (без знака) (*jump on below/not above or equal*)
Флаг *CF* = 1



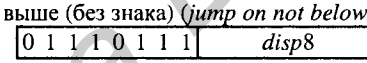
JNB/JAE/JNC — перейти, если не ниже/если выше или равно (без знака) (*jump on not below/above or equal*)
Флаг *CF* = 0



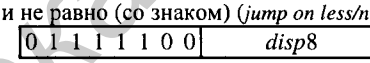
JBE/JNA — перейти, если ниже или равно/если не выше (без знака) (*jump on below or equal/not above*)
Флаг $CF \vee ZF = 1$



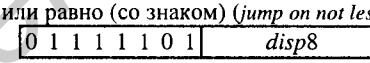
JNBE/JA — перейти, если не ниже и не равно/если выше (без знака) (*jump on not below or equal/above*)
Флаги $CF \vee ZF = 0$



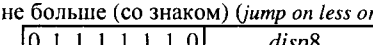
JL/JNGE — перейти, если меньше/если не больше и не равно (со знаком) (*jump on less/not greater or equal*)
Флаг $SF \oplus OF = 1$



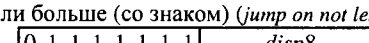
JNL/JGE — перейти, если не меньше/если больше или равно (со знаком) (*jump on not less/greater or equal*)
Флаги $SF \oplus OF = 0$ ($SF = OF$)



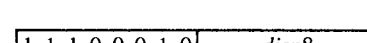
JLE/JNG — перейти, если меньше или равно/если не больше (со знаком) (*jump on less or equal/not greater*)
Флаги $SF \oplus OF \vee ZF = 1$



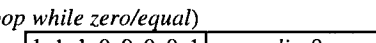
JNLE/JG — перейти, если не меньше и не равно/если больше (со знаком) (*jump on not less or equal/greater*)
Флаги $SF \oplus OF \vee ZF = 0$



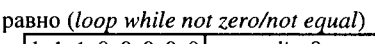
LOOP — зациклить CX раз (*loop CX times*)



LOOPZ/LOOPE — зациклить, пока нуль/равно (*loop while zero/equal*)



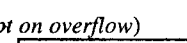
LOOPNZ/LOOPNE — зациклить, пока не нуль/не равно (*loop while not zero/not equal*)



JCXZ — перейти, если CX = 0 (*jump on CX zero*)

INT — прервать (*interrupt* — прерывание)
Определенного типа (*type* ≠ 03*h*)
Типа 3 (*type* = 03*h*)

INTO — прервать, если есть переполнение (*interrupt on overflow*)
Выполняется только при *OF* = 1



IRET — возвратиться из прерывания (*interrupt return*)

Команды управления процессором	7 *Байт 1* 0	7 *Байт 2* 0
CLC — сбросить флаг переноса (*clear carry*)		
$CF \leftarrow 0$	* 1 1 1 1 1 0 0 0	
CMC — инвертировать флаг переноса (*complement carry*)		
$CF \leftarrow \overline{CF}$	* 1 1 1 1 0 1 0 1	
STC — установить флаг переноса (*set carry*)		
$CF \leftarrow 1$	* 1 1 1 1 1 0 0 1	
CLD — сбросить флаг направления (*clear direction*)		
$DF \leftarrow 0$	* 1 1 1 1 1 1 0 0	
STD — установить флаг направления (*set direction*)		
$DF \leftarrow 1$	* 1 1 1 1 1 1 0 1	
CLI — сбросить флаг прерывания (*clear interrupt*)		
$IF \leftarrow 0$	* 1 1 1 1 1 0 1 0	
STI — установить флаг прерывания (*set interrupt*)		
$IF \leftarrow 1$	* 1 1 1 1 1 0 1 1	
HLT — остановить (*halt*)	1 1 1 1 0 1 0 0	
WAIT — ожидать активного значения сигнала *TEST* (*wait*)	1 0 0 1 1 0 1 1	
ESC — переключиться на сопроцессор (*escape — to external device*)	1 1 0 1 1 × × ×	*mod* × × × *r/m*
LOCK — префикс блокировки шины (*bus lock prefix*)	1 1 1 1 0 0 0 1	

Примечание: символом "*" отмечены команды, не требующие дополнительного пояснения.

Описание команд. Многие команды МП 8086/8088 выполняют те же операции, что и команды МП 8080/8085 (например, команды MOV, AND, OR и др.). Различия таких команд заключаются лишь в методах адресации операндов, размерах операндов (байт/слово) и способах записи некоторых однотипных для МП 8086/8088 и 8080/8085 команд на языке ассемблера. Так, если регистр содержит адрес операнда, находящегося в памяти, то он заключается в прямые скобки (на языке ассемблера для МП 8086/8088). Ниже приведено детальное описание всех команд, не отмеченных символом "*", в последовательности, соответствующей последовательности вышеприведенного описания их форматов, за исключением случаев, противоречащих логике изложения материала. Так, команды передачи данных PUSH, POP, PUSHF и POPF будут описаны в группе команд передачи управления (описание всех команд, использующих стек, сосредоточено в одном месте).

1. *Команды передачи данных*

Команды MOV *dst, src*. Эти команды выполняют операции передачи данных:

$$dst \leftarrow src.$$

Можно использовать как непосредственные операнды *imm*, так и операнды, находящиеся в аккумуляторе (*A*), регистрах (*reg*), сегментных регистрах (*seg*) и в памяти (*M*).

Выполняются команды MOV за 10 тактов (A → *M*; *M* → A), 2 такта (*reg* → *reg*; *reg* → *seg* SS, DS, ES; *seg* → *reg*), 8 + *ea* тактов (*M* → *reg*; *M* → *seg* SS, DS, ES), 9 + *ea* тактов (*reg* → *M*; *seg* → *M*), 4 такта (*imm* → *reg*) и 10 + *ea* тактов (*imm* → *M*). Операндом-получателем не может быть сегментный регистр кода *CS*.

Пример 4:

```
        MOV  CL, 240          ; CL ← 240d = F0h — непосредственный операнд
        MOV  BX, 38h          ; BX ← 0038h — непосредственный операнд
        MOV  CH, BL           ; CH ← BL = 38h
        MOV  DI, BX           ; DI ← BX = 0038h
        MOV  [DI+10h], 5A3Dh  ; M(DI+10h) ← 5A3Dh — слово (EA = DI + 10h = 0048h)
        MOV  AX, CS           ; AX ← CS — сегментный регистр кода
        MOV  DX, ES:[BX]      ; DX ← M(BX), ES: — префикс замены сегмента DS на ES
segES   MOV  DX, [BX]         ; Эквивалентная форма предыдущей команды
```

Квадратные скобки используются для указания адреса памяти ([ ] — оператор индексирования). В поле комментария для памяти *M* всегда будет указываться лишь эффективный адрес *EA*, так как сегментный регистр, используемый по умолчанию для вычисления физического адреса, определяется табл. 4.9. В последних двух командах был использован *префикс замены сегмента*, поэтому *M*(BX) означает *M*(16·ES+BX), где BX = *EA*. Далее в примерах программ, поясняющих назначение команд, будут использоваться, как правило, только регистровые и непосредственные операнды, так как операции, выполняемые командами, не зависят от режима адресации. Это позволяет использовать в программах конкретные числовые примеры, не усложняя их директивами ассемблера, определяющими имена данных в памяти.

Команды XCHG *dst, src* и NOP. Команда XCHG выполняет операцию обмена операндов:

$$dst \leftrightarrow src.$$

Если в операции участвует операнд, находящийся в памяти, то на время обмена устанавливается значение сигнала $\overline{LOCK} = 0$, независимо от наличия префикса LOCK (шина блокируется до конца выполнения команды). Команда XCHG AX, AX используется как команда NOP (пустая операция) — машинный код равен 90*h* (см. табл. 4.11).

Выполняется команда XCHG за 3 такта (*reg* ↔ A), 4 такта (*reg* ↔ *reg*) и 11 + *ea* тактов (*reg* ↔ *M*). Команда NOP выполняется за 3 такта.

Пример 5: XCHG BX, SI; BX ↔ SI.

Команды IN AL/AX, *port* и IN AL/AX, DX. Эти команды осуществляют ввод данных из внешних устройств (*I/O*):

$$AL \leftarrow I/O(port), \quad AX \leftarrow I/O(port), \quad AL \leftarrow I/O(DX), \quad AX \leftarrow I/O(DX).$$

Команды имеют два формата: *длинный* и *короткий* (двухбайтная и однобайтная команды). Ввод данных производится только в регистры AL (ввод байта) или AX (ввод слова).

Выполняются команды за 10 тактов (длинный формат) и 8 тактов (короткий формат).

Пример 6:

```
IN  AL, port  ; длинный формат, AL ← I/O(port), где port = 00 ÷ FFh
IN  AX, port  ; длинный формат (фиксированный порт), AX ← I/O(port)
IN  AL, DX    ; короткий формат, AL ← I/O(DX), где DX — адрес I/O
IN  AX, DX    ; короткий формат (переменный порт), AX ← I/O(DX)
```

Команда IN *port* аналогична одноименной команде МП 8080/8085 — адресуется $2^8 = 256$ устройств ввода. В однобайтных командах адрес порта находится в 16-разрядном регистре DX, поэтому адресуется $2^{16} = 65536$ устройств ввода.

Команда OUT *port*, AL/AX и OUT DX, AL/AX. Эти команды осуществляют вывод данных в внешние устройства (*I/O*):

$$I/O(port) \leftarrow AL, \quad I/O(port) \leftarrow AX, \quad I/O(DX) \leftarrow AL, \quad I/O(DX) \leftarrow AX.$$

Команды имеют два формата: *длинный* и *короткий* (двухбайтная и однобайтная команды) Вывод данных производится только из регистров AL (вывод байта) или AX (вывод слова).

Выполняются команды за 10 тактов (длинный формат) и 8 тактов (короткий формат).

Пример 7:

```
OUT  port, AL  ; длинный формат, I/O(port) ← AL (port = 00 ÷ FFh)
OUT  port, AX  ; длинный формат (фиксированный порт), I/O(port) ← AX
OUT  DX, AL    ; короткий формат, I/O(DX) ← AL (DX — адрес I/O)
OUT  DX, AX    ; короткий формат (переменный порт), I/O(DX) ← AX
```

Команда OUT *port* аналогична одноименной команде МП 8080/8085 — адресуется $2^8 = 256$ устройств вывода. В однобайтных командах адрес порта находится в 16-разрядном регистре DX, поэтому адресуется $2^{16} = 65536$ устройств вывода.

Команда XLAT/XLATB *name*. Эта команда (XLAT и XLATB — синонимы) заменяет содержимое регистра AL байтом из таблицы перекодировки (максимальная длина таблицы равна 256 байт):

$$AL \leftarrow M(BX+AL),$$

где *M* — *Memory* (память), BX — начальный адрес таблицы, исходное значение AL = 00 ÷ FF*h* (содержимое AL используется как индекс таблицы, адресуемой регистром BX). Команда XLAT очень удобна для выборки данных из 256-байтной таблицы. Например, если в таблицу записать коды *ASCII*, то команду XLAT можно использовать для преобразования двоичных чисел из регистра AL в коды *ASCII*.

Выполняется команда XLAT за 11 тактов.

Пример 8:

```
LEA   BX, tabl       ; BX ← адрес начала таблицы tabl (tabl — символическое имя операнда)
MOV   AL, 12         ; AL ← 0Ch  (директивы и операторы языка ассемблера см. в § 4.4)
XLAT  tabl           ; AL ← M(BX + tabl), команду XLAT здесь можно использовать
  ∴                  ;                                  без операнда tabl
MOV   BX, offset tabl ; BX ← адрес начала таблицы tabl (то же самое, что и LEA  BX, tabl)
MOV   AL, 12         ; AL ← 0Ch
XLAT  ES:tabl        ; AL ← M(BX + tabl), ES:tabl — замена сегментного регистра DS на ES.
```

Операнд *name* является *необязательным* (фиктивным) — не транслируется ассемблером в машинный код, т. е. этот операнд играет роль краткого комментария, используемого для удобства чтения программы, написанной на языке ассемблера, в которой используется несколько таблиц преобразований (обычно *name* — имя таблицы).

Выражение ES:*name* в языке ассемблера употребляется для переопределения сегментного регистра, используемого по умолчанию, на ES. Фиктивный операнд *name* в команде XLAT также пригоден для этого. Переопределение сегментного регистра можно выполнить с помо-

щью префикса замены сегмента и в не стандартной форме: segES XLAT. Пример использования команды XLAT имеется в *задаче* 10.

Команда LEA *r*16, *addr*. Эта команда вычисляет эффективный адрес *EA* операнда, находящегося в памяти, в соответствии с заданным режимом адресации и помещает его значение в указанный регистр:

$$r16 \leftarrow addr.$$

В символике команды использовано специальное обозначение *addr* = *src* для конкретизации операнда-источника (он может находиться только в памяти).

Выполняется команда за 2 + *ea* тактов.

Пример 9: LEA BP, [BX][SI]; BP ← BX + SI = *EA*.

Команды LDS *r*16, *mem* и LES *r*16, *mem*. Эти команды загружают полный указатель из памяти и записывают его в пару регистров DS: *r*16 или ES: *r*16 соответственно:

$$r16 \leftarrow M(addr),\ DS \leftarrow M(addr+2) \quad \text{и} \quad r16 \leftarrow M(addr),\ ES \leftarrow M(addr+2).$$

В символике команды использовано специальное обозначение *mem* = *src* для конкретизации операнда-источника (он может находиться только в памяти *M*).

Выполняются команды за 16 + *ea* тактов.

Пример 10:

```
LDS   SI, [BX] ; SI ← M(BX), DS ← M(BX + 2)
LES   DI, [BP] ; DI ← M(BP), ES ← M(BP + 2)
```

2. *Арифметические команды*

В МП 8086/8088 можно использовать пять типов представления исходных чисел, над которыми производятся арифметические операции:

целые числа без знака — все разряды байта и слова определяют величину числа;

целые числа со знаком — числа представлены в дополнительном коде (старшие разряды байта и слова задают знак числа);

упакованные BCD-числа (*Binary Coded Decimal* — код 8–4–2–1) — в одном байте содержится двухразрядное десятичное число от 00 до 99*d* (для каждой десятичной цифры используется одна тетрада);

неупакованные BCD-числа — в одном байте содержится одноразрядное десятичное число от 00 до 09*d* (старшая тетрада содержит код 0000*b*);

десятичные ASCII-числа (*ASCII — American Standard Code for Information Interchange* — американский стандартный код для информационного обмена) — в одном байте содержится одноразрядное десятичное число, представленное его *ASCII*-кодом от 30*h* до 39*h* для цифр от 0 до 9 (старшая тетрада содержит код 0011);

Независимо от формата представления чисел *ALU* при выполнении команд сложения, вычитания, умножения и деления производит над ними операции как над двоичными кодами. Поэтому для получения правильного результата при использовании трех последних типов чисел нужно выполнять специальные команды коррекции.

Команда NEG *dst*. Эта команда используется для изменения знака числа, представленного в дополнительном коде. Команда выполняет операцию:

$$dst \leftarrow 0 - dst$$

(вычисление *двоичного дополнения — Two's Complement*), т. е. модуль *dst* отрицательного числа преобразуется в дополнительный код. Эта операция эквивалентна операции поразрядного инвертирования числа и сложения с 1: $dst \leftarrow \overline{dst} + 1$. Флаг переноса *CF* устанавливается в 1, если операнд не равен 0. Очевидно, что повторное выполнение команды NEG дает исходный операнд, т. е. команду NEG можно использовать для получения модуля числа по его дополнительному коду. Флаг переполнения *OF* устанавливается в 1, если *dst* = –128 (байт) и –32768 (слово).

Выполняется команда за 3 такта (*reg*) и 16 + *ea* тактов (*M*).

Пример 11: NEG BX; NEG CH; NEG [BP + SI].

Задача 1. Найти дополнительный код десятичного числа $X = -825d$. Результат поместить в регистр *CX*. *Ответ* для проверки: $|X| = 339h$, $[-825d]_Д = 0 - 339h = FCC7h$. *Решение*:

```
MOV  CX, 825d   ; CX ← 825d = 0339h — модуль числа X
NEG  CX         ; CX ← FCC7h — дополнительный код числа X
 ∴              ; Проверка особых случаев
MOV  CL, 80h    ; CX ← 80h — дополнительный код числа –128 = –2^7
NEG  CL         ; CX = 80h — число не изменяется, OF ← 1 — переполнение
 ∴
MOV  CX, 8000h  ; CX ← 8000h — дополнительный код числа –32768 = –2^15
NEG  CX         ; CX = 8000h, OF ← 1 — переполнение
```

Команды ADD *dst*, *src*, ADC *dst*, *src*, DAA и AAA. Команды ADD и ADC выполняют операции сложения двоичных чисел:

$$dst \leftarrow dst + src \text{ и } dst \leftarrow dst + src + CF,$$

где CF — значение флага переноса *CF* (ADD — сложение без переноса, ADC — сложение с переносом). Операнды могут иметь различную разрядность:

$$dst8 \leftarrow dst8 + src8,\quad dst16 \leftarrow dst16 + src16,\quad dst16 \leftarrow dst16 + im16,\quad dst16 \leftarrow dst16 + im8,$$

где *im*8 — непосредственный операнд с расширением знака для сложения чисел, представленных в дополнительном коде (например, команда ADD BX, 0FFC1*h* транслируется в машинный код 83C3C1 — старший байт FF операнда *im*8 игнорируется). Числа могут быть без знака и со знаком. Если при сложении возникает перенос из старшего разряда байта (слова), то он фиксируется во флаге *CF*, т. е. флаг *CF* является расширителем разрядности чисел. Это позволяет вычислять сумму двух чисел, представленных *m* байтами (словами): сложение младших байт (слов) выполняется командой ADD, а последовательное сложение остальных байт (слов) — командой ADC, или только командой ADC, если предварительно сбросить флаг переноса *CF* в 0 командой CLC.

Команда DAA используется для десятичной коррекции содержимого регистра AL после сложения упакованных *BCD*-чисел с помощью команды ADD AL, *src*8. После коррекции в регистре AL получается сумма в упакованном *BCD*-формате. Если сумма превышает число 99*d*, то флаг переноса *CF* устанавливается в 1 (вес 100*d*).

Команда AAA используется для коррекции содержимого регистра AL после сложения неупакованных *BCD*-чисел и десятичных *ASCII*-чисел с помощью команды ADD AL, *src*8. Команда AAA выполняет операции:

AL ← 0#, где # — десятичная сумма (# = 0 … 9);
AH ← AH + 1, CF ← 1 и AF ← 1, если сумма больше 9;
AH ← AH, CF ← 0 и AF ← 0, если сумма меньше 10.

Выполняются команды ADD и ADC за 3 такта (*reg* → *reg*), 9 + *ea* тактов (*M* → *reg*), 16 + *ea* тактов (*reg* → *M*), 4 такта (*imm* → *reg*; *imm* → A) и 17 + *ea* тактов (*imm* → *M*). Команды DAA и AAA выполняются за 4 такта. Стрелка "→" указывает операнд-получатель результата операции.

Задача 2. Найти суммы десятичных чисел без знака 126*d* + 217*d* и 42483*d* + 27787*d*. *Ответ* для проверки: 126*d* + 217*d* = 343*d* = 157*h* и 42483*d* + 27787*d* = 70270*d* = 1 127E*h*. *Решение*:

```
MOV DL, 126d   ; DL ← 126d = 7Eh,   217d = D9h
ADD DL, 217d   ; DL ← DL + D9h = 57h = 87d,   CF = 1 — вес 2^8 = 256d (256 + 87 = 343d)
 ∴             ; В следующем фрагменте CF = 1 — вес 2^16 ⇒ 65536d + 4734d = 70270d
MOV SI, 42483d ; SI ← 42483d = A5F3h,   27787d = 6C8Bh
ADD SI, 27787d ; SI ← SI + 6C8Bh = 127Eh = 4734d, CF = 1
```

Задача 3. Вычислить сумму 64-разрядных двоичных чисел без знака

$$X = \text{BE58 1D32 FA77 C90D}h \text{ и } Y = \text{7C4A 1938 AD36 3943}h$$

(числа представлены четырьмя словами). Начальные адреса чисел (младших слов) равны *addr*1 и *addr*2. Сумму поместить на место второго слагаемого. *Ответ* для проверки: сумма равна 1 3AA2 366B A7AE 0250*h* (четыре слова и один байт для переноса). *Решение*:

```
      LEA  BX, addr1 ; BX ← начальный адрес первого слагаемого
      LEA  SI, addr2 ; SI ← начальный адрес второго слагаемого
      MOV  CX, 4     ; CX ← 0004h — число слов слагаемых
      CLC            ; CF ← 0 — сброс флага переноса
L1:   MOV  AX, [BX]  ; AX ← M(BX) = C90Dh, FA77h, 1D32h, BE58h
      ADC  [SI], AX  ; M(SI) ← M(SI) + AX + CF = 0250h, A7AEh, 366Bh, 3AA2h
      INC  BX        ; BX ← BX + 1
      INC  BX        ; BX ← BX + 1
      INC  SI        ; SI ← SI + 1
      INC  SI        ; SI ← SI + 1
      LOOP L1        ; CX ← CX – 1 и переход, пока CX не равно 0
      LAHF           ; AH ← low byte PSW = SF ZF 0 AF 0 PF 1 CF
      AND  AH, 1     ; AH ← 0000 000×, где × — флаг переноса CF = 0 или 1
      MOV  [SI], AH  ; M(SI) = M(addr2 + 8) ← CF
```

Задача 4. Найти суммы в дополнительном коде чисел

$$X + Y = -39d + 126d \text{ и } X + Y = -1999d + 1936d.$$

Ответ для проверки: $[-39d + 126d]_д = [87d]_д = 57h$ и $[-1999d + 1936d]_д = [-63d]_д = \text{FFC1}h$. *Решение*:

```
MOV DL, 39d    ; DL ← 39d = 27h — модуль числа X, 126d = 7Eh
NEG DL         ; DL ← D9h — дополнительный код числа X
ADD DL, 126d   ; DL ← DL + 7Eh = 57h = 87d
 ∴
MOV SI, 1999d  ; SI ← 1999d = 7CFh — модуль числа X
NEG SI         ; SI ← F831h — дополнительный код числа X
ADD SI, 1936d  ; SI ← SI + 790h = FFC1h — дополнительный код суммы (отрицательное
NEG SI         ; SI ← 003Fh = 63d — модуль суммы                          число)
```

При сложении в дополнительном коде чисел, имеющих одинаковый знак, может возникнуть переполнение разрядной сетки. В этом случае будет получен неправильный результат, и флаг *OF* будет установлен в 1. Эта ошибка не является катастрофической, так как если флаг переноса *CF* принять в качестве знакового разряда (считать, что сумма представлена девятью разрядами, т. е. диапазон представимых чисел увеличен вдвое), то получится правильный дополнительный код суммы. Флаг *OF* следует использовать для проверки правильности результата сложения после каждой команды ADD, если числа представлены в дополнительном коде.

Задача 5. Найти сумму в дополнительном коде чисел $X + Y = (-17508d) + (-20065d)$. *Ответ* для проверки: в этом случае сумма модулей чисел

$$|X| + |Y| = 17508d + 20065d = 37573d > 2^{15} = 32768,$$

поэтому возникнет переполнение разрядной сетки. *Решение*:

```
        MOV  SI, 17508d  ; SI ← 17508d = 4464h — модуль числа X
        NEG  SI          ; SI ← BB9Ch — дополнительный код числа X
        MOV  AX, 20065d  ; AX ← 20065d = 4E61h — модуль числа Y
        NEG  AX          ; AX ← B19Fh — дополнительный код числа Y
        ADD  AX, SI      ; AX ← AX + SI = 6D3Bh, CF ←1, OF ← 1 (переполнение разрядной
        JO   ERROR       ; Переход по значению флага OF = 1                    сетки)
          ∴              ; Продолжение программы
ERROR:    ∴              ; Принятие решения по ошибочным значениям операндов
```

Конечно, в подобных программах должны использоваться не непосредственные операнды, значения которых заведомо дают переполнение, а переменные, значения которых могут и не вызвать переполнения.

Задача 6. Найти сумму упакованных *BCD*-чисел $95d + 78d$. *Ответ* для проверки: $95d + 78d = 173d$. *Решение*:

```
MOV  AL, 95h  ; AL ← 95d
ADD  AL, 78h  ; AL ← AL + 78d = 0Dh, CF = 1, AF = 0
DAA           ; Десятичная коррекция: AL ← 73d, CF = 1 — вес 100d, AF = 1
```

Задача 7. Найти сумму 8-разрядных упакованных *BCD*-чисел $X = 83452097d$ и $Y = 69873836d$ (числа представлены четырьмя байтами). Начальные адреса чисел (младших байт) равны *addr*1 и *addr*2. Сумму поместить на место второго слагаемого. *Ответ* для проверки: сумма равна 1 5332 5933 (пять байт). *Решение*:

```
      LEA  BX, addr1  ; BX ← начальный адрес первого слагаемого
      LEA  SI, addr2  ; SI ← начальный адрес второго слагаемого
      MOV  CX, 4      ; CX ← 0004h — число циклов сложения
      CLC             ; CF ← 0 — сброс флага переноса
L1:   MOV  AL, [BX]   ; AL ← M(BX) = 97d, 20d, 45d, 83d
      ADC  AL, [SI]   ; AL ← AL + M(SI) + CF
      DAA             ; Десятичная коррекция
      MOV  [SI], AL   ; M(SI) ← десятичная сумма двух разрядов
      INC  BX         ; BX ← BX + 1
      INC  SI         ; SI ← SI + 1
      LOOP L1         ; CX ← CX – 1 и переход, пока CX не равно 0
      LAHF            ; AH ← low byte PSW = SF ZF 0 AF 0 PF 1 CF
```

```
AND   AH, 1      ; AH ← 0000 000×, где × — флаг переноса CF = 0 или 1
MOV   [SI], AH   ; M(SI) = M(addr2 + 4) ← CF
```

Задача 8. Найти суммы неупакованных *BCD*-чисел 05*d* + 08*d* и десятичных *ASCII*-чисел 35*h* и 38*h*. *Ответ* для проверки: 05*d* + 08*d* = 13*d* и 5*d* + 8*d* = 13*d* (в десятичных эквивалентах). *Решение*:

```
MOV   AX, 0605h ; AH ← 06d, AL ← 05d
ADD   AL, 08h   ; AL ← AL + 08h = 0Dh, CF ← 0, AF ← 0
AAA             ; AL ← 03, AH ← AH + 1 = 07d, CF ← 1, AF ← 1
 ∴
MOV   AX, 3635h ; AH ← 36h — код ASCII цифры 6, AL ← 35h — код ASCII цифры 5
ADD   AL, 38h   ; AL ← AL + 38h = 6Dh, CF = 0, AF = 0
AAA             ; AL ← 03, AH ← AH + 1 = 37h, CF ← 1, AF ← 1
OR    AL, 30h   ; AL ← 33h — код ASCII цифры 3 (либо команда ADD AL, 30h)
```

Задача 9. Вычислить сумму 8-разрядных неупакованных десятичных *ASCII*-чисел

3930 3735 3336 3233 + 3632 3838 3636 3935 = 01 0701 0303 0209 0504

(слагаемые представлены восемью байтами в *ASCII*-кодах, а сумма — девятью байтами в неупакованном *BCD*-формате). Начальные адреса чисел (старших разрядов) равны *ascii*1 и *ascii*2. Сумму поместить на место второго слагаемого. *Решение*:

```
      MOV   CX, 8          ; CX ← 8 — указание разрядности чисел
      LEA   SI, ascii1 + 7 ; SI ← адрес младшего разряда первого операнда
      LEA   DI, ascii2 + 7 ; DI ← адрес младшего разряда второго операнда
      CLC                  ; CF ← 0
L1:   MOV   AH, 0          ; AH ← 0
      MOV   AL, [SI]       ; AL ← M(SI)
      ADC   AL, [DI]       ; AL ← AL + M(DI) + CF
      AAA                  ; ASCII-коррекция
      MOV   [DI], AL       ; M(DI) ← AL — BCD-код суммы двух разрядов
      DEC   SI             ; SI ← SI – 1
      DEC   DI             ; DI ← DI – 1
      LOOP  L1             ; CX ← CX – 1 и переход, пока CX не равно 0
      MOV   [DI], AH       ; M(DI) = M(ascii2 – 1) ← CF (флаг переноса)
```

Данные, вводимые с клавиатуры и выводимые на дисплей компьютера, имеют *ASCII*-формат. Команды *ASCII*-коррекции результатов сложения (AAA), вычитания (AAS), умножения (AAM) и деления (AAD) позволяют выполнять эти операции, не прибегая к преобразованию входных данных в двоичные числа. Следующая программа поясняет сложение чисел в *ASCII*-формате с получением суммы также в *ASCII*-формате, непосредственно пригодном для вывода на дисплей.

Задача 10. Вычислить сумму 8-разрядных неупакованных десятичных *ASCII*-чисел

3930 3735 3336 3233 + 3632 3838 3636 3935 = 31 3731 3333 3239 3534

(слагаемые и сумма представлены восемью и девятью байтами в *ASCII*-кодах). Начальные адреса чисел (старших разрядов) равны *ascii*1 и *ascii*2. Сумму поместить на место второго слагаемого. Начальный адрес таблицы *ASCII*-чисел 30 31 32 33 34 35 36 37 38 39 для преобразования неупакованных *BCD*-цифр равен *tabl*. *Решение*:

```
        LEA   BX, tabl      ; BX ← начальный адрес таблицы преобразования
        MOV   CX, 8         ; CX ← 8 — число циклов сложения
        LEA   SI, ascii1 + 7 ; SI ← адрес младшего разряда первого операнда
        LEA   DI, ascii2 + 7 ; DI ← адрес младшего разряда второго операнда
        CLC                 ; CF ← 0
L1:     MOV   AH, 0         ; AH ← 0
        MOV   AL, [SI]      ; AL ← M(SI)
        ADC   AL, [DI]      ; AL ← AL + M(DI) + CF
        AAA                 ; ASCII-коррекция
        XLAT                ; AL ← ASCII-код одного разряда суммы
        MOV   [DI], AL      ; M(DI) ← AL — ASCII-код одного разряда суммы
        DEC   SI            ; SI ← SI – 1
        DEC   DI            ; DI ← DI – 1
        LOOP  L1            ; CX ← CX – 1 и переход, пока CX не равно 0
        MOV   AL, AH        ; AL ← AH — перенос из старшего разряда суммы
        XLAT                ; AL ← ASCII-код переноса (30h или 31h)
        MOV   [DI], AL      ; M(DI) = M(ascii2 – 1) ← ASCII-код переноса
```

Команды SUB *dst*, *src*, SBB *dst*, *src*, DAS и AAS. Команды SUB и SBB выполняют операции вычитания двоичных чисел

$$dst \leftarrow dst - src \quad \text{и} \quad dst \leftarrow dst - src - \text{CF},$$

где CF — значение флага заема *CF* (SUB — вычитание без заема, SBB — вычитание с заемом *Borrow* = CF). Операнды могут иметь различную разрядность:

$$dst8 \leftarrow dst8 - src8, \quad dst16 \leftarrow dst16 - src16, \quad dst16 \leftarrow dst16 - im16, \quad dst16 \leftarrow dst16 - im8,$$

где *im*8 — непосредственный операнд с расширением знака для вычитания чисел, представленных в дополнительном коде (например, команда SUB BX, 003F*h* транслируется в машинный код 83EB3F — старший байт 00 операнда *im*8 игнорируется). Числа могут быть без знака и со знаком. Если при вычитании возникает заем из старшего разряда байта (слова), то он фиксируется во флаге *CF*. Это позволяет вычислять разность двух чисел, представленных *m* байтами (словами): вычитание младших байт (слов) выполняется командой SUB, а последовательное вычитание остальных байт (слов) — командой SBB, или только командой SBB, если предварительно сбросить флаг переноса *CF* в 0 командой CLC.

Команда DAS используется для десятичной коррекции содержимого регистра AL после вычитания упакованных *BCD*-чисел с помощью команды SUB AL, *src*8. После коррекции в регистре AL получается разность в упакованном *BCD*-формате. Если AL < *src*8, то флаг переноса *CF* устанавливается в 1 (вес –100*d*) и значение AL ← 100*d* – | AL – *src*8 |.

Команда AAS используется для коррекции содержимого регистра AL после вычитания однобайтовых неупакованных *BCD*-чисел и десятичных *ASCII*-чисел с помощью команды SUB AL, *src*8. Команда AAA выполняет операции:

AL ← 0#, где # — десятичная сумма (# = 0 … 9);
AH ← AH – 1, CF ← 1 и *AF* ← 1, если разность AL – *src*8 < 0;
AH ← AH, CF ← 0 и *AF* ← 0, если разность AL – *src*8 ≥ 0.

Выполняются команды SUB и SBB за 3 такта (*reg* → *reg*), 9 + *ea* тактов (*M* → *reg*), 16 + *ea* тактов (*reg* → *M*), 4 такта (*imm* → *reg*; *imm* → A) и 17 + *ea* тактов (*imm* → *M*). Команды DAS и

AAS выполняются за 4 такта. Стрелка "→" указывает операнд-получатель результата операции.

Задача 11. Вычислить разность 64-разрядных двоичных чисел без знака

$$X = \text{BE58 1D32 FA77 C90D}h \text{ и } Y = \text{7C4A 1938 AD36 3943}h$$

(числа представлены четырьмя словами). Младшие слова чисел расположены по адресам *addr*1 и *addr*2. Сумму поместить на место второго слагаемого. *Ответ* для проверки: разность равна 0 420E 03FA 4D41 8FCA*h* (четыре слова и один байт для переноса). *Решение*: в *задаче* 3 команду ADC заменить командой SBB.

Если *dst* ≥ *src* (*dst* и *src* — двоичные числа без знака), то разность *dst* – *src* является положительным числом или числом 0 и флаг CF = 0. Если же *dst* < *src*, то разность *dst* – *src* является отрицательным числом и флаг CF = 1 (заем) имеет вес -2^8 для байта и -2^{16} для слова. Например, для 8-разрядных операндов

$$dst - src = 0 - 255d = 0000\ 0000 - 1111\ 1111 = 1\ .\ 0000\ 0001$$

(слева от точки указано значение флага *CF*), т. е. младшие 8 разрядов представляют собой положительное число. С другой стороны, если разряд *CF* считать знаковым, то разность будет представлять собой дополнительный код числа –255*d*, но знаковый разряд вышел за пределы 8-разрядной сетки.

Если диапазон представления двоичных чисел ограничить соотношениями

$$dst \le 2^7 - 1 = 127,\ src \le 2^7 = 128 \text{ для байта и } dst \le 2^{15} - 1 = 32767,\ src \le 2^{15} = 32768 \text{ для слова,}$$

то знаковый разряд будет находиться в старшем разряде байта и слова (знаковый разряд не выходит за пределы соответствующей разрядной сетки). В этом случае разность всегда будет представлена в дополнительном коде без переполнения разрядной сетки — значение флага *CF* будет совпадать со значением старшего разряда байта и слова, а значит, состояние флага *CF* просто игнорируется. Действительно,

$$dst - src = dst + (0 - src) = [dst]_д + [-src]_д,$$

т. е. операции вычитания соответствует операция сложения дополнительных кодов чисел *dst* и –*src* с представлением суммы в дополнительном коде.

Если числа представлены *m* байтами (словами), то знаковый разряд будет находиться в старшем разряде старшего байта (слова), и все сказанное выше справедливо для таких чисел с учетом нового диапазона их абсолютных значений. Это и есть числа со знаком.

Задача 12. Решить *задачу* 1 с помощью команды SUB. *Решение*:

```
SUB  CX, CX    ; CX ← 0
SUB  CX, 825d  ; CX ← 0 – 825d = FCC7h — дополнительный код числа X
```

Задача 13. Решить вторую часть *задачи* 3 с помощью команды SUB. *Решение*:

```
MOV  SI, 1936d  ; SI ← 1936d = 790h
SUB  SI, 1999d  ; SI ← SI – 7CEh = FFC1h — дополнительный код разности
NEG  SI         ; SI ← 003Fh = 63d — модуль разности
```

Пример 12:

```
MOV  AL, 95h    ; AL ← 95d.  Вычисление разности 95d – 78d = 17d
SUB  AL, 78h    ; AL ← AL – 78d = 1Dh, CF = 0, AF = 1
DAS             ; AL ← 17d, CF = 0, AF = 1
```

```
  ∴
MOV  AL, 78h  ; AL ← 78d.  Вычисление разности  78d – 95d = –17d
SUB  AL, 95h  ; AL ← AL – 95d = E3h, CF = 1, AF = 0
DAS           ; AL ← 83d, CF = 1 (вес –100d), AF = 0
  ∴
MOV  AL, 0    ; AL ← 0.  Вычисление разности  0 – 99d = –99d
SUB  AL, 99h  ; AL ← AL – 99d = 67h, CF = 1, AF = 1
DAS           ; AL ← 01d, CF = 1 (вес –100d), AF = 1
```

Задача 14. Найти разности неупакованных *BCD*-чисел 05*d* – 08*d* и десятичных *ASCII*-чисел 35*h* и 38*h*. *Ответ* для проверки: 05*d* – 08*d* = –3*d* и 5*d* – 8*d* = –3*d* (в десятичных эквивалентах). *Решение*:

```
MOV  AX, 0605h  ; AL ← 05d
SUB  AL, 08d    ; AL ← AL – 08h = FDh, CF ← 1, AF ← 1
AAS             ; AL ← 07, AH ← AH – 1 = 05d, CF = 1, AF = 1
  ∴
MOV  AX, 3635h  ; AH ← 36h — код ASCII цифры 6, AL ← 35h — код ASCII цифры 5
SUB  AL, 38h    ; AL ← AL – 38h = FDh, CF ← 1, AF ← 1
AAS             ; AL ← 07, AH ← AH – 1 = 35h, CF = 1, AF = 1
OR   AL, 30h    ; AL ← 37h — код ASCII цифры 7 (либо команда ADD AL, 30h)
```

Задача 15. Найти дополнительный код числа X = –7E05 DCF6*h* (модуль отрицательного числа представлен 31 двоичным разрядом, а старший разряд старшего слова — знаковый). Результат поместить в регистры SI:DI. *Ответ* для проверки: 0 – 7E05 DCF6*h* = 81FA 230A*h* — дополнительный код. *Решение*:

```
XOR  DI, DI      ; DI ← 0
SUB  DI, 0DCF6h  ; DI ← 0 – DCF6h = 230Ah, CF ← 1
MOV  SI, 0       ; SI ← 0 (команда MOV флаги не изменяет)
SBB  SI, 7E05h   ; SI ← 0 – 7E05h – CF = 81FAh, CF ← 1 — игнорируется
```

Команды SUB и SBB позволяют вычислять дополнительный код отрицательных чисел, представленных любым числом *m* байт или слов в то время, как команда NEG способна вычислять дополнительный код только для одного байта и слова.

Команда CMP src_1, src_2. Эта команда производит сравнение операндов src_1 и src_2 с помощью операции вычитания для определения существующих между ними соотношений (равно, не равно, больше, меньше или равно):

$$src_1 - src_2.$$

Операнды могут иметь различную разрядность:

$$src8_1 - src8_2, \quad src16_1 - src16_2, \quad src16_1 - im16_2, \quad src16_1 - im8,$$

где *im*8 — непосредственный операнд с расширением знака для вычитания чисел, представленных в дополнительном коде (например, команда CMP DX, 0FF8E*h* транслируется в машинный код 83FA8E — старший байт FF операнда *im*8 игнорируется). Числа могут быть без знака и со знаком. Если $src_1 < src_2$, то флаг *CF* устанавливается в 1, а если $src_1 = src_2$, то флаг *ZF* устанавливается в 1. Команда CMP подобна команде SUB, но разность $src_1 - src_2$ не запоминается (операнды src_1 и src_2 не изменяются). Результатом выполнения команды CMP являются значения флагов, используемых в командах условных переходов.

Выполняется команда CMP за 3 такта (*reg – reg*), 9 + *ea* тактов (*M – reg*; *reg – M*), 10 + *ea* тактов (*M – imm*), 4 такта (*reg – imm*; A – *imm*).

Пример 13:

```
MOV   CH, 9Ah
CMP   CH, 0Bh      ; CH – 0Bh = 8Fh ⇒ CF = 0, ZF = 0, SF = 1, PF = 0, OF = 0
  ∴
MOV   BX, 4D7Ch
CMP   BX, 4D7Ch    ; BX – 4D7Ch = 0 ⇒ CF = 0, ZF = 1, SF = 0, PF = 1, OF = 0
  ∴
MOV   DX, 3D7Ch
CMP   DX, 0FF8Eh   ; DX – FF8Eh = 3DEEh ⇒ CF = 1, ZF = 0, SF = 0, PF = 1, OF = 0
  ∴
MOV   DL, 7Ch
CMP   DL, 9Eh      ; DL – 9Eh = DEh ⇒ CF = 1, ZF = 0, SF = 1, PF = 1, OF = 1
```

Значение флага паритета *PF* определяется числом единиц в младшем байте результата выполнения команды. Например, в третьем фрагменте в слове результата 3DEE*h* содержится нечетное число единиц, но флаг *PF* = 1, так как в младшем байте EE*h* содержится четное число единиц.

Если в последнем фрагменте числа представлены в дополнительном коде $[X]_д$ = 7C*h* и $[Y]_д$ = 9E*h*, то X = +124*d*, Y = –62*h* = –98*d* и X – Y = +222*d* > 127*d* — переполнение разрядной сетки (флаг OF = 1). В этом случае флаги *OF* и *SF* можно использовать для выполнения условных переходов. Но эти же двоичные коды 7C*h* и 9E*h* в другой задаче могут использоваться как числа без знака. Тогда X = 124*d*, Y = 158*d* и разность X – Y = –34*d*. В этом случае флаги *OF* и *SF* для выполнения условных переходов использовать нельзя. Почему нельзя использовать флаг *SF*, поясняет первый фрагмент: X = 9A*h* = 154*d*, Y = 0B*h* = 11*d* и разность X – Y = +143*d*, а значение флага *SF* = 1 указывает, что число отрицательное.

Команда CBW. Эта команда преобразует байт со знаком, находящийся в регистре AL, в слово со знаком AX:

$$AH_7 \ldots AH_1AH_0 \leftarrow AL_7 \ldots AL_7AL_7,$$

т. е. производится расширение знака числа, представленного в дополнительном коде — значение старшего (знакового) разряда AL_7 записывается во все разряды регистра AH. Если в регистре AL разряд 7 равен 0 (число положительное), то устанавливается значение AH = 00*h*, а если разряд 7 равен 1 (число отрицательное), то устанавливается значение AH = FF*h*.

Выполняется команда за 2 такта.

Команда CWD. Эта команда преобразует слово со знаком, находящееся в регистре AX, в двойное слово со знаком DX:AX:

$$DX_{15} \ldots DX_1DX_0 \leftarrow AX_{15} \ldots AX_{15}AX_{15},$$

т. е. производится расширение знака числа, представленного в дополнительном коде. Регистр DX используется как расширитель 16-разрядного двоичного числа AX до 32-разрядного двоичного числа DX:AX. Если в регистре AX разряд 15 (знаковый разряд) равен 0 (число положительное), то устанавливается значение DX = 0000*h*, а если разряд 15 равен 1 (число отрицательное), то устанавливается значение DX = FFFF*h*.

Выполняется команда за 5 тактов.

Задача 16. Вычислить сумму в дополнительном коде слова BX и двойного слова в памяти, расположенного по адресу 2000*h*, и двойное слово в памяти заменить вычисленной суммой. *Решение*:

```
MOV  SI, 2000h    ; SI ← 2000h — адрес первого байта двойного слова
MOV  AX, BX       ; AX ← BX
CWD               ; Выравнивание длины слагаемых (слово AX → двойное слово DX:AX)
ADD  [SI], AX     ; M(SI) ← AX + M(SI) — вычисление суммы младших слов
ADC  [SI + 2], DX ; M(SI + 2) ← DX + M(SI + 2) + CF — вычисление суммы старших слов
```

Например, если значения операндов BX = 91B7*h* и *M*(2003 … 2000) = 87C4 0D36*h*, то DX:AX = FFFF 91B7*h* и *M*(2003 … 2000) = 87C4 0D36 + FFFF 91B7 = 87C3 9EED*h*.

Команды MUL *src* и AAM. Команда MUL выполняет умножение целых чисел без знака. Перемножаемые числа могут быть байтами и словами. Использование непосредственных операндов не допускается. Один из сомножителей должен находиться в регистре AL (умножение на *src*8) или в регистре AX (умножение на *src*16). Произведение чисел поступает в регистр AX или в регистры DX:AX соответственно:

$$AX \leftarrow AL \times src8, \quad DX{:}AX \leftarrow AX \times src16.$$

Регистр DX используется как расширитель 16-разрядного регистра AX до 32-разрядного регистра DX:AX. Флаги *CF* и *OF* устанавливаются в 0, если старший байт произведения AH = 0 (сомножители — байты) или старшее слово произведения DX = 0 (сомножители — слова). В противном случае эти флаги устанавливаются в 1. Проверив состояния этих флагов, можно определить, размещается ли произведение байт (слов) в одном байте (слове). Остальные флаги могут модифицироваться, но их состояния не предсказуемы.

Команда AAM используется для коррекции содержимого регистра AX после умножения одноразрядных неупакованных *BCD*-чисел (AL × *src*8). Умножать *ASCII*-числа нельзя, если они предварительно не переведены в неупакованный *BCD*-формат. При коррекции производится деление содержимого регистра AL на 10*d* и загрузка частного (*quotient*) в регистр AH, а остатка (*remainder*) — в регистр AL:

$$AH \leftarrow quot(AL/10d), \quad AL \leftarrow rem(AL/10d).$$

Выполняется команда MUL за 70 … 77 тактов (8-разрядный регистр), 118 … 133 тактов (16-разрядный регистр), (76 … 83) + *ea* тактов (8-разрядный операнд в памяти) и (124 … 139) + *ea* тактов (16-разрядный операнд в памяти). Команда AAM выполняется за 83 такта.

Пример 14: MUL CL; MUL CX; MUL [SI].

Задача 17. Вычислить произведение целых чисел без знака 142*d* и 246*d*, представимых одним байтом. *Ответ* для проверки: 142*d* × 246*d* = 34932*d*. *Решение*:

```
MOV  BH, 246d     ; BH ← F6h = 246d
MOV  AL, 142d     ; AL ← 8Eh = 142d
MUL  BH           ; AX ← AL × BH = 8874h = 34932d
```

Задача 18. Вычислить произведение целых чисел без знака 51 730*d* и 64 928*d*, представимых двумя байтами (словами). *Ответ* для проверки: 51 730*d* × 64 928*d* = 3 358 725 440*d*. *Решение*:

```
MOV  CX, 64928d   ; CX ← FDA0h = 64928d
MOV  AL, 51730d   ; AX ← 6A12h = 51730d
MUL  CX           ; DX:AX ← C8321540h = 3 358 725 440d
```

Задача 19. Вычислить произведение неупакованных *BCD*-чисел 08*d* и 07*d*. *Решение*:

```
MOV  CL, 7  ; CL ← 07h
MOV  AL, 8  ; AL ← 08h
MUL  CL     ; AX ← 07 × 08 = 0038h = 56d
AAM         ; AX ← 0506 — неупакованный BCD-код произведения
```

Команда IMUL *src*. Эта команда выполняет умножение целых чисел со знаком. Перемножаемые числа, представленные в дополнительном коде, могут быть байтами и словами. Использование непосредственных операндов не допускается (в МП 80286 и выше — допускается). Произведение чисел также представляется в дополнительном коде. Для одного из сомножителей и произведения используются те же регистры, что и при выполнении команды MUL:

$$AX \leftarrow AL \times src8, \quad DX{:}AX \leftarrow AX \times src16.$$

Флаги *CF* и *OF* устанавливаются в 0, если при умножении байт (слов) старший байт (слово) произведения является расширением знака младшего байта (слова). В противном случае эти флаги устанавливаются в 1. Проверив состояния этих флагов, можно определить, размещается ли произведение в одном байте (слове).

Выполняется команда за 80 ... 98 тактов (8-разрядный регистр), 128 ... 154 тактов (16-разрядный регистр), (86 ... 104) + *ea* тактов (8-разрядный операнд в памяти) и (134 ... 160) + *ea* тактов (16-разрядный операнд в памяти).

Пример 15: MUL CL; MUL CX; MUL [BP].

Задача 20. Написать программы вычисления произведений однобайтных чисел

$$P_1 = (-91d) \times (-114d) = +10374d \quad \text{и} \quad P_2 = (+69d) \times (-114d) = -7866d.$$

Решение:

```
MOV  BH, 91d   ; BH ← 91d = 5Bh — модуль числа –91d
NEG  BH        ; BH ← 0 – BH = A5h — дополнительный код числа –91d
MOV  AL, 114d  ; AL ←114d = 72h — модуль числа –114d
NEG  AL        ; AL ← 0 – AL = 8Eh — дополнительный код числа –114d
IMUL BH        ; AX ← 2886h — дополнительный код P1 (AX = +10374d)
 ∴
MOV  BH, 69d   ; BH ← 69d = 45h — дополнительный код
MOV  AL, 114d  ; AL ←114d = 72h — модуль числа –114d
NEG  AL        ; AL ← 0 – AL = 8Eh — дополнительный код числа –114d
IMUL BH        ; AX ← E146h — дополнительный код P2 (число отрицательное)
MOV  CX, AX    ; CX ← AX
NEG  CX        ; CX ← 0 – E146h = 1EBAh = 7866d — модуль P2
```

Задача 21. Вычислить произведение *P* чисел *X* = –24451*d* и *Y* = +31974*d*, заданных словами в дополнительном коде: $[X]_д$ = A07D*h* и $[Y]_д$ = 7CE6*h*. Модуль произведения записать в регистры DI:SI. *Ответ* для проверки: *P* = (–24451*d*) × (+31974*d*) = –781 796 274*d*. *Решение*:

```
MOV  DX, 0A07Dh ; DX ← A07Dh — дополнительный код числа X = –24451d
MOV  AX, 7CE6h  ; AX ← 7CE6h — дополнительный код числа  Y = +31974d
IMUL DX         ; DX:AX ← D166BC4Eh — дополнительный код произведения P
XOR  SI, SI     ; SI ← 0
SUB  SI, AX
```

```
MOV   DI, 0        ; DI ← 0
SBB   DI, DX       ; DI:SI ← 2E9943B2h — модуль произведения P = –781 796 274d
```

Команды DIV *src* и AAD. Команда DIV выполняет операцию деления целых чисел без знака. Непосредственные операнды использовать нельзя. Делимое должно находиться в регистре AX (деление на *src*8) или в регистрах DX:AX (деление на *src*16). Частное (*quotient*) и остаток (*remainder*) от деления поступают в регистры AL и AH или AX и DX соответственно:

$$\text{AL} \leftarrow quot(\text{AX} / src8),\ \text{AH} \leftarrow rem(\text{AX} / src8) \ \text{ или}$$
$$\text{AX} \leftarrow quot(\text{DX:AX} / src16),\ \text{DX} \leftarrow rem(\text{DX:AX} / src16).$$

Остаток всегда меньше делителя ($rem < src8$ и $rem < src16$). После выполнения команды DIV состояния флагов непредсказуемы.

Если частное от деления *quot* требует для своего представления больше 8 разрядов при *src*8 или 16 разрядов при *src*16 (переполнение разрядной сетки), то автоматически генерируется прерывание INT 0 ошибки деления ("*divide by zero*" — деление на 0; см. рис. 4.26), выводящее на экран сообщение: *Your program caused a divide overflow error. If the problem persists, contact your program vendor* (выводимый текст сообщения зависит от типа компьютера). Переполнение может возникнуть только при выполнении условий:

$$\text{AH} \geq src8 \ \text{ и } \ \text{DX} \geq src16,$$

где AH и DX — старший байт и слово делимого (разрядность частного *quot* должна быть не больше разрядности делителей *src*8 и *src*16). Значит, с помощью операции сравнения операндов можно предотвратить выполнение деления в случаях, вызывающих прерывание INT 0.

Команда AAD используется для коррекции перед делением AX / *src*8 — неупакованного двухразрядного *BCD*-числа в регистре AX на неупакованное *BCD*-число *src*8. Коррекция заключается в преобразовании неупакованного двухразрядного *BCD*-числа в регистре AX в двоичное число (AL ← AL + AH ×10, AH ← 0). Делить *ASCII*-числа нельзя, если они не переведены в неупакованный *BCD*-формат.

Выполняется команда DIV за 80…90 тактов (8-разрядный регистр), 144…162 такта (16-разрядный регистр), (86…96) + *ea* тактов (8-разрядный операнд в памяти) и (150…168) + *ea* тактов (16-разрядный операнд в памяти). Команда AAD выполняется за 60 тактов.

Задача 22. Написать программы вычисления частного $quot(X_k / Y_k)$ и остатка $rem(X_k / Y_k)$, где $X_1 = 41984d$, $Y_1 = 165d$ (8-разрядный операнд *src*) и $X_2 = 4272225536d$ = FEA4FD00*h*, $Y_2 = 65189d$ (16-разрядный операнд *src*). *Ответ* для проверки:

$quot(X_1 / Y_1) = 254d$ = FE*h*, $rem(X_1 / Y_1) = 74d = 4Ah$;
$quot(X_2 / Y_2) = 65535d$ = FFFF*h*, $rem(X_2 / Y_2) = 64421d$ = FBA5*h*.

Решение:

```
MOV   BH, 165d     ; BH ← 165d = A5h — делитель Y1
MOV   AX, 41984d   ; AX ← 41984d = A400h — делимое X1
DIV   BH           ; AL ← quot(X1 / Y1) = FEh = 254d, AH ← rem(X1 / Y1) = 4Ah = 74d
  ∴
MOV   BX, 65189d   ; BX ← 65189d = FEA5h — делитель Y2
MOV   AX, 0FD00h   ; AX ← FD00h — младшее слово делимого X2
MOV   DX, 0FEA4h   ; DX ← FEA4h — старшее слово делимого X2
DIV   BX           ; AX ← quot(X2/Y2) = FFFFh = 65535d,
                   ; DX ← rem(X2/Y2) = FBA5h = 64421d
```

С помощью операции сравнения операндов можно предотвратить выполнение деления в случаях, вызывающих прерывание INT 0:

```
        CMP  DX, src16 ; DX – src16 (проверка операндов на предмет "divide by zero")
        JNB  L_OVER    ; Переход по значению флага CF = 0 (DX ≥ src16)
        DIV  src16     ; Деление DX:AX на src16
         ∴             ; Продолжение программы
L_OVER:  ∴             ; Принятие решения по ошибочным значениям операндов
```

Аналогично выполняется и анализ условия AH ≥ *src*8.

Задача 23. Написать программы вычисления частного и остатка для операций деления неупакованных *BCD*-чисел 0601/07 (*scr*8 = 07 > AH = 06) и 0905 / 07 (*scr*8 = 07 < AH = 09). *Решение*:

```
MOV  DH, 7      ; DH ← 07
MOV  AX, 0601h  ; AX ← 0601
AAD             ; AX ← 003Dh = 61d (коррекция операнда AX)
DIV  DH         ; AX ← 0508, AL = quot(0601 / 07) = 08, AH ← rem(0601 / 07) = 05
 ∴
MOV  DL, 7      ; DL ← 07
MOV  AX, 0905h  ; AX ← 0905
AAD             ; AX ← 005Fh = 95d
DIV  DL         ; AX ← 040D, 0Dh = 13d — не BCD-число (нарушено условие scr8 > AH)
```

Команда IDIV *src*. Эта команда выполняет операцию деления целых чисел со знаком (чисел, представленных в дополнительном коде). Использование непосредственных операндов не допускается. Делимое должно находиться в регистре AX (деление на *src*8) или в регистрах DX:AX (деление на *src*16). Дополнительные коды частного (*quotient*) и остатка (*remainder*) от деления поступают в регистры AL и AH или AX и DX соответственно:

$$\text{AL} \leftarrow quot(\text{AX}/src8),\ \ \text{AH} \leftarrow rem(\text{AX}/src8) \text{ или}$$
$$\text{AX} \leftarrow quot(\text{DX:AX}/src16),\ \ \text{DX} \leftarrow rem(\text{DX:AX}/src16)$$

(здесь и далее для упрощения записи полагается, что все операнды и результат представлены в дополнительном коде, например, DX:AX = $[\text{DX:AX}]_д$, *src*16 = $[src16]_д$, *quot*(AX/*src*8) = *quot* = = $[quot]_д$ и т. д.).

Остаток имеет тот же знак, что и делимое, а его абсолютное значение всегда меньше, чем абсолютное значение делителя. Состояния шести изменяющихся флагов непредсказуемы. При переполнении разрядной сетки автоматически генерируется прерывание INT 0 ошибки деления ("*divide by zero*" — деление на 0; см. рис. 4.26). Переполнение может возникнуть только при выполнении условий:

$$quot(\text{AX} / src8) < -128d, \qquad quot(\text{AX} / src8) > +127d;$$
$$quot(\text{DX:AX} / src16) < -32768d, \qquad quot(\text{DX:AX} / src16) > 32767d.$$

Выполняется команда IDIV за 101 ... 112 тактов (8-разрядный регистр), 165 ... 184 тактов (16-разрядный регистр), (107 ... 118) + *ea* тактов (8-разрядный операнд в памяти) и (171 ... 190) + *ea* тактов (16-разрядный операнд в памяти).

Задача 24. Написать программы вычисления частного $quot(X_k / Y_k)$ и остатка $rem(X_k / Y_k)$ для операций деления чисел, где

$X_1 = +11938d$, $Y_1 = -117d$ (8-разрядный операнд *src*) и
$X_2 = -2A394399h$, $Y_2 = -26361d$ (16-разрядный операнд *src*).

Вычислить модули частного и остатка. *Ответ* для проверки:

$quot(X_1 / Y_1) = -102d = -66h$, $rem(X_1 / Y_1) = +4$;
$quot(X_2 / Y_2) = +26\ 872d = +68F8h$, $rem(X_2 / Y_2) = -23\ 137d = -5A61h$.

Решение:

```
    MOV  BH, 117d    ; BH ← 117d = 75h — модуль делителя Y1
    NEG  BH          ; BH ← 0 – BH = 8Bh — дополнительный код делителя Y1
    MOV  AX, 11938d  ; AX ← 11938d = 2EA2h — дополнительный код делимого X1
    IDIV BH          ; AL ← [quot]Д = 9Ah, AH ← [rem]Д = 04h
    MOV  DL, AL
    NEG  DL          ; DL ← 0 – DL = 66h = 102d — модуль частного quot(X1 / Y1)
     ∴
    MOV  BX, 26361d  ; BX ← 26361d = 66F9h — модуль делителя Y2
    NEG  BX          ; BX ← 0 – BX = 9907h — дополнительный код делителя Y2
    XOR  AX, AX      ; AX ← 0
    SUB  AX, 4399h   ; AX ← 0 – 4399h = BC67h — младшее слово [X2]Д
    MOV  DX, 0       ; DX ← 0
    SBB  DX, 2A39h   ; DX ← 0 – 2A39h – CF = D5C6h — старшее слово [X2]Д
                     ; DX:AX = D5C6 BC67h = [X2]Д — дополнительный код делимого X2
    IDIV BX          ; AX ← [quot]Д = 68F8h, DX ← [rem]Д = A59Fh,
    MOV  SI, AX      ; SI ← AX
    MOV  DI, DX      ; DI ← DX
    TEST SI, 8000h   ; SI & 8000h — проверка знакового разряда частного
    JZ   L1          ;      (команду TEST см. ниже)
    NEG  SI          ; SI = | quot | = 68F8h = 26872d
L1: TEST DI, 8000h   ; DI & 8000h — проверка знакового разряда остатка
    JZ   L2
    NEG  DI          ; DI = | rem | = 5A61h = 23137d
L2:  ∴
```

Конечно, в этой программе с приведенным способом вычисления модулей частного и остатка должны использоваться не непосредственные операнды, а переменные, которые могут дать частное и остаток заранее неизвестного знака.

3. *Логические команды*

Команды AND *dst*, *src*, OR *dst*, *src*, XOR *dst*, *src* и NOT *dst*. Эти команды выполняют над операндами логические двухместные операции И (*AND* — конъюнкция &), ИЛИ (*OR* — дизъюнкция ∨), исключающее ИЛИ (XOR — *exclusive OR*; сумма по модулю два ⊕) и одноместную операцию НЕ (*NOT* — *complement Boolean* — булево дополнение, отрицание, инверсия) соответственно:

$$dst \leftarrow dst\ \&\ src,\ dst \leftarrow dst \vee src,\ dst \leftarrow dst \oplus src,\ dst \leftarrow \overline{dst},$$

Все логические операции выполняются поразрядно над байтами и словами. Например,

$$A \,\&\, B = (a_k \ldots a_1 a_0) \,\&\, (b_k \ldots b_1 b_0) = (a_k \,\&\, b_k) \ldots (a_1 \,\&\, b_1)(a_0 \,\&\, b_0), \quad \overline{A} = \overline{a}_k \ldots \overline{a}_1 \overline{a}_0,$$

где $k = 7$ или 15 для 8-разрядных и 16-разрядных операндов *dst* и *src*; a_i и b_i — двоичные разряды операндов; $a_i \,\&\, b_i$ и $\overline{a_i}$ — двоичные (0 или 1) разряды результата.

Выполняются команды AND, OR и XOR за 3 такта (*reg* → *reg*), 9 + *ea* тактов (*M* → *reg*), 16 + *ea* тактов (*reg* → *M*), 17 + *ea* тактов (*imm* → *M*), 4 такта (*imm* → *reg*; *imm* → A). Команда NOT выполняется за 3 такта (*reg*), 16 + *ea* тактов (*M*). Стрелка "→" указывает операнд-получатель результата операции.

Задача 25. В регистре BH сбросить в 0 разряды 4 и 1, установить в 1 разряд 5 и проинвертировать разряды 7 и 2. *Решение*:

```
AND  BH, 0EDh  ; BH ← BH & EDh, EDh = 1110 1101  (b4 & 0 = 0, b1 & 0 = 0)
OR   BH, 20h   ; BH ← BH ∨ 20h,  20h = 0010 0000  (b5 ∨ 1 = 1)
XOR  BH, 84h   ; BH ← BH ⊕ 84h,  84h = 1000 0100  (b7 ⊕ 1 = b̄7, b2 ⊕ 1 = b̄2)
```

Команда TEST *src*$_1$, *src*$_2$. Эта команда производит над операндами src_1 и src_2 операцию поразрядной конъюнкции:

$$src_1 \,\&\, src_2.$$

Команда TEST подобна команде AND, но результат $src_1 \,\&\, src_2$ не запоминается (операнды src_1 и src_2 не изменяются). Результат выполнения команды TEST отражается только путем модификации флагов, используемых в командах условных переходов.

Выполняется команда TEST за 3 такта (*reg* & *reg*), 9 + *ea* тактов (*M* & *reg*), 11 + *ea* тактов (*M* & *imm*), 4 такта (A & *imm*) и 5 тактов (*reg* & *imm*).

Задача 26. Запретить прерывания, если разряд 5 в регистре CL равен 1. *Решение*:

```
        TEST CL, 20h    ; CL & 20h, 20h = 0010 0000
        JZ   CL5_1
        CLI             ; Флаг прерываний IF ← 0
CL5_1:  ∴
```

Команды сдвигов SHL/SAL *dst*, *cnt*, SHR *dst*, *cnt* и SAR *dst*, *cnt*. Эти команды производят сдвиг разрядов операнда *dst* влево (команда SHL/SAL; SHL и SAL — синонимы) или вправо (команды SHR и SAR) на один разряд при *cnt* = 1 или на число разрядов, предварительно заданное в регистре CL, т. е. *cnt* = 1 или *cnt* = CL.

Операции сдвига разрядов операнда *dst* наглядно поясняет рис. 4.24 ($k = 7$ или 15 — для 8-разрядных и 16-разрядных операндов *dst* соответственно): SHL — логический сдвиг влево, SAL — арифметический сдвиг влево, SHR — логический сдвиг вправо, SAR — арифметический сдвиг вправо.

Пример 16:

```
SHL  DL, 1      ; cnt = 1 — сдвиг содержимого регистра DL на один разряд влево
SHR  SI, 1      ; cnt = 1 — сдвиг содержимого регистра SI на один разряд вправо
SAL  BH, CL     ; cnt = CL — сдвиг содержимого регистра BH на CL разрядов влево
SAR  DI, CL     ; cnt = CL — сдвиг содержимого регистра DI на CL разрядов вправо
```

Можно задавать значения CL = 0 … 255*d* (0 — сдвиг не производится, 1 — сдвиг на один разряд, как и при *cnt* = 1). Например, команды SHL DX, 1 и SHL DX, CL производят сдвиг влево содержимого регистра DX на один разряд, если CL = 1. Естественно, что в этом случае предпочтительнее использовать команду SHL DX, 1. В МП 80286/80386 число сдвигов определяется пятью младшими разрядами регистра CL, т. е. остатком *rem*(CL / 32) = 0 … 31.

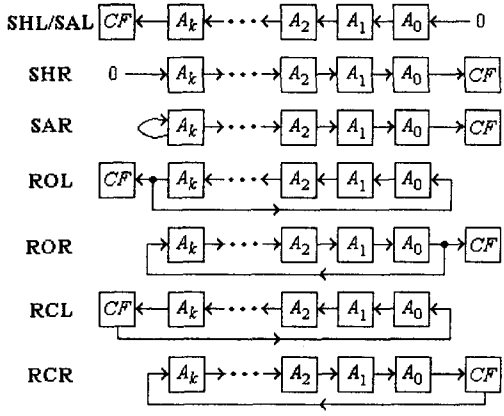


Рис. 4.24. Команды сдвигов

Команды арифметических сдвигов (SAL — при каждом сдвиге в младший разряд A_0 поступает 0 и SAR — старший разряд A_k при сдвигах не изменяется) позволяют производить быстрое умножение и деление целых чисел без знака и со знаком на 2^m.

Все команды сдвигов воздействуют на все шесть флагов условий (арифметические флаги). Флаг переноса *CF* участвует во всех операциях сдвига операнда *dst* — от типа команды зависит лишь место подключения флага *CF* к операнду. Этот флаг наиболее важен как результат выполнения команд сдвигов. Флаги нуля *ZF*, паритета *PF* и знака *SF* сбрасываются и устанавливаются по обычным правилам, а состояние флага *AF* не определено.

Флаг переполнения *OF* содержит полезную информацию только в том случае, если число сдвигов *cnt* = 1. В этом случае состояние флага *OF* устанавливается в 1, если при сдвиге изменяется значение старшего разряда A_k при выполнении команд SHL/SAL и SHR (при выполнении команды SAR значение разряда A_k никогда не изменяется — петля на рис. 4.24). Если число сдвигов *cnt* ≠ 1, то значение флага *OF* не определено.

Выполняются команды сдвига SHL/SAL, SHR и SAR за 2 такта (регистр, *cnt* = 1), 8 + 4/*bit* тактов (регистр, *cnt* = CL), 15 + *ea* тактов (память, *cnt* = 1), 20 + *ea* + 4/*bit* тактов (память, *cnt* = CL), где 4/*bit* означает 4 × CL тактов.

Пример 17:

```
MOV   DL, 63h     ; DL ← 0 1 1 0 0 0 1 1 = 63h
SHL   DL, 1       ; DL ← 1 1 0 0 0 1 1 0 = C6h,   OF = 1, CF = 0
SAR   DL, 1       ; DL ← 1 1 1 0 0 0 1 1 = E3h,   OF = 0, CF = 0
SHR   DL, 1       ; DL ← 0 1 1 1 0 0 0 1 = 71h,   OF = 1, CF = 1
MOV   CL, 2
SHL   DL, CL      ; DL ← 1 1 0 0 0 1 0 0 = C4h,   OF = 1, CF = 1
SAR   DL, CL      ; DL ← 1 1 1 1 0 0 0 1 = F1h,   OF = 1, CF = 0
SAL   DL, CL      ; DL ← 1 1 0 0 0 1 0 0 = C4h,   OF = 1, CF = 1
MOV   CL, 4
MOV   SI, 54B3h   ; SI ← 0101 0100 1011 0011 = 54B3h
SAL   SI, CL      ; SI ← 0100 1011 0011 0000 = 4B30h, CF = 1
```

Задача 27. Умножить число без знака X = 107*d* на 2^6 и разделить число без знака Y = B9E7*h* на 2^5 с потерей остатка *rem*. *Ответ* для проверки: $P = 107d \times 2^6 = 6848d$ и *quot* = B9E57*h*/2^5 = 5CF*h*. *Решение*:

```
MOV   CL, 6
MOV   DX, 107d    ; DX ← 107d = 006Bh
SAL   DX, CL      ; DX ← 107d × 2^6 = 1AC0h = 6848d = P
  ∴
MOV   CL, 5
MOV   DI, 0B9E7h  ; DI ← 0B9E7h = 47591d
SHR   DI, CL      ; DI ← 0B9E7h / 2^5 = 5CFh = 1487d = quot
```

Задача 28. Умножить число со знаком в регистре AL на 2^6 и разделить число со знаком в регистре SI на 2^5 с потерей остатка *rem* (числа представлены в дополнительном коде). Вычислить модули произведения и частного *quot*. *Решение*:

Метка	Команда	Комментарий	Пример 1	Пример 2
		; *Численные примеры*: $[\mathrm{AL}]_д$ =	$[+99d]_д = 63h$	$[-77d]_д = \mathrm{B}3h$
	MOV CL, 6	; 6 — умножение на $2^6 = 64$	⇓	⇓
	CBW	; AX ← AL с расшир. знака	AX = 0063*h*	AX = FFB3*h*
	SAL AX, CL	; AX ← $[\mathrm{AL} \times 2^6]_д = [P]_д$	AX = 18C0*h*	AX = ECC0*h*
	MOV DI, AX	; DI ← AX	DI = 18C0*h*	DI = ECC0*h*
	TEST DI, 8000*h*	; DI & 8000*h* — проверка	$\mathrm{DI}_{15} = 0$	$\mathrm{DI}_{15} = 1$
	JZ L1	знака	Переход на L1	Нет перехода на L1
	NEG DI	;		DI = 1340*h*
L1:	∴	; DI = $\lvert P \rvert$	$\lvert P \rvert = 18\mathrm{C}0h = 6336d$	$\lvert P \rvert = 1340h = 4928d$
	∴	; *Численные примеры*: $[\mathrm{SI}]_д$ =	$[+31582d]_д = 7\mathrm{B}5\mathrm{E}h$	$[-18969d]_д = \mathrm{B}5\mathrm{E}7h$
	MOV CL, 5		⇓	⇓
	SAR SI, CL	; SI ← $[\mathrm{SI} / 2^5]_д = [quot]_д$	SI = 03DA*h*	SI = FDAF*h*
	MOV DI, SI		DI = 03DA*h*	DI = FDAF*h*
	TEST DI, 8000*h*	; DI & 8000*h* — проверка	$\mathrm{DI}_{15} = 0$	$\mathrm{DI}_{15} = 1$
	JZ L2	знака	Переход на L2	Нет перехода на L2
	NEG DI	;		DI = 251*h*
L2:	∴	; DI = $\lvert quot \rvert$	$\lvert quot \rvert = 3\mathrm{DA}h = 986d$	$\lvert quot \rvert = 251h = 593d$
		; *Потерян остаток*:	*rem* = +30*d*	*rem* = +7

Команды циклических сдвигов ROL *dst*, *cnt*, ROR *dst*, *cnt*, RCL *dst*, *cnt* и RCR *dst*, *cnt*. Эти команды производят сдвиг разрядов операнда *dst* влево (ROL и RCL) или вправо (ROR и RCR) на один разряд при *cnt* = 1 или на число разрядов, предварительно заданное в регистре CL, т. е. *cnt* = 1 или *cnt* = CL.

В МП 80286/80386 число сдвигов определяется пятью младшими разрядами регистра CL, т. е. остатком *rem*(CL/32) = 0 … 31.

Операции сдвига разрядов операнда *dst* наглядно поясняет рис. 4.24 (*k* = 7 или 15 — *dst*8 и *dst*16): ROL — циклический сдвиг влево, ROR — циклический сдвиг вправо, RCL — циклический сдвиг влево через флаг переноса *CF*, RCR— циклический сдвиг вправо через флаг переноса *CF*.

Команды циклических сдвигов воздействуют только на флаги переноса *CF* и переполнения *OF*. Если число сдвигов *cnt* ≠ 1, то значение флага *OF* не определено. При задании значения *cnt* = 1 значение флага *OF* устанавливается в 1, если при сдвиге изменяется значение старшего разряда A_k.

Выполняются команды сдвига SHL/SAL, SHR и SAR за 2 такта (регистр, *cnt* = 1), 8 + 4/*bit* тактов (регистр, *cnt* = CL), 15 + *ea* тактов (память, *cnt* = 1), 20 + *ea* + 4/*bit* тактов (память, *cnt* = CL), где 4/*bit* означает 4 × CL тактов.

Пример 18:

```
MOV   DL, 63h    ; DL ← 0 1 1 0 | 0 0 1 1 = 63h
ROL   DL, 1      ; DL ← 1 1 0 0 | 0 1 1 0 = C6h,   OF = 1,  CF = 0
ROR   DL, 1      ; DL ← 0 1 1 0 | 0 0 1 1 = 63h,   OF = 1,  CF = 0
RCR   DL, 1      ; DL ← 0 0 1 1 | 0 0 0 1 = 31h,   OF = 0,  CF = 1
MOV   CL, 2
ROL   DL, CL     ; DL ← 1 1 0 0 | 0 1 0 0 = C4h,   OF = 0,  CF = 0
```

```
RCR   DL, CL     ; DL ← 0 0 1 1 | 0 0 0 1 = 31h,   OF = 0, CF = 0
RCL   DL, CL     ; DL ← 1 1 0 0 | 0 1 0 0 = C4h,   OF = 0, CF = 0
MOV   CL, 4
MOV   SI, 54B3h  ; SI ← 0101 0100 1011 0011 = 54B3h
ROL   SI, CL     ; SI ← 0100 1011 0011 0101 = 4B35h, CF = 1
```

Задача 29. Разделить число в регистре SI, представленное в дополнительном коде, на 2^5 без потери остатка *rem*. *Решение*:

```
        ;  Численные примеры: [SI]д =   [+31582d]д = 7B5Eh      [–18969d]д = B5E7h
    XOR  DL, DL ; DL ← 0                        ⇓                      ⇓
    MOV  CL, 5  ; CL ← 5
L1: SAR  SI, 1  ; SI ← [SI / 2]д (деление на 2)
    RCR  DL, 1  ; DL ← CF (сдвиг остатка)
    LOOP L1     ; SI ← [quot]д              [quot]д = 03DAh        [quot]д = FDAFh
    MOV  CL, 3  ; CL ← 3                    | quot | = 3DAh = 986d  | quot | = 251h = 593d
    SHR  DL, CL ; DL ← rem                  rem = 1Fh = +30d       rem = +7
```

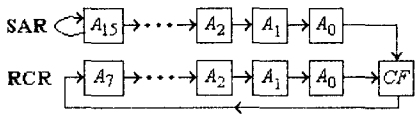


Рис. 4.25. Схема вычисления частного и остатка

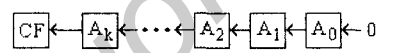
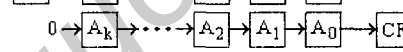
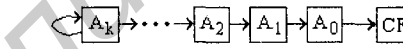
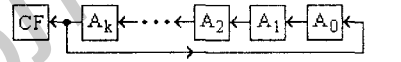
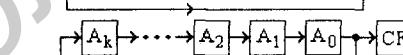
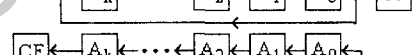
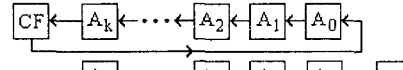
Вычисление частного *quot* и остатка *rem* выполняется по алгоритму, изображенному на рис. 4.25.

Табл. 4.13 содержит краткое описание команд МП 8086/8088, позволяющее быстро получить справку по операциям, выполняемым командами, и их воздействиям на флаги.

Таблица 4.13. **Краткое описание системы команд МП 8086/8088**

Команды передачи данных		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
MOV *dst*, *src*	*dst* ← *src*	–	–	–	–	–	–	–	–	–
PUSH *src*	SP ← SP – 2, *M*(SP) ← *src*	–	–	–	–	–	–	–	–	–
POP *dst*	*dst* ← *M*(SP), SP ← SP + 2	–	–	–	–	–	–	–	–	–
XCHG *dst*, *src*	*dst* ↔ *src*	–	–	–	–	–	–	–	–	–
IN *dst*, *src*	AL ← *I/O*(*port*), AL ← *I/O*(DX), AX ← *I/O*(*port*), AX ← *I/O*(DX)	–	–	–	–	–	–	–	–	–
OUT *dst*, *src*	*I/O*(*port*) ← AL, *I/O*(DX) ← AL, *I/O*(*port*) ← AX, *I/O*(DX) ← AX.	–	–	–	–	–	–	–	–	–
XLAT	AL ← *M*(BX + AL)	–	–	–	–	–	–	–	–	–
LEA *dst*, *src*	*r*16 ← *addr*	–	–	–	–	–	–	–	–	–
LDS *dst*, *src*	*r*16 ← *M*(*addr*), DS ← *M*(*addr* + 2)	–	–	–	–	–	–	–	–	–
LES *dst*, *src*	*r*16 ← *M*(*addr*), ES ← *M*(*addr* + 2)	–	–	–	–	–	–	–	–	–
LAHF	AH ← *low* byte PSW	–	–	–	–	–	–	–	–	–
SAHF	*Low byte* PSW ← AH	–	–	–	–	*r*	*r*	*r*	*r*	*r*
PUSHF	SP ← SP – 2, *M*(SP) ← PSW	–	–	–	–	–	–	–	–	–
POPF	PSW ← *M*(SP), SP ← SP + 2	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*

Продолжение таблицы 4.13

Арифметические команды		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
ADD *dst*, *src*	*dst* ← *dst* + *src*	+	–	–	–	+	+	+	+	+
ADC *dst*, *src*	*dst* ← *dst* + *src* + CF	+	–	–	–	+	+	+	+	+
INC *dst*	*dst* ← *dst* + 1	+	–	–	–	+	+	+	+	–
AAA	*ASCII*-коррекция для сложения	×	–	–	–	×	×	+	×	+
DAA	Десятичная коррекция для сложения	×	–	–	–	+	+	+	+	+
SUB *dst*, *src*	*dst* ← *dst* – *src*	+	–	–	–	+	+	+	+	+
SBB *dst*, *src*	*dst* ← *dst* – *src* – CF	+	–	–	–	+	+	+	+	+
DEC *dst*	*dst* ← *dst* – 1	+	–	–	–	+	+	+	+	–
NEG *dst*	*dst* ← 0 – *dst*	+	–	–	–	+	+	+	+	+
CMP src_1, src_2	src_1 – src_2	+	–	–	–	+	+	+	+	+
AAS	*ASCII*-коррекция для вычитания	×	–	–	–	×	×	+	×	+
DAS	Десятичная коррекция для вычитания	×	–	–	–	+	+	+	+	+
MUL *src*	AX ← AL × *src*8, DX : AX ← AX × *src*16	+	–	–	–	×	×	×	×	+
IMUL *src*	AX ← AL × *src*8, DX : AX ← AX × *src*16	+	–	–	–	×	×	×	×	+
AAM	*ASCII*-коррекция для умножения	×	–	–	–	+	+	×	+	×
DIV *src*	AL ← *quot*(AX/*src*8), AH ← *rem*(AX/*src*8)	×	–	–	–	×	×	×	×	×
IDIV *src*	AX ← *quot*(DX:AX/*src*16), DX ← *rem*(DX:AX/*src*16)	×	–	–	–	×	×	×	×	×
AAD	*ASCII*-коррекция для деления	×	–	–	–	+	+	×	+	×
CBW	$AH_7 \ldots AH_1AH_0 \leftarrow AL_7 \ldots AL_7AL_7$	–	–	–	–	–	–	–	–	–
CWD	$DX_{15} \ldots DX_1DX_0 \leftarrow AX_{15} \ldots AX_{15}AX_{15}$	–	–	–	–	–	–	–	–	–
Логические команды		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
SHL/SAL *dst*, *cnt*		+	–	–	–	+	+	×	+	+
SHR *dst*, *cnt*		+	–	–	–	+	+	×	+	+
SAR *dst*, *cnt*		+	–	–	–	+	+	×	+	+
ROL *dst*, *cnt*		+	–	–	–	–	–	–	–	+
ROR *dst*, *cnt*		+	–	–	–	–	–	–	–	+
RCL *dst*, *cnt*		+	–	–	–	–	–	–	–	+
RCR *dst*, *cnt*		+	–	–	–	–	–	–	–	+
AND *dst*, *src*	*dst* ← *dst* & *src* — логическое И	0	–	–	–	+	+	×	+	0
TEST src_1, src_2	src_1 & src_2 — логическое И	0	–	–	–	+	+	×	+	0
NOT *dst*	*dst* ← $\overline{dst}$ — логическое НЕ	–	–	–	–	–	–	–	–	–
OR *dst*, *src*	*dst* ← *dst* ∨ *src* — логическое ИЛИ	0	–	–	–	+	+	×	+	0
XOR *dst*, *src*	*dst* ← *dst* ⊕ *src* — исключающее ИЛИ	0	–	–	–	+	+	×	+	0

Продолжение табл. 4.13

Команды манипуляции цепочками		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
REP/REPZ/REPE	Повторять пока не ноль/не равно (инд. $z = 1$)	–	–	–	–	–	–	–	–	–
REPNZ/REPNE	Повторять пока равно (индикатор $z = 0$)	–	–	–	–	–	–	–	–	–
MOVS/MOVSB/ MOVSW	Пересылка строки байт/слов из памяти в память	–	–	–	–	–	–	–	–	–
CMPS/CMPSB/ CMPSW	Сравнение двух строк байт/слов в памяти	+	–	–	–	+	+	+	+	+
SCAS/SCASB/ SCASW	Просмотр строки байт/слов: AL/AX – (*M*)	+	–	–	–	+	+	+	+	+
LODS/LODSB/ LODSW	Загрузка в AL/AX строки байт/слов из памяти	–	–	–	–	–	–	–	–	–
STOS/STOSB/ STOSW	Запись в память строки байт/слов из AL/AX	–	–	–	–	–	–	–	–	–
Команды передачи управления		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
CALL *target*	Вызов процедуры (подпрограммы) – *near*, *far*	–	–	–	–	–	–	–	–	–
RET	Возврат из процедуры – *near*, *far*	–	–	–	–	–	–	–	–	–
JMP *target*	Безусловный переход – *short*, *near*, *far*	–	–	–	–	–	–	–	–	–
JE/JZ *sL*	Переход, если флаг ZF = 1	–	–	–	–	–	–	–	–	–
JNE/JNZ *sL*	Переход, если флаг ZF = 0	–	–	–	–	–	–	–	–	–
JS *sL*	Переход, если флаг SF = 1	–	–	–	–	–	–	–	–	–
JNS *sL*	Переход, если флаг SF = 0	–	–	–	–	–	–	–	–	–
JO *sL*	Переход, если флаг OF = 1	–	–	–	–	–	–	–	–	–
JNO *sL*	Переход, если флаг OF = 0	–	–	–	–	–	–	–	–	–
JP/JPE *sL*	Переход, если флаг PF = 1	–	–	–	–	–	–	–	–	–
JNP/JPO *sL*	Переход, если флаг PF = 0	–	–	–	–	–	–	–	–	–
JB/JNAE/JC *sL* *	Переход, если флаг CF = 1	–	–	–	–	–	–	–	–	–
JNB/JAE/JNC *sL* *	Переход, если флаг CF = 0	–	–	–	–	–	–	–	–	–
JBE/JNA *sL* *	Переход, если флаги CF ∨ ZF = 1	–	–	–	–	–	–	–	–	–
JNBE/JA *sL* *	Переход, если флаги CF ∨ ZF = 0	–	–	–	–	–	–	–	–	–
JL/JNGE *sL* **	Переход, если флаг SF ⊕ OF = 1	–	–	–	–	–	–	–	–	–
JNL/JGE *sL* **	Переход, если флаги SF ⊕ OF = 0 (SF = OF)	–	–	–	–	–	–	–	–	–
JLE/JNG *sL* **	Переход, если флаги SF ⊕ OF ∨ ZF = 1	–	–	–	–	–	–	–	–	–
JNLE/JG *sL* **	Переход, если флаги SF ⊕ OF ∨ ZF = 0	–	–	–	–	–	–	–	–	–

* — при выполнении операций над числами без знака, ** — при выполнении операций над числами со знаком;
A — *above* (выше), B — *below* (ниже), C — *carry* (перенос), E — *equally* (равно), G — *greater* (больше),
L — *less* (меньше), O — *overflow* — переполнение, P — *parity* — паритет (равенство),
PE — *even parity* (четный паритет), PO — *odd parity* (нечетный паритет), S — *sign* (знак), Z — *zero* (нуль),
sL — *short Label* (короткий переход по метке).

LOOP *sL*	Зациклить CX раз	–	–	–	–	–	–	–	–	–
LOOPZ/LOOPE *sL*	Зациклить, пока нуль/равно	–	–	–	–	–	–	–	–	–
LOOPNZ/LOOPNE *sL*	Зациклить, пока не нуль/не равно	–	–	–	–	–	–	–	–	–
JCXZ *sL*	Перейти, если CX = 0	–	–	–	–	–	–	–	–	–
INT *type*	Прерывание (*type* = 00 ÷ FF*h*)	–	–	0	0	–	–	–	–	–
INT 3	Прерывание (*type* = 03*h*)	–	–	0	0	–	–	–	–	–
INTO	Прерывание при переполнении	–	–	0	0	–	–	–	–	–
IRET	Возвратиться из прерывания	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*

Продолжение табл. 4.13

Команды управления процессором		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
CLC	CF ← 0	–	–	–	–	–	–	–	–	0
CMC	CF ← $\overline{CF}$	–	–	–	–	–	–	–	–	+
STC	CF ← 1	–	–		–	–	–	–	–	1
CLD	DF ← 0	–	0	–	–	–	–	–	–	–
STD	DF ← 1	–	1	–	–	–	–	–	–	–
CLI	IF ← 0	–	–	0	–	–	–	–	–	–
STI	IF ← 1	–	–	1	–	–	–	–	–	–
NOP	Пустая операция	–	–	–	–	–	–	–	–	–
HLT	Останов МП	–	–	–	–	–	–	–	–	–
WAIT	Ожидать значения сигнала $\overline{TEST}$ = 0	–	–	–	–	–	–	–	–	–
ESC	Переключение на сопроцессор	–	–	–	–	–	–	–	–	–
LOCK	Блокировка шины	–	–	–	–	–	–	–	–	–

Примечание: "+" — значение флага определяется результатом операции, "–" — значение флага не изменяется, "0" — значение флага сбрасывается в 0, "1" — значение флага устанавливается в 1, "*r*" — возвращение значения флага из стека, "×" — неопределенное значение флага.

Начиная с этого момента, неподготовленному читателю следует бегло ознакомиться с материалом, изложенным в § 4.4 (по крайней мере, нужно знать назначение директив определения данных и понимать применение имен переменных в ассемблерных программах).

4. *Команды манипуляции цепочками*

В пяти однобайтовых командах манипуляции цепочками (табл. 4.13) используется неявная адресация двух операндов, находящихся в памяти или в памяти и аккумуляторе. Для повторения выполнения этих команд используется префикс повторения REP — число повторений предварительно задается в регистре *CX* (например, REP MOVSB — пересылка цепочки длиной в CX байт из одной области памяти в другую). При каждом выполнении цепочечной команды автоматически производится декремент содержимого регистра *CX*. При получении значения CX = 0 повторение выполнения команды прекращается.

Операнды могут быть байтами (B) и словами (W). На языке ассемблера для цепочечных команд можно использовать любой из трех форматов:

MOVSB, CMPSB, SCASB, LODSB, STOSB — операндами являются байты;
MOVSW, CMPSW, SCASW, LODSW, STOSW — операндами являются слова;
MOVS *dst*, *src*, CMPS *src*, *dst*, SCAS *dst*, LODS *src*, STOS *dst*,

где *dst* и *src* — фиктивные операнды, указывающие ассемблеру тип данных, заблаговременно определенный директивами ассемблера (см. § 4.4). Команды с одним операндом *dst* или *src* неявно предполагают, что вторым операндом является аккумулятор AL/AX. Операндам *dst* и *src* можно присваивать любые имена, определенные директивами ассемблера как байт или слово. Использование фиктивных операндов облегчает также чтение программы, так как с именами операндов всегда связаны адреса их расположения в памяти. Например, команда REP MOVS *string*2, *string*1 указывает, что цепочка байт/слов с начальным адресом *string*1 пересылается по адресу (начальному) *string*2, конечно, если эти имена были определены соответствующим образом. Директивами определения данных задается и тип переменных *string*2 и

*string*1 — байты или слова. Команды, содержащие операнды *dst* и *src*, ассемблером транслируются в один из первых двух форматов в зависимости от типа операнда (байт или слово).

Каждая из цепочечных команд без префикса повторения REP выполняет элементарную операцию одной или двух пересылок операндов (байт/слов), расположенных в памяти. Поэтому цепочечные команды называются *примитивами* (эти же операции пересылок могут быть легко реализованы и другими командами).

Таблица 4.14. **Адресация цепочек**

Операнд	Адрес	Автоинкремент (флаг DF = 0)	Автодекремент (флаг DF = 1)
*src*8	DS : SI	SI = SI + 1	SI = SI – 1
*src*16	DS : SI	SI = SI + 2	SI = SI – 2
*dst*8	ES : SI	DI = DI + 1	DI = DI – 1
*dst*16	ES : SI	DI = DI + 2	DI = DI – 2

Использование примитивов для выполнения операций над длинными цепочками позволяет существенно уменьшить затрачиваемое на это время — при выполнении примитива автоматически производится адресация следующего элемента в цепочке. В табл. 4.14 приведен способ адресации операндов, реализованный в примитивах и не зависящий от используемого формата команды. Цепочки операндов могут находиться как в одном сегменте (ES = DS), так и в разных не перекрывающихся сегментах (ES ≠ DS). В цепочечных командах для операнда-источника *src* можно использовать префикс замены сегментного регистра, а в качестве сегментного регистра операнда-получателя *dst* всегда используется регистр *ES*.

Префиксы повторения REP/REPZ/REPE и REPNZ/REPNE. С цепочечными командами MOVS, LODS или STOS, не изменяющими флаги условий, достаточно использовать префикс повторения REP, задающий число повторений, указанное в регистре *CX*. В цепочечных командах CMPS и SCAS, производящих сравнение символов в двух цепочках и сравнение символов одной цепочки с символом в аккумуляторе, для окончания повторений добавочно используется состояние флага *ZF*. Поэтому имеется два альтернативных префикса повторений:

REP/REPZ/REPE (синонимы; повторять пока ноль/повторять пока равно) — повторение выполнения цепочечных команд пока CX ≠ 0 и флаг ZF = 1;

REPNZ/REPNE (синонимы; повторять пока не ноль/повторять пока не равно) — повторение выполнения цепочечных команд пока CX ≠ 0 и флаг ZF = 0.

Исходное состояние флага *ZF* не влияет на число повторений — проверка состояния флага *ZF* производится только *после* очередного выполнения команды CMPS или SCAS. Проверка значения CX на нуль проводится *перед* очередным выполнением любой цепочечной команды, поэтому, если сразу задать значение CX = 0, то команда не будет выполнена ни разу. В принципе, для префикса повторения можно использовать только две альтернативных мнемоники REP и REPNZ.

Если выполнение цепочечной команды будет прервано между повторениями, то после возврата из прерывания ее выполнение будет корректно продолжено.

Выполняются префиксы повторений за 2 такта.

Команды MOVS *dst*, *src*, MOVSB и MOVSW. Эти команды выполняют пересылку цепочек (строк) байт или слов:

$$M(\text{ES:DI}) \leftarrow M(\text{DS:SI}),\ \text{SI} \leftarrow \text{SI} + m,\ \text{DI} \leftarrow \text{DI} + m, \text{ если DF} = 0;$$
$$M(\text{ES:DI}) \leftarrow M(\text{DS:SI}),\ \text{SI} \leftarrow \text{SI} - m,\ \text{DI} \leftarrow \text{DI} - m, \text{ если DF} = 1,$$

где *M* — *Memory* (память), $m = 1$ для байт и $m = 2$ для слов.

Выполняются команды за 18 тактов (без повторения) и 9 + 17/*rep* тактов (с повторением), где 17/*rep* означает 17 × CX тактов.

Задача 30. Переслать таблицу векторов прерывания персонального компьютера, находящуюся по адресам 0000 : 0000 ... 0000 : 03FF*h*, в область памяти, определяемую программой пользователя. *Решение*:

```
; Пересылка таблицы векторов прерываний для команд INT type
s_seg   segment  para STACK     ; Сегмент стека
        db       32 dup ('St')  ; Стек — 32 слова
s_seg   ends                    ; Конец сегмента стека
d_seg   segment                 ; Сегмент данных
buff    db       256 dup('Buff') ; Buffer address = 0000 ÷ 03FFh (512 слов)
d_seg   ends                    ; Конец сегмента данных
c_seg   segment                 ; Сегмент кода
        assume   CS: c_seg, DS: d_seg, SS: s_seg, ES: d_seg
main    proc     far            ; Начало процедуры main
        push     ds             ; Запись в стек начального адреса PSP (Program Segment Prefics)
        sub      ax, ax         ;      и 0000h (для возврата в DOS)
        push     ax
        mov      ax, d_seg
        mov      ds, ax         ; Инициализация сегментного регистра DS
        mov      es, ax         ; ES = DS
; Передача в буфер buff таблицы векторов прерываний команд INT type
        PUSH     DS             ; SP ← SP – 2, M(SP) ← DS (сохранение в стеке DS)
        XOR      AX, AX         ; AX ← 0000h
        MOV      DS, AX         ; DS ← 0000h
        MOV      SI, AX         ; SI ← 0000h, DS:SI = 0000:0000h — начальный адрес таблицы
        LEA      DI, buff       ; DS ← адрес буфера buff, ES:DI — начальный адрес буфера
        MOV      CX, 200h
        CLD                     ; DF ← 0 (автоинкремент)
REP     MOVSW                   ; M(ES:buff) ← M(DS:SI), SI = SI + 2, DI = DI + 2
        POP      DS             ; DS ← M(SP), SP ← SP + 2 (извлечение DS из стека)
        RET
main    endp                    ; Конец процедуры main
c_seg   ends                    ; Конец сегмента кода
        end      main           ; Конец программы
```

В этой программе использована команда MOVSW, явно указывающая на пересылку слов, поэтому не имеет значения, что буфер *buff* определен как байтовый. Программа написана с учетом требований языка ассемблера — программа транслируется в загрузочный модуль (*exe*-файл), что позволяет просмотреть векторы прерываний с помощью отладчика программ (см. § 4.4). Так, для системного прерывания по ошибке деления INT 0 вектор (адрес) вызова подпрограммы обработки прерывания CS : IP = 2F2A : 0258 в *PC Pentium II* (естественно, на разных компьютерах будут получаться разные вектора, если их операционные системы не совпадают). Эта подпрограмма выводит на экран дисплея сообщение: *Your program caused a divide overflow error. If the problem persists, contact your program vendor* (*Ваша программа вызвала ошибку переполнения при делении. Если проблема продолжает существовать, войдите в контакт с вашим продавцом программы*).

Команды CMPS *src*, *dst*, CMPSB и CMPSW. Эти команды выполняют сравнение цепочек (строк) байт или слов:

$$M(\text{DS:SI}) - M(\text{ES:DI}), \quad \text{SI} \leftarrow \text{SI} + m, \text{DI} \leftarrow \text{DI} + m, \text{если DF} = 0;$$
$$M(\text{DS:SI}) - M(\text{ES:DI}), \quad \text{SI} \leftarrow \text{SI} - m, \text{DI} \leftarrow \text{DI} - m, \text{если DF} = 1,$$

где $m = 1$ для байт и $m = 2$ для слов (операнды не изменяются, а устанавливаются лишь флаги в соответствии с результатом сравнения).

Выполняются команды за 22 такта (без повторения) и 9 + 22/*rep* тактов (с повторением), где 22/*rep* означает 22 × CX тактов.

Задача 31. Найти адреса первых от начала не совпадающих символов в двух цепочках *str*1 и *str*2 длиною 32 байта или слова. *Решение*:

```
        CLD             ; DF ← 0 — сравнение цепочек от начала к концу
        LEA   SI, str1  ; SI ← начальный адрес строки (цепочки) str1
        LEA   DI, str2  ; DI ← начальный адрес строки str2
        MOV   CX, 32    ; CX ← 32d — число символов в строке
REPE    CMPS  str1, str2; M(DS:SI) – M(ES:DI), SI ← SI + m, DI ← DI + m , CX ← CX – 1
        JE    L1        ; Переход на метку L1, если все символы совпали
        DEC   SI        ; SI ← SI – 1 — адрес несовпадающего символа в строке str1 (для байт)
        DEC   DI        ; DI ← DI – 1 — адрес несовпадающего символа в строке str2 (для байт)
         ∴              ; Для слов декремент регистров следует сделать два раза
L1:      ∴
```

Задача 32. Найти адреса первых от конца совпадающих символов в двух цепочках *str*1 и *str*2 длиною 32 байта или слова. *Решение*:

```
        STD             ; DF ← 1 — сравнение цепочек от конца к началу
        LEA   SI, str1 + 31 ; SI ← конечный адрес строки (цепочки) str1
        LEA   DI, str2 + 31 ; DI ← конечный адрес строки str2
        MOV   CX, 32    ; CX ← 32d — число символов в строке
REPNE   CMPS  str1, str2; M(DS:SI) – M(ES:DI), SI ← SI + m, DI ← DI + m , CX ← CX – 1
        JNE   L1        ; Переход на метку L1, если все символы не совпали
        INC   SI        ; SI ← SI + 1 — адрес совпадающего символа в цепочке str1 (для байт)
        INC   DI        ; DI ← DI + 1 — адрес совпадающего символа в цепочке str2 (для байт)
         ∴              ; Для слов инкремент регистров следует сделать два раза
L1:      ∴
```

Команды SCAS *dst*, SCASB и SCASW. Данные команды выполняют сканирование (просмотр) цепочки (строки) байт или слов:

$$\text{AL/AX} - M(\text{ES:DI}), \quad \text{DI} \leftarrow \text{DI} + m \ (m = 1 \text{ для байт}, m = 2 \text{ для слов}), \text{ если DF} = 0;$$
$$\text{AL/AX} - M(\text{ES:DI}), \quad \text{DI} \leftarrow \text{DI} - m \ (m = 1 \text{ для байт}, m = 2 \text{ для слов}), \text{ если DF} = 1,$$

т. е. производится сравнение содержимого аккумулятора AL или AX с байтом или словом в памяти по адресу ES:DI (сами операнды не изменяются, а устанавливаются лишь флаги в соответствии с результатом сравнения). Эти команды используются в тех случаях, когда в цепочке необходимо найти заданный символ.

Выполняются команды за 15 тактов (без повторения) и 9 + 15/*rep* тактов (с повторением), где 15/*rep* означает 15 × CX тактов.

Задача 33. Найти адрес первого от начала символа '&' в цепочке *string* длиною 32 байта. *Решение*:

```
        CLD                 ; DF ← 0 — сканирование цепочки от начала к концу
        MOV   AL, '&'       ; AL ← 26h — ASCII-код символа &
        LEA   DI, string    ; DI ← начальный адрес строки string
        MOV   CX, 32        ; CX ← 32d — число символов в строке
REPNE   SCAS  string        ; AL – M(ES:DI), DI ← DI + 1, CX ← CX – 1
        JNE   L1            ; Переход на метку L1, если символ не найден
        DEC   DI            ; DI ← DI – 1 — адрес символа '&' в цепочке string
          ∴
L1:       ∴
```

Команды LODS *src*, LODSB и LODSW. Эти команды выполняют пересылку цепочек (строк) байт или слов в аккумулятор:

$$AL/AX \leftarrow M(DS:SI),\ SI = SI + m, \text{ если } DF = 0;$$
$$AL/AX \leftarrow M(DS:SI),\ SI = SI - m, \text{ если } DF = 1,$$

где $m = 1$ для байт и $m = 2$ для слов, т. е. производится пересылка в аккумулятор AL или AX байта или слова из памяти с адресом DS:SI. Хотя и разрешается использовать эту команду с префиксом повторения, этого почти никогда не делается из-за отсутствия полезного результата (в регистр AL или AX при каждом повторении команды LODS загружается новое значение байта или слова с потерей предыдущего значения).

Выполняются команды за 12 тактов (без повторения) и 9 + 13/*rep* тактов (с повторением), где 13/*rep* означает 13 × CX тактов.

Команды STOS *dst*, STOSB и STOSW. Эти команды выполняют пересылку цепочек (строк) байт или слов:

$$M(ES:DI) \leftarrow AL/AX,\ DI = DI + m, \text{ если } DF = 0;$$
$$M(ES:DI) \leftarrow AL/AX,\ DI = DI - m, \text{ если } DF = 1,$$

где $m = 1$ для байт и $m = 2$ для слов, т. е. производится пересылка из аккумулятора AL или AX байта или слова в память по адресу DS:SI. Эти команды полезны в тех случаях, когда некоторую область памяти необходимо заполнить заданным символом (инициализация блока памяти).

Выполняются команды за 11 тактов (без повторения) и 9 + 10/*rep* тактов (с повторением), где 10/*rep* означает 10 × CX тактов.

Задача 34. Заполнить блок памяти *string* длиною 32 слова символами '&$'. *Решение*:

```
        CLD                 ; DF ← 0 — сканирование цепочки от начала к концу
        MOV   AX, '&$'      ; AX ← 2624h, где 26h и 24h — ASCII-коды символов & и $
        LEA   DI, string    ; DI ← начальный адрес строки string
        MOV   CX, 32        ; CX ← 32d — число слов в строке
REP     STOS  string        ; M(ES:DI) ← AX, DI ← DI + 2, CX ← CX – 1
```

Все примитивы производят над элементами цепочек только "пассивные" действия — не преобразуют сами элементы, что требуется, например, в редакторах текстов. Совместное использование примитивов LODS и STOS позволяет эффективно производить и преобразование элементов цепочек. Префиксы повторения при этом использовать нельзя — для организации цикла наиболее подходит команда LOOP.

Задача 35. Написать программу изменения в цепочке *string* длиною 32 байта прописных латинских букв на строчные. *Решение*:

$41h = 0100\ 0001 \ldots 5Ah = 0101\ 1010$ — *ASCII*-коды прописных латинских букв от *A* до *Z*,

$61h = 0110\ 0001 \ldots 7Ah = 0111\ 1010$ — *ASCII*-коды строчных латинских букв от *a* до *z*,

т. е. во всех элементах цепочки требуется установить в 1 разряд 5, что можно выполнить с помощью команды OR AL, 20*h* (20*h* = 0010 0000). На основании этого программа будет иметь вид:

```
      CLD             ; DF ← 0 — преобразование цепочки от начала к концу
      LEA   SI, string ; SI ← начальный адрес строки string
      MOV   DI, SI    ; DI ← SI
      MOV   CX, 32    ; CX ← 32d — число циклов
L1:   LODSB           ; AL ← M(DS:SI), SI ← SI + 1
      OR    AL, 20h   ; AL ← AL ∨ 20h (20h = 0010 0000)
      STOSB           ; M(ES:DI) ← AL ∨ 20h, DI ← DI + 1
      LOOP  L1        ; CX ← CX – 1 и переход, если CX ≠ 0
```

5. Команды передачи управления

Некоторые команды передачи управления требуют временного запоминания информации, и затем считывания ее в обратном порядке. Эта задача решается посредством реализации в МП-системах *стека*, представляющего собой память типа *LIFO* (*last-in, first-out* — последним вошел, первым вышел). Такая память называется иначе *стеком включения/извлечения*. Наиболее важное применение стека связано с процедурами (подпрограммами) и программными и аппаратными прерываниями, требующими автоматического сохранения адресов возврата. Кроме этого стек используется и для временного запоминания данных (команды PUSH — включение в стек, POP — извлечение из стека).

Для неявной адресации стека используется специальный регистр *SP* (*Stack Pointer*) — указатель стека. При включении (*push*) данных в стек производится автоматический декремент указателя стека *SP*, а при извлечении (*pop*) — инкремент. Адрес последнего включенного в стек элемента называется *вершиной стека — TOS* (*Top of Stack*).

Команды PUSH *src*16 и POP *dst*16, PUSHF и POPF. Эти команды относятся к группе команд передачи данных, но они тесно связаны по применению и принципу выполнения операций с некоторыми командами передачи управления — выполняют операции включения и извлечения данных из стека. Стек оперирует только словами и в командах нельзя использовать непосредственные операнды, но все остальные режимы адресации в командах PUSH и POP разрешены. Данные команды являются единственными, кроме команды MOV, в которых допускается задавать сегментный регистр как операнд. Однако в команде POP регистр CS указывать нельзя. Если операнды *src*16 и *dst*16 находятся в памяти, то команды PUSH и POP выполняют операцию пересылки типа память–память подобно примитиву MOVS.

При выполнении команд PUSH и POP производятся действия:

$$SP \leftarrow SP - 2, \text{ а затем } M(SP) \leftarrow src16,$$

$$dst16 \leftarrow M(SP), \text{ а затем } SP \leftarrow SP + 2$$

соответственно. Это же справедливо и для команд PUSHF и POPF, только *src*16 = *PSW* и *dst*16 = *PSW* (*Program Status Word* — регистр признаков). После одинакового числа включений

и извлечений вершина стека *TOS* оказывается в первоначальном положении. Следует соблюдать правило: нельзя извлекать из стека какие-либо данные, не включив их сначала в стек. И всегда необходимо помнить о "тайном" смысле терминов "включение" (в стек) и "извлечение" (из стека), подразумевающих автоматическое изменение содержимого указателя стека *SP* (включение и извлечение — это не то же самое, что запись и чтение данных из памяти).

Физический адрес стека определяется содержимым пар регистров SS:SP или SS:BP, причем SP служит неявным указателем стека для всех операций включения и извлечения, а SS — сегментным регистром стека. Регистр BP используется, главным образом, для произвольных обращений к стеку. Содержимое SS является самым младшим адресом (границей) области стека и называется *базой стека.* Первоначальное значение указателя стека SP, устанавливаемое при инициализации, является наибольшим смещением, которого может достигать стек. Это значение определяет максимальное число слов, которое может быть включено в стек (минимальное значение SP = 0). Местоположение стека задается операционной системой или пользовательской программой.

Временное сохранение операндов в стеке эффективнее использования для этих целей памяти в области сегмента данных: команды PUSH и POP короче других команд и не нужно задавать адрес памяти в явном виде (инкремент и декремент SP производится автоматически).

Команда PUSH выполняется за 11 тактов (*reg*), 10 тактов (*seg*) и 16 + *ea* тактов (*M*), команда POP — за 8 тактов (*reg* и *seg* SS, DS, ES) и 17 + *ea* тактов (*M*), команда PUSHF — за 10 тактов, команда POPF — за 8 тактов.

Команды CALL *target* и RET. Основным назначением стека является автоматическое запоминание адресов возврата из *процедур* (подпрограмм), вызываемых командами CALL *target* и INT *type*. Адрес возврата — это находящийся в указателе инструкции IP или в регистрах CS:IP адрес команды, следующей за командой вызова процедуры. Процедуры всегда должны заканчиваться командой возврата RET, автоматически извлекающей из стека адрес возврата и загружающей его в указатель инструкции IP или в регистры CS:IP.

Команда CALL предназначена для внутрисегментного и межсегментного вызова процедур *proc* (*procedure* — процедура, подпрограмма), которые могут находиться как внутри текущего сегмента кода (*near-proc* — внутрисегментный вызов), так и вне него (*far-proc* — межсегментный вызов):

SP ← SP – 2, *M*(SP) ← IP, IP ← *target* (*near*);

SP ← SP – 2, *M*(SP) ← CS, SP ← SP – 2, *M*(SP) ← IP, IP ← *target*1; CS ← *target*2 (*far*),

где *target* — одно или два слова адреса передачи управления. Режимы адресации (способы задания адреса *target*) команд CALL такие же, что и для команд JMP (см. § 4.2, рис. 4.23) — нет только короткого (*short*) перехода (вызова). Вызов может быть прямым или косвенным, внутрисегментным или межсегментным. Команд условных вызовов процедур нет.

Соответственно вызывающим процедуры командам CALL имеются команды RET, выполняющие внутрисегментные и межсегментные возвраты. Команды RET выполняют операции со стеком в обратном направлении:

IP ← *M*(SP), SP ← SP + 2 (*near-return*);

IP ← *M*(SP), SP ← SP + 2, CS ← *M*(SP), SP ← SP + 2 (*far-return*).

В команде RET может быть задан непосредственный операнд *im*16, если через стек в вызванную процедуру передавались необходимые ей данные (параметры). В этом случае после извлечения из стека адреса возврата выполняется операция SP + *im*16, что соответствует удале-

нию из стека отработавших параметров (модифицируется вершина стека для обхода этих параметров). Значение *im*16 должно быть равно числу слов данных, включенных в стек командами PUSH, перед вызовом процедуры, которой эти данные необходимы. Вызванная процедура получает доступ к параметрам, адресуя область памяти стека содержимым пары регистров SS:BP, не извлекая их из стека командами POP.

В виде процедур обычно оформляются большие часто повторяющиеся программные фрагменты — в памяти нужно хранить только одну копию процедуры, а вызываться она может много раз из разных мест программы. Процедуры обеспечивают основное средство разделения кода программы на модули, которые можно легко по отдельности разрабатывать, тестировать и документировать. Процедуры можно хранить в библиотеках и использовать многими программами.

Команда CALL выполняется за 19 тактов (внутрисегментный прямой вызов), 16 тактов (внутрисегментный косвенный вызов через регистр), 21 + *ea* тактов (внутрисегментный косвенный вызов через память), 28 тактов (межсегментный прямой вызов) и 37 + *ea* тактов (межсегментный косвенный вызов).

Команда RET выполняется за 8 тактов (внутрисегментный возврат), 12 тактов (внутрисегментный возврат с непосредственным операндом), 18 тактов (межсегментный возврат) и 17 тактов (межсегментный возврат с непосредственным операндом).

Пример 19. Передача управления при выполнении одноуровневых процедур:

```
          ∴
          CALL  proc_1  ;
          MOV   DL, 75  ;
          ∴
proc_1    PUSH  AX      ;   Начало процедуры proc_1
          PUSH  BX
          PUSH  CX      ;   Сохранение содержимого регистров AX, BX и CX
          ∴
          ∴
          POP   CX      ;   Восстановление содержимого регистров AX, BX и CX
          POP   BX
          POP   AX
          RET           ;   Конец процедуры proc_1
```

Содержимое тех регистров, которые использует процедура, будет модифицировано, а значит необходимо запомнить содержимое этих регистров до его изменения, а перед самым выходом из процедуры восстановить. Для этого используются команды PUSH и POP. Содержимое стека извлекается из него и загружается в регистры МП в порядке, обратном тому, в каком они включались в стек при выполнении программы.

Если процедура обращалась к стеку (выполняла команды PUSH и POP), то указатель стека SP должен быть возвращен на прежнее значение и действия со стеком не должны были разрушить адрес возврата. При работе с процедурами программист должен внимательно следить за правильным использованием стека, в противном случае возможны возвраты из процедур по бессмысленным адресам.

Процедуры могут иметь любое число уровней вложенности, от числа которых зависит, в частности, минимальный размер стека. Размер стека устанавливается программистом, исходя из потребностей разрабатываемой программы.

Пример 20. Передача управления при выполнении двухуровневых процедур:

В этом примере пояснена последовательность передач управления при выполнении двухуровневой процедуры (цифрами ①, ②, ③ и ④ помечен порядок передачи управления).

В *задаче* 36 приведен пример оформления на языке ассемблера процедуры CALL *c_intr near*. Конечно, вводить в программу такую процедуру (содержит всего две команды и используется только один раз) бессмысленно — число команд не сократилось, а увеличилось. *Задача* 36 предназначена в основном для демонстрации манипуляций с векторами прерываний команд INT *type*.

Задача 36. Написать программу вывода на экран дисплея собственного сообщения при возникновении ошибки деления с последующим восстановлением системного вектора прерывания INT 0. *Решение*:

```
s_seg    segment page STACK  ; Сегмент стека
         dw      8 dup (0E291h, 0AAA5h, 5320h, 2150h) ; Стек SP! — ASCII символы
s_seg    ends                ; Конец сегмента стека
d_seg    segment             ; Сегмент данных
Message  db      10, 10, ' Чтоб в арифметике упрочиться,', 13, 10
         db      ' О-о-о, Юный Неофит!', 13, 10
         db      ' Пиши не так, как хочется,', 13, 10
         db      ' А как Zero велит!', 10, 10, '(нажмите любую клавишу)', '$'
; Управляющие ASCII-коды:
;    13d = 0Dh — возврат каретки (сдвиг курсора в левую позицию экрана)
;    10d = 0Ah — перевод строки (сдвиг курсора вниз на одну позицию)
;    $ — задание конца строки, выводимой на дисплей
d_seg    ends                ; Конец сегмента данных
c_seg    segment             ; Сегмент кода
         assume  CS: c_seg, DS: d_seg, SS: s_seg, ES: d_seg
main     proc    far         ; Начало процедуры main
         push    ds          ; Запись в стек начального адреса PSP (Program Segment Prefics)
         sub     ax, ax      ;      и 0000h (для возврата в DOS)
         push    ax
         mov     ax, d_seg
         mov     ds, ax      ; Инициализация сегментного регистра DS
         mov     es, ax      ; ES = DS
```

```
; Получение системного вектора прерывания CSC:IPC для INT 0
        CALL   c_intr       ; AH ← 35h — номер функции, AL ← type = 00
; Исходное значение указателя стека SP = SP0
        PUSH   BX           ; SP ← SP – 2, M(SP0 – 2) ← IPC (сохранение в стеке CSC:IPC)
        PUSH   ES           ; SP ← SP – 2, M(SP0 – 4) ← CSC
; Замена системного вектора прерывания пользовательским CSU:IPU
        PUSH   DS
        LEA    DX, zero
        MOV    AX, c_seg
        MOV    DS, AX
        MOV    AX, 2500h    ; AH ← 25h — номер функции, AL ← type = 00
        INT    21h          ; M(0000) ← IPU = offset zero, M(0002) ← CSU = c_seg
        POP    DS           ; DS ← M(SP0 – 6), SP ← SP + 2
; INT 21h, функция DOS 25h — установка вектора прерывания
; Деление целых чисел с переполнением ("divide by zero")
        MOV    BL, 0A4h     ; BH ← A4h = 164d
        MOV    AX, 0A400h   ; AX ← A400h = 41984d
        DIV    BL           ; "divide by zero" ⇒ системное прерывание INT 0
        MOV    AH, 8
        INT    21h          ; Ожидание нажатия любой клавиши
; INT 21h, функция DOS 08h — ввод с клавиатуры символа в ASCII-коде без эха
; Восстановление системного вектора прерывания для INT 0
        CLI                 ; IF ← 0 — запрет аппаратных прерываний
        MOV    SI, DS
        POP    AX           ; AX ← M(SP0 – 4) = CSC, SP ← SP + 2
        MOV    DS, AX       ; DS ← CSC
        POP    DX           ; DX ← M(SP0 – 2) = IPC, SP ← SP + 2
        MOV    AX, 2500h    ; AH ← 25h — номер функции, AL ← type = 00
        INT    21h          ; M(0000) ← IPC, M(0002) ← CSC
; INT 21h, функция DOS 25h — установка (восстановление) вектора прерывания
        MOV    DS, SI
        STI                 ; IF ← 1 — разрешение аппаратных прерываний
        RET
main    endp                ; Конец процедуры main
; Подпрограмма для получения системного вектора прерывания
c_intr  proc   near         ; Начало процедуры c_intr
        MOV    AX, 3500h    ; AH ← 35h — номер функции, AL ← type = 00
        INT    21h          ; BX ← M(0000) = IPC, ES ← M(0002) = CSC
                            ; ES:BX = CSC:IPC — адрес подпрограммы INT 0
; INT 21h, функция DOS 35h — получение вектора прерывания
        RET
c_intr  endp                ; Конец процедуры c_intr
; Подпрограмма, вызываемая системным прерыванием INT 0
zero    proc   far          ; Начало процедуры прерывания zero
        POP    AX           ; Изменение адреса возврата на команду, следующую за
        ADD    AX, 2        ;      командой DIV, вызывающей проблему "divide by zero"
```

```
        PUSH    AX
        LEA     DX, Message ; DX — начальный адрес символьной строки
        MOV     AH, 9       ;     Вывод на экран дисплея сообщения Message:
        INT     21h         ; Чтоб в арифметике упрочиться,
                            ; О-о-о, Юный Неофит!
                            ; Пиши не так, как хочется,
                            ; А как Zero велит! (нажмите любую клавишу)
; INT 21h, функция DOS 09h — вывод на дисплей ASCII-строки
        IRET
zero    endp                ; Конец процедуры прерывания zero
c_seg   ends                ; Конец сегмента кода
        end     main        ; Конец программы
```

Команды INT *type* и IRET. В системе команд имеется три типа команд *программного прерывания*: INT 3, INTO (однобайтовые команды) и INT *type* (двухбайтовые команды), где *type* = 0 … 255. Прерывания подразделяются на *внутренние*, которые вызываются состоянием МП или одной из перечисленных команд, входящих в программу, и *внешние* — инициируемые сигналом, подаваемым в МП от других компонент системы (обычно запросы прерываний поступают от контроллера прерываний 8259*A* и по входу немаскируемого запроса прерывания *NMI*). Типичное внутреннее прерывание инициируется, например, при делении числа на нуль, а типичные внешние прерывания — сигналами на входах *NMI* и *INTR* микропроцессора. Команды прерываний по назначению аналогичны командам CALL вызова процедур (подпрограмм). Вызываемая командой прерывания подпрограмма называется *процедурой прерывания*.

Некоторыми видами прерываний управляют флаги IF и TF. Если условия для прерывания удовлетворяются и необходимые флаги установлены, МП завершает текущую команду, а затем реализует последовательность прерывания:

$$\text{SP} \leftarrow \text{SP} - 2,\ M(\text{SP}) \leftarrow \text{PSW},\ \text{TF} \leftarrow 0,\ \text{IF} \leftarrow 0,$$
$$\text{SP} \leftarrow \text{SP} - 2,\ M(\text{SP}) \leftarrow \text{CS},\ \text{SP} \leftarrow \text{SP} - 2,\ M(\text{SP}) \leftarrow \text{IP},$$
$$\text{IP} \leftarrow M(4 \times type),\ \text{CS} \leftarrow M(4 \times type + 2),$$

где *PSW* — регистр признаков, *TF* — флаг трассировки, *IF* — флаг разрешения прерываний, $4 \times type$ — адрес первого слова вектора прерывания, $4 \times type + 2$ — адрес второго слова вектора прерывания. Новое содержимое регистров *IP* и *CS* определяет начальный адрес выполняемой процедуры прерывания.

Двойное слово, загружаемое в указатель инструкции *IP* и сегментный регистр кода *CS*, называется *вектором прерывания*. Для хранения двойного слова требуются 4 байта, поэтому вектора прерываний могут занимать первые 1024 байта памяти и их нельзя использовать для других целей. Некоторые из 256 типов прерываний резервируются операционной системой и должны инициализироваться после включения компьютера. Пользователи приспосабливают остальные типы прерываний в соответствии со своими требованиями. В *задаче* 36 было показано, как можно использовать и системные векторы прерываний для вызова своих процедур обслуживания прерываний.

После обслуживания прерывания возврат в прерванную программу осуществляется командой IRET, которая извлекает из стека значения *IP*, *CS* и *PSW*:

$$\text{IP} \leftarrow M(\text{SP}),\ \text{SP} \leftarrow \text{SP} + 2,\ \text{CS} \leftarrow M(\text{SP}),\ \text{SP} \leftarrow \text{SP} + 2,\ \text{PSW} \leftarrow M(\text{SP}),\ \text{SP} \leftarrow \text{SP} + 2$$

в обратном порядке по отношению к их включению в стек.

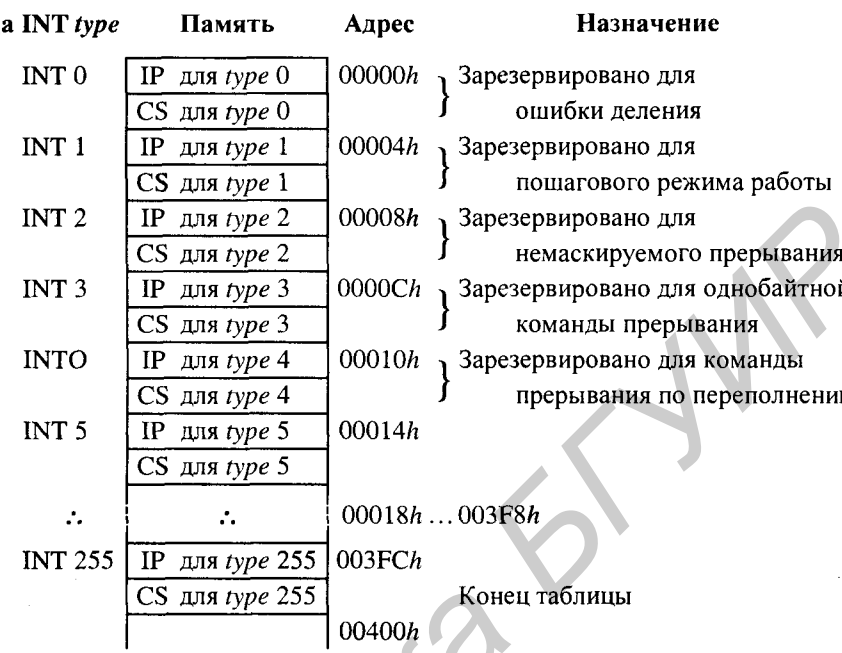


Рис. 4.26. Векторная система прерываний

Типы прерываний представлены на рис. 4.26, иллюстрирующем размещение векторов прерываний в памяти. Только первые пять типов определены явно, а остальные отведены для команд программных прерываний (INT *type*, используемых в программах) или внешних аппаратных прерываний *type*, поступающих от контроллера прерываний 8259*A*. Некоторые типы прерываний в персональных компьютерах *IBM PC* используются операционной системой *MS-DOS* (*Microsoft Disk Operating System*) и базовой системой ввода-вывода *BIOS* (*Basic Input-Output System*), обеспечивающей управление периферийным оборудованием компьютера. Например, команда INT 21*h* вызывает для выполнения функцию *MS-DOS*, номер которой должен быть предварительно задан в регистре AH (см. § 4.5), а команда INT 10*h* — функцию *BIOS*.

Прерывание типа 0 вызывается автоматически по ошибке деления — компьютер включает текущее содержимое регистров *PSW*, *CS* и *IP* в стек, загружает регистры *IP* и *CS* из содержимого памяти по адресам 00000 и 00002 и продолжает выполнение программы с адреса, определяемого новым содержимым регистров *IP* и *CS*. Прерывание из-за ошибки деления возникает независимо от состояния флагов *IF* и *TF*, когда в командах DIV или IDIV частное превышает диапазон представления чисел.

Прерывание типа 1 обеспечивает пошаговую работу и только им управляет флаг *TF*. Если значение флага *TF* = 1, то по окончании следующей команды возникает прерывание, которое вызывает процедуру обработки прерывания, адрес которой определяется содержимым ячеек памяти 00004 ... 00007.

Прерывание типа 2 вызывается независимо от состояния флага *IF* сигналом, поступающим на вход *NMI* (*Non-maskable Interrupt*) микропроцессора.

Остальные типы прерываний вызываются либо командами прерываний, включенными в программу (этими прерываниями флаг *IF* не управляет), либо внешними прерываниями, за-

просы которых поступают на вход *INTR* микропроцессора (разрешением и запретом этих прерываний управляет флаг *IF*).

Команда INT 3 часто применяется при отладке, когда пошаговая работа дает больше информации, чем требуется. Помещая команды INT 3 в контрольных точках программы (*Breakpoint interrupt*), программист может использовать процедуру прерывания для вывода на экран дисплея или на принтер результата выполнения программы до контрольной точки.

Команда INTO имеет тип 4 и вызывает прерывание, если только значение флага OF = 1. Ее часто помещают сразу после арифметической команды, способной вызвать ошибку переполнения, и при наличии переполнения осуществляется специальная обработка этой ошибки. В отличие от деления на нуль переполнение не вызывает автоматического прерывания — прерывание необходимо явно определять командой INTO.

Команда IRET осуществляет возврат из процедуры прерывания. Она аналогична команде RET, но кроме адреса возврата извлекает из стека исходное содержимое PSW.

Команда INT 3 выполняется за 52 такта, команда INTO — за 53 такта (OF = 1) и за 4 такта (OF = 0), команда INT type ≠ 3 — за 51 такт. Команда IRET выполняется за 24 такта.

Задача 37. Написать программу умножения 16-разрядных двоичных чисел AX × DX с выводом на экран дисплея произведения DX:AX в пошаговом режиме. *Решение*:

```
s_seg   segment page STACK   ; Сегмент стека
        dw      32 dup (?)
s_seg   ends                 ; Конец сегмента стека
d_seg   segment              ; Сегмент данных
M_DX    db      10, ' DX = 0000h, AX = 0000h', 13, 10
        db      10, ' Нажмите любую клавишу', 10, 13
tabl    db      '0123456789ABCDEF'
; Управляющие ASCII-коды:
;     13d = 0Dh — возврат каретки (сдвиг курсора в левую позицию экрана)
;     10d = 0Ah — перевод строки (сдвиг курсора вниз на одну позицию)
;     $ — задание конца строки, выводимой на дисплей
d_seg   ends                 ; Конец сегмента данных
c_seg   segment              ; Сегмент кода
        assume  CS: c_seg, DS: d_seg, SS: s_seg, ES: d_seg
main    proc    far          ; Начало процедуры main
        push    ds           ; Запись в стек начального адреса PSP (Program Segment Prefics)
        sub     ax, ax       ;      и 0000h (для возврата в DOS)
        push    ax
        mov     ax, d_seg
        mov     ds, ax       ; Инициализация сегментного регистра DS
        mov     es, ax       ; ES = DS
; Получение системного вектора прерывания CSC:IPC для INT 1
        MOV     AX, 3501h    ; AH ← 35h — номер функции, AL ← type = 01
        INT     21h          ; BX ← M(0000) = IPC, ES ← M(0002) = CSC
                             ; ES:BX = CSC:IPC — адрес подпрограммы INT 1
; INT 21h, функция DOS 35h — получение вектора прерывания
; Исходное значение указателя стека SP = SP0
        PUSH    BX           ; SP ← SP – 2, M(SP0 – 2) ← IPC (сохранение в стеке CSC:IPC)
        PUSH    ES           ; SP ← SP – 2, M(SP0 – 4) ← CSC
```

```
                ; Замена системного вектора прерывания пользовательским CSU:IPU
        PUSH    DS
        LEA     DX, step    ; DX ← адрес (смещение) процедуры прерывания step
        MOV     AX, c_seg
        MOV     DS, AX
        MOV     AX, 2501h   ; AH ← 25h — номер функции, AL ← type = 01
        INT     21h         ; M(0000) ← IPU = offset step, M(0002) ← CSU = c_seg
        POP     DS          ; DS ← M(SP0 – 6), SP ← SP + 2
; INT 21h, функция DOS 25h — установка вектора прерывания
; Перевод программы в пошаговый режим
        PUSHF               ; Включение в стек регистра признаков PSW
        POP     AX          ; AX ← PSW — извлечение PSW из стека в регистр AX
        OR      AH, 1       ; AH ← AH ∨ 0000 0001 (TF ← 1 — задание режима трассировки)
        PUSH    AX          ; Включение в стек нового значения PSW
        POPF                ; PSW ← новое значение PSW (TF = 1)
; INT 21h, функция DOS 08h — ввод с клавиатуры символа в ASCII-коде без эха
; Умножение 16-разрядных чисел
        LEA     BX, tabl
        MOV     DX, 7BC4h   ; DX ← 7BC4h
        MOV     AX, 95A3h   ; AX ← 95A3h
        MUL     DX          ; DX:AX ← AX × DX = 4857 E1CCh — произведение
; Вывод программы из пошагового режима
        PUSHF               ; Включение в стек регистра признаков PSW
        POP     AX          ; AX ← PSW — извлечение PSW из стека в регистр AX
        AND     AH, 0FEh    ; AH ← AH & 1111 1110 (TF ← 0 — запрет режима трассировки)
        PUSH    AX          ; Включение в стек нового значения PSW
        POPF                ; PSW ← новое значение PSW (TF = 0)
; Восстановление системного вектора прерывания для INT 1
        CLI                 ; IF ← 0 — запрет аппаратных прерываний
        MOV     SI, DS
        POP     AX          ; AX ← M(SP0 – 4) = CSC, SP ← SP + 2
        MOV     DS, AX      ; DS ← CSC
        POP     DX          ; DX ← M(SP0 – 2) = IPC, SP ← SP + 2
        MOV     AX, 2501h   ; AH ← 25h — номер функции, AL ← type = 01
        INT     21h         ; M(0000) ← IPC, M(0002) ← CSC
; INT 21h, функция DOS 25h — установка (восстановление) вектора прерывания
        MOV     DS, SI
        STI                 ; IF ← 1 — разрешение аппаратных прерываний
        RET
main    endp                ; Конец процедуры main
; Подпрограмма преобразования двоичных чисел в ASCII-числа
conv    proc    near        ; Начало процедуры conv
        MOV     CH, 2       ; CH ← 2 — число циклов повторения
L1:     MOV     SI, AX      ; SI ← AX
        AND     AL, 0Fh     ; AL ← AL & 0000 1111
        XLAT                ; AL ← ASCII-код младшей тетрады AL
```

```
        MOV     [DI], AL    ; Запись ASCII-кода младшей тетрады AL в буфер M_DX
        MOV     AX, SI      ; AX ← SI — восстановление AL
        MOV     CL, 4       ; CL ← 4 — число сдвигов
        SHR     AL, CL      ; AL ← 0000 A7A6A5A4
        XLAT                ; AL ← ASCII-код старшей тетрады AL
        DEC     DI          ; DI ← DI – 1
        MOV     [DI], AL    ; Запись ASCII-кода старшей тетрады AL в буфер M_DX
        DEC     DI          ; DI ← DI – 1
        MOV     AL, AH      ; AL ← AH
        DEC     CH          ; CH ← CH – 1
        JNZ     L1
        RET
conv    endp                ; Конец процедуры conv
; Подпрограмма, вызываемая пользовательским прерыванием INT 1
step    proc    far         ; Начало процедуры прерывания step
        PUSH    AX
        PUSH    DX
        LEA     DI, M_DX+22 ; DI ← адрес младшего разряда ASCII-числа для AX
        CALL    conv
        MOV     AX, DX
        LEA     DI, M_DX+10 ; DI ← адрес младшего разряда ASCII-числа для DX
        CALL    conv
        LEA     DX, M_DX    ; DX ← начальный адрес символьной строки
        MOV     AH, 9       ; Вывод на дисплей сообщения M_DX:
        INT     21h         ; DX = ××××, AX = ××××
; INT 21h, функция DOS 09h — вывод на дисплей ASCII-строки
        MOV     AH, 8
        INT     21h         ; Ожидание нажатия любой клавиши
; INT 21h, функция DOS 08h — ввод с клавиатуры символа в ASCII-коде без эха
        POP     DX
        POP     AX
        IRET
step    endp                ; Конец процедуры прерывания step
c_seg   ends                ; Конец сегмента кода
        end     main        ; Конец программы
```

Установки флага трассировки выполнены без разрушения остальных флагов. Если же разрушение допустимо, то команды PUSHF и POP AX можно исключить. После выполнения команды MUL DX на экран дисплея выводится сообщение:

$$DX = 4857h,\ AX = E1CCh.$$

Не составляет труда написать программу вывода в пошаговом режиме на экран дисплея содержимого всех регистров и используемой в программе памяти данных.

Задача 38. Выполнить инициализацию контроллера прерываний 8259*A* (рис. 4.27), задав базовое значение типа прерывания $type_B$ = C8*h* и приняв адреса его портов равными E4*h* и E6*h*.
Решение:

```
MOV   AL, 13h      ; AL ← ICW1 = 13h = 0001 0011 (см. рис. 3.49)
OUT   0E4h, AL     ; I/O(A4h) ← ICW1
MOV   AL, 0C8h     ; AL ← ICW2 = C8h = 1100 1000 (см. рис. 3.50)
OUT   0E6h, AL     ; I/O(A6h) ← ICW2
MOV   AL, 0Fh      ; AL ← ICW4 = 0Fh = 0000 1111 (см. рис. 3.53)
OUT   0E6h, AL     ; I/O(A6h) ← ICW4
```

После этого по адресам 4 × (C8*h* + *i*) следует записать векторы прерываний, по которым будут вызываться процедуры обработки внешних прерываний, поступающих от 8 внешних устройств по входам IR_i ($i = 0 \dots 7$).

Шина данных контроллера прерываний D_{7-0} подключена к младшей части AD_{7-0} локальной мультиплексной шины адреса-данных МП, поэтому младший разряд A_0 шины адреса МП не может быть использован для адресации контроллера (при значении $A_0 = 1$ передача данных производится по старшей части шины AD_{7-0}). По этой причине на вход A_0 контроллера прерываний следует подать сигнал A_1 адресной шины МП, а значит адреса двух портов при использовании команд вывода OUT *port*, AX будут определяться значением *port* = $A_7A_6 \dots A_2A_1$0 = E4*h* и E6*h* ($A_0 = 0$).

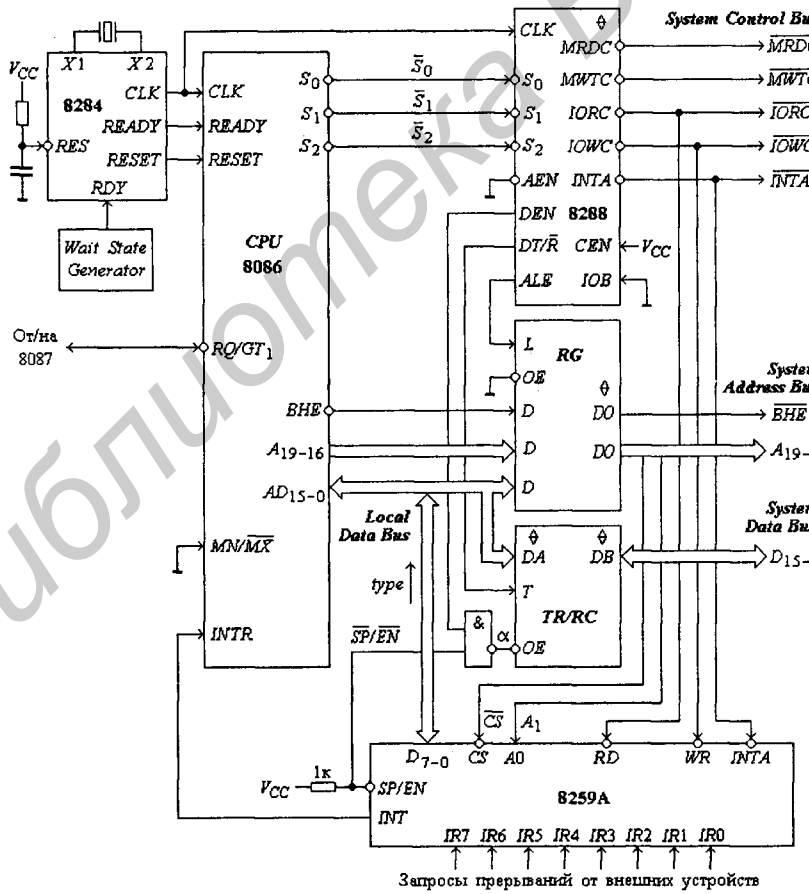


Рис. 4.27. Центральное процессорное устройство с контроллером прерываний

Сигнал выбора кристалла $\overline{CS} = \overline{A}_7 \vee \overline{A}_6 \vee \overline{A}_5 \vee A_4 \vee A_3 \vee \overline{A}_2$ должен вырабатываться адресным дешифратором (значение сигнала $\overline{CS} = 0$ при значениях *port* = E4*h* и E6*h*). При инициализации контроллера прерываний следует задать буферированный режим (см. § 3.5). Управление (включение/выключение) приемопередатчиком *TR/RC* производится сигналом

$$\alpha = \overline{DEN \cdot \overline{SP} / \overline{EN}} = \begin{cases} 0 - \text{при обращении к системной шине данных,} \\ 1 - \text{при обращении к контроллеру прерываний,} \end{cases}$$

так как при обращении к контроллеру прерываний, работающему в буферированном режиме, на его выходе $\overline{SP}/\overline{EN}$ автоматически устанавливается значение 0. Значение сигнала $\alpha = 1$ выключает (переводит в *Z*-состояние) приемопередатчик *TR/RC*. Аппаратное квитирование ввода-вывода по прерыванию при значении флага разрешения прерываний IF = 1 определяется схемой:

$$IR_i \sqcap \Rightarrow INT \sqcap \Rightarrow IF \sqcup, \ \overline{INTA} = \sqcup\!\sqcap\!\sqcup\!\sqcap \Rightarrow type = \text{C8}h + i \text{ поступает в МП.}$$

Чтение типа прерывания *type* = C8*h* + *i* из контроллера прерываний безадресное — выполняется сигналом $\overline{INTA}$, который больше нигде не используется.

Команда JMP *target*. Команда безусловного перехода JMP передает управление по адресу, определяемому операндом *target*. Переходы могут быть типа *short* (короткий), *near* (близкий, или внутрисегментный) и *far* (далекий, или межсегментный). Диапазон адресов коротких переходов относительно адреса следующей команды составляет –128 ... +127 байт. Методы адресации переходов были описаны в § 4.2 (см. рис. 4.23).

Пример 21:

```
Адрес   Машинный код
0012  EB 0C              JMP   short L1   ; IP ← IP + disp8 (disp8 = 0Ch)
0014  EB 0A 90           JMP   L1         ; Следовало задать атрибут short
0017  EB 07 90 90 90     JMP   far ptr L1 ; Неверно задан атрибут far
001C  FF 27              JMP   [BX]       ; IP ← M(BX) — переход типа near
001E  FF 2F              JMP   dword ptr [BX] ; IP ← M(BX), CS ← M(BX+2)
0020  B0 09          L1: MOV   AL, 9      ;        (косвенный переход типа far)
0022  8B 17              MOV   DX, [BX]
0024  EB FA              JMP   short L1   ; IP ← IP + disp8 (disp8 = FAh)
0026  EB F8              JMP   L1         ; Атрибут short не потребовался
0028  EB F6              JMP   far ptr L1 ; Ассемблер исправил ошибку в
                                          ;              задании атрибута
002A  FF E3              JMP   BX         ; IP ← BX — косвенный переход типа near
```

В этом примере приведен результат трансляции ассемблером команд переходов, записанных в различных формах. Если атрибуты *short* или *far* не указаны, то ассемблер по умолчанию считает переход близким (типа *near*). Трансляция производится от начала к концу программы. Метка L1 в шести командах соответствует короткому переходу, но для команд перехода вперед ассемблер еще не знает адреса перехода и потому резервирует одну ячейку памяти для близко-

го перехода JMP L1 и три ячейки памяти для далекого перехода JMP *far* ptr L1. Машинный код 90*h* соответствует команде NOP — лишние команды. Для команд перехода назад ассемблеру уже известен адрес перехода, поэтому он исправляет ошибочное указание атрибута *far* в предпоследней команде. Таким образом, неправильное задание атрибутов команд безусловных переходов транслятор не обнаруживает — не выводит предупреждения об ошибке, что может привести к увеличению длины программы.

Трансляция команд прямого межсегментного перехода JMP *far* ptr L1 была бы выполнена в соответствии с атрибутом *far*, если бы метка L1 находилась в другом сегменте кода:

IP ← смещение L1 в сегменте, CS ← сегментный адрес L1.

Команды JMP выполняются за 15 тактов (внутрисегментный и межсегментный прямой переходы), за 11 тактов (внутрисегментный косвенный переход через регистр), за 18 + *ea* (внутрисегментный косвенный переход через память) и за 24 + *ea* (межсегментный косвенный переход).

Команды условных переходов. Все эти команды двухбайтовые. Второй байт представляет собой смещение *disp*8 — число со знаком. Переходы могут быть только типа *short* (короткие) в диапазоне адресов –128 ... +127 байт относительно адреса следующей команды. Переходы осуществляются только по меткам. Описание команд условных переходов приведено в табл. 4.15 (большинство команд имеют один или два синонима). Выполняется переход или не выполняется, зависит от состояния флагов. При переходе выполняется действие:

IP ← IP + *disp*8.

Если указанный в программе условный переход выходит из диапазона –128 ... +127 байт, то при трансляции ассемблер выдаст сообщение об ошибке. В этом случае команду условного перехода необходимо заменить двумя командами: командой условного перехода с альтернативным условием перехода и следующей за ней командой безусловного перехода типа *near*.

Пример 22:

```
    JZ    L2        ; Если расстояние до метки L2 превышает +127 байт, то команду JZ
    MOV   [BX], AL  ; необходимо заменить двумя командами JNZ и JMP
    ∴               ;      (длина программы увеличивается на три байта)
    JNZ   L0
    JMP   L2
L0: MOV   [BX], AL
    ∴               ; ∴— другие команды
L2: ∴
```

Мнемоника команд условных переходов начинается с буквы J (*jump*), а остальные буквы содержат информацию об условии перехода (см. табл. 4.13). Последние восемь команд условных переходов в табл. 4.15 ориентированы на использование после команд сравнения CMP src_1, src_2, выполняющих операцию сравнения $src_1 - src_2$ чисел без знака и со знаком. Эти команды реализуют решения (переходы), опирающиеся на операторы отношений "<", ">", "≤" и "≥".

Команды условных переходов выполняются за 16 тактов (переход есть) и за 4 такта (перехода нет).

Таблица 4.15. **Команды условных переходов и циклов**

Команды переходов	Назначение команды	Условие перехода
JE/JZ	Перейти, если нуль/если равно	ZF = 1
JNE/JNZ	Перейти, если не нуль/если не равно	ZF = 0
JS	Перейти, если знак установлен	SF = 1
JNS	Перейти, если знак сброшен	SF = 0
JO	Перейти, если есть переполнение	OF = 1
JNO	Перейти, если нет переполнения	OF = 0
JP/JPE	Перейти, если есть паритет/если четный паритет	PF = 1
JNP/JPO	Перейти, если нет паритета/если нечетный паритет	PF = 0
JB/JNAE/JC	Перейти, если ниже/если не выше и не равно (без знака)	CF = 1
JNB/JAE/JNC	Перейти, если не ниже/если выше или равно (без знака)	CF = 0
JBE/JNA	Перейти, если ниже или равно/если не выше (без знака)	CF ∨ ZF = 1
JNBE/JA	Перейти, если не ниже и не равно/если выше (без знака)	CF ∨ ZF = 0
JL/JNGE	Перейти, если меньше/если не больше и не равно (со знаком)	SF ⊕ OF = 1
JNL/JGE	Перейти, если не меньше/если больше или равно (со знаком)	SF ⊕ OF = 0
JLE/JNG	Перейти, если меньше или равно/если не больше (со знаком)	SF ⊕ OF ∨ ZF = 1
JNLE/JG	Перейти, если не меньше и не равно/если больше (со знаком)	SF ⊕ OF ∨ ZF = 0

Команды циклов	Назначение команды	Условие перехода
LOOP	Зациклить CX раз (перейти, если CX ≠ 0)	CX ≠ 0
LOOPZ/LOOPE	Зациклить, пока нуль/равно	CX ≠ 0 и ZF = 1
LOOPNZ/LOOPNE	Зациклить, пока не нуль/не равно	CX ≠ 0 и ZF = 0
JCXZ	Перейти, если CX = 0	CX = 0

Команды циклов LOOP, LOOPZ/LOOPE и LOOPNZ/LOOPNE. В этих командах в качестве счетчика циклов используется регистр CX. Команды двухбайтовые — второй байт представляет собой смещение *disp*8 (число со знаком). Переходы могут быть только типа *short* (короткие) в диапазоне адресов –128 … +127 байт относительно адреса следующей команды. Переходы осуществляются только по меткам. Описание команд циклов приведено в табл. 4.15. Выполняется переход или не выполняется, зависит от состояния счетчика циклов CX и флага *ZF*. При переходе выполняются действия:

$$CX \leftarrow CX - 1,\ IP \leftarrow IP + disp8.$$

Команда LOOP выполняется за 17/5 тактов (переход есть/перехода нет), команда LOOPZ/LOOPE — за 18/6 тактов, команда LOOPNZ/LOOPNE — за 19/5 тактов.

Команды циклов LOOPZ/LOOPE и LOOPNZ/LOOPNE обычно используются после команд CMP и SUB. Применение команды LOOP было дано в *задачах* 3, 7, 9, 10, 29 и 35.

Команда JCXZ. Эта команда является по существу командой условного перехода, которая проверяет на 0 содержимое регистра CX. Эта команда отнесена к командам циклов (табл. 4.15), поскольку обычно она применяется в начале цикла, чтобы пропустить тело цикла, когда начальное содержимое регистра CX равно нулю.

Команда JCXZ выполняется за 18/6 тактов (переход есть/перехода нет).

6. *Команды управления процессором*

Команды управления флагами *CF, DF* и *IF*. Описание семи команд управления флагом переноса *CF*, флагом направления *DF* и флагом прерываний *IF* приведено в табл. 4.13. Флаг направления *DF* используется в командах манипуляции цепочками, а флаг прерываний *IF* разрешает (*IF* = 1) или запрещает (*IF* = 0) обслуживание запросов прерываний, поступающих на вход *INTR* МП (рис. 4.27).

Изменять состояния флагов *SF*, *ZF*, *AF*, *PF* и *CF* (рис. 4.28) можно также с помощью команд LAHF и SAHF, выполняющих операции:

7	6	5	4	3	2	1	0
SF	*ZF*	0	*AF*	0	*PF*	1	*CF*

AH ← *low byte PSW* (команда LAHF),
low byte PSW ← AH (команда SAHF).

Рис. 4.28. Младший байт флагов

Загрузив командой LAHF младший байт регистра признаков *PSW* в аккумулятор AH, с помощью логических операций AND, OR и XOR можно задать значения любого из пяти флагов: 0, 1, инверсное либо исходное значение. Затем командой SAHF модифицированные значения флагов возвращаются в регистр признаков *PSW*.

Выполняются команды управления флагами за 2 такта, а команды LAHF и SAHF — за 4 такта.

Команда HLT. Эта команда производит останов МП, переводя его в состояние ожидания сигнала сброса *RESET* = 1 или сигнала немаскируемого прерывания *NMI* = 1. После выполнения команды HLT содержимое регистров CS:IP указывает на команду, следующую за командой HLT. Если в состоянии ожидания значение флага прерываний *IF* = 1, то работу МП возобновляет и по запросу по входу маскируемых прерываний *INTR*. Команда HLT часто используется в диагностических программах МП-систем.

Команда HLT выполняется за 2 такта.

Команда WAIT. Эта команда переводит МП 8086/8088 в состояние ожидания от арифметического сопроцессора 8087 активного уровня (0) сигнала $\overline{TEST}$ = *BUSY* (см. рис. 4.15). Команда WAIT используется для обеспечения синхронизации совместной работы МП 8086/8088 и *NDCP* 8087.

Команда выполняется за 3 + 5 × *n* тактов (*n* — число циклов ожидания значения сигнала $\overline{TEST}$ = 0).

Команда ESC. Данная команда используется для передачи управления от МП 8086/8088 арифметическому сопроцессору 8087. Машинный код команды ESC = 11011×××× является первым байтом команд сопроцессора 8087, содержащих от двух до четырех байт. Для обеспечения синхронизации совместной работы МП 8086/8088 и 8087 программист должен перед каждой командой ESC помещать команду WAIT (либо автоматически добавляет транслятор).

Команда ESC выполняется за 2 такта (регистр) и 8 + *ea* тактов (память).

Команда LOCK. Эта команда является однобайтовым префиксом, который может предшествовать любой команде. По префиксу LOCK микропроцессор вырабатывает активный уровень (0) сигнала блокировки шины $\overline{LOCK}$ на время выполнения команды, непосредственно следующей за префиксом. Сигнал $\overline{LOCK}$ используется в мультипроцессорных системах для аппаратного управления доступом нескольких МП к общим ресурсам (обычно к общей памяти).

В этом случае сигнал $\overline{LOCK}$ подается на арбитр системной шины 8289, который запрещает ее использование другими МП до тех пор, пока значение сигнала $\overline{LOCK} = 0$.

В однопроцессорных системах команда с префиксом LOCK делает системную шину на время ее выполнения недоступной для любого внешнего устройства или события, включая запросы прерывания и передачи данных по прямому доступу к памяти.

Команда LOCK выполняется за 2 такта.

Описание системы команд МП 8086/8088 и дополнительные примеры программ на языке ассемблера можно найти в [13 ÷ 15].

4.4. Директивы ассемблера

Текст исходной программы на языке ассемблера, называемый *исходным модулем*, подготавливается с помощью обычного текстового редактора, имеющегося в дисковой операционной системе (*DOS — Disk Operating System*). Все символы такого текста представлены в *ASCII*-кодах (см. табл. 1.9 на с. 51). Файлу исходного модуля обычно присваивается имя *name*.asm (*name* — имя, составленное не более чем из 8 допустимых символов).

Общие сведения о языке ассемблера. В настоящее время различными фирмами создано много версий ассемблера как программного продукта. Все версии имеют много общего, так как они являются диалектами одного языка ассемблера. Здесь будет кратко рассмотрен *Turbo Assembler* (*TASM*) *Version* 3.2 фирмы *Borland International*.

Текст исходного модуля на языке ассемблера состоит из *операторов ассемблера*, каждый из которых занимает отдельную строку этого текста, содержащую не более 128 символов. Имеется два типа операторов: *инструкции* и *директивы*. Инструкции при трансляции преобразуются в машинные коды, которые исполняются после загрузки в память компьютера *загрузочного модуля* программы, имеющего расширение .com или .exe (*com-программы* и *exe-программы* или *com-файлы* и *exe-файлы*). Директивы (*псевдокоманды*) языка ассемблера не преобразуются в машинные коды, а только управляют *процессом ассемблирования* (трансляции) — преобразования текста исходной программы в коды объектного модуля (расширение .obj). Ассемблер интерпретирует и обрабатывает операторы один за другим, генерируя последовательность из машинных кодов команд МП и байтов данных.

Запись инструкции на языке ассемблера состоит из четырех полей в свободном формате: поля метки, поля кода операции (КОП), поля операндов и поля комментария:

[Метка:]	КОП	[Операнд1[, Операнд2]]	[; Комментарий]	← *Инструкции ассемблера*
[Имя]	Директива	[Выражение]	[; Комментарий]	← *Директивы ассемблера*

(элементы, указанные в квадратных скобках, могут отсутствовать). Метка — это идентификатор, присваиваемый первому байту того оператора, в котором она появляется, а КОП — это мнемоническое обозначение инструкций (команд) МП. Для директив ассемблера эти поля называются полем имени, полем директивы, полем выражения и полем комментария соответственно. Операнды могут описываться выражениями, составленными из чисел и символических имен с помощью *операторов выражений*. Пробелы вводятся произвольно (свободный формат), но минимум один пробел должен быть после кода операции. В двухоперандных командах операнды должны быть разделены запятой, а поле комментария должно начинаться с точки с запятой.

Директивы ассемблера действуют лишь во время компиляции программы и позволяют устанавливать режимы трансляции, задавать структуру сегментирования программы, опреде-

лять содержимое полей данных, управлять печатью листинга программы, обеспечивать условную трансляцию и выполнять некоторые другие функции.

Конструкции языка ассемблера содержат *идентификаторы* и *ограничители*. Идентификатор представляет собой набор букв, цифр и символов _, ?, $, @, % или &, не начинающийся с цифры. Идентификатор должен полностью размещаться на одной строке и может содержать от 1 до 31 символа. Друг от друга идентификаторы отделяются пробелом или ограничителем, которым считается любой недопустимый в идентификаторе символ (например, запятая, точка с запятой, двоеточие, круглые и квадратные скобки). Посредством идентификаторов представляются *переменные*, *метки* и *имена*.

Переменные идентифицируют хранящиеся в памяти данные. Все переменные имеют три атрибута: SEGMENT — сегмент, соответствующий тому сегменту, который ассемблировался, когда была определена переменная; OFFSET — смещение, являющееся адресом данного поля памяти относительно начала сегмента (относительным адресом); TYPE — тип данных, определяющий число байтов, с которыми производятся операции при использовании переменной.

Метка является частным случаем переменной, когда известно, что определяемая ею память содержит машинный код инструкции МП. На нее можно ссылаться в командах переходов и вызовов процедур. Метка имеет два атрибута: SEGMENT и OFFSET. Метка для команд переходов в поле метки должна заканчиваться двоеточием.

Имена не идентифицируют данных в памяти, а используются в программе только для указания замены их при трансляции присвоенным им директивами = или EQU значением.

Некоторые идентификаторы, называемые *ключевыми словами*, имеют фиксированный смысл и должны употребляться только в соответствии с этим. Такими словами являются: директивы ассемблера, инструкции и имена регистров МП, операторы выражений и типы данных (табл. 4.16 — инструкции МП не включены). Ключевые слова нельзя использовать для идентификации переменных, меток и имен.

Операнды, представляющие собой числа, должны обязательно начинаться с цифры (например, 0E843*h*), а идентификаторы операндов (символические имена) — с буквы. Для обозначения систем счисления служат буквы:

Таблица 4.16. **Зарезервированные имена**

Имена регистров, мнемоники директив, операторов и атрибутов										
.8086	and	cs	dx	.errb	far	ifidn	local	org	seg	tbyte
.186	assume	cx	else	.errdef	ge	ifnb	low	%out	segment	.tfcond
.286	ax	db	end	.errdif	group	ifndef	lt	page	.sfcond	this
.386	bh	dd	endif	.erre	gt	include	macro	proc	shl	title
.486	bl	dh	endm	.erridn	high	irp	mask	ptr	short	type
.8087	bp	di	endp	.errnb	if	irpc	mod	public	shr	.type
.287	bx	dl	ends	.errndef	if1	label	name	purge	si	width
.387	byte	dq	eq	..errnz	if2	.lall	ne	qword	size	word
.487	ch	ds	equ	es	ifb	le	near	.radix	sp	.xall
=	cl	dt	.err	even	ifdef	length	not	record	ss	.xcref
ah	comment	dw	.err1	exitm	ifdif	.lfcond	offset	rept	struc	.xlist
al	.cref	dword	.err2	extrn	ife	.list	or	sall	subttl	xor

B или *b* — двоичная система счисления,
O или *Q* (*o* или *q*) — восьмеричная система счисления,
D или *d* — десятичная система счисления,
H или *h* — 16-ричная система счисления

(по умолчанию принимается десятичная система счисления, т. е. указатель *d* можно опускать). В тексте программы на языке ассемблера можно использовать как прописные, так и строчные буквы — ассемблер их не различает.

Ассемблирование, компоновка и отладка программ. Для преобразования исходного модуля в *загрузочный модуль*, выполнимый на персональном компьютере *IBM PC*, можно использовать, например, системный программный пакет *Turbo Assembler Version* 3.2 фирмы *Borland International*. В этот пакет входят программы:

tasm.exe (*Turbo Assembler*) — *ассемблер*, транслирующий исходный модуль *name*.asm в машинные коды и генерирующий файлы *name*.obj, *name*.lst и *name*.map; файл *name*.obj (*объектный модуль*) используется программой *tlink*.exe для создания загрузочного модуля; файл *name*.lst — *листинг* результата трансляции с указанием всех обнаруженных ошибок, помогающий исправить текст исходного модуля (листинг содержит и текст исходной программы); файл *name*.map содержит *карту распределения памяти*, используемой создаваемой программой;

tlink.exe (*Turbo Link*) — *компоновщик*, создающий из одного или нескольких объектных модулей загрузочный модуль *name*.exe или *name*.com;

td.exe (*Turbo Debugger*) — *отладчик*, позволяющий выполнять программу частями и в пошаговом режиме для обнаружения неуловимых другими средствами ошибок в разрабатываемой программе (доступно для анализа — выводится на экран — содержимое всех регистров, флагов и используемой памяти).

Объектный модуль генерируется при запуске из командной строки команды

tasm.exe *name*.asm /L

(генерируются файлы *name*.obj, *name*.lst и *name*.map; при отсутствии параметра /L файл *name*.lst не создается). Загрузочные модули *name*.exe и *name*.com генерируются при запуске из командной строки команд

tlink.exe *name*.obj [/Tde] и tlink.exe *name*.obj /Tdc

соответственно (для генерации загрузочных модулей *name*.exe и *name*.com исходные модули *name*.asm должны быть подготовлены в различных формах). Если вместо параметра /Tdc задать параметр /Tdd, то будет создан файл *name*.dll (dll — *Dynamic Link Library — динамически подключаемая библиотека* для программ, выполняемых под управлением *Windows*). Для запуска отладчика необходимо последовательно выполнить команды

tlink.exe *name*.obj /v и td.exe *name*.exe

(первая команда генерирует файл *name*.exe, а вторая запускает отладчик; параметр /v дает указание компоновщику ввести в файл *name*.exe дополнительную информацию, необходимую для работы отладчика).

Исходный модуль и листинг *exe*-программы. В *примере* 1 приведен отредактированный (исключена несущественная информация и добавлен заголовок) листинг программы, не содержащей ошибок. Из полученного при ассемблировании объектного модуля может быть получен загрузочный модуль — *exe*-файл, выполнимый на персональном компьютере *IBM PC*.

Пример 1. Программа вычисления суммы и разности 8-разрядных десятичных чисел *X* и *Y*, вводимых с клавиатуры, с выводом результата *Z* на дисплей:

Адрес	*Машинный код*	*Имя*	*Директива*
			page 64, 128 ; 64 — число строк на странице листинга
			; 128 — число символов в строке
			title Ввод с клавиатуры / вывод на дисплей (файл 4x04_01*e*.asm)
0000		*s_seg*	segment para stack 'Stack' ; ***<u>Сегмент стека</u>***
0000	08*(E291 AAA5 5320 2150)		dw 8 dup (0E291*h*, 0AAA5*h*, 5320*h*, 2150*h*) ; Стек SP!
0040		*s_seg*	ends ; для визуализации расположения стека в *Turbo Debugger*
0000		*d_seg*	segment ; *<u>Адреса переменных</u>*: ***<u>; Сегмент данных</u>***
0000	08*(??)	*ascii*1	db 8 dup (?) ; *ascii*1 = 0000*h* (для *ASCII*-кодов числа *X*)
0008	00	*sign*	db (?) ; *sign* = 0008*h* (для знака "+" или "–")
0009	08*(??)	*ascii*2	db 8 dup (?) ; *ascii*2 = 0009*h* (для *ASCII*-кодов числа *Y*)
0011	0A 0A*(??)	*addsub*	db 10, 10 dup (?) ; *addsub* = 0011*h* (для *ASCII*-кодов +/–, 0/1, *Z*)
001C	0D 0A 24		db 13, 10, '$' ;
001F	30 31 32 33 34 35	*tabl*	db '0123456789' ; *tabl* = 001F*h* — таблица конвертора *BCD/ASCII*
	36 37 38 39	;	Управляющие *ASCII*-коды:
		;	13*d* = 0D*h* — возврат каретки (сдвиг курсора в левую позицию экрана)
		;	10*d* = 0A*h* — перевод строки (сдвиг курсора вниз на одну позицию)
		;	$ — задание конца строки, выводимой на дисплей
		;	***Символьные строки для вывода на дисплей сообщения (Message)***
0029	20 87 A4 E0 A0 A2 E1 E2 A2 E3 A9 E2 A5 2C 20 A3 AE E1 AF AE A4 A0 21 0D 0A 0A 24	*Mess*	db ' Здравствуйте, господа!', 13, 10, 10, '$'
0044	20 82 A2 A5 A4 A8 E2 A5 20 E1 20 AA AB A0 A2 A8 A0 E2 E3 E0 EB 20 38 2D E0 A0 A7 E0 EF A4 AD EB A5 20 E7 A8 E1 AB A0 20 A8 20 AE AF A5 E0 A0 E6 A8 EE 0D 0A	*Mess*1	db ' Введите с клавиатуры 8-разрядные числа и операцию', 13, 10
0078	20 20 20 A2 20 22 A6 A5 E1 E2 AA AE AC 22 20 E4 AE E0 AC A0 E2 A5 3A 20 58 58 58 58 58 58 58 58 23 59 59 59 59 59 59 59 59		db ' в "жестком" формате: XXXXXXXX#YYYYYYYY'
00A1	20 28 23 20 2D 20 AE AF A5 E0 A0 E6 A8 EF 20 22 2B 22 20 A8 AB A8 20 22 2D 22 29 0D 0A 0A		db ' (# — операция "+" или "–")', 13, 10,10
00BF	20 20 20 28 AA AB A0 A2 A8 E8 A8 3A 20 45 6E 74 65 72 20 2D 20 A2 EB AF AE AB AD A8 E2 EC 2C 20 45 73 63 20 2D 20 A2 EB E5 AE		db ' (клавиши: *Enter* — выполнить, *Esc* — выход)', 13, 10, 10, '$'

```
      A4 29 0D 0A 0A 24
00EF  0A 20 82 EB AF AE  Mess2  db      10, ' Выполнить новое вычисление? (Y)', 13, 10,10, '$'
      AB AD A8 E2 EC 20
      AD AE A2 AE A5 20
      A2 EB E7 A8 E1 AB
      A5 AD A8 A5 3F 20
      28 59 29 0D 0A 0A
      24
0114                     d_seg ends      ; Конец сегмента данных

0000                     c_seg segment                                  ; Сегмент кода
                         assume CS: c_seg, DS: d_seg, SS: s_seg, ES: d_seg
0000                     main  proc  far         ; Начало процедуры main типа far
0000  1E                       PUSH  DS          ; Запись в стек начального адреса PSP
0001  2B C0                    SUB   AX, AX      ;      (Program Segment Prefics) и 0000h
0003  50                       PUSH  AX          ;      (для возврата в DOS)
0004  B8 0000s                 MOV   AX, d_seg   ;
0007  8E D8                    MOV   DS, AX      ; Инициализация сегментного регистра DS
                         ; Вывод на дисплей приглашения к работе
0009  BA 0029r                 LEA   DX, Mess    ; DX — начальный адрес символьной строки
000C  B4 09                    MOV   AH, 9       ;      Вывод на дисплей сообщения Mess:
000E  CD 21                    INT   21h         ; Здравствуйте, господа!
                         ; INT 21h, функция DOS 09h — вывод на дисплей ASCII-строки
0010  BA 0044r           L8:   LEA   DX, Mess1 ; DX — начальный адрес символьной строки
0013  B4 09                    MOV   AH, 9       ;      Вывод на дисплей сообщения Mess1:
0015  CD 21                    INT   21h         ; Введите с клавиатуры 8-разрядные числа и операцию
                         ; в "жестком" формате: XXXXXXXX#YYYYYYYY (# — операция "+" или "–")
                         ;      (клавиши Enter — выполнить, Esc — выход)
                         ; Ввод с клавиатуры данных и операции "+" или "–"
0017  BB 0000r                 LEA   BX, ascii1  ; BX ← адрес первого слагаемого
001A  B4 01              L3:   MOV   AH, 1
001C  CD 21                    INT   21h         ; Ожидание ввода данных с клавиатуры
                         ; INT 21h, функция DOS 01h — ввод с клавиатуры символа в ASCII-коде с эхом
001E  3C 0D                    CMP   AL, 0Dh     ; 0Dh— ASCII-код клавиши Enter (ввод закончен)
0020  74 0C                    JZ    L1
0022  3C 1B                    CMP   AL, 1Bh     ; 1Bh — ASCII-код клавиши Esc (отказ от ввода)
0024  75 03                    JNZ   L0
0026  E9 0088                  JMP   L2          ; Закончить работу
0029  88 07              L0:   MOV   [BX], AL    ; Запись в память ASCII-кода нажатой клавиши
002B  43                       INC   BX
002C  EB EC                    JMP   L3          ; Продолжение ввода
                         ; Инициализация регистров и анализ типа операции ("+" или "–")
002E  BB 001Fr           L1:   LEA   BX, tabl    ; BX ← начальный адрес таблицы преобразования
0031  BD 001Br                 LEA   BP, addsub + 10 ; BP ← адрес младшего разряда суммы/разности
0034  32 D2                    XOR   DL, DL      ; DL ← 0 — знак суммы/разности
0036  B9 0008                  MOV   CX, 8       ; CX ← 8 — число циклов сложения
0039  BE 0007r                 LEA   SI, ascii1 + 7 ; SI ← адрес младшего разряда первого операнда
003C  BF 0008r                 LEA   DI, sign    ; DI ← адрес символа операции "+" или "–"
003F  80 3D 2D                 CMP   [DI], byte ptr 2Dh      ; 2Dh — ASCII-код символа "–" (вычитание)
0042  BF 0010r                 LEA   DI, ascii2 + 7 ; DI ← адрес младшего разряда второго операнда
0045  F8                       CLC               ; CF ← 0
0046  74 1A                    JZ    L4          ; Переход на вычисление разности
```

```
                    ; Вычисление суммы в ASCII-формате
0048  B4 00       L5:  MOV  AH, 0      ; AH ← 0
004A  8A 04            MOV  AL, [SI]   ; AL ← M(SI)
004C  12 05            ADC  AL, [DI]   ; AL ← M(SI) + M(DI) + CF
004E  37               AAA             ; ASCII-коррекция одного разряда суммы
004F  D7               XLAT            ; AL ← ASCII-код одного разряда суммы
0050  3E: 88 46 00     MOV  DS: [BP], AL ; M(DI) ← AL — ASCII-код одного разряда суммы
0054  4E               DEC  SI         ; SI ← SI – 1
0055  4F               DEC  DI         ; DI ← DI – 1
0056  4D               DEC  BP         ; BP ← BP – 1
0057  E2 EF            LOOP L5         ; CX ← CX – 1 и переход, пока CX не равно 0
0059  8A C4            MOV  AL, AH     ; AL ← AH — перенос из старшего разряда суммы
005B  D7               XLAT            ; AL ← ASCII-код переноса (30h или 31h)
005C  3E: 88 46 00     MOV  DS: [BP], AL ; M(BP) = M(sign) ← ASCII-код переноса CF
0060  EB 27            JMP  short L6   ; Переход на вывод суммы на дисплей
                    ; Вычисление разности X – Y (X ≥ Y) или Y – X (X < Y) в ASCII-формате
0062  8A 04       L4:  MOV  AL, [SI]   ; AL ← M(SI)
0064  1A 05            SBB  AL, [DI]   ; AL ← M(SI) – M(DI) – CF
0066  3F               AAS             ; ASCII-коррекция одного разряда разности
0067  D7               XLAT            ; AL ← ASCII-код одного разряда разности
0068  3E: 88 46 00     MOV  DS: [BP], AL ; M(BP) ← AL — ASCII-код разряда разности
006C  4E               DEC  SI         ; SI ← SI – 1
006D  4F               DEC  DI         ; DI ← DI – 1
006E  4D               DEC  BP         ; BP ← BP – 1
006F  E2 F1            LOOP L4         ; CX ← CX – 1 и переход, пока CX не равно 0
0071  3E: C6 46 00 30  MOV  DS: [BP], byte ptr 30h ; M(sign) ← 0
0076  73 11            JNC  L6         ; Переход на вывод разности на дисплей
0078  FE C2            INC  DL         ; DL ← 1
007A  BE 0010r         LEA  SI, ascii2 + 7 ; SI ← адрес младшего разряда второго операнда
007D  BF 0007r         LEA  DI, ascii1 + 7 ; DI ← адрес младшего разряда первого операнда
0080  BD 001Br         LEA  BP, addsub + 10 ; BP ← адрес младшего разряда суммы/разности
0083  B9 0008          MOV  CX, 8      ; CX ← 8 — число циклов сложения
0086  F8               CLC             ; CF ← 0
0087  EB D9            JMP  L4
                    ; Запись знака суммы/разности
0089  4D           L6:  DEC  BP
008A  22 D2            AND  DL, DL     ; DL ← DL & DL (0 — знак "+", 1 — знак "–")
008C  3E: C6 46 00 2D  MOV  DS: [BP], byte ptr 2Dh ; M(sign–1) ← 2Dh — ASCII-код знака "–"
0091  75 05            JNZ  L7
0093  3E: C6 46 00 2B  MOV  DS: [BP], byte ptr 2Bh ; M(sign–1) ← 2Bh — ASCII-код знака "+"
                    ; Вывод на дисплей результата (суммы или разности)
0098  BA 0011r     L7:  LEA  DX, addsub ; DX ← адрес выводимой на дисплей строки
009B  B4 09            MOV  AH, 9      ; Вывод на дисплей результата: суммы или разности
009D  CD 21            INT  21h
009F  BA 00EFr         LEA  DX, Mess2 ; DX — начальный адрес символьной строки
00A2  B4 09            MOV  AH, 9      ; Вывод на дисплей сообщения Mess2:
00A4  CD 21            INT  21h        ; Повторить? (Y)
00A6  B4 08            MOV  AH, 8
00A8  CD 21            INT  21h        ; Ожидание ввода данных с клавиатуры
                    ; INT 21h, функция DOS 08h — ввод с клавиатуры ASCII-символа без эха
```

```
00AA  3C 59          CMP   AL, 59h   ; 59h — ASCII-код символа Y
00AC  75 03          JNZ   L2        ; Выход в DOS
00AE  E9 FF5E        JMP   L8        ; Выполнить новое вычисление суммы или разности
00B1  CB       L2:   RET
00B2           main  endp
00B2           c_seg ends            ; Конец сегмента кода
                     end   main      ; Конец программы
```

Эта программа предназначена для вычисления суммы и разности 8-разрядных десятичных чисел *X* и *Y*, вводимых с клавиатуры, и вывода на дисплей результата вычислений в диалоговом режиме:

Здравствуйте, господа!
Введите с клавиатуры 8-разрядные числа и операцию в "жестком" формате:
XXXXXXXX#YYYYYYYY (# — операция "+" или "–")
(клавиши: *Enter* — выполнить, *Esc* — выход)
57390314–84609275
–027218961
Выполнить новое вычисление? (Y)

Здесь полужирным шрифтом выделены сообщения, выдаваемые персональным компьютером, а обычным шрифтом — вводимые с клавиатуры данные.

Приведенная программа 4*x*04_01*e*.asm (файл имеется на дискете) позволяет достаточно просто изучить назначение основных директив языка ассемблера. В исходном модуле программы можно использовать как строчные, так и прописные буквы (курсив, полужирный шрифт, специальные знаки в поле комментария, например, ← и др., конечно, недоступны в редакторах текста, по необходимости использующих *ASCII*-коды символов).

Исходная программа может содержать ошибки двух типов: *синтаксические* и *семантические*. Синтаксическая ошибка (*syntax error*) — это ошибка, заключающаяся в нарушении грамматических правил языка или неправильной записи конструкции языка. Трансляторы обнаруживают все без исключения ошибки этого типа, а некоторые и исправляют. Семантическая ошибка (*semantic error*) — это смысловая ошибка, не связанная с нарушением синтаксиса. К числу таких ошибок относятся: неправильное описание алгоритма решения задачи, неверное определение переменных, несогласованность исходных данных с алгоритмом, неверное использование правильно записанных конструкций языка и др. Программа, содержащая семантические ошибки, выполняться будет, но решать она будет другую задачу, отличающуюся от поставленной. Для выявления семантических ошибок используется отладчик, позволяющий проанализировать выполнение программы в пошаговом режиме или с точками останова. В вышеприведенной программе нет как синтаксических, так и семантических ошибок.

Для удобства распознавания директив и команд МП в исходной программе директивы набраны строчными буквами, а команды — прописными буквами. Символические имена постоянных данных и переменных набраны курсивными строчными буквами.

Любая программа, написанная на языке ассемблера, содержит, по крайней мере, один сегмент — сегмент кода (до 64 Кбайт), в котором располагаются команды программы, данные и стек. Типовая программа для генерации *exe*-файла содержит три сегмента — сегмент кода, сегмент данных и сегмент стека. Текущие физические адреса в сегментах кода и стека определяются парами регистров CS:IP (сегментный регистр *CS* и указатель инструкции *IP*) и SS:SP (сегментный регистр *SS* и указатель стека *SP*). Текущий физический адрес в сегменте данных опре-

деляется парами DS:*EA*, SS:*EA* или ES:*EA* при указании префикса замены сегмента (см. рис. 4.22), где *EA* — эффективный адрес, вычисляемый в соответствии с используемым режимом адресации данных (см. табл. 4.9). Смещения внутри сегментов (содержимое регистров, указанных после двоеточия) полностью определяются написанной программой, а инициализация сегментных регистров производится операционной системой при загрузке *exe*-программы в свободную область оперативной памяти персонального компьютера. Одна и та же программа может загружаться по разным адресам в зависимости от числа ранее загруженных резидентных программ (*exe*-файлы — переместимые).

В табл. 4.17 приведены директивы определения сегментов и процедур, а в табл. 4.18 — директивы определения данных, используемых в сегменте данных. Описание всех директив языка ассемблера имеется в [16 ÷ 18]. Здесь же будет дано описание только наиболее часто используемых в простых программах директив и операторов языка ассемблера.

Таблица 4.17. **Директивы определения сегментов и процедур**

Имя	*Директива*	*Выражение*	*Комментарий*
s_seg	segment	para stack 'Stack'	; *Segment Definition* — определение сегмента стека
	∴	∴	; ← **Директива определения размера стека**
s_seg	ends		; *End Segment* — конец сегмента стека
d_seg	segment		; *Segment Definition* — определение сегмента данных
∴	∴	∴	; ← **Директивы определения данных**
d _seg	ends		; *End Segment* — конец сегмента данных
c_seg	segment		; *Segment Definition* — определение сегмента кода
	assume	CS: *c_seg*, DS: *d_seg*, SS: *s_seg*, ES: *d_seg*	; Привязка сегментов к сегментным регистрам
main	proc	*far*	; *Procedure Definition* — определение процедуры
∴	∴	∴	; ← **Исходный текст программы**
main	endp		; *End Procedure* — конец процедуры
c_seg	ends		; *End Segment* — конец сегмента кода
	end	*main*	; *End* — конец программы

Таблица 4.18. **Директивы определения данных**

Имя	*Директива*	*Выражение*	*Комментарий*
*addr*0	db	5A*h*, 0C7*h*, 75	; *Define Byte* — определить байт ; (в память записываются 3 байта)
tabl	db	'0123456789'	; *Define Byte* — определить байт ; (в память записываются 10 байт *ASCII*-кодов 30 … 39)
*var*1	dw	0ABCD*h*, 1999	; *Define Word* — определить слово ; (в память записываются два слова)
*var*2	dw	20*h* dup (?)	; *Define Word* — в памяти резервируется 32 слова
	dd	4-байтные числа	; *Define Double Word* — определение двойного слова
	dq	8-байтные числа	; *Define Quad Word* — определение четырех слов
	dt	10-байтные числа	; *Define Ten Byte* — определение 10 байт
alpha	equ	75*h*	; *alpha* = 0075*h*
beta	equ	0D7CE*h*	; *beta* = 0D7Ce*h*

1. *Директивы сегментирования программы*

Программа на языке ассемблера состоит из последовательности программных сегментов (сегменты стека, данных и кода), заканчивающейся директивой END. Начало и конец любого сегмента задаются директивами SEGMENT и ENDS. Программный сегмент представляет собой последовательность инструкций (команд МП) и/или полей данных, адресуемых относительно одного сегментного регистра CS, DS, SS или ES, присваиваемого сегменту директивой ASSUME.

Директивами PROC и ENDP в сегментах кода выделяются процедуры, позволяющие использовать часть программного кода многократно без его дублирования в разных частях программы. Процедуры являются основным средством разделения кода программы на модули, которые легко по отдельности разрабатывать, тестировать и документировать.

Сегменты могут объединяться в группу при помощи директивы GROUP. Директивы ORG и EVEN позволяют управлять адресами размещения команд МП.

Директивы SEGMENT и ENDS. Директива SEGMENT (*Segment Definition* — определение сегмента) имеет формат:

seg_name SEGMENT [*align*] [*combine*] ['*class*']

(параметры, заключенные в квадратные скобки, являются необязательными). Имя сегмента *seg_name* обязательно должно быть указано — назначается любое уникальное имя, не совпадающее с зарезервированными. Сегмент — это набор команд МП и/или данных, адреса которых вычисляются относительно одного и того же сегментного регистра CS, DS, SS или ES.

Директива ENDS (*End Segment* — конец сегмента) имеет формат:

seg_name ENDS,

где *seg_name* — то же самое имя, что и в директиве SEGMENT (см. табл. 4.17: *s_seg*, *d_seg* и *c_seg* — имена сегментов стека, данных и кода). Таким образом, директивы SEGMENT и ENDS задают начало и конец сегмента, адресуемого относительно одного из сегментных регистров SS, DS, ES и CS. Между этими директивами располагаются другие директивы ассемблера, в частности, директивы определения данных для сегмента данных и директивы определения процедур для сегмента кода. Исходный модуль может иметь несколько директив SEGMENT с одним и тем же именем, определяющих некоторые части целого сегмента. Все эти части компонуются в один сегмент. Параметры таких директив SEGMENT не должны противоречить друг другу.

Директива SEGMENT может содержать три типа необязательных параметров: *align* (выравнивание), *combine* (объединение, или комбинирование) и '*class*' (класс, или категория). Если они присутствуют, то должны описываться в указанном порядке, хотя любые из них могут и отсутствовать.

Параметр *align* сообщает компоновщику границу адреса относительно предыдущего сегмента, начиная с которого сегмент будет загружаться в память. Этот параметр может иметь значения:

BYTE — выравнивание не выполняется (сегмент располагается, начиная со следующего байта);

WORD — начало сегмента выравнивается на границу следующего слова;

DWORD — начало сегмента выравнивается на границу следующего двойного слова;

PARA — начало сегмента выравнивается на границу следующего параграфа (16 байт);

PAGE — начало сегмента выравнивается на границу следующей страницы (256 байт);

MEMPAGE — начало сегмента выравнивается на границу следующей страницы памяти (4096 байт).

Значение PARA принимается по умолчанию, если параметр *align* не задан. Действительный начальный адрес сегмента не вычисляется до загрузки программы в компьютер (тип выравнивания только накладывает на него ограничение).

Параметр *combine* сообщает компоновщику способ объединения одноименных сегментов, находящихся в нескольких объектных модулях. Этот параметр может иметь значения:

PRIVATE — сегмент не будет объединяться с сегментами, находящимися в других модулях и имеющих такое же имя;

PUBLIC — все сегменты с одинаковыми именами и классами, находящиеся в разных модулях, объединяются в один непрерывный сегмент. Команды или данные нового сегмента будут адресоваться относительно одного сегментного регистра, а все смещения будут вычисляться относительно начала этого сегмента;

STACK — все сегменты с одинаковыми именами и классами объединяются в один непрерывный сегмент. Этот тип комбинирования отличается от типа PUBLIC лишь тем, что адресация в новом сегменте будет производиться относительно сегментного регистра SS (регистр SP при этом устанавливается на конец сегмента). Такой тип комбинирования обычно имеют сегменты стека. Параметр STACK автоматически обеспечивает инициализацию регистров SS и SP, и пользователю не нужно включать в свою программу команды для установки этих регистров. Если не указано ни одного сегмента типа STACK, то компоновщик выдаст предупреждение, что сегмент стека не найден (Warning: *No stack*). Если параметр STACK не был задан, пользователь должен сам программным способом загрузить в регистры SS и SP необходимые адреса;

COMMON — для всех сегментов с одинаковыми именами и классами компоновщик устанавливает общий базовый адрес (адреса данных и команд в каждом из таких сегментов являются смещениями по отношению к общему стартовому адресу). Сегменты будут загружаться в память, начиная с одного адреса (таким способом можно создавать оверлейные программы). Размер области загрузки определяется самым большим сегментом;

AT *address* — задает абсолютный стартовый адрес сегмента. Адреса всех меток и переменных сегмента вычисляются относительно фиксированного параметра *address* (определение абсолютного сегмента). Параметр *address* является 16-разрядным адресом начала параграфа для сегмента. Адрес может быть представлен любым допустимым выражением, не содержащим ссылок вперед. Сегмент с этим типом комбинирования обычно не содержит программного кода или инициализируемых данных, а включает в себя только адресные значения, фиксированные для компьютера (видеобуфер, таблица векторов прерывания, *ROM BIOS* и др.). Тип комбинирования AT нельзя использовать для загрузки кодов или данных в определяемую им область памяти.

Значение PRIVATE принимается по умолчанию, если параметр *combine* не задан.

Параметр '*class*' сегмента используется компоновщиком для задания порядка следования сегментов в памяти. Сегменты одного класса загружаются в память один после другого до того, как начнут загружаться сегменты другого класса. В качестве класса сегмента может быть указано любое допустимое имя, заключенное в апострофы. Поскольку класс сегмента рассматривается как идентификатор, то в программе он не может быть определен еще где-либо. Наиболее часто для сегментов стека и кода используются идентификаторы '*Stack*' и '*Code*'.

Если параметр '*class*' не указан, то компоновщик копирует сегменты в загрузочный модуль в той последовательности, в которой они расположены в объектном модуле. Эта последовательность сохраняется до тех пор, пока компоновщик не обнаружит два или более сегмента

одного класса, после чего компоновщик начинает объединение сегментов. Сегменты одного класса копируются в последовательные блоки загрузочного модуля.

Все сегменты имеют класс — сегменты, для которых параметр '*class*' не указан, считаются принадлежащими к классу с пустым именем и копируются в последовательные блоки памяти вместе с такими же сегментами. Число сегментов, принадлежащих к одному классу, не ограничено, но их суммарный объем не должен превышать 64 Кбайт.

Директива ASSUME. Эта директива используется для привязки сегментов к сегментным регистрам, т. е. для каждого сегмента задает сегментный регистр, который будет использоваться по умолчанию для вычисления исполнительных адресов всех меток и переменных, определенных под заданным именем сегмента. Директива ASSUME (присваивать) имеет два формата:

ASSUME *seg_reg*: *seg_name* [, ...] ... [, ...] или ASSUME NOTHING,

где *seg_reg* — любой сегментный регистр из CS, DS, ES и SS; *seg_name* — имя сегмента, предварительно определенное директивой SEGMENT (см. табл. 4.17), имя группы, определенное директивой GROUP, или ключевое слово NOTHING (ничего). Если используется ключевое слово NOTHING, то предшествующий выбор сегмента аннулируется. При использовании в исходном модуле второго формата директивы ASSUME аннулируется выбор сегментов для всех четырех сегментных регистров.

Директива GROUP. Эта директива используется для сбора сегментов с разными именами, но одного типа (код, данные, стек), под одно имя. Директива ASSUME имеет формат:

gr_name GROUP *seg_name* [, ...] ... [, ...],

где *gr_name* — имя группы, *seg_name* — имя сегмента, заранее определенное директивой SEGMENT. Объединение нескольких сегментов в одну группу с заданным именем позволяет адресовать все метки и переменные в этих сегментах относительно начала группы, а не начала содержащего их сегмента.

Директива GROUP не влияет на порядок загрузки сегментов, который зависит от классов сегментов и их расположения в объектном файле. Сегменты одной группы не обязательно будут занимать непрерывную область памяти. Они могут быть перемешаны с сегментами, не принадлежащими этой группе. Однако расстояние между первым и последним элементами группы не должно превышать 64 Кбайт. Если сегменты группы расположены последовательно, группа может занимать до 64 Кбайт памяти.

Имя группы можно использовать в директиве ASSUME и в качестве префикса замены сегмента, если предварительно имя группы присвоено сегментному регистру директивой ASSUME. В исходном модуле имя группы может быть употреблено только с одной директивой GROUP.

Директивы ORG и EVEN. Для определения относительного адреса программных элементов в сегментах данных и кода ассемблер использует счетчик адресов. Текущее значение адреса обозначается символом $ (можно использовать в директивах). Начальное значение счетчика адресов равно 0000*h*. Все программные элементы (данные и команды МП) располагаются по смежным адресам в том порядке, в котором они следуют в программе. В итоге сегмент представляет собой непрерывную область памяти, занятую данными и/или командами МП. Директива ORG (*Origin* — начало) позволяет изменять значение счетчика адресов и соответственно адрес следующего определяемого элемента. Директива ORG имеет следующий формат:

ORG *expression*,

где *expression* — выражение, значение которого присваивается счетчику адресов (указателю позиции). Адреса следующих за директивой ORG инструкций МП и/или данных будут начи-

наться с указанного значения. Значением выражения должно быть двухбайтовое абсолютное число. Выражение должно быть преобразовано к абсолютному числу при первом проходе ассемблера, следовательно, нельзя использовать ссылки вперед. В качестве элемента выражения может быть использован знак указателя позиции $, обозначающий текущее значение счетчика адресов. Это позволяет вводить смещение следующего элемента относительно предыдущего, используя выражение $ + *N*, где *N* — некоторое число.

Пример 2:

```
  ∴          ; Пусть addr_l — адрес последнего элемента перед директивой ORG
org   $ + 8  ; $ — этот символ обозначает текущее значение счетчика адресов
  ∴          ; Тогда addr_l + 8 — адрес первого элемента после директивы ORG
org   100h   ; Директива ORG устанавливает для команды JMP относительный адрес
JMP   main   ;       в текущем сегменте, равный 100h
```

Директива EVEN (четный, выравнивать) выравнивает следующее за ней поле данных или инструкцию МП на границу слова. Если без директивы EVEN следующий байт располагается по нечетному адресу, то ассемблер генерирует команду NOP в ответ на директиву EVEN, смещая байт на четный адрес. Эта директива имеет формат (параметры отсутствуют):

EVEN.

Директивы PROC и ENDP. Обычно эти директивы используются для описания в кодовом сегменте процедур pr_name, вызываемых командами CALL pr_name. Процедура представляет собой набор инструкций и директив, составляющих некоторую подпрограмму в кодовом сегменте. Директива PROC (*Procedure* — процедура) имеет формат:

pr_name PROC [*attribute*],

где *pr_name* — имя процедуры; *attribute* — атрибут (необязательный), указывающий расстояние между командой CALL *pr_name* и процедурой *pr_name*. Атрибут может быть двух типов:

NEAR — внутрисегментный (близкий) вызов,
FAR — межсегментный (далекий) вызов.

По умолчанию (атрибут не указан) используется тип NEAR. Допускается вложенность процедур с любым числом уровней. Процедура должна содержать хотя бы одну команду RET для возврата из нее — адрес возврата извлекается из стека в соответствии с указанным атрибутом NEAR (только IP) или FAR (IP и CS).

Директива ENDP (*End Procedure* — конец процедуры) имеет формат:

pr_name ENDP,

где *pr_name* — то же самое имя, что и в директиве PROC. Таким образом, директивы PROC и ENDP задают начало и конец процедур.

Имя процедуры *pr_name* имеет те же возможности, что и метка — может быть использовано как операнд в командах переходов, командах вызовов процедур и командах циклов. Примеры процедур были приведены в *задаче* 37 (§ 4.3, с. 399). Процедуре могут быть переданы параметры. Согласно стандартным соглашениям параметры размещаются в стеке.

Пример 3:

```
PUSH  DX      ; SP ← SP – 2, M(SP) ← DX (передача параметра 2)
PUSH  BX      ; SP ← SP – 2, M(SP) ← BX (передача параметра 1)
CALL  pr_par  ; Вызов процедуры pr_par (SP ← SP – 2, M(SP) ← IP)
```

```
        MOV  CL, DH
          ∴
pr_par  proc  near
        PUSH BP          ; SP ← SP – 2, M(SP) ← BP (сохранение указателя базы)
        MOV  BP, SP      ; BP ← SP (загрузка регистра базы содержимым указателя стека)
        MOV  BX, [BP+4]  ; BX ← параметр 1
        MOV  DX, [BP+6]  ; DX ← параметр 2
          ∴
        POP  BP          ; BP ← M(SP), SP ← SP + 2 (восстановление указателя базы)
        RET  4           ; Возврат на команду MOV  CL, DH с "уничтожением" параметров
pr_par  endp
```

Директива END. Эта директива используется для указания конца исходного модуля — все предложения в исходном модуле, следующие за этой директивой, ассемблер игнорирует. Директива END (*End* — конец) имеет формат:

END [*expression*],

где *expression* (выражение) — необязательный параметр, определяющий точку входа программы, в которую будет передано управление при ее запуске на выполнение. Значением этого выражения должен быть адрес в одном из программных сегментов исходного модуля. В исходном модуле может быть определена только одна точка входа. Если программа состоит из нескольких модулей, то только в одном модуле можно задать точку входа. Модуль, в котором задана точка входа, называется главным.

Если точка входа не задана, то при попытке выполнения программы могут возникнуть ошибки. Поэтому директиву END без параметров рекомендуется применять лишь для сегментов, содержащих только поля данных.

Исходный модуль *com*-программы. В начале этого параграфа был приведен исходный модуль, иллюстрирующий сегментирование программы для получения после компоновки загрузочного модуля в виде *exe*-файла. Для получения *com*-файла сегментирование программы выполняется иначе. Отметим вначале различия между *exe*- и *com*-программами:

exe-программа может иметь любой размер, в то время как *com*-программа ограничена размером одного сегмента (не более 64 Кбайт); объем *com*-файла всегда меньше объема соответствующего *exe*-файла (в частности из-за отсутствия 512-байтового заголовка *exe*-файла);

в *com*-программе стек генерируется автоматически, а все данные размещаются в сегменте кода, т. е. в *com*-программе требуется описание только одного сегмента — сегмента кода; директивой assume все остальные сегментные регистры инициализируются значением сегментного регистра *CS*;

в начале сегмента кода *exe*-программы производится включение в стек содержимого сегментного регистра DS и числа 0000*h*, которые извлекаются из стека командой RET — последней командой сегмента кода (это обеспечивает передачу управления операционной системе *DOS*), а также выполняется инициализация сегментного регистра DS начальным значением сегмента данных; в *com*-программе этого делать не нужно;

операционная система *DOS* при загрузке *exe*- и *com*-файлов резервирует в памяти 256 (100*h*) байт для префикса программного сегмента *PSP* (*Program Segment Prefix*) и для *com*-программ все сегментные регистры инициализируются на начало *PSP*, поэтому директивой ORG 100*h* следует задать начало сегмента кода; при загрузке *com*-файла значение 100*h* записывается в указатель инструкции *IP*.

В *примере* 4 приведен текст исходного модуля *com*-программы, выполняющей ту же самую задачу, что и рассмотренная в начале этого параграфа *exe*-программа.

Пример 4:

```
c_seg   segment                 ; Начало исходного модуля и сегмента кода
        assume  CS: c_seg, DS: c_seg, SS: c_seg, ES: c_seg
        org     100h    ;       (сегментные регистры CS = DS = SS = ES)
begin:  JMP     main    ; Точка входа в программу по директиве end begin
ascii1  db      8 dup (?)       ; Начало данных
sign    db      (?)     ;       (определения данных такие же, что и в exe-программе)
          ∴             ; ← Остальные данные такие же, что и в exe-программе
Mess2   db      10, ' Выполнить новое вычисление? (Y)', 13, 10, 10, '$'   ; Конец данных
main    proc    near    ; Начало процедуры main
        LEA     DX, Mess
          ∴             ; ← Остальная часть сегмента кода такая же,
        JMP     L8      ;                       что и в exe-программе
L2:     RET
main    endp            ; Конец процедуры main
c_seg   ends            ; Конец сегмента кода
        end     begin   ; Конец программы (begin указывает точку входа программы)
```

Команда JMP *main* использована для обхода данных, размещенных в начале сегмента кода. Размер этих *exe*- и *com*-файлов составляет 1042 байт (*exe*-файл) и 449 байт (*com*-файл).

2. Директивы определения данных

Директивы определения данных DB (байта), DW (слова), DD (двойного слова), DQ (квадрослова) и DT (10 байт) служат для задания размеров, содержимого и местоположения полей данных в памяти, используемых в программе. В отличие от других директив эти директивы генерируют объектный код (данные копируются в объектный файл). Эти директивы имеют формат:

[*name*] DL *value_*0 [, *value_*1] … [, *value_k*],

где *name* — имя переменной, L = B, W, D, Q или T, *value_m* (*m* = 0, 1, …, *k*) — данные, а квадратные скобки означают необязательные параметры. Если параметр *name* имеется, то транслятор генерирует переменную *name*, имеющую тип BYTE, WORD, DWORD, QWORD или TBYTE для директив DB, DW, DD, DQ и DT соответственно.

С переменной *name* ассемблер связывает смещение (относительный адрес) первого байта данных *value_*0 относительно начала сегмента данных. Смещение же любого операнда *value_m* будет определяться величиной *value_m* + *m* (знак + — оператор арифметического сложения). Выражение *value_m* + *m* можно использовать в инструкциях (командах) МП для *прямой адресации* любого из операндов, заданных директивами определения данных. При ассемблировании производится строгий контроль типов операндов и при несоответствии их в двухоперандных командах транслятор выдает сообщение: Operand types do not match (несоответствие типов операндов).

Директива DB используется напрямую только для МП 8086, директивы DW и DD — как для МП 8086 (*CPU*), так и для арифметического сопроцессора 8087 (*NDCP — Numeric Data CoProcessor*), а директивы DQ и DT — только для *NDCP* 8087. С помощью директивы переопределения типа данных LABEL все эти директивы можно использовать и для МП 8086.

Типы данных арифметического сопроцессора 8087. Используемые в *NDCP* представления чисел с плавающей точкой совместимы со стандартом *IEEE Floating Point Standard* 754. Сопроцессор оперирует численными данными в семи форматах (рис. 4.29), относящимся к трем классам:

двоичные целые (*Word Integer* — слово целое, *Short Integer* — короткое целое, *Long Integer* — длинное целое);

упакованные десятичные целые (*Packed BCD* — упакованное десятичное);

двоичные вещественные числа, или *числа с плавающей точкой* (*Short Real* — короткое вещественное, *Long Real* — длинное вещественное, *Temporary Real* — временное вещественное).

Во всех форматах старший (левый) разряд отведен для знака числа со стандартным кодированием: 0 означает плюс, а 1 — минус.

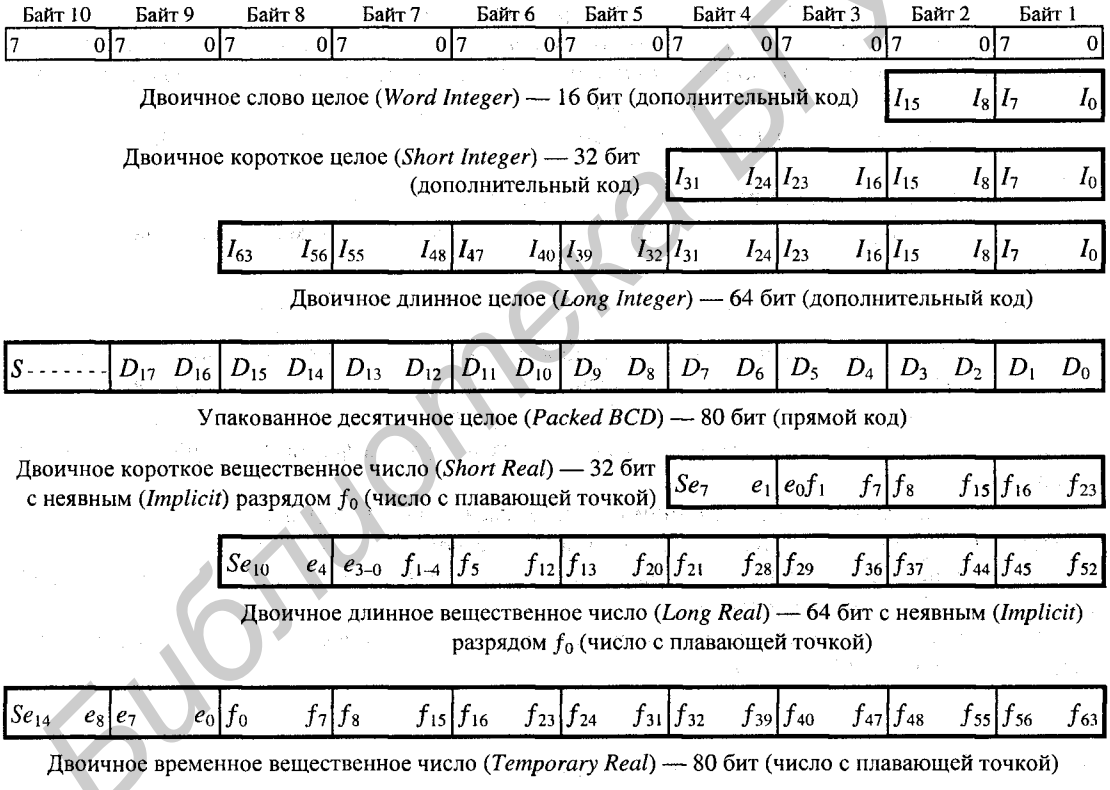


Рис. 4.29. Типы данных арифметического сопроцессора 8087

Форматы целых двоичных чисел отличаются только длиной (точностью — *Precision*) и, следовательно, диапазоном (*Range*) допустимых чисел. Только в этих форматах для представления чисел применяется *дополнительный код* (*Two's Complement*). Формат целого слова соответствует основному численному формату МП 8086. Наибольшие положительное и отрицательное (по модулю) числа в дополнительном коде имеют вид: 011 … 1 и 100 … 0.

Диапазоны допустимых целых чисел в трех форматах определяются значениями:

$-2^{15} \div +2^{15} - 1$, или $-3{,}2768 \cdot 10^4 \div +3{,}2767 \cdot 10^4$;

$-2^{31} \div +2^{31} - 1$, или $-2{,}1\ 4748\ 3648 \cdot 10^9 \div +2{,}1\ 4748\ 3647 \cdot 10^9$;

$-2^{63} \div +2^{63} - 1$, или $-9{,}22\ 3372\ 0368\ 5477\ 5808 \cdot 10^{18} \div +9{,}22\ 3372\ 0368\ 5477\ 5807 \cdot 10^{18}$.

В упакованном десятичном формате каждый байт содержит две десятичные цифры в коде 8–4–2–1. В этом формате десятичные целые числа представляются в *прямом коде*. Старший разряд старшего (левого) байта отведен для знака числа, а остальные разряды этого байта игнорируются, но при записи в память в них помещаются нули. Значение числа, представленного в упакованном десятичном формате, равно

$$(-1)^S(D_{17} \ldots D_2 D_1 D_0), \text{ причем } +0 = 00 \ldots 000, \ -0 = 10 \ldots 000.$$

При наличии в тетраде запрещенных комбинаций 1010 ÷ 1111 результат операции с десятичным операндом не определен — *NDCP* не контролирует правильность десятичных цифр.

Формат вещественных чисел содержит три поля: *S* (*Sign*) — *поле знака* (один бит), *E* (*Exponent*) — *поле порядка* и *F* (*Fraction*) — *поле мантиссы* (см. рис. 4.29). Значащие цифры числа находятся в поле мантиссы, поле порядка показывает фактическое положение двоичной точки в разрядах мантиссы, а разряд *S* определяет знак числа. Разрядность мантиссы определяет точность (*Precision*) представления чисел. Мантисса, называемая также *дробью*, представляется в *прямом коде*.

В сопроцессоре используется представление мантиссы в *нормализованной форме* — ее старший разряд f_0 равен 1. Следовательно, за исключением числа 0 мантисса представляет собой дробь

$$f_0.f_1f_2 \ldots f_n = 1.f_1f_2 \ldots f_n, \ \ f_i = 0 \text{ или } 1 \ \ (i = 1, 2, \ldots, n),$$

целая часть (f_0) которой равна 1. Благодаря нормализации устраняются старшие нули в числах, меньших 1, что увеличивает количество значащих цифр мантиссы при ее фиксированной длине. В коротком и длинном вещественных форматах разряд f_0 отсутствует — при передачах чисел и хранении их в памяти разряд f_0 используется в скрытой форме (f_0 *Implicit* — неявный разряд f_0).

Числа во временном вещественном формате имеют явный разряд f_0. Числа во временном вещественном формате называются также числами с *расширенной точностью*. Числа с плавающей точкой длиной 32 и 64 бита (короткое и длинное вещественные) называются числами с *одинарной* и *двойной точностью*.

Порядок *E* представляется в *смещенной форме* — он равен *истинному порядку exp*, увеличенному на значение *bias* (*смещение*):

$$E = exp + bias,$$

где $bias = 2^7 - 1 = 127$ для коротких вещественных чисел, $bias = 2^{10} - 1 = 1023$ для длинных вещественных чисел, $bias = 2^{14} - 1 = 16383$ для временных вещественных чисел.

Смещенный порядок *E* является целым положительным, или беззнаковым, числом и называется *характеристикой числа*. В сопроцессоре минимальное ($E = 000 \ldots 00$) и максимальное ($E = 111 \ldots 11$) значения смещенного порядка зарезервированы для некоторых специальных чисел (не для нормализованных чисел — см. табл. 4.27 на с. 471). Значение числа *N*, представленного в вещественных форматах, равно

$$(-1)^S \times 2^{E - bias} \times f_0 . f_1 f_2 \ldots f_n,$$

где $n = 23$, 52 и 63 для соответствующих форматов.

Точность представления чисел в вещественных форматах определяется разрядностью мантиссы F, а диапазон допустимых чисел — разрядностью порядка E:

$10^{\pm 38}$ (*Short Real*), $10^{\pm 308}$ (*Long Real*), $10^{\pm 4932}$ (*Temporary Real*) — диапазон допустимых чисел.

Пример 5. Представим десятичное число –973943.078125 в трех вещественных форматах. Его точное значение в двоичной системе счисления равно –11101101110001110111.000101, а истинный порядок $exp = +19$. Смещенный же порядок $E = exp + bias = 146$ (*Short Real*), 1042 (*Long Real*) и 16402 (*Temporary Real*). Из рис. 4.29 следует, что все вещественные форматы с учетом явного и неявного разряда f_0 имеют вид:

$$Se_k \dots e_1 e_0 f_0 f_1 \dots f_n, \quad +0 = 00 \dots 0000 \dots 0, \quad -0 = 10 \dots 0000 \dots 0,$$

где $n = 23$ и $k = 7$ для короткого формата, $n = 52$ и $k = 10$ для длинного формата, $n = 63$ и $k = 14$ для временного формата, S — знаковый разряд. Переведя порядок E в двоичную систему счисления, легко получить представление десятичного числа –973943.078125 в трех вещественных форматах (табл. 4.19). В коротком вещественном формате это число не может быть представлено точно — теряются два младших двоичных разряда. В 16-ричной системе счисления кодированные значения числа –973943.078125 будут равны C96D C771, C12D B8EE 2800 0000 и C012 EDC7 7140 0000 0000 для короткого, длинного и временного форматов соответственно.

Таблица 4.19. **Кодирование десятичного числа –973943.078125 в трех вещественных форматах**

Формат числа	S	Порядок E	Мантисса F
		e_{14} e_{10} e_7 e_0	f_0 f_1 f_{23} f_{52} f_{63}
1	1	10010010	– 11011011100011101110001
2	1	10000010010	– 11011011100011101110001010000000000 ... 00000000
3	1	100000000010010	1 11011011100011101110001010000000000 ... 00000000 00000000000

Примечание: 1 — *Short Real*, 2 — *Long Real*, 3 — *Temporary Real* (форматы числа).

Модуль числа $N = Se_k \dots e_1 e_0 f_0 f_1 \dots f_n$, находится в диапазоне

$$2^{-bias+1} \le |N| \le 2^{bias+1} \times (2 - 2^{-n}) \approx 2^{bias+2}, \; bias = 2^k - 1,$$

а значит

$$\left.\begin{array}{l} |N|_{min} = 2^{-126} = 1{,}17549435082228750796873653722222e{-}38 \\ |N|_{max} \approx 2^{+128} = 3{,}40282366920938463463374607431768e{+}38 \end{array}\right\} \textit{Short Real}\ (k = 7, n = 23);$$

$$\left.\begin{array}{l} |N|_{min} = 2^{-1022} = 2{,}22507385850720138309023271733e{-}308 \\ |N|_{max} \approx 2^{+1024} = 1{,}79769313486231590772930519078e{+}308 \end{array}\right\} \textit{Long Real}\ (k = 10, n = 52);$$

$$\left.\begin{array}{l} |N|_{min} = 2^{-16382} = 3{,}36210314311209350626267781732e{-}4932 \\ |N|_{max} \approx 2^{+16384} = 1{,}18973149535723176508575932663e{+}4932 \end{array}\right\} \textit{Temporary Real}\ (k = 14, n = 63).$$

Временный вещественный формат используется для представления чисел внутри *NDCP*, и все вычисления производятся в этом формате аппаратно с очень высокой точностью. Числа во временном вещественном формате можно записывать в память, что приходится делать для хранения промежуточных результатов из-за нехватки внутренних регистров (*NDCP* имеет всего восемь 80-разрядных регистров для хранения исходных операндов и результатов вычислений).

Независимо от исходного формата при загрузке числа из памяти в регистр сопроцессора оно автоматически преобразуется во временный вещественный формат, а при записи в память осуществляется обратное преобразование в формат получателя — любой из остальных шести форматов. Поэтому элементарные и быстрые для МП 8086 команды загрузки и запоминания двоичных и десятичных целых чисел выполняются сопроцессором достаточно долго. Благодаря аппаратному преобразованию всех форматов данных во временный вещественный формат программист может не заботиться об их преобразованиях. Конечно, при возвращении результата в память программист должен выбрать формат получателя, необходимый для восприятия результата с требуемой точностью. Выполнение вычислений с большей точностью (64-разрядная мантисса), чем точность форматов получателя (максимум 53-разрядная мантисса), обеспечивает для большинства достаточно сложных задач исключение накапливаемых ошибок округления промежуточных результатов.

Директива DB. Эта директива размещает и инициализирует один или более байтов данных в памяти. Директива DB (*Define Byte* — определение байта) имеет формат:

[*name*] DB *value_*0 [, *value_*1] ... [, *value_k*],

где *name* — необязательное имя переменной типа BYTE, с которым ассемблер связывает смещение (относительный адрес) байта данного *value_*0 в сегменте данных. С этого адреса в сегменте данных последовательно будут располагаться все заданные байты *value_m* ($m = 0 \dots k$). Аргументы *value_m* могут быть заданы одним из следующих способов:

целое число (например, 0E7*h*);
константное выражение (например, 4 * 18, где * — оператор умножения);
строковая константа (например, 'Pentium');
оператор DUP (например, 10 DUP (?) — резервирование 10 байт в памяти);
знак вопроса ? (указание ассемблеру оставить начальное значение неопределенным).

Символы строковой константы, заключенные в апострофы, транслируются в *ASCII*-коды в соответствии с табл. 1.9. Если в строковую константу входит апостроф, то его необходимо повторить два раза, например, 'Об''ект', что соответствует слову Об'ект. Строковая константа может быть любой длины, при условии, что она умещается вместе с директивой на одной строке текста программы (всего 128 символов). Символы строки входят в сегмент данных в порядке их следования, т. е. первый символ имеет меньший адрес, последний — больший.

Пример 6 (считаем, что данные расположены с начала сегмента данных, т. е. смещение переменной *alpha* равно 0000*h* — для уменьшения текста программы директивы определения сегментов опущены):

```
alpha db  95h, 93h, 'IBM  PC', ?, 10110b, ?, 36 ; 95 93 49 42 4D 20 50 43 00 16 00 24h (12 байт)
   db  4 * 18, 5 dup (?), 2 dup (55, 0AAh)     ; 48 00 00 00 00 00 37 AA 37 AA (10 байт)
                                               ;          (4 × 18 = 72d = 48h)
   MOV  DL, alpha + 4     ; DL ← M(alpha + 4) = M(0004) = 4Dh — ASCII-код символа M
; Здесь операнд alpha + 4 является адресом и используется режим прямой адресации
   MOV  BX, offset alpha + 4 ; BX ← 0004h — смещение (относительный адрес) символа M
   LEA  SI, alpha + 4     ; SI ← 0004h — смещение (относительный адрес) символа M
   MOV  AH, alpha + 12    ; AH ← M(alpha + 12) = M(000C) = 48h
   MOV  BX, alpha + 4     ; Ошибка: переменная M(alpha+4) типа BYTE, а не типа WORD
```

Транслятор знак "?" заменяет значением 00*h* и команду LEA командой MOV. Оператор OFFSET вычисляет смещение (относительный адрес) переменной *alpha*+4 относительно начала

сегмента данных, т. е. операнд offset *alpha*+4 является непосредственным операндом. На этом основании транслятор и заменяет команду LEA на команду MOV. В первых четырех командах вторые операнды можно заключать в квадратные скобки. Сказанное справедливо и для остальных директив определения данных за исключением допустимой длины строковой константы.

Оператор DUP. Этот оператор используется вместе с директивами DB, DW, DD, DQ и DT для задания нескольких одинаковых начальных значений. Оператор DUP (*Duplicate* — дублировать) имеет формат

count DUP (*value*_1 [, *value*_2] ... [, *value_k*]),

где *count* — аргумент (целое положительное число), задающий число раз, которое нужно продублировать начальные значения, перечисленные в круглых скобках. Каждая величина *value_m* может быть любым выражением, имеющим значением целое число, символьную константу, знак вопроса ? или другой оператор DUP (допускается до 17 уровней вложенности операторов DUP). При трансляции знак вопроса ? заменяется значением 00*h*.

Пример 7:

```
data1  db 4 dup ('var1')     ; 76 61 72 31 76 61 72 31 76 61 72 31 76 61 72 31 — данные после
                             ;     трансляции (76h 61h 72h 31h — ASCII-коды букв слова var1)
       db 8 dup (?)                ; 8 неинициализированных (неопределенных) байтов
data2  db 8 dup (30h, 31h)         ; 16 байтов, инициализированных значениями 30h 31h
data3  db 2 dup (4 dup (41h, 42h)) ; 16 байтов, инициализированных значениями 41h 42h
       db 3 dup (37*10/3)          ; 3 байта, инициализированных значением 7Bh = 123
```

Оператор OFFSET. Этот оператор (*Offset* — смещение) имеет формат

OFFSET *expression*

и вычисляет смещение выражения *expression* относительно начала сегмента (вычисляет относительный адрес). Выражение *expression* может быть переменной или меткой. Команда LEA выполняет аналогичные действия, но без использования оператора OFFSET.

Конструкцию offset *expression* можно использовать в инструкциях (командах) МП в качестве *непосредственного операнда.*

Директива DW. Эта директива размещает и инициализирует одно или более слов данных в памяти. Директива DW (*Define Word* — определение слова, или двух байт) имеет формат:

[*name*] DW *value*_0 [, *value*_1] ... [, *value_k*],

где *name* — необязательное имя переменной типа WORD, с которым ассемблер связывает смещение (относительный адрес) младшего байта данного *value*_0 в сегменте данных. С этого адреса в смежных ячейках памяти последовательно будут располагаться все заданные слова *value_m* (m = 0, 1, ..., k): в первую ячейку памяти помещается младший байт слова, а в следующую (с большим адресом) — старший байт слова. Аргументы *value_m* могут быть заданы одним из следующих способов:

целое число — *Word Integer* (например, 0F0E7*h*);
константное выражение — *Word Integer* (например, 318 / 65, где / — оператор деления);
адресное выражение (например, *near_array*);
строковая константа — только один или два символа (например, 'D' или 'PC');
оператор DUP (например, 7 DUP (?) — резервирование 7 × 2 = 14 байт в памяти);
знак вопроса ? (указание ассемблеру оставить начальное значение слова неопределенным).

Строковая константа типа WORD должна содержать не более двух символов. Последний (или единственный) символ строки сохраняется в младшем байте слова, а старший байт слова содержит первый символ или 0, если строковая константа односимвольная.

Пример 8 (считаем, что данные расположены с начала сегмента данных, т. е. смещение переменной *beta* равно 0000*h*):

```
beta    dw      8595, 1998h, ?, 0D4C1h, 65*318  ; 2193 1998 0000 D4C1 50BEh (5 слов)
        dw      'D', 'PC', ?, 3 dup (75)        ; 0044 5043 0000 004B 004B 004B (6 слов)
gam_b   LABEL   byte            ; Смещение переменной gam_b равно 0022d = 0016h
  ; Смещение переменной gam_b типа BYTE равно смещению gam_w
gam_w   dw      1, 2, 3, 4, 5, 6, gam_w, L1   ; 0001 0002 0003 0004 0005 0006 0016 ××××  (8 слов)
omega   dw      ?               ; Резервирование одного слова в памяти (смещение равно 0028h)
          ∴     ; gam_w — адресная переменная типа near, ×××× — адрес метки L1 типа near
        MOV     DX, beta + 6    ; DX ← M(beta + 6) = M(0006) = D4C1h
  ; Здесь операнд alpha + 6 является адресом и используется режим прямой адресации
        JMP     SHORT L1        ; SHORT  L1 — короткий адрес перехода
        MOV     BX, offset beta+6 ; BX ← 0006h — смещение операнда D4C1h
        LEA     SI, gam_w + 6   ; SI ← 001Ch — смещение операнда 0004h
L1:     MOV     AL, gam_b + 6   ; AL ← M(gam_b + 6) = M(001C) = 04h
        MOV     AL, gam_w + 6   ; Ошибка: переменная M(gam_w+6) типа word, а не типа byte
          ∴     ; Описание директивы LABEL и оператора PTR приведено ниже
        MOV     CH, byte PTR beta+2   ; CH ← M(beta + 2) = 98h — младший байт слова 1998h
        MOV     AH, byte PTR beta+3   ; AH ← M(beta + 3) = 19h — старший байт слова 1998h
        MOV     AH, byte PTR [beta+3] ; то же самое
        MOV     omega, AX             ; M(omega) = M(0026h) ← AX
        MOV     AX, omega – gam_b     ; AX ← 0010h
        MOV     omega, AL             ; Ошибка: Operand types do not match
        MOV     AX, omega + gam_b     ; Ошибка: Can't add relative quantities
                                      ;         (нельзя суммировать относительные величины)
```

Директива LABEL. Эта директива (*Label* — метка, переменная) имеет формат

name LABEL *type*

и создает новую переменную или метку *name* типа *type*, присваивая ей текущее значение счетчика адресов. Типом может быть BYTE, WORD, DWORD, QWORD, TBYTE (для переменных, определяющих операнды в памяти), NEAR или FAR (для меток).

Использование директивы LABEL для создания переменной *gamma_b* типа BYTE было приведено в *примере* 8 — переменные *gamma_b* и *gamma_w* относятся к одним и тем же данным (8 слов) и позволяют обращаться к ним побайтно и пословно.

Оператор PTR. Этот оператор (PTR — *Pointer* — указатель) имеет формат:

type PTR *expression*,

где *expression* — выражение, которое может быть переменной или меткой, *type* = BYTE, WORD, DWORD, QWORD, TBYTE (для переменных), NEAR или FAR (для меток). Как и директива LABEL, оператор PTR изменяет тип переменной или метки с ее стандартного типа на тип *type*, но только временно — для одной инструкции (команды) МП, в которой этот оператор использован.

Оператор PTR обычно применяется для доступа к переменной, имеющей тип, отличный от указанного при ее определении (используемый по умолчанию), например, для доступа к младшему или старшему байту переменной типа WORD. Оператор PTR также используется для явного описания типа переменной или метки, являющейся ссылкой вперед. Использование оператора PTR для пересылки младшего и старшего байт переменной типа WORD было приведено в *примере* 8.

Оператор SHORT. Этот оператор (*Short* — короткий) имеет формат

SHORT *label*

и устанавливает тип SHORT метки *label* (короткий переход — в пределах –128 ÷ +127 байт). Команды переходов JMP, в которых указан оператор SHORT на один байт короче, чем команды, использующие метки типа NEAR (по умолчанию). Использование оператора SHORT для указания короткого перехода было приведено в *примере* 8.

Директива DD. Эта директива размещает и инициализирует одно или более двойных слов данных в памяти. Директива DD (*Define Double Word* — определение двойного слова, или четырех байт) имеет формат:

[*name*] DD *value_*0 [, *value_*1] ... [, *value_k*],

где *name* — необязательное имя переменной типа DWORD, с которым ассемблер связывает смещение (относительный адрес) младшего байта данного *value_*0 в сегменте данных. Аргументы *value_m* могут быть заданы одним из следующих способов ($m = 0 \ldots k$):

целое число — *Short Integer* (например, 19363943*h*);

константное выражение — *Short Integer* (например, 465 * 377);

вещественное число — *Short Real* (например, –1500.5625 или –1500562.5*e*–3, где *e* = –3 — порядок числа; для положительных чисел и порядков знак "+" можно опускать);

кодированное вещественное число — *Short Real* (например, 80F00000*r*, где *r* — указатель вещественного числа в формате *NDCP*);

адресное выражение (например, *far_array*);

строковая константа — только один или два символа (например, 'D', 'PC');

оператор DUP (например, 10 DUP (?) — резервирование 10 × 4 = 40 байт в памяти);

знак вопроса ? (указание ассемблеру оставить начальное значение неопределенным).

Вещественные числа типа ×××.××× транслятор преобразует в формат *Short Real*. Кодированные вещественные числа типа ×××*r* транслятор проверяет на допустимые значения кодов.

Строковая константа типа DWORD должна содержать не более двух символов. Последний (или единственный) символ строки сохраняется в младшем байте младшего слова, а старший байт младшего слова содержит первый символ или 0, если строковая константа односимвольная. Остальные байты заполняются нулями.

Пример 9 (считаем, что данные расположены с начала сегмента данных, т. е. смещение переменной *alpha* равно 0000*h*):

```
alpha   dd      'D', 'PC'                   ; 44 00 00 00  ♦ 43 50 00 00
        dd      +1500.5625, -3827.653       ; 00 92 BB 44  ♦ 73 3A 6F C5 — Short Real
        dd      +1500562.5e-3, -38.27653e2  ; 00 92 BB 44  ♦ 73 3A 6F C5 — Short Real
        dd      80F00000r, 421AF2C9r        ; 00 00 F0 80  ♦ C9 F2 1A 42 — Short Real
beta_w  label   word                        ; Смещение переменной beta_w равно 0032d = 0020h
                ; Смещение переменной beta_w типа WORD равно смещению beta_d
beta_d  dd      2C1E61F7h, 4D33BB03h        ; F7 61 1E 2C  ♦ 03 BB 33 4D — Short Integer
```

```
buff    dd      2 dup (?)          ; Резервирование 2 × 4 = 8 байт памяти
gamma   dq      98465 * 523 – 377 / 3  ; 6E C8 11 03 — Short Integer (311C86Eh = 51497070d)
num_1   dd      007E4339r          Ошибка: Illegal number (недопустимое число)
num_2   dd      007E4339h          39 43 7E 00 — Short Integer
          ∴
        MOV     DX, beta_w + 3  ; DX ← M(beta_w + 3) = M(0023) = 032Ch
; Здесь операнд beta_w является адресом и используется режим прямой адресации
        MOV     BX, offset beta_w + 2  ; BX ← 0022h — смещение операнда 2C1Eh
        LES     BP, beta_d + 1  ; BP ← M(0021) = 1E61h, ES ← M(0023) = 032Ch
        MOV     AX, beta_d      ; Ошибка: переменная M(beta_d) типа dword, а не типа word
          ∴     ; Преобразование двоичного слова целого в короткое вещественное число
        FILD    beta_w    ; NDCP ← 61F7h — загрузка в NDCP слова
        FST     buff      ; M(buff) ← 46C3EE00h = 01000110110000111110111000000000 —
; знак S = 0, порядок E = exp + bias = 10001101 = 141d, exp = 141 – 127 = 14, мантисса
; F = 1.10000111110111000000000, т. е. в буфер buff записывается исходное число
; (1100001111101111.000000000 = 61F7h) в кодированном формате
          ∴     ; Вычисление квадратного корня из числа 1500.5625
        FLD     alpha + 8 ; NDCP ← 44BB9200 = 0100 0100101110111001001000000000 —
; знак S = 0, порядок E = exp + bias = 10001001 = 137d, exp = 137 – 127 = 10, мантисса
; F = 1.01110111001001000000000, т. е. в NDCP передается число
; 10111011100.1001000000000 = 1500.5625
        FSQRT   alpha + 8 ; Вычисление квадратного корня из числа 1500.5625
        FST     buff + 4  ; M(buff + 4) ← 421AF2C9h = 01000010000110101111001011001001 —
; знак S = 0, порядок E = exp + bias = 10000100 = 132d, exp = 132 – 127 = 5, мантисса
; F = 1.00110101111001011001001, т. е. в буфер buff записывается кодированное число, кото-
; рому соответствует двоичное число 100110.101111001011001001 = 38.737094879150390625
```

В комментариях к определению данных байты данных расположены в том же порядке, в каком они располагаются в *дампе* (*дамп* — данные, получаемые при загрузке памяти): первый байт (44*h*) по относительному адресу 0000*h*, а последний байт (4D*h*) по относительному адресу 002F*h*. Для удобства чтения четырехбайтовых операндов они разделены знаком ♦. В распечатках дампа значения байтов приводятся в 16-ричной системе счисления. Для получения численного значения операнда байты данных следует читать в обратном порядке — справа налево.

В табл. 4.20 приведены значения чисел 2^{-n} для перевода дробной части десятичных чисел в двоичный код.

Таблица 4.20. **Значения чисел 2^{-n}**

n	Значение 2^{-n}	*n*	Значение 2^{-n}	*n*	Значение 2^{-n}	*n*	Значение 2^{-n}
01	0.5	06	0.015625	11	0.00048828125	16	0.0000152587890625
02	0.25	07	0.0078125	12	0.000244140625	17	0.00000762939453125
03	0.125	08	0.00390625	13	0.0001220703125	18	0.000003814697265625
04	0.0625	09	0.001953125	14	0.00006103515625	19	0.0000019073486328125
05	0.03125	10	0.0009765625	15	0.000030517578125	20	0.00000095367431640625

Использованные в примере команды *NDCP* (F×××) описаны в § 4.7. Извлечение квадратного корня из десятичного числа 1500.5625, представленного в коротком вещественном форма-

те, дает число 38.737094879150390625 с точными значениями только шести разрядов после точки. Более точное значение

$$\sqrt{1500.5625} = 38{,}737094625178073620498949458 6506.$$

Директива DQ. *Эта директива размещает и инициализирует одно или более счетверенных слов (квадрослов) данных в памяти. Директива DQ (Define Quad Word — определение четырех* слов, или восьми байт) имеет формат:

[*name*] DQ *value_0* [, *value_1*] ... [, *value_k*],

где *name* — необязательное имя переменной типа QWORD, с которым ассемблер связывает смещение (относительный адрес) младшего байта данного *value_0* в сегменте данных. Аргументы *value_m* могут быть заданы одним из следующих способов ($m = 0 \ldots k$):

целое число — *Long Integer* (например, 0ABCD193639431938*h*);

константное выражение — *Long Integer* (например, 465 * 23 – 377 / 3);

вещественное число — *Long Real* (например, –1500.5625 или –15.005625*e*+2, где *e* = +2 — порядок числа; для положительных чисел и порядков знак "+" можно опускать);

кодированное вещественное число — *Long Real* (например, 80F0 000000000000*r*, где *r* — указатель числа в формате *NDCP*);

строковая константа — только один или два символа (например, 'D' или 'PC');

оператор *DUP* (например, *10h DUP (?)* — резервирование $16 \times 8 = 128$ байт в памяти);

знак вопроса ? (указание ассемблеру оставить начальное значение неопределенным).

Вещественные числа типа ×××.××× транслятор преобразует в формат *Long Real*. Кодированные вещественные числа типа ×××*r* транслятор проверяет на допустимые значения кодов.

Строковая константа типа QWORD должна содержать не более двух символов. Последний (или единственный) символ строки сохраняется в младшем байте младшего слова, а старший байт младшего слова содержит первый символ или 0, если строковая константа односимвольная. Остальные байты заполняются нулями.

Пример 10 (считаем, что данные расположены с начала сегмента данных, т. е. смещение переменной *alpha* равно 0000*h*):

```
alpha   dq    'D', 'PC'                ; 44 00 00 00 00 00 00 00 ♦ 43 50 00 00 00 00 00 00
        dq    +1500.5625, -3827.653    ; 00 00 00 00 40 72 97 40 ♦ 93 18 04 56 4E E7 AD C0
        dq    80100000 00000000r       ; 00 00 00 00 00 00 10 80 ← Long Real ↑
        dq    10000000000000A7r        ; A7 00 00 00 00 00 00 10 ← Long Real
beta_w  label word                     ; Смещение переменной beta_w типа word равно 0030h
beta_d  label dword                    ; Смещение переменной beta_d типа dword равно 0030h
  ; Смещение переменных beta_w и beta_d типов WORD и DWORD равно смещению beta_q
beta_q  dq    2C1E61F74D33BB03h        ; 03 BB 33 4D F7 61 1E 2C — Long Integer
buff    dq    ?   ; Резервирование 8 байт памяти
gamma   dq    98465 * 523 – 377/3      ; 6EC8110300000000 — Long Integer
          ∴
        MOV   DX, beta_w + 3 ; DX ← M(beta_w + 3) = M(0033) = F74Dh
          ; Здесь операнд beta_w является адресом и используется режим прямой адресации
        MOV   BX, offset beta_w + 2  ; BX ← 0032h — смещение операнда 4D33h
        LES   BP, beta_d + 1  ; BP ← M(0031) = 33BBh, ES ← M(0033) = F74Dh
          ∴   ; Вычисление квадратного корня из числа 1500.5625
```

```
        FLD   alpha + 16      ; NDCP ← M(alpha + 16) = M(0010) = 40977240 00000000h —
;  знак S = 0, порядок E = exp + bias = 10000001001 = 1033d, exp = 1033 – 1023 = 10d,
;  мантисса F = 1. 0111 0111 0010 0100 0000 ... 0, т. е. в NDCP передается число
;  10111011100.1001000000 ... 0 = 1500.5625
        FSQRT alpha + 8       ; Вычисление квадратного корня из числа 1500.5625
        FST   buff            ; M(buff) = M(0038) ← 40435E59 1DDE993Eh из NDCP —
;  знак S = 0, порядок E = exp + bias = 10000000100 = 1028d, т. е. exp = 1028 – 1023 = 5,
;  мантисса F = 1.00110101111001011001000111011011111010011001001111110, т. е. в буфер
;  buff записывается кодированное число, которому соответствует двоичное число
;  100110.10111100101100100011101110111101001100100111110 — ср. с примером 9
```

Директива DT. Эта директива размещает и инициализирует один или более 10-байтовых операндов данных в памяти. Директива DT (*Define Ten Byte* — определение 10 байт) имеет формат:

[*name*] DT *value_*0 [, *value_*1] ... [, *value_k*],

где *name* — необязательное имя переменной типа TBYTE, с которым ассемблер связывает смещение (относительный адрес) младшего байта данного *value_*0 в сегменте данных. Аргументы *value_m* могут быть заданы одним из следующих способов ($m = 0 \ldots k$):

целое число (например, 12345678901234567890*d*, 0C7623FE46055557777D0*h*);
упакованное десятичное число — *Packed BCD* (например, –123456789012345678);
вещественное число — *Temporary Real* (например, 1500.75);
кодированное вещественное число — *Temporary Real* (например, 80F0000000000000000*r*, где *r* — указатель вещественного числа в формате *NDCP*);
строковая константа — только один или два символа (например, 'D' или 'PC');
оператор DUP (например, 10*h* DUP (?) — резервирование 16 × 10 = 160 байт в памяти);
знак вопроса ? (указание ассемблеру оставить начальное значение неопределенным).

Вещественные числа типа ×××.××× транслятор преобразует в формат *Temporary Real*. Кодированные вещественные числа типа ×××*r* транслятор проверяет на допустимые значения кодов. Если в десятичной константе указатель системы счисления *d* отсутствует, то она транслируется в упакованное десятичное число (*Packed BCD*).

Строковая константа типа TBYTE должна содержать не более двух символов. Последний (или единственный) символ строки сохраняется в младшем байте младшего слова, а старший байт младшего слова содержит первый символ или 0, если строковая константа односимвольная. Остальные байты заполняются нулями.

Пример 11 (считаем, что данные расположены с начала сегмента данных, т. е. смещение переменной *alpha* равно 0000*h*):

```
alpha   dt   'D', 'PC'                ; 0000000000000000044h, 0000000000000005043h
        dt   +987654321123456789      ; 0098765432112345678 9h — Packed BCD
buff    dq   ?                        ; Резервирование 8 байт памяти
beta    dt   1500.5625                ; 4009BB92000000000000h — Temporary Real
num_1   dt   –987654321123456789      ; 8098765432112345678 9h — Packed BCD
num_2   dt   –987654321123456789h     ; FF6789ABCDEEDCBA9877h — дополнит. код
num_3   dt   98765432100123456789d    ; 00055AA54D369FD23515h
num_4   dt   2C1E61F74D33BB030075r    ; 2C1E61F74D33BB030075h
num_5   dt   2C1E61F74D33BB030075h    ; 2C1E61F74D33BB030075h
num_6   dt   0F1E61F74D33BB030075r    ; Ошибка: Illegal number (недопустимое число)
```

```
num_7 dt      0F1E61F74D33BB030075h  ; 0F1E61F74D33BB030075h
        ∴     ; Преобразование упакованного BCD-числа в длинное вещественное число
      FBLD  alpha + 20    ; NDCP ← M(alpha + 20) = M(0014) = 0098765432112345678 9
      FST   buff          ; M(buff) = M(001E) ← 43AB69B4 BE96E2EEh =
; = 0100001110101011011010011011010010111110100101101110001011101110 из NDCP
```

В комментариях указаны данные, получаемые после трансляции сегмента данных.

Порядок $E = exp + bias = 1082d$, т. е. истинный порядок $exp = 1082 - 1023 = 59d$, а мантисса F — 53-разрядная. Значит преобразование выполнено с потерей точности.

Директивы EQU и =. Эти директивы не определяют элемент данных в памяти, а лишь вводят символические имена для чисел и текста, с которыми удобнее оперировать, так как в имена всегда можно вложить содержательный смысл. Директива EQU (*Equate* — приравнять) имеет формат

name EQU *expression*

и создает *абсолютное имя* (16-разрядное двоичное число от 0 до 65535*d*), *алиас* (*alias* — псевдоним, альтернативное имя) или *текстовое имя* (строковая константа или текст) путем присваивания имени *name* указанного выражения *expression*. Абсолютные имена используются в командах МП для задания *непосредственных операндов*. Каждое вхождение имени в исходном модуле ассемблер замещает значением выражения — числом, алиасом или текстом. Имя, как всегда, должно быть уникальным и не может быть переопределено. Выражение *expression* может быть задано одним из следующих способов:

целое число (например, 3943*h*) — создается абсолютное имя;
вещественное число (например, –1500.565 или –150056.5e–3) — создается текстовое имя;
константное выражение (например, 465 * 37) — создается абсолютное имя;
адресное выражение (например, *near_array*) — создается алиас;
мнемоника команды (MOV BP, 100*h*) — создается текстовое имя;
строковая константа (например, 'abCD') — создается текстовое имя;
текст (например, word ptr; текст не должен начинаться с цифры) — создается текстовое имя.

Директива = (приравнять) имеет формат

name = *expression*

и создает *абсолютное имя* (16-разрядное двоичное число от 0 до 65535*d*) путем присваивания имени *name* вычисленного значения выражения *expression*. Каждое вхождение имени в исходном модуле ассемблер замещает значением выражения — числом. Имя должно быть уникальным и может быть переопределено в любом месте программы. Выражение *expression* может быть задано одним из следующих способов:

целое число (например, 3943*h*);
константное выражение (например, 465 * 37);
адресное выражение (например, *near_array*);
строковая константа из одного или двух символов (например, 'ТЯ').

Пример 12 (считаем, что данные расположены с начала сегмента данных, т. е. смещение переменной *alpha* равно 0000*h*):

```
string  equ   'AbCd'              ; string — текстовое имя (строковая константа)
real    equ   –1500.5625          ; real = –1500.5625 — текстовое имя (вещественное число)
alpha   db    39h, 43h, 0C5h, 77, string  ; 39 43 C5 4D 41 62 43 64
```

```
omega   dw    65000, 0E8A2h     ; E8 FD ♦ A2 E8   (65000d = FDE8h)
short_r dd    real              ; 00 92 BB C4
long_r  dq    real              ; 00 00 00 00 40 72 97 C0
buff    dd    ?                 ; Резервирование 4 байт памяти
sizeBF  =     $ – buff          ; sizeBF = 0004h, $ — текущее значение счетчика адресов
beta    =     3943h             ; beta = 3943h — абсолютное имя
prod    =     465 * 37          ; prod = 17205d = 4335h — абсолютное имя
omg     =     omega             ; omg = omega — алиас
cop     equ   MOV BP, 100h      ; cop = MOV BP, 100h — текстовое имя
        MOV   BX, beta          ; BX ← 3943h
        MOV   SI, omg + 2       ; SI ← M(omg + 2) = M(000A) = E8A2h
        MOV   CX, prod          ; CX ← 4335h
        MOV   DL, LOW prod      ; DL ← 35h — передача младшего байта слова 4335h
        MOV   AL, HIGH prod     ; AL ← 43h — передача старшего байта слова 4335h
ptrw    equ   word ptr          ; ptrw = word ptr — текстовое имя
        MOV   ptrw buff, beta   ; M(buff) = M(0018) ← 3943h
        cop                     ; Имя cop транслируется в машинные коды команды MOV BP,100h
omg     =     0DAh              ; omg = 00DAh — абсолютное имя
        MOV   BH, omg           ; BH ← DAh — байт
        MOV   AX, omg           ; AX ← 00DAh — слово
        MOV   word ptr buff, omg ; M(buff) ← 00DAh — слово
        MOV   byte ptr buff + 1, omg ; M(buff + 1) ← DAh — байт
omg     =     'ТЯ'              ; omg = 929Fh , 92h и 9Fh — ASCII-коды букв Т и Я
sizeLR  =     buff – long_r     ; sizeLR = 0008h — число байт в переменной long_r
```

Если какая-либо константа используется в программе большое число раз, а затем потребуется изменить ее значение, то это значительно проще сделать одной директивой EQU, чем изменять числа по всей программе. В последнем случае велика также вероятность ошибки при внесении в программу изменений (в больших программах некоторые константы могут использоваться десятки раз).

Операторы HIGH и LOW. Эти операторы имеют форматы:

HIGH *expression* (выделение старшего байта слова *expression*),
LOW *expression* (выделение младшего байта слова *expression*).

Использование операторов HIGH и LOW было приведено в *примере* 12.

Пример 13:

```
        MOV   DI, low 8973/5    ; DI ← 0002h (8973/5 = 1794d = 702h)
        MOV   BL, high 8973/5   ; BL ← 07h
```

Порядок старшинства операторов. Порядок старшинства операторов определяет последовательность, в соответствии с которой будет вычисляться выражение. Операторы, имеющие большее старшинство, будут выполняться раньше операторов, имеющих меньшее старшинство. В табл. 4.21 операторы расположены в порядке уменьшения старшинства. Оператор ( ) — круглые скобки — изменяет порядок старшинства других операторов: часть выражения, заключенная в круглые скобки выполняется первой.

Операторы, стоящие в одной строке, имеют одинаковое старшинство и последовательно выполняются в порядке слева направо при их использовании в одном выражении. Круглые

скобки можно использовать несколько раз для выделения необходимого числа частей выражения.

Форматы и применение операторов № 4 ÷ № 10 ассемблера фирмы *Avocet Systems, Inc* для МП 8080/8085 были подробно рассмотрены в § 1.8 (с. 66 – 68). Эти операторы имеют такое же назначение и в ассемблере *Turbo Assembler Version* 3.2 фирмы *Borland International* для МП 8086/8088 (за исключением сообщений об ошибках).

Таблица 4.21. **Порядок старшинства операторов**

№	Операторы	Примечание
01	LENGTH, SIZE, WIDTH, MASK, ( ), [ ], < >	Операторы типа переменной и др.
02	PTR, OFFSET, SEG, TYPE, THIS	Операторы типа переменной
03	HIGH, LOW	Операторы типа переменной
04	+ (*unary*), – (*unary*)	Арифметические операторы
05	*, /, MOD, SHL, SHR	Арифметич. операторы и операторы сдвига
06	+ (сложение), – (вычитание)	Арифметические операторы
07	EQ, NE, LT, LE, GT, GE	Операторы отношений
08	NOT	Логический оператор
09	AND	Логический оператор
10	OR, XOR	Логические операторы
11	SHORT, .TYPE	Операторы типа переменной

Пример 14:

```
beta  equ   5Eh                ; beta = 5Eh
      MOV   BL, high 8973/5    ; BL ← 07h (8973/5 = 1794d = 0702h)
      MOV   BL, low high 8973/5 ; то же самое
      MOV   DX, –5E63h         ; DX ← A19Dh — дополнительный код числа –5E63h
      MOV   AX, 177 mod 9/2    ; AX ← 0003h = 177 mod 9/2 = (177 mod 9)/2 = 6/2 = 03
      MOV   CH, 177 mod (9/2)  ; CH ← 01h, так как 177 mod (9/2) = 177 mod 4 = 01
      MOV   CL, 16h shl 3      ; CL ← B0h = 1011 0000 (16h = 0001 0110)
      AND   CL, 0D5h shr 2     ; CL ← CL & 35h = B0h & 35h = 0011 0000 = 30h
      MOV   BX, 5E63h xor 7B84h ; BX ← 5E63h ⊕ 7B84h = 25E7h
      MOV   AL, beta ne 5Eh    ; AL ← ▶beta ne 5Eh◀ = 00h (false)
      MOV   AH, beta eq 5Eh    ; AH ← ▶beta eq 5Eh◀ = FFh (true)
      MOV   SI, beta ne 5Eh    ; SI ← ▶beta ne 5Eh◀ = 0000h (false)
      MOV   DI, beta eq 5Eh    ; DI ← ▶beta eq 5Eh◀ = FFFFh (true)
```

Здесь символами ▶ … ◀ обозначены значения, присваиваемые выражениям транслятором. Значению *true* (*истина*) соответствует число FFFFh (дополнительный код числа –1), а значению *false* (*ложь*) — число 0000h. Операторы отношений используются в условных директивах, управляющих трансляцией программы, для проверки условия, определяющего, какой из двух следующих за директивой блоков программы следует транслировать. В командах МП использовать операторы отношений в качестве операндов нет необходимости, но если это по каким-либо причинам сделано, то следует знать, что логическим значениям *true* и *false* при представлении их байтами/словами будут соответствовать числа FFh/FFFFh и 00h/0000h.

Использование операторов в ассемблерных программах позволяет многие вычисления переложить на транслятор и тем самым значительно уменьшить как объем памяти, занимаемый программой, так и время выполнения программы. В частности, перевод чисел в дополнительный код очень просто выполняется оператором унарный минус.

Оператор THIS. Этот оператор имеет формат

THIS *type*,

где *type* = BYTE, WORD, DWORD, QWORD, TBYTE, NEAR или FAR. Оператор THIS создает операнд типа *type*, для которого адрес (сегмент и смещение) равен текущему значению счетчика адресов.

Оператор THIS, как правило, используется с директивами EQU и = для создания меток и переменных (как и директива LABEL). Употребление оператора THIS приведено в *примере* 15.

Операторы TYPE, LENGTH, SIZE и SEG. Эти операторы имеют форматы:

TYPE *expression*,
LENGTH *variable*,
SIZE *variable*,
SEG *expression*.

Оператор TYPE возвращает число байтов, необходимых для хранения переменной того типа, каким является выражение *expression*: 1 — для типа BYTE, 2 — для типа WORD, 4 — для типа DWORD, 8 — для типа QWORD, 10 — для типа TBYTE. Для меток возвращает числа FFFF*h* (для метки типа NEAR) и FFFE*h* (для метки типа FAR).

Оператор LENGTH возвращает число, предшествующее оператору DUP в описании переменной (вложенные операторы DUP в расчет не принимаются), а для остальных переменных возвращает число 1.

Оператор SIZE возвращает число байтов в переменной, равное произведению значений LENGTH и TYPE.

Оператор SEG возвращает адрес сегмента, в котором определено выражение *expression*.

Выражение *expression* может быть переменной, меткой, именем сегмента или группы и любым другим символьным именем. Используется этот оператор в программах, состоящих из нескольких отдельно ассемблируемых сегментов.

Пример 15:

```
d_seg   segment                          ; Сегмент данных
alpha   dw      8595, 1998h, 7, 0D4C1h   ;  93 21 ♦ 98 19 ♦ 07 00 ♦ C1 D4
        dw      0E4D1h, 1234h, 2 dup (?) ;  D1 E4 ♦ 34 12 ♦ 00 00 ♦ 00 00
beta    equ     this word    ; d_seg : 0010h — смещение переменной beta типа word
L3      =       this word    ; d_seg : 0010h — смещение переменной L3 типа word
gamma   dd      12345678h, 87654321h     ;  78 56 34 12 ♦ 21 43 65 87
buff    dw      4 dup(2 dup (3 dup(?)))  ; Резервирование 24 слов памяти
d_seg   ends                             ; Конец сегмента данных
c_seg   segment                          ; Сегмент кода
        assume  CS: c_seg, DS: d_seg
main    proc    far                      ; Начало процедуры main типа far
        LES   SP, gamma      ; SP ← M(gamma) = 5678h, ES ← M(gamma + 2) = 1234h
        MOV   AX, beta       ; AX ← M(beta) = 5678h
        JMP   short L1       ; Переход типа short (короткий) — метка L1 однобайтовая
```

```
        JMP  L2                  ; Переход типа near (близкий) — метка L2 двухбайтовая
        MOV  CL, length gamma    ; CL ← 0001h
        MOV  CH, length buff     ; CH ← 0004h
L1:     MOV  BL, size gamma      ; BL ← 04h
        MOV  SI, size buff       ; SI ← 0008h
        MOV  buff, size d_seg    ; M(buff) ← 0048h = 72d — число байт в сегменте данных
        MOV  buff + 2, size c_seg ; M(buff + 2) ← 002Dh = 45d — число байт в сегменте кода
L2:     MOV  buff + 4, type alpha ; M(buff + 4) ← 0002h          ;  до данной команды
        MOV  buff + 8, type gamma ; M(buff + 8) ← 0004h
        MOV  buff + 10, type buff ; M(buff + 10) ← 0002h
L4      =    this near   ; c_seg : ××××  — смещение переменной L3 типа near в сегменте кода
    ┌─→ MOV  buff + 12, type L1  ; M(buff + 12) ← FFFFh
    │   MOV  buff + 14, type L2  ; M(buff + 14) ← FFFFh
    │   MOV  buff + 16, type L3  ; M(buff + 16) ← 0002h
    │   MOV  DX, type main       ; DX ← FFFEh
    │   MOV  buff + 18, seg gamma ; M(buff + 18) ← адрес сегмента данных
    │   JMP  L3 + 1              ; IP ← M(L3 + 1) = M(0011) = 3456h — переход типа near
    │   JMP  beta + 1            ; IP ← M(beta + 1) = M(0011) = 3456h — переход типа near
    │   JMP  gamma               ; IP ← M(gamma) = 5678h, CS ← M(gamma + 2) = 1234h
    └──JMP  L4                   ; (переход типа far)
        ∴
```

3. *Условные директивы*

Условные директивы IF××××, ELSE××××, ELSE и ENDIF используются для управления условным ассемблированием блоков операторов — трансляция блока зависит от проверки заданного условия (×××× — некоторая мнемоника, определяющая тип проверяемого директивой условия; сами же условия задаются дополнительными выражениями, которые могут быть оценены во время ассемблирования).

Имеется шесть пар взаимосвязанных условных директив IF×××× и ELSE××××:

IF *expression*, ELSEIF *expression* — условие выполняется, если *expression* = *true* (не равно 0; см. *пример* 16),

IFE *expression*, ELSEIFE *expression* — условие выполняется, если *expression* = *false* (равно 0); выражение *expression* должно иметь абсолютное значение, и не содержать ссылок вперед;

IF1 и ELSEIF1 — условие выполняется, если производится первый проход ассемблирования,

IF2 и ELSEIF2 — условие выполняется, если производится второй проход ассемблирования; эти директивы проверяют только номер прохода (первый или второй), поэтому они не имеют выражения;

IFDEF *name* и ELSEIFDEF *name* — условие выполняется, если имя *name* было предварительно определено,

IFNDEF *name* и ELSEIFNDEF *name* — условие выполняется, если имя (*name* — переменная, метка, символьное имя) еще не было определено; если имя определено как ссылка вперед, то IFDEF ложно на первом проходе и истинно на втором, а IFNDEF истинно на первом проходе и ложно на втором;

IFB <*argument*> и ELSEIFB <*argument*> — условие выполняется, если аргумент *argument* отсутствует,

IFNB <*argument*> и ELSEIFNB <*argument*> — условие выполняется, если аргумент *argument* имеется; эти директивы используются обычно для условного ассемблирования макросов, в зависимости от наличия или отсутствия параметров в макровызове; аргументом может быть любое имя, число или выражение (аргумент необходимо заключать в угловые скобки **< >**);

IFIDN <*argument*1>,<*argument*2> и ELSEIFIDN <*argument*1>,<*argument*2> — условие выполняется, если аргументы идентичны,

IFDIF <*argument*1>,<*argument*2> и ELSEIFDIF <*argument*1>,<*argument*2> — условие выполняется, если аргументы 1 и 2 различны; аргументы, которыми могут быть имена, числа и выражения, сравниваются посимвольно (при этом учитывается и различие строчных и прописных букв); аргументы необходимо заключать в угловые скобки **< >**; эти директивы необходимы для тестирования параметров, передаваемых макроопределению;

IFIDNI <*argument*1>,<*argument*2> и ELSEIFIDNI <*argument*1>,<*argument*2> — условие выполняется, если аргументы идентичны,

IFDIFI <*argument*1>,<*argument*2> и ELSEIFDIFI <*argument*1>,<*argument*2> — условие выполняется, если аргументы 1 и 2 различны; аргументы, которыми могут быть имена, числа и выражения, сравниваются посимвольно (при этом различие строчных и прописных букв не учитывается); аргументы необходимо заключать в угловые скобки **< >**; эти директивы необходимы для тестирования параметров, передаваемых макроопределению.

Оформление блоков условного ассемблирования имеет вид (директивы ELSE×××× и ELSE необязательны):

```
IF××××      ; Блок операторов ① транслируется, если условие, заключенное в директиве
  ∴①        ;     IF×××× выполняется (тогда остальные блоки не транслируются)
[ELSE××××   ; Блок операторов ② транслируется, если условие, заключенное в директиве
  ∴         ;     IF×××× не выполняется, а условие, заключенное в директиве ELSE××××
  ∴②]       ;     выполняется
[ELSE××××   ; Блок операторов ③ транслируется, если условия, заключенные в первых двух
  ∴         ;     директивах не выполняются, а условие, заключенное в данной директиве
  ∴③]       ;     ELSE×××× выполняется
[ELSE       ; Блок операторов ④ транслируется, если условия, заключенные в предыдущих
  ∴④]       ;     директивах не выполняются
ENDIF       ; Конец условной директивы
```

Всегда транслируется только один блок операторов — первый блок от начала, у которого условие, заключенное в директиве выполняется. Число директив ELSE×××× не ограничено. Если ни одно из условий не выполняется, то транслируется последний блок. Конец любой условной директивы задается директивой ENDIF.

Можно использовать вложенные условные директивы (до 255 уровней вложенности). Вложенная директива ELSE соответствует ближайшей из директив IF××××, у которой нет директивы ELSE. Условные директивы часто используются в макрокомандах. Примеры условных директив см. в файле *if_else*.asm на прилагаемой к учебному пособию дискете.

4. *Директивы макрокоманд*

Директивы макрокоманд предоставляют мощное средство для облегчения труда программиста, связанного с вводом текста исходного модуля программы, и делают сложные программы на языке ассемблера более понятными. Группу директив макрокоманд составляют восемь директив ассемблера:

MACRO, LOCAL, EXITM, ENDM, PURGE и REPT, IRP, IRPC

Директива MACRO позволяет повторяющимся блокам программы, различающимися только некоторыми параметрами, присвоить имя с перечнем формальных параметров (*dummy or formal parameter*), а затем любое число раз использовать только имя с указанием фактических параметров (*actual parameter*) вместо повторения текста всего блока в разных местах программы. Транслятор же вместо имен блоков подставит все команды МП, входящие в этот блок, и введет фактические параметры — транслятор по имени блока создаст его макрорасширения (*Macro Expansion*). Определенные директивой MACRO *блоки программы* называются *макрокомандами* (*Macro Command, Macro Instruction*), или *макросами* (*Macros*).

Оформление макроопределений имеет вид:

```
Name_M  MACRO   Par1, Par2, Par3, Par4, ...   ; Директива MACRO (Macro Definition)
        LOCAL   Name1, Name2, Name3, ...     ; Директива LOCAL (Definition Local Name)
          ∴①            ; ① — операторы ассемблера (инструкции и директивы)
        [EXITM]         ; Директива прекращения макрогенерации (Exit Macro Generation)
          ∴②            ; ② — операторы ассемблера (инструкции и директивы)
        ENDM            ; Директива конца макроопределения (End Macro Definition)
          ∴             ; Основная программа
        Name_M  Mg11, Mg12, Mg13, ...        ; Макрорасширение 1 (Macro Generation)
          ∴             ; Основная программа
        Name_M  Mg21, Mg22, Mg23, ...        ; Макрорасширение 2 (Macro Expansion 2)
        [PURGE  Name_M]                 ; Удаление макроопределения Name_M из памяти
```

Директива MACRO связывает с именем макрокоманды *Name_M* некоторое число формальных параметров *Par*1, *Par*2, *Par*3, … . Вызов макрокоманды производится строкой, содержащей имя макрокоманды *Name_M* и такое же число фактических параметров (допустимое число параметров ограничено только длиной строки операторов ассемблера, равной 120 символам). Транслятор имеет генератор макрорасширений, который формальные параметры *Par*# (# = 1, 2, 3, …) заменяет фактическими параметрами *Mg*1# для макрорасширения 1 и *Mg*2# для макрорасширения 2. Соответствие формальных и фактических параметров определяется последовательностью их задания в макроопределении и макрорасширении. Можно использовать вложенные макрокоманды с любым числом уровней. Макрокоманды могут вызывать другие макросы.

Директива LOCAL объявляет перечисленные в ней имена переменных и констант действительными только внутри макроопределения, а генератор макрорасширений заменяет их уникальными именами ??××××, где ×××× = 0000*h* ÷ FFFF*h* — 4-разрядные 16-ричные числа. Это исключает появление в программе одинаковых имен, имеющих разные численные значения.

Директива EXITM немедленно прекращает генерацию транслятором макрорасширения, если требуется получить только его часть, и возвращает управление оператору, следующему за оператором вызова макроса. При этом в первой части макрокоманды не должно быть ссылок на имена, определенные во второй ее части. Эта директива обычно используется совместно с ус-

ловными директивами для досрочного выхода из макроопределения. Директиву EXITM можно включать в блоки, генерируемые директивами повторения REPT, IRP и IRPC. Если директива EXITM находится во вложенных макросах или во вложенных директивах повторения REPT, IRP и IRPC, то управление возвращается в блок внешнего уровня.

Директива ENDM указывает транслятору конец макрокоманды и конец действия локальных имен. Эта же директива задает конец директив повторения REPT, IRP и IRPC.

Директива PURGE *Name_M*1 [, *Name_M*2, …] (*purge* — очищать, удалять, освобождать) удаляет макроопределения *Name_M*1, *Name_M*2, … из памяти. Директива PURGE используется при ассемблировании больших программ при недостаточном объеме оперативной памяти. Директивой PURGE можно удалять ненужные макросы из библиотеки. При вызове удаленного командой PURGE макроса транслятор выдает сообщение об ошибке (для *примера* 16):

Error 4*x*04_16.asm(42) Illegal instruction.

Пример 16 (файл 4*x*04_16.asm в основном соответствует *примеру* 10 из § 1.8, с. 76):

```
d_seg   segment                 ; Сегмент данных
Alpha   equ     39h             ; Alpha = 39h
Beta    db      2Ch, 'IBM PC'   ; M(Beta) = 2Ch, M(Beta+3) = 4Dh — ASCII-код буквы M
d_seg   ends
c_seg   segment                 ; Сегмент кода
        assume  CS: c_seg, DS: d_seg
main:   mov     ax, d_seg       ; Инициализация сегментного регистра DS
        mov     ds, ax
Add3    macro   Par1, Par2, Par3 ; Macro Definition Add3
        local   LJ1, LJ2, Beta  ; Definition Local Name LJ1, LJ2, Beta
;   Register CX = [Par1 – Par2 – Beta + M(Par3)]д — дополнительный код результата
Beta    equ     5               ; Beta = 5
        MOV     CH, 0           ; CH ← 0
        MOV     CL, Par1        ; CL ← Par1
        SUB     CL, Par2 + Beta ; CL ← Par1 – Par2 – Beta
        JNC     short LJ1
        DEC     CH              ; CH ← CH – 1, если CY = 1
LJ1:    ADD     CL, Par3        ; CL ← Par1 – Par2 – Beta + M(Par3)
        JNC     short LJ2
        INC     CH              ; CH ← CH + 1, если CY = 1
LJ2:    ; Условная директива IF и вложенная директива повторения IRP
        if      Omega           ; Условная директива
        exitm                   ; Выход из макроса, если Omega = true
        endif                   ; Конец условной директивы
        irp     regs, <AX, CX, SI, DS>  ; Ассемблирование параметров <AX, CX, SI, DS>
        PUSH    regs
        endm                    ; Конец ассемблирования параметров
        endm                    ; End Macros — CX = [Par1 – Par2 – Beta + M(Par3)]д
Gamma   equ     9Eh             ; Gamma = 9Eh
; Macro Expansion 1 (Macros Add3): Par1 = Alpha + Ah, Par2 = Gamma, Par3 = Beta
Omega   =       1               ; Omega = true
        Add3    Alpha + Ah, Gamma, Beta
```

```
; Macro Expansion 2 (Macros Add3): Par1 = 55h, Par2 = Alpha, Par3 = Beta +3
Omega   =       0               ; Omega = false
        Add3    55h, Alpha, Beta + 3
        purge   Add3            ; Удаление макроопределения Add3 из памяти
c_seg   ends                    ; Конец сегмента кода
        end     main            ; Конец программы
```

Имя *Beta* используется в программе два раза: в основной программе оно определено как адрес числа 2C*h*, а в макрокоманде — как байт-константа, но конфликта между ними не возникает, так как имя *Beta* в макрокоманде объявлено локальным — ее действие не выходит за пределы макрокоманды.

После трансляции листинг (только для макрорасширений) с исключенными столбцами адресов и машинных кодов будет иметь вид (транслятор заменяет формальные параметры на фактические не только в программе, но и в комментарии):

```
          ∴
Gamma   equ     9Eh             ; Gamma = 9Eh
; Macro Expansion 1 (Macros Add3): Par1 = Alpha + 0Ah, Par2 = Gamma, Par3 = Beta
Omega   =       1               ; Omega = true
        Add3    Alpha + Ah, Gamma, Beta
        MOV     CH, 0           ; CH ← 0
        MOV     CL, Alpha + 0Ah ; CL ← Alpha + 0Ah
        SUB     CL, Gamma + ??0002 ; CL ← Alpha + 0Ah – Gamma – ??0002
        JNC     short ??0000
        DEC     CH              ; CH ← CH – 1, если CY = 1
??0000: ADD     CL, Beta        ; CL ← Alpha + 0Ah – Gamma – ??0002 + M(Beta)
        JNC     short ??0001
        INC     CH              ; CH ← CH + 1, если CY = 1
??0001:         ; Условная директива IF и вложенная директива повторения IRP
; Macro Expansion 2 (Macros Add3): Par1 = 55h, Par2 = Alpha, Par3 = Beta +3
Omega   =       0               ; Omega = false
        Add3    55h, Alpha, Beta + 3
        MOV     CH, 0           ; CH ← 0
        MOV     CL, 55h         ; CL ← 55h
        SUB     CL, Alpha + ??0005 ; CL ← 55h – Alpha – ??0005
        JNC     short ??0003
        DEC     CH              ; CH ← CH – 1, если CY = 1
??0003: ADD     CL, Beta + 3    ; CL ← 55h – Alpha – ??0005 + M(Beta + 3)
        JNC     short ??0004
        INC     CH              ; CH ← CH + 1, если CY = 1
??0004:         ; Условная директива IF и вложенная директива повторения IRP
        PUSH    AX
        PUSH    CX
        PUSH    SI
        PUSH    DS
        purge   Add3            ; Удаление макроопределения Add3 из памяти
          ∴
```

Директивы повторения REPT, IRP и IRPC имеют формат:

```
REPT    expression          ; Начало директивы повторения REPT
  ∴①                        ; Блок ① — операторы ассемблера (инструкции и директивы)
ENDM                        ; Конец директивы повторения REPT
  ∴∴
IRP     par,<arg1[,arg2]...> ; Начало директивы повторения IRP
  ∴②                        ; Блок ② — операторы ассемблера (инструкции и директивы)
ENDM                        ; Конец директивы повторения IRP
  ∴∴
IRP     parameter, string   ; Начало директивы повторения IRPC
  ∴③                        ; Блок ③ — операторы ассемблера (инструкции и директивы)
ENDM                        ; Конец директивы повторения IRPC
```

Директива REPT *expression* повторяет блок ① операторов ассемблера (инструкций и директив) столько раз, сколько задает выражение *expression*. Выражение не должно содержать внешних или неопределенных имен и должно давать 16-разрядное целое число без знака.

Директива IRP *parameter*, <*argument*1 [, *argument*2] ... > повторяет блок ② операторов ассемблера (инструкций и директив) по одному разу для всех значений параметра *parameter*, перечисленных в угловых скобках <*argument*1 [, *argument*2] ... >, заменяя на каждой итерации все вхождения *parameter* в блок ② операторов на текущее его значение (см. *пример* 16). Может быть задано произвольное число аргументов (*argument*) и параметр *parameter* может встречаться в блоке ② любое число раз. Если список аргументов < ... > пуст, то операторы не ассемблируются. Список аргументов должен быть заключен в угловые скобки (аргументы в списке разделяются запятыми). Аргументом может быть любой разрешенный символ, строка, текст, числовая или символьная константа.

Директива IRPC *parameter*, *string* повторяет блок ③ операторов ассемблера (инструкций и директив) по одному разу для каждого символа строки *string*, заменяя на каждой итерации все вхождения параметра *parameter* на текущий символ строки *string*. Параметр *parameter* может встречаться в теле блока любое число раз. Строка может состоять из букв, цифр и других символов.

Пример 17 (в основном соответствует *примеру* 18 из § 1.8, с. 88):

```
d_seg  segment          ; Сегмент данных
; 1. Директива повторения REPT expression — генерация таблицы кода Грея
Tgray  label   byte     ; Tgray = 0000h — начальный адрес таблицы кода Грея (см. табл. 1.14)
gray   =       -1       ; gray = -1
       rept    256      ; 256 — число повторений трансляции блока операторов
gray   =       gray + 1 ; gray = 00h ... FFh      Генерация кода Грея (см. рис. 1.26) ↓
       db      gray xor gray shr 1 ;              D7D6D5D4D3D2D1D0 ⊕ 0 D7D6D5D4D3D2D1
       endm             ; Конец директивы повторения REPT
; 2. Директива повторения IRPC parameter, string
P      equ     1        ; P = 01h          резервируется (не нужно использовать директиву DS)
S      equ     0FFh     ; S = FFh
N      equ     0        ; N = 00h
Sym    equ     this byte ; Sym = 0100h — начальный адрес массива генерируемых данных
       irpc    Sym, PnPpppsssPpnPppsssPpsPsP      ; 24 символа в строке
       db      Sym      ; Запись в память 24 байт (чисел 0, 1 и FFh), начиная с адреса 0100h
       endm             ; Конец директивы повторения IRPC
```

```
; 3. Директива повторения IRP par, <arg1 [, arg2, ... ]>
Par     label   byte    ; Par1 = 0118h — начальный адрес массива генерируемых данных
Beta    equ     55h     ; Beta = 55h
        irp     Par,  <P, p, Beta, S, s, Beta, N, n>  ; В память, начиная с адреса 0118h,
        db      Par     ;                 записываются 8 байт (числа 01h, 00h, 55h и FFh)
        endm            ; Конец директивы повторения IRP
d_seg   ends
        end
```

В табл. 4.22 приведен указатель директив, операторов и атрибутов языков ассемблера для МП 8080/8085 и 8086/8088, рассмотренных в данном учебном пособии и позволяющий производить быстрое отыскание их описания.

Таблица 4.22. **Указатель директив и операторов ассемблеров *AVSIM85/TASM***

Директива	Стр.	Директива	Стр.	Директива	Стр.	Директива	Стр.
.8086 [1]	63/441	END	56/419	IFE	*no*/435	ORG	60/417
.186, .286	*no*/441	ENDIF	82/435	IFIDN	*no*/436	PAGE [6]	62/410
.386, .486	*no*/441	ENDM	76/438	IFIDNI	*no*/436	PROC	*no*/418
.8087, .287	*no*/442	ENDP	*no*/418	IFNB	*no*/436	PTR	*no*/426
.387, .487	*no*/442	ENDS	*no*/415	IFNDEF	*no*/435	PURGE	*no*/438
$	63/417	EQ	68/433	IRP [4], IRPC [5]	86/440	QWORD	*no*/420
+, –, *, /	66/433	EQU	64/431	LABEL	*no*/426	REPT	86/440
= [2]	66/431	EVEN	*no*/417	LE	68/433	SEG	56/434
AND	67/433	EXITM	76/437	LENGTH	*no*/434	SEGMENT [7]	56/415
ASSUME	*no*/417	FAR	*no*/418	LOCAL	76/437	SHL	67/433
BYTE	*no*/420	GE	68/433	LOW	67/432	SHORT	*no*/427
DB	63/424	GROUP	*no*/417	LT	68/433	SHR	67/433
DD	*no*/427	GT	68/433	MACRO	76/437	SIZE	*no*/434
DQ	*no*/429	HIGH	67/432	MOD	66/433	TBYTE	*no*/420
DT	*no*/430	IF	82/435	NE	68/433	THIS	*no*/434
DUP [3]	64/425	IF1, IF2	*no*/435	NEAR	*no*/418	TITLE	62/410
DW	63/425	IFB	*no*/436	NOT	67/433	TYPE	*no*/434
DWORD	*no*/420	IFDEF	*no*/435	OFFSET	*no*/425	WORD	*no*/420
ELSE	82/435	IFDIF, IFDIFI	*no*/436	OR	67/433	XOR	67/433

Примечание: [1] $CHIP(8085) и $CHIP(Z80), [2] TEQ, [3] DS, [4] %FOR *Var* in *List*, [5] %FOR *Var* chars *String*, [6] $PAGINATE и $PAGEWIDTH, [7] DEFSEG — близкие по назначению директивы ассемблера *AVSIM85*

5. *Директивы выбора моделей МП и NDCP*

Директивы выбора моделей МП и *NDCP* активизируют наборы инструкций для определенных типов основного микропроцессора (МП) и арифметического сопроцессора (*NDCP*). Эти директивы следует помещать в начале исходного файла программы — это гарантирует ассемблирование всех команд программы с использованием одного и того же набора инструкций.

Директивы выбора модели МП. Данные директивы имеют форматы:

.8086 или P8086; .186 или P186; .286 или P286 или .286c или .286p
.386 или P386 или .386c или .386p; .486 или P486 или .486c или .486p

Буква "c" дает указание ассемблировать инструкции только незащищенного режима, а буква "p" — инструкции и защищенного режима в дополнение к инструкциям незащищенного режима. По умолчанию устанавливается режим трансляции для МП 8086.

Директивы выбора модели *NDCP*. Данные директивы имеют формат:

.8087 или P8087; .287 или P287; .387 или P387; .487 или P487.

По умолчанию тип *NDCP* устанавливается в соответствии с заданным типом МП (для МП 80186 используется *NDCP* 8086). Если в программе написать директивы .8086 и .286, то транслятор выдаст предупреждение: Auxiliary processor incompatible with main processor (*вспомогательный процессор несовместим с основным*).

Заключение. Ниже для изучения режимов адресации данных приведен листинг программы с сообщениями ассемблера об ошибках, иллюстрирующий все доступные в МП 8086/8088 режимы адресации данных, графически изображенные на рис. 4.21 (файл *adr_data*.asm на прилагаемой к учебному пособию дискете).

```
Адрес  Машинный код  Имя      Директива
0000                 d_seg    segment                  ; Сегмент данных
0000   ABCD          addr0    dw      0ABCDh           ; disp = 0000h, M(addr0+1, addr0) = ABCDh
0002   5A            addr1    db      5Ah              ; disp = 0002h, M(addr1) = 5Ah
0003   A2            addr2    db      0A2h             ; disp = 0003h, M(addr2) = A2h
0004   20*(????)     var1     dw      20h dup (?)      ; disp = 0004h
0044   10*(??)       var2     db      10h dup (?)      ; disp = 0044h
0054                 d_seg    ends                     ; Конец сегмента данных
0000                 c_seg    segment                  ; Сегмент кода
                              assume  CS: c_seg, DS: d_seg
       =1234         alpha    equ     1234h            ; alpha = 1234h
       =00D7         beta     equ     0D7h             ; beta = 00D7h
     ; 1. Непосредственная адресация: 8- или 16-разрядное данное является частью команды
0000   B1 F0                  MOV     CL, 240          ; CL ← 240d = F0h
0002   B1 D7                  MOV     CL, beta         ; CL ← D7h
0004   B1 00                  MOV     CL, 256d         ; Ошибка: dst — 8-разрядный,
**Error** Constant too large (слишком большая константа)  ;          src — 16-разрядный
0006   BD 0052                MOV     BP, 52h          ; BP ← 0052h
0009   BF 0057                MOV     DI, 57h          ; DI ← 0057h
000C   25 FF00                AND     AX, 0FF00h       ; AX ← AX & FF00h
000F   81 F2 1234             XOR     DX, alpha        ; DX ← DX ⊕ 1234h
0013   2C 30                  SUB     AL, 30h          ; AL ← AL – 30h
0015   80 F9 D7               CMP     CL, beta         ; CL – D7h
                              ; Постбайт mod КОП r/m = F9 = 11 111 001 = 11 CMP CL
0018   BB 0002r               MOV     BX, offset addr1 ; BX ← disp16 = 0002h
     ; 2. Прямая адресация: эффективный адрес EA данного является частью команды
001B   89 1E 0048r            MOV     [var1 + 44h], BX ; M(0048h) ← BX
001F   89 1E 0043r            MOV     [var1 + 3Fh], BX ; M(0043h) ← BX
0023   8A 2E 0002r            MOV     CH, addr1        ; CH ← M(addr1) = 5Ah
0027   8A 2E 0002r            MOV     CH, [addr1]      ;    то же самое
                              ; Физический адрес = 16 × DS + disp16, где disp16 = 0002h
                              ; Постбайт mod reg r/m = 2Eh = 00 101 110 = 00 CH 110
002B   BA 1234                MOV     DX, [alpha]      ; Ошибка: это непосредственная, а не
                              ; прямая адресация (скобки не следует ставить — игнорируются)
*Warning* [Constant] assumed to mean immediate constant (alpha — непосредственный операнд)
```

```
002E | C6 06 0044r D7      MOV  var2, beta          ; M(var2) ← beta = D7h
0033 | C6 06 0044r D7      MOV  [var2], beta        ;   то же самое
0038 | A2 0044r            MOV  var2, AL            ; M(var2) ← AL
003B | 89 1E 0009r         MOV  [var1 + 5], BX      ; M(0009h) ← BX
003F | 89 1E 0009r         MOV  var1 + 5, BX        ;   то же самое
                       ; Постбайт mod reg r/m = 1Eh = 00 011 110 = 00 BX 110
0043 | 89 1E 0004r         MOV  [var1], BL          ; Ошибка: dst — 16 разрядов, src — 8 разрядов
**Error** Operand types do not match (несоответствие типов операндов)
0047 | 89 1E 0005r         MOV  [var1 + 1], BX      ; M(0005h) ← BX
004B | 89 3E 0009r         MOV  [var1 + 5], BH.     ; Ошибка: dst — 16 разрядов, src — 8 разрядов
**Error** Operand types do not match (несоответствие типов операндов)
004F | 09 06 0009r         OR   var1 + 5, apha      ; Ошибка: неверно написано имя alpha
**Error** Undefined symbol: APHA (символ APHA не определен)
0053 | 81 0E 0009r 1234    OR   var1 + 5, alpha     ; M(0009h) ← M(0009h) ∨ 1234h
0059 | 81 0E 0009r 00D7    OR   var1 + 5, beta      ; M(0009h) ← M (0009h) ∨ 00D7h
                       ; Постбайт mod КОП r/m = 0Eh = 00 001 110 = 00 OR 110
005F | FE 0E 0044r         DEC  var2                ; M(var2) ← M (var2) – 1
                       ; Постбайт mod КОП r/m = 0Eh = 00 001 110 = 00 DEC 110
          ; 3. Регистровая адресация: данное находится в регистре AX, BX, CX, DX, SI, DI,
                       ; BP, SP, AL, AH, BL, BH, CL, CH, DL или DH
0063 | 8B F3            MOV  SI, BX              ; SI ← BX
0065 | 03 CF            ADD  CX, DI              ; CX ← CX + DI
0067 | 23 C8            AND  CL, AX              ; Ошибка: dst — 8 разрядов, src — 16 разрядов
**Error** Operand types do not match (несоответствие типов операндов)
0069 | 32 C4            XOR  AL, AH              ; AL ← AL ⊕ AH
          ; 4. Косвенная регистровая адресация: эффективный адрес EA = BX, SI или DI
006B | 02 27            ADD  AH, [BX]            ; AH ← AH + M(BX)
006D | 03 0D            ADD  CX, [DI]            ; CX ← CX + M(DI)
006F | 88 0C            MOV  [SI], CL            ; M(SI) ← CL
0071 | 8B 14            MOV  DX, [SI]            ; DX ← M(SI)
0073 | 81 27 1234        AND  [BX], alpha         ; M(BX) ← M(BX) & alpha = M(BX) & 1234h
0077 | 81 27 00D7        AND  [BX], beta          ; M(BX) ← M(BX) & beta — байт или слово?
*Warning* Argument needs type override (необходимо указать тип операнда)
007B | 01 56 00          ADD  [BP], DX            ; M(BP) ← M(BP) + DX
          ; 5. Косвенная регистровая относительная адресация:
                       ; EA = {BX, BP, SI или DI} + {disp8 или disp16}
007E | 8B 46 0A         MOV  AX, [BP][10]        ; AX ← M(BP + 10), disp8 = 10d
0081 | 8B 46 0A         MOV  AX, [BP+10]         ;   то же самое
0084 | 8B 46 00         MOV  AX, [BP] 10         ; Ошибка: смещение 10 — в скобки [ ]!
**Error** Extra characters on line (лишние символы в строке)
0087 | 01 8F 0003r      ADD  [BX] [offset addr2], CX ; M(BX + 3) ← M(BX + 3) + CX
                       ; Постбайт mod reg r/m = 8Fh = 10 001 111 = 10 CX 111
008B | 83 07 00          ADD  [BX] offset addr2, CX ; Ошибка: смещение addr2 — в скобки [ ]!
**Error** Too few operands to instruction (слишком много операндов в команде)
*Warning* Argument needs type override (необходимо указать тип операнда)
008E | 01 8F 0003r       ADD  [BX] [addr2], CX    ; Ошибка: нет оператора offset!
**Error** Operand types do not match (несоответствие типов операндов)
0092 | 01 8F 0006r       ADD  [BX] [offset addr2+3], CX ; M(BX + 6) ← M(BX + 6) + CX
0096 | 01 8F 1234        ADD  [BX] [alpha], CX    ; M(BX + 1234h) ← M(BX + 1234h) + CX
009A | 01 8F 1234        ADD  [BX + alpha], CX    ;   то же самое
```

```
009E | 01 8F 1234          ADD  [BX][offset alpha], CX ; использовать оператор offset нет смысла
00A2 | 01 B7 00D7          ADD  [BX + beta], SI    ; M(BX + 00D7h) ← M(BX + 00D7h) + SI
                     ; Постбайт mod reg r/m = B7h = 10 110 111 = 10 SI 111
00A6 | 83 07 00          ADD  [BX] beta, SI      ; Ошибка: смещение beta — в скобки [ ]!
**Error** Too few operands to instruction (слишком много операндов в команде)
*Warning* Argument needs type override (необходимо указать тип операнда)
                ; 6. Базовая индексная адресация: EA = {BX или BP} + {SI или DI}
00A9 | 01 38              ADD  [BX] [SI], DI      ; M(BX + SI) ← M(BX + SI) + DI
00AB | 01 38              ADD  [BX + SI], DI      ;   то же самое
00AD | 81 3B 4C05         CMP  [BP] [DI], 4C05h   ; M(BP + DI) – 4C05h
00B1 | 83 3B 43           CMP  [BP] [DI], 43h     ; M(BP + DI) – 43h — байт или слово?
*Warning* Argument needs type override (необходимо указать тип операнда)
00B4 | 81 29 1234         SUB  [BX] [DI], alpha   ; M(BX + DI) ← M(BX + DI) – 1234h
00B8 | 81 29 00D7         SUB  [BX][DI], beta ; M(BX+DI) ← M(BX+DI) – beta — байт или слово?
*Warning* Argument needs type override (необходимо указать тип операнда)
                ; 7. Относительная базовая индексная адресация:
                     ; EA = {BX или BP} + {SI или DI} + {disp8 или disp16}
00BC | 01 B8 0003r        ADD  [BX][offset addr2][SI], DI ; M(BX+3+SI) ← M(BX+3+SI) + DI
00C0 | 01 B8 0003r        ADD  [BX+offset addr2+SI], DI ;   то же самое
00C4 | 01 B8 1234         ADD  [BX][alpha][SI], DI  ; M(BX+1234+SI) ← M(BX+1234+SI) + DI
00C8 | 3E: 81 BB 0009r 4C05  CMP  [BP + offset var1 + 5 + DI], 4C05h ; M(BP + 9 + DI) – 4C05h
00CF | 3E: 83 BB 0002r 43    CMP  [BP][offset addr1][DI], 43h; M(BP+2+DI) – 43h — байт или слово?
*Warning* Argument needs type override (необходимо указать тип операнда)
00D5 | 81 A9 00D7 1234      SUB  [BX][DI][beta], alpha; M(BX+DI+D7h) ← M(BX+DI+D7h) – 1234h
00DB | 81 A9 00D7 1234      SUB  [beta][BX][DI], alpha ; M(D7h+BX+DI) ← M(D7h+BX+DI) – 1234h
00E1 | 81 A9 00D7 1234      SUB  [BX+beta+DI], alpha ; M(BX+D7h+DI) ← M(BX+D7h+DI) – 1234h
00E7 |                  c_seg ends                  ; Конец сегмента кода
                            end                     ; Конец программы
```

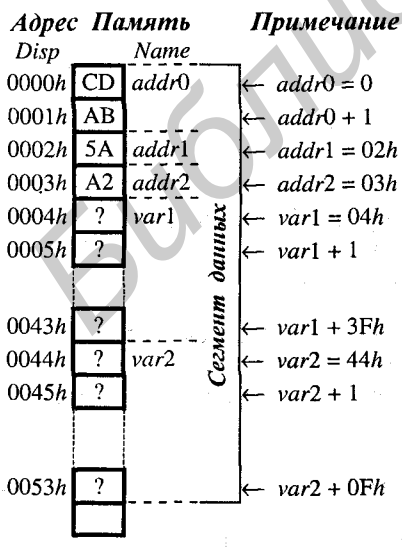


Рис. 4.30. Распределение данных

На рис. 4.30 показано распределение данных в памяти для вышеприведенной программы. С символическими именами данных в памяти связаны две величины: смещение (*displacement*) и данное, соответствующее этому смещению. Для различения этих величин используется оператор *offset* (смещение), который указывает, что символическому имени соответствует адрес памяти (см. листинг по адресу 0018).

Несмотря на эффективность многих компиляторов языков высокого уровня еще существуют области применения МП, в которых приоритет отдается языку ассемблера. К их числу относятся, например, программы обработки прерываний и обслуживания устройств ввода-вывода, а также процедуры, критичные по времени исполнения или по требуемой памяти. Программы ассемблерного уровня могут осуществлять доступ к ресурсам системы, недоступным для языков высокого уровня: регистрам, средствам распределения памяти и маскирования прерываний, специфическим средствам аппаратного обеспечения.

4.5. Функции *DOS*

Операционная система *MS-DOS* (*Microsoft Disk Operating System*) персональных компьютеров типа *IBM PC* состоит из двух основных компонентов: базовой системы ввода-вывода (*BIOS — Basic Input-Output System*) и собственно дисковой операционной системы (*DOS*).

Назначение *DOS* и *BIOS*. Базовая система ввода-вывода, размещаемая в постоянном запоминающем устройстве *ROM BIOS*, обеспечивает управление периферийным оборудованием компьютера, поэтому *BIOS* для разных модификаций персональных компьютеров *IBM PC* отличаются друг от друга. Базовая система ввода-вывода содержит набор резидентных драйверов периферийных устройств компьютера, таких, как клавиатура и экран терминала (консоль), магнитная дисковая память, принтер, последовательный порт, часы. Программы *BIOS*, обеспечивая управление периферийным оборудованием на самом низком ("физическом") уровне, путем обращения к портам, регистрам и аппаратным буферам, являются аппаратно-зависимыми.

Дисковая операционная система обеспечивает пользователю управление компьютером на более высоком уровне, чем *BIOS*, упрощая запуск и завершение программ, управление памятью, обслуживание файловой системы и др. В логическом плане *DOS* располагается между *BIOS* и программой пользователя, облегчая программные обращения к аппаратуре.

В распоряжение пользователя предоставлены функции *DOS* (табл. 4.23) и *BIOS*, вызываемые с помощью программных прерываний INT *type*.

Таблица 4.23. **Функции *DOS* (INT *type*)**

type	Функция	Подфункция	Назначение
21*h*	01*h*	—	Ввод символа с эхом
21*h*	02*h*	—	Вывод символа
21*h*	05*h*	—	Вывод символа на принтер
21*h*	06*h*	—	Прямой ввод-вывод
21*h*	07*h*	—	Нефильтрованный ввод без эха
21*h*	08*h*	—	Ввод символа без эха
21*h*	09*h*	—	Вывод строки
21*h*	0A*h*	—	Буферизованный ввод с клавиатуры
21*h*	0B*h*	—	Проверка состояния ввода
21*h*	0C*h*	—	Очистка входного буфера и ввод
21*h*	0E*h*	—	Выбор диска
21*h*	1A*h*	—	Установка адреса области обмена с диском
21*h*	1C*h*	—	Получение информации о заданном диске
21*h*	1F*h*	—	Получение адреса блока параметров текущего диска
21*h*	25*h*	—	Установка вектора прерывания
21*h*	2A*h*	—	Получение системной даты
21*h*	2B*h*	—	Установка системной даты
21*h*	2C*h*	—	Получение времени
21*h*	2D*h*	—	Установка системного времени
21*h*	2E*h*	—	Установка флага проверки
21*h*	2F*h*	—	Получение адреса области обмена с диском
21*h*	30*h*	—	Получение версии *DOS*
21*h*	31*h*	—	Завершение программы и сохранение ее резидентной в памяти
21*h*	1B*h*	—	Получение информации о текущем диске
21*h*	32*h*	—	Получение адреса блока параметров заданного диска

Продолжение табл. 4.23

type	Функция	Подфункция	Назначение
21*h*	33*h*	00*h* и 01*h*	Получение или установка состояния *Break*
21*h*	33*h*	02*h*	Получение и установка состояния *Break*
21*h*	33*h*	05*h*	Получение дисковода загрузки
21*h*	34*h*	—	Получение адреса флага занятости *DOS* (флага *InDOS*)
21*h*	35*h*	—	Получение вектора прерывания
21*h*	36*h*	—	Получение объема свободного пространства на диске
21*h*	39*h*	—	Создание каталога
21*h*	3A*h*	—	Удаление каталога
21*h*	3B*h*	—	Смена текущего каталога
21*h*	3C*h*	—	Создание или усечение файла
21*h*	3D*h*	—	Открытие файла
21*h*	3E*h*	—	Закрытие файла
21*h*	3F*h*	—	Чтение из файла или устройства
21*h*	40*h*	—	Запись в файл или в устройство
21*h*	41*h*	—	Удаление файла
21*h*	42*h*	—	Установка указателя в файле
21*h*	43*h*	—	Получение или установка атрибутов файла
21*h*	45*h*	—	Дублирование дескриптора файла
21*h*	46*h*	—	Принудительное дублирование дескриптора файла
21*h*	47*h*	—	Получение текущего каталога
21*h*	48*h*	—	Выделение блока памяти
21*h*	49*h*	—	Освобождение блока памяти
21*h*	4A*h*	—	Изменение размера выделенного блока памяти
21*h*	4B*h*	—	Запуск программы (функция *Exec*)
21*h*	4C*h*	—	Завершение процесса с кодом возврата
21*h*	4D*h*	—	Получение кода возврата и типа завершения
21*h*	4E*h*	—	Поиск первого файла с заданной спецификацией
21*h*	4F*h*	—	Поиск следующего файла
21*h*	50*h*	—	Установка идентификатора текущего процесса
21*h*	51*h*	—	Получение идентификатора текущего процесса
21*h*	52*h*	—	Получение адреса списка списков
21*h*	54*h*	—	Получение флага проверки
21*h*	56*h*	—	Переименование файла
21*h*	57*h*	00*h*	Получение даты и времени создания файла
21*h*	57*h*	01*h*	Установка даты и времени создания файла
21*h*	59*h*	—	Получение расширенной информации об ошибке
21*h*	5A*h*	—	Создание временного файла
21*h*	5B*h*	—	Создание нового файла
21*h*	5D*h*	06*h*	Получение адреса области текущих данных *DOS*
21*h*	5D*h*	0A*h*	Установка расширенной информации об ошибке
21*h*	62*h*	—	Получение идентификатора текущего процесса
21*h*	67*h*	—	Установка числа дескрипторов
21*h*	68*h*	—	Сброс буферов *DOS* в файл
21*h*	69*h*	—	Получение или установка серийного номера тома
25*h*	—	—	Абсолютное чтение с диска

Задав в регистрах общего назначения МП параметры и номер функции *DOS* или *BIOS* и выполнив команду INT с определенным значением *type*, пользователь из своей программы запускает на выполнение требуемую системную функцию. После возврата из программного прерывания в некоторых регистрах МП может содержаться результат выполнения функции.

Подробное описание функций *DOS* и *BIOS*, а также многочисленные примеры их использования, можно найти в [19], здесь же приведем только краткую характеристику функций *DOS*.

INT 21*h*, функция 01*h* — *ввод символа с эхом*. Вводит символ из устройства стандартного ввода и отображает его на устройстве стандартного вывода. При отсутствии символа ждет ввода. Допустимо перенаправление ввода. Если ввод не перенаправлен, выполняет обработку комбинации клавиш *Ctrl+C* (прекращение выполнения работающей программы при одновременном нажатии клавиш *Ctrl* и *C*). Если ввод перенаправлен, выполняет обработку комбинации клавиш *Ctrl+C* лишь при включенном режиме *BREAK* (*BREAK = ON* в файле *config*.sys). Для чтения расширенного кода *ASCII* требуется повторное выполнение функции.

При вызове: AH = 01*h*
При возврате: AL = байт входных данных

INT 21*h*, функция 02*h* — *вывод символа*. Выводит символ на экран. Допустимо перенаправление вывода. Выполняет обработку комбинации клавиш *Ctrl+C* перед выводом каждого 64-го символа. Коды *ASCII* 07*h* (звонок), 08*h* (шаг назад), 09*h* (табуляция), 0D*h* (возврат каретки) и 0A*h* (перевод строки) рассматриваются как управляющие и выполняются соответствующие им действия.

При вызове: AH = 02*h*
DL = байт данных

INT 21*h*, функция 05*h* — *вывод символа на принтер*. Выводит символ на стандартный принтер. Если принтер занят, функция ждет его освобождения. Выполняет обработку комбинации клавиш *Ctrl+C* при вводе ее с клавиатуры.

При вызове: AH = 05*h*
DL = байт данных

INT 21*h*, функция 06*h* — *прямой ввод-вывод*. Вводит из устройства стандартного ввода или выводит на устройство стандартного вывода коды символов. В режиме вывода коды *ASCII* 07*h* (звонок), 0D*h* (возврат каретки) и 0A*h* (перевод строки) рассматриваются как управляющие и выполняются соответствующие им действия. Код 08*h* (возврат на шаг) отрабатывается, только если вывод не перенаправлен. Допустимо перенаправление ввода-вывода. Для чтения расширенного кода *ASCII* требуется повторное выполнение функции. При отсутствии символа функция не ждет его ввода, а возвращает управление в программу.

При вызове: AH = 06*h*
DL = код символа (00*h* … FE*h* — при выводе; FF*h* — при вводе)
При возврате: AL = код символа (при вводе); если символа нет, то флаг ZF = 1

INT 21*h*, функция 07*h* — *нефильтрованный ввод без эха*. Вводит символ из устройства стандартного ввода без его отображения. При отсутствии символа ждет ввода. Допустимо перенаправление ввода. Не выполняет обработку комбинации клавиш *Ctrl+C*. Для чтения расширенного кода *ASCII* требуется повторное выполнение функции.

При вызове: AH = 07*h*
При возврате: AL = байт входных данных

INT 21*h*, функция 08*h* — *ввод символа без эха.* Вводит символ из устройства стандартного ввода без его отображения. При отсутствии символа ждет ввода. Допустимо перенаправление ввода. Для чтения расширенного кода *ASCII* требуется повторное выполнение функции. Если ввод не перенаправлен, выполняет обработку комбинации клавиш *Ctrl+C*. Если ввод перенаправлен, выполняет обработку комбинаций клавиш *Ctrl+C* при включенном режиме *BREAK*.

При вызове: AH = 08*h*
При возврате: AL = байт входных данных

INT 21*h*, функция 09*h* — *вывод строки.* Выводит строку символов на устройство стандартного вывода. Строка должна заканчиваться символом $. Допустимо перенаправление вывода. Допустимо использование *Esc*-последовательностей. Коды *ASCII* 07*h* (звонок), 08*h* (шаг назад), 0D*h* (возврат каретки) и 0A*h* (перевод строки) рассматриваются как управляющие и выполняются соответствующие им действия. Выполняет обработку комбинации клавиш *Ctrl+C* при вводе ее с клавиатуры перед выводом каждого 64-го символа.

При вызове: AH = 09*h*
DS:DX = адрес строки

INT 21*h*, функция 0A*h* — *буферизованный ввод с клавиатуры.* Вводит строку байт из устройства стандартного ввода в буфер пользователя с отображением на устройстве стандартного вывода. Строка должна заканчиваться символом возврата каретки (0D*h*). Допустимо перенаправление ввода. Если ввод не перенаправлен, выполняет обработку комбинации клавиш *Ctrl+C*. Если ввод перенаправлен, выполняет обработку комбинации клавиш *Ctrl+C* при включенном режиме *BREAK*.

При вызове: AH = 0A*h*
DS:DX = адрес буфера
При возврате: Данные помещены в буфер. Формат буфера:
байт 0 — ожидаемая длина строки,
байт 1 — фактическая длина введенной строки,
байт 2 и далее — строка, заканчивающаяся символом 0D*h*

INT 21*h*, функция 0B*h* — *проверка состояния ввода.* Проверяет наличие символа от устройства стандартного ввода. Допустимо перенаправление ввода. Если ввод не перенаправлен, выполняет обработку комбинации клавиш *Ctrl+C*. Если ввод перенаправлен, выполняет обработку комбинации клавиш *Ctrl+C* при включенном режиме *BREAK*.

При вызове: AH = 0B*h*
При возврате: AL = 00*h*, если функция символ не ждет;
AL = FF*h*, если функция символ ждет

INT 21*h*, функция 0C*h* — *очистка входного буфера и ввод.* Очищает кольцевой буфер клавиатуры и активизирует функцию ввода. Допустимо перенаправление ввода.

При вызове: AH = 0C*h*
AL = номер требуемой функции ввода
(допустимы функции 01, 07, 08, 0A*h*)
DS:DX = адрес буфера (если AL = 0A*h*)
При возврате: AL = байт входных данных
(если при вызове AL = 0A*h*, данные помещаются в буфер)

INT 21*h*, функция 0E*h* — *выбор диска.* Назначает текущий диск и возвращает число логических дисководов в системе.

При вызове: AH = 0E*h*
AL = код дисковода (0 = дисковод *A*, 1 = дисковод *B* и т. д.)
При возврате: AL = число логических дисководов в системе

INT 21*h*, функция 19*h* — *получение текущего диска.* Возвращает код текущего диска.

При вызове AH = 19*h*
При возврате: AL = код текущего диска (0 = диск *A*, 1 = диск *B* и т. д.)

INT 21*h*, функция 1A*h* — *установка адреса области обмена с диском.* Позволяет определить адрес дисковой области передачи (*DTA — Disk Transfer Address*) для последующих операций с блоками управления файлами.

При вызове: AH = 1A*h*
DS:DX = адрес *DTA*

INT 21*h*, функция 1B*h* — *получение информации о текущем диске.* Возвращает характеристики текущего диска.

При вызове: AH = 1B*h*
При возврате: AL = количество секторов в кластере
CX = количество байтов в секторе
DX = общее количество кластеров на диске
DS:BX → байт описания носителя:
FF*h* — дискета 320 Кбайт, FE*h* — дискета 160 Кбайт
FD*h* — дискета 360 Кбайт, FC*h* — дискета 180 Кбайт
F9*h* — дискета 1,2 Мбайт, F8*h* — жесткий диск, F0*h* — другие

INT 21*h*, функция 1C*h* — *получение информации о заданном диске.* Возвращает характеристики заданного диска.

При вызове: AH = 1C*h*
При возврате: AL = количество секторов в кластере
CX = количество байтов в секторе
DX = общее количество кластеров на диске
DS:BX → байт описания носителя (см. функцию 1B*h*)

INT 21*h*, функция 1F*h* — *получение адреса блока параметров текущего диска.* Позволяет получить детальную информацию о параметрах текущего диска.

При вызове: AH = 1F*h*
При возврате: AL = 00*h* (успешное выполнение)
DS:BX → блок параметров диска:
смещение 00*h* — 3 байта команды перехода на программу начальной загрузки
смещение 03*h* — 8 байт идентификатора изготовителя
смещение 0B*h* — 2 байта размера сектора в байтах
смещение 0D*h* — 1 байт размера кластера в секторах
смещение 0E*h* — 2 байта числа зарезервированных секторов
смещение 10*h* — 1 байт числа *FAT* (*File Allocation Table*)

смещение 11*h* — 2 байта числа записей в корневом каталоге
смещение 13*h* — 2 байта числа секторов в томе
смещение 15*h* — 1 байт описания носителя
смещение 16*h* — 2 байта размера *FAT* в секторах
смещение 18*h* — 2 байта числа секторов на дорожке
смещение 1A*h* — 2 байта числа сторон (головок)
смещение 1C*h* — 4 байта числа скрытых секторов
смещение 27*h* — 4 байта серийного номера тома
смещение 2B*h* — 11 байт метки тома

При ошибке: AL = FF*h* (недопустимый дисковод)

INT 21*h*, функция 25*h* — *установка вектора прерывания*. Позволяет заполнить вектор прерывания адресом программы обработки прерываний.

При вызове: AH = 25*h*
AL = номер вектора прерывания
DS:DX = адрес программы обработки прерываний

INT 21*h*, функция 2A*h* — *получение системной даты*. Позволяет получить значение текущей даты.

При вызове: AH = 2A*h*
При возврате: CX = год (от 1980 до 2099)
DH = месяц (от 1 до 12)
DL = день (от 1 до 31)
AL = день недели (0 — воскресенье и т. д.)

INT 21*h*, функция 2B*h* — *установка системной даты*. Позволяет изменить дату внутреннего календаря.

При вызове: AH = 2B*h*
CX = год (от 1980 до 2099)
DH = месяц (от 1 до 12)
DL = день (от 1 до 31)
При возврате: AL = 0 (успешное выполнение)
При ошибке: AL = FF*h* (недопустимая дата, системная дата не изменилась)

INT 21*h*, функция 2C*h* — *получение времени*. Позволяет получить значение текущего времени.

При вызове: AH = 2C*h*
При возврате: CH = часы (от 0 до 23)
CL = минуты (от 0 до 59)
DH = секунды (от 0 до 59)

INT 21*h*, функция 2D*h* — *установка системного времени*. Позволяет изменить время внутренних часов.

При вызове: AH = 2D*h*
CH = часы (от 0 до 23),
CL = минуты (от 0 до 59)
DH = секунды (от 0 до 59)

При возврате: AL = 00*h* (успешное выполнение)
При ошибке: AL = FF*h* (недопустимое время, системное время не изменилось)

INT 21*h*, функция 2E*h* — *установка флага проверки.* Изменяет состояние флага проверки записи на диск.

При вызове: AH = 2E*h*
AL = 00*h* — установить флаг проверки
AL = 01*h* — сбросить флаг проверки

INT 21*h*, функция 2F*h* — *получение адреса области обмена с диском.* Возвращает адрес текущей области обмена с диском (*DTA — Disk Transfer Address*).

При вызове: AH = 2F*h*
При возврате: ES:DX = адрес *DTA*

INT 21*h*, функция 30*h* — *получение версии DOS.* Возвращает номер используемой версии *MS-DOS*.

При вызове: AH = 30*h*
AL = 00*h* — в BH вернуть номер *OEM* (v. 5.0+)
AL = 01*h* — в BH вернуть флаг версии (v. 5.0+)
При возврате: AL = номер основной версии,
AH = номер подверсии
BH = номер *OEM*:
00*h* — *IBM*, 16*h* — *DEC*
99*h* — архитектура *STARLITE*, FF*h* — *Phoenix*
BH = флаг версии:
08*h* — DOS находится в ПЗУ
10*h* — DOS находится в области старшей памяти (*HMA*)

OEM (*Original Equipment Manufacturer*) — фирма-изготовитель комплексного оборудования, поставщик систем; фирма, которая покупает части и комплектующие у других производителей, производит на их основе оборудование по своим собственным оригинальным разработкам и реализующая его под своей маркой;

HMA (*High Memory Area*) — область верхней памяти; первые 64 Кбайт расширенной (*extended*) памяти; область используется операционными системами *DOS* (версия 5.0 и выше) и *Windows*.

INT 21*h*, функция 31*h* — *завершение программы и сохранение ее резидентной в памяти.* Завершает выполнение активной программы, резервируя при этом для завершаемой программы указанный объем памяти и возвращая управление родительскому процессу. Возвращает в *DOS* код возврата. В процессе завершения сбрасывает на диск буферы, закрывает дескрипторы, восстанавливает из ячеек *PSP* векторы 22*h*, 23*h* и 24*h*.

При вызове: AH = 31*h*
AL = код возврата
DX = объем резервируемой памяти (в параграфах)

INT 21*h*, функция 32*h* — *получение адреса блока параметров заданного диска.* Позволяет получить детальную информацию о параметрах заданного диска.

При вызове: AH = 32*h*
DL = номер диска (00*h* — по умолчанию, 01*h* — *A*: и т. д.)

При возврате: AL = 00*h* (успешное выполнение)
DS:BX → блок параметров диска (см. INT 21*h*, функция 1F*h*)
При ошибке: AL = FF*h* (недопустимый дисковод)

PSP (*Program Segment Prefix* — префикс программного сегмента) — специальная область оперативной памяти размером 256 (100*h*) байт; *PSP* используется в программе для хранения имен файлов и параметров, введенных из командной строки при запуске программы на выполнение, объема доступной памяти, переменных окружения системы и т. д.

INT 21*h*, функция 33*h*, подфункции 00*h* и 01*h* — *получение или установка состояния Break*. Позволяет определить или задать условия реакции *DOS* на ввод с клавиатуры комбинации клавиш *Ctrl+C* или *Ctrl+Break*. Функция не использует внутренние стеки *DOS* и поэтому реентерабельна.

При вызове: AH = 33*h*
AL = 00*h* — получить состояние *Break*
AL = 01*h* — установить состояние *Break*
DL = 00*h* — состояние *Break* выключено
(проверка только для функций *DOS* 01 … 0C*h*)
DL = 01*h* — состояние *Break* включено
(проверка для всех функций *DOS*)
При возврате: DL = текущее состояние *Break* (если при вызове AL = 00*h*):
00*h* — состояние *Break* выключено (*OFF*)
01*h* — состояние *Break* включено (*ON*)

Реентерабельность (*reenterability*) — свойство загрузочного модуля, позволяющее двум или более задачам использовать его одновременно.

INT 21*h*, функция 33*h*, подфункция 02*h* — *получение и установка состояния Break*. Позволяет определить и задать условия реакции *DOS* на ввод с клавиатуры комбинации клавиш *Ctrl+C* или *Ctrl+Break* в одной операции. Функция не использует внутренние стеки *DOS* и поэтому реентерабельна.

При вызове: AX = 3302*h*
DL = 00*h* — состояние *Break* выключено
(проверка только для функций *DOS* 01 … 0C*h*)
DL = 01*h* — состояние *Break* включено
(проверка для всех функций *DOS*)
При возврате: DL = прошлое состояние *Break*:
00*h* — состояние *Break* выключено (*OFF*)
01*h* — состояние *Break* включено (*ON*)

INT 21*h*, функция 33*h*, подфункция 05*h* — *получение дисковода загрузки*. Определяет дисковод, с которого была загружена система.

При вызове: AX = 3305*h*
При возврате: DL = дисковод загрузки (01*h* — *A*: и т. д.)

INT 21*h*, функция 34*h* — *получение адреса флага занятости DOS* (*флага InDOS*). Возвращает адрес байта области текущих данных *DOS* (*SDA*), содержащего флаг *InDOS*.

При вызове: AH = 34*h*
При возврате: ES:BX → однобайтовый флаг *InDOS*

SDA (*Swappable Data Area*) — область свопируемых данных; в этой области находятся все три стека *DOS* (*Swapping* — перекачка, обмен).

INT 21*h*, функция 35*h* — *получение вектора прерывания*. Возвращает содержимое указанного вектора прерывания.

При вызове: AH = 35*h*
AL = номер вектора прерывания
При возврате: ES:BX = адрес программы обработки прерывания

INT 21*h*, функция 36*h* — *получение объема свободного пространства на диске*. Возвращает информацию о дисководе, из которой можно вычислить емкость носителя и объем незанятого пространства. Потерянные кластеры считаются занятыми.

При вызове: AH = 36*h*
DL = код дисковода (00*h* — текущий, 01*h* — *A*: и т. д.)
При возврате: AX = число секторов в кластере
BX = число свободных кластеров
CX = размер сектора в байтах
DX = полное число кластеров на диске
При ошибке: AX = FFFF*h*

INT 21*h*, функция 39*h* — *создание каталога*. Создает каталог в конце указанного пути.

При вызове: AH = 39*h*
DS:DX = адрес пути в виде строки *ASCIIZ*
При ошибке: CF = 1, AX = код ошибки

Строка *ASCIIZ* — символьная строка, завершающаяся двоичным нулем.

INT 21*h*, функция 3A*h* — *удаление каталога*. Удаляет указанный каталог.

При вызове: AH = 3A*h*
DS:DX = адрес каталога в виде строки *ASCIIZ*
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 3B*h* — *смена текущего каталога*. Устанавливает новый текущий каталог.

При вызове: AH = 3B*h*
DS:DX = адрес каталога в виде строки *ASCIIZ*
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 3C*h* — *создание или усечение файла*. Создает новый файл с указанной спецификацией. Если указанный файл существует, он усекается до нулевой длины. В любом случае файл открывается и возвращается его дескриптор для дальнейших операций над файлом.

При вызове: AH = 3C*h*
CX = атрибуты файла (могут комбинироваться):
1 — только для чтения, 2 — скрытый
4 — системный, 8 — метка тома
20*h* — атрибут архива
DS:DX = адрес спецификации файла в виде строки *ASCIIZ*

При возврате: AX = дескриптор
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 3D*h* — *открытие файла*. Открывает файл с указанной спецификацией. Возвращает дескриптор для последующих операций над файлом. Устанавливает указатель на начало файла (байт 0). Если к режиму доступа добавлено 80*h*, дескриптор наследуется дочерним процессом. В противном случае дескриптор не наследуется дочерним процессом.

При вызове: AH = 3D*h*
AL = режим доступа:
0 — чтение
1 — запись
2 — запись и чтение
DS:DX = адрес спецификации файла в виде строки *ASCIIZ*
При возврате: AX = дескриптор
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 3E*h* — *закрытие файла*. Сбрасывает на диск внутренние буферы файла, закрывает файл и освобождает дескриптор. Если файл был модифицирован, в записи каталога устанавливаются новые значения длины файла, а также даты и времени создания файла.

При вызове: AH = 3E*h*
BX = дескриптор
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 3F*h* — *чтение из файла или устройства*. Пересылает из файла данные в буфер пользователя и модифицирует указатель. При чтении из символьного устройства в режиме *ASCII* читается строка указанной длины, либо до символа возврата каретки, если он встретился раньше.

При вызове: AH = 3F*h*
BX = дескриптор
CX = число пересылаемых байтов
DS:DX = адрес буфера пользователя
При возврате: AX = число переданных байтов
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 40*h* — *запись в файл или в устройство*. Пересылает в файл данные из буфера пользователя и модифицирует указатель. Если при вызове CX = 0, длина файла устанавливается в соответствии с текущим положением указателя. Если перед выводом с клавиатуры вводится *Ctrl+C* и режим *BREAK* включен (*BREAK = ON*), выполняется обработка *Ctrl+C*.

При вызове: AH = 40*h*
BX = дескриптор
CX = число пересылаемых байтов
DS:DX = адрес буфера пользователя
При возврате: AX = число переданных байтов
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 41*h* — *удаление файла.* Удаляет указанный файл.

При вызове: AH = 41*h*
DS:DX = спецификация файла в виде строки *ASCIIZ*
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 42*h* — *установка указателя в файле.* Позволяет установить текущее положение указателя на любой байт файла для выполнения последующих операций прямого доступа к файлу (чтения или записи).

При вызове: AH = 42*h*
AL = режим установки указателя:
00*h* — абсолютное смещение от начала файла
01*h* — знаковое смещение от текущего положения указателя
02*h* — знаковое смещение от конца файла
BX = дескриптор
CX = старшая часть смещения
DX = младшая часть смещения
При возврате: DX = старшая часть возвращенного указателя
AX = младшая часть возвращенного указателя

INT 21*h*, функция 43*h* — *получение или установка атрибутов файла.* Позволяет получить или изменить значения атрибутов файла или каталога. Файл нельзя преобразовать в каталог или метку тома. Каталог можно сделать скрытым.

При вызове: AH = 43*h*
AL = 00*h* — для получения атрибутов
AL = 01*h* — для установки атрибутов
CX = атрибуты файла (могут комбинироваться):
0001*h* — только для чтения, 0002*h* — скрытый
0004*h* — системный, 0020*h* — атрибут архивации
DS:DX = адрес спецификации файла или каталога
При возврате: CX = возвращаемые атрибуты файла (если при вызове AL = 0)
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 45*h* — *дублирование дескриптора файла.* Создает новый дескриптор файла, который связан с заданным файлом или устройством через тот же элемент системной таблицы файлов (*SFT*).

При вызове: AH = 45*h*
BX = дескриптор файла
При возврате: AX = новый дескриптор
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 46*h* — *принудительное дублирование дескриптора файла.* Принудительно объявляет указанный дескриптор дубликатом заданного. Если дескриптор в BX был открыт, он закрывается.

При вызове: AH = 46*h*
BX = дескриптор файла
CX = дескриптор, который должен стать дубликатом первого
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 47*h* — *получение текущего каталога*. Возвращает строку *ASCIIZ* с полным путем (от корневого каталога) к текущему каталогу, включая его имя. Не возвращается обозначение текущего дисковода и корневого каталога (знак \).

При вызове:	AH = 47*h*
	DL = код дисковода (0 — текущий, 1 — *A*: и т. д.)
	DS:SI = адрес буфера размером 64 байта
При возврате:	буфер заполнен спецификацией текущего каталога
При ошибке:	CF = 1, AX = код ошибки

INT 21*h*, функция 48*h* — *выделение блока памяти*. Выделяет блок памяти и возвращает его сегментный адрес.

При вызове:	AH = 48*h*
	BX = требуемое число параграфов памяти
При возврате:	AX = сегментный адрес выделенного блока
При ошибке:	CF = 1, AX = код ошибки
	BX = размер наибольшего доступного блока памяти в параграфах

INT 21*h*, функция 49*h* — *освобождение блока памяти*. Освобождает блок памяти и передает его системе для использования другими программами.

При вызове:	AH = 49*h*
	ES = сегментный адрес освобождаемого блока
При ошибке:	CF = 1, AX = код ошибки

INT 21*h*, функция 4A*h* — *изменение размера выделенного блока памяти*. Уменьшает или увеличивает размер выделенного блока памяти.

При вызове:	AH = 4A*h*
	BX = требуемый размер блока в параграфах
	ES = сегментный адрес модифицируемого блока
При ошибке:	CF = 1, AX = код ошибки
	BX = размер наибольшего доступного блока памяти в параграфах

INT 21*h*, функция 4B*h* — *запуск программы* (*функция Exec*). Позволяет родительскому процессу, в частности, активной прикладной программе, загрузить и запустить другую программу (дочерний процесс). После завершения запущенной программы управление возвращается родительскому процессу.

При вызове:	AH = 4B*h*
	AL = 00*h* — загрузить и выполнить программу
	AL = 01*h* — загрузить и не выполнять программу
	AL = 03*h* — загрузить оверлей, ES:BX = адрес блока параметров
	DS:DX = адрес спецификации запускаемой программы в виде строки *ASCIIZ*
При ошибке:	CF = 1, AX = код ошибки

INT 21*h*, функция 4C*h* — *завершение процесса с кодом возврата*. Завершает текущий процесс (программу), помещая указанный код завершения в предназначенный для него байт области текущих данных *DOS* (*SDA*). В процессе завершения освобождает всю выделенную процессу память, сбрасывает на диск буферы, закрывает все открытые дескрипторы, из ячеек *PSP* восстанавливает векторы 22*h*, 23*h* и 24*h*.

При вызове: AH = 4C*h*
AL = код возврата

INT 21*h*, функция 4D*h* — *получение кода возврата и типа завершения.* Используется родительским процессом после возврата из дочернего процесса, активизированного функцией *Exec* (INT 21*h*, функция 4B*h*), для получения из области текущих данных *DOS* (*SDA*) кодов возврата и завершения процесса. Код возврата в *SDA* в результате использования данной функции очищается, поэтому ее можно вызывать только один раз.

При вызове: AH = 4D*h*
При возврате: AH = тип завершения:
00*h* — нормальное завершение с помощью INT 20*h* или INT 21*h* (функции 00*h* или 4C*h*)
01*h* — пользователь ввел *Ctrl+C*
02*h* — завершение через драйвер критической ошибки
03*h* — завершение с помощью INT 21*h* (функции 31*h* или 27*h*)
AL = код возврата, передаваемый из дочернего процесса

INT 21*h*, функция 4E*h* — *поиск первого файла с заданной спецификацией.* Осуществляет поиск в указанном каталоге первого файла, соответствующего указанному шаблону групповой операции. Если CX = 0, производится поиск только нормальных файлов (без атрибутов). Если CX = 8, производится поиск только метки тома. При указании других атрибутов или их комбинаций производится поиск файлов с указанными атрибутами и нормальных файлов.

При вызове: AH = 4E*h*
CX = атрибуты искомых файлов (могут комбинироваться):
1 — только для чтения, 2 — скрытый
4 — системный, 8 — метка тома
10*h* — каталог, 20*h* — атрибут архивации
DS:DX = адрес спецификации искомого файла
При возврате: Имя файла и расширение заносятся в область обмена с диском в байты 1E*h* … 2A*h*
При ошибке: CF = 1
AX = код ошибки

INT 21*h*, функция 4F*h* — *поиск следующего файла.* Осуществляет поиск следующего файла после того, как функция 4E*h* нашла первый файл, соответствующий указанному шаблону групповой операции. Используется только после успешного выполнения функции 4E*h*. Если предполагается, что файлов, соответствующих шаблону, может быть больше двух, функцию следует выполнять многократно до получения CF = 1 (файлов, соответствующих шаблону, больше нет).

При вызове: AH = 4F*h*
При возврате: Имя файла и расширение заносятся в область обмена с диском в байты 1E*h* … 2A*h*
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 50*h* — *установка идентификатора текущего процесса.* Позволяет записать в область текущих данных *DOS* адрес *PSP* программы, которую требуется объявить

текущей, функция не использует внутренние стеки *DOS* и поэтому реентерабельна. Функция документирована начиная с *v.* 5.0.

При вызове: AH = 50*h*
BX = сегментный адрес *PSP* процесса, объявляемого текущим

INT 21*h*, функция 51*h* — *получение идентификатора текущего процесса.* Позволяет получить адрес *PSP* программы, которую *DOS* считает текущей. Функция не использует внутренние стеки *DOS* и поэтому реентерабельна. Функция документирована начиная с *DOS v.* 5.0.

При вызове: AH = 51*h*
При возврате: BX = сегментный адрес *PSP* текущего процесса

INT 21*h*, функция 52*h* (недокументирована) — *получение адреса списка списков.* Возвращает адрес "списка списков" — базовой системной таблицы, содержащей информацию о ряде других таблиц *DOS*.

При вызове: AH = 52*h*
При возврате: ES:BX = адрес списка списков

INT 21*h*, функция 54*h* — *получение флага проверки.* Позволяет получить состояние флага *DOS "verify"* проверки правильности записи на диск.

При вызове: AH = 54*h*
При возврате: AL = флаг проверки: 00*h* — выключен, 01*h* — включен

INT 21*h*, функция 56*h* — *переименование файла.* Переименовывает файл или перемещает его в другой каталог на том же диске. Допустимо переименование каталога. Недопустимо использование шаблонов групповых операций.

При вызове: AH = 56*h*
DS:DX = адрес текущей спецификации файла
ES:DI = адрес новой спецификации файла
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 57*h*, подфункция 00*h* — *получение даты и времени создания или модификации файла.* Позволяет получить дату и время создания файла, записанные в каталоге. Файл должен быть предварительно создан или открыт.

При вызове: AX = 5700*h*
BX = дескриптор
При возврате: CX = новое время. Разряды:
0 … 4 — двухсекундные интервалы
5*h* … A*h* — минуты
B*h* … F*h* — часы
DX = дата. Разряды:
0 … 4 — день
5 … 8 — месяц
9*h* … F*h* — год относительно 1980
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 57*h*, подфункция 01*h* — *установка даты и времени создания файла.* Позволяет модифицировать дату и время создания файла, записанные в каталоге. Файл должен быть предварительно создан или открыт.

При вызове:	AX = 5701*h*
	BX = дескриптор
	CX = новое время (см. функцию 57*h*, подфункцию 00*h*)
	DX = новая дата (см. функцию 57*h*, подфункцию 00*h*)
При ошибке:	CF = 1, AX = код ошибки

INT 21 *h*, функция 59*h* — *получение расширенной информации об ошибке.* Позволяет получить после предыдущего неудачного вызова *DOS* детальную информацию об ошибке, которая помещается системой в поля области текущих данных (*SDA*). Функция разрушает содержимое регистров CL, DX, SI, DI, BP, DS и ES. Функцию можно вызывать в обработчике критической ошибки.

При вызове:	AH = 59*h*
	BX = 0000*h*
При возврате:	AX = расширенный код ошибки (табл. 4.24)
	BH = класс ошибки
	BL = рекомендуемое действие
	CH = местоположение ошибки

Таблица 4.24. **Коды ошибок**

Расширенный код ошибки (регистр AX)			
00*h*	Нет ошибки	16*h*	Неизвестная команда
01*h*	Неверный номер функции	17*h*	Ошибка контрольной суммы (*CRC*)
02*h*	Файл не найден	18*h*	Неверная длина структуры запроса
03*h*	Путь не найден	19*h*	Ошибка поиска дорожки
04*h*	Слишком много открытых файлов	1A*h*	Неизвестный носитель (не *DOS*-диск)
05*h*	Доступ запрещен	1B*h*	Сектор не найден
06*h*	Неверный дескриптор	1C*h*	В принтере нет бумаги
07*h*	Разрушен блок управления памятью	1D*h*	Ошибка записи
08*h*	Недостаточно памяти	1E*h*	Ошибка чтения
09*h*	Неверный адрес блока памяти	1F*h*	Общий отказ
0A*h*	Ошибка окружения	20*h*	Нарушение разделения
0B*h*	Неправильный формат	21*h*	Нарушение закрытия файла
0C*h*	Неправильный код доступа	22*h*	Недопустимая смена дискеты
0D*h*	Неправильные данные	23*h*	Отсутствует блок управления *FCB*
0E*h*	Неизвестное устройство	24*h*	Переполнение разделяемого буфера
0F*h*	Неправильный дисковод	25*h*	Несоответствие кодовой страницы
10*h*	Попытка удалить текущий каталог	26*h*	Невозможно завершить ввод из файла
11*h*	Не то же устройство	27*h*	Недостаточно места на диске
12*h*	Нет больше файлов	50*h*	Файл уже существует
13*h*	Диск защищен от записи	52*h*	Невозможно создать каталог
14*h*	Неизвестное устройство	28*h* ... 5A*h* — зарезервированные и сетевые ошибки	
15*h*	Дисковод не готов		

Продолжение табл. 4.24

Класс ошибки (регистр BH)			
01*h*	Нехватка ресурсов	07*h*	Ошибка прикладной программы
02*h*	Файл или запись заперты	08*h*	Не найдено
03*h*	Доступ запрещен	09*h*	Неверный формат
04*h*	Ошибка системной программы	0A*h*	Заперто
05*h*	Отказ оборудования	0B*h*	Ошибка носителя
06*h*	Отказ системы	0C*h*	Уже существует
Рекомендуемое действие (регистр BL)			
01*h*	Повтор	05*h*	Немедленно прекратить выполнение
02*h*	Задержанный повтор	06*h*	Игнорировать
03*h*	Запрос пользователю на повторный ввод	07*h*	Повтор после вмешательства пользователя
04*h*	Прекратить выполнение после очистки ресурсов		
Местоположение ошибки (регистр CH)			
01*h*	Неизвестное	04*h*	Таймаут последовательного порта
02*h*	Блочное устройство (ошибка диска)	05*h*	Ошибка памяти
03*h*	Сетевая ошибка		

Примечание: *CRC* (*Cyclic Redundancy Check*) — контроль с использованием циклического избыточного кода (метод обнаружения ошибок). Передатчик выполняет вычисления по определенному алгоритму над содержимым передаваемого блока данных и добавляет полученный результат к блоку. Приемник выполняет аналогичные вычисления над принимаемым блоком данных — ошибка, если принятое и вычисленное значения *CRC* не совпадают.

INT 21*h*, функция 5A*h* — *создание временного файла.* Создает файл с указанными атрибутами в указанном каталоге и возвращает дескриптор и имя файла. Имя файлу назначается системой. При завершении программы файл не удаляется. Функцию удобно использовать, если в программе требуется создать большое и не определенное заранее количество файлов, конкретные имена которых не имеют особого значения.

При вызове: AH = 5A*h*
CX = атрибуты файла (могут комбинироваться):
1 — только для чтения, 2 — скрытый
4 — системный, 20*h* — атрибут архива
DS:DX = адрес спецификации каталога в виде строки *ASCIIZ*
При возврате: AX = дескриптор
DS:DX = адрес полной спецификации файла в виде строки *ASCIIZ*
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 5B*h* — *создание нового файла.* Создает новый файл с указанной спецификацией и атрибутами и возвращает дескриптор. Если указанный файл уже существует, функция завершается с ошибкой.

При вызове: AH = 5B*h*
CX = атрибуты файла (могут комбинироваться):
1 — только для чтения, 2 — скрытый, 4 — системный
8 — метка тома, 20*h* — атрибут архива
DS:DX = адрес спецификации файла в виде строки *ASCIIZ*
При возврате: AX = дескриптор
При ошибке: CF = 1, AX = код ошибки

INT 21*h*, функция 5D*h*, подфункция 06*h* — *получение адреса области текущих данных DOS* (*недокументированная функция*). Возвращает адрес области текущих данных *DOS* (*SDA — Swappable Data Area*), в которой хранится ряд системных переменных и, в частности, находятся все три стека *DOS*.

При вызове:	AX = 5D06*h*
При возврате:	DS:SI → SDA
	CX = размер в байтах части *SDA*, которая должна сохраняться при переходе на другой процесс, если прерывается функция *DOS*
	DX = размер в байтах части *SDA*, которая должна сохраняться при переходе на другой процесс во всех случаях
При ошибке:	CF = 1, AX = код ошибки

INT 21*h*, функция 5D*h*, подфункция 0A*h* — *установка расширенной информации об ошибке*. Позволяет восстановить в *SDA* расширенную информацию об ошибке. Предварительно эта информация должна быть получена из *SDA* с помощью функции *DOS* 59*h* и сохранена в программе. Функция используется в резидентных обработчиках аппаратных прерываний.

При вызове:	AX = 5D0A*h*
	DS:DX → трехсловный список параметров, составляющих расширенную информацию об ошибке

INT 21*h*, функция 62*h* — *получение идентификатора текущего процесса*. Позволяет получить адрес *PSP* программы, которую *DOS* считает текущей. Функция не использует внутренние стеки *DOS* и поэтому реентерабельна. Идентична функции 51*h*.

При вызове:	AH = 62*h*
При возврате:	BX = сегментный адрес *PSP* текущего процесса

INT 21*h*, функция 67*h* — *установка числа дескрипторов*. Устанавливает максимальное число файлов и устройств, которые могут быть одновременно открыты текущим процессом. Фактически функция создает новую таблицу файлов задания *JFT* (*Job File Table*), копируя в ее начало исходную *JFT*, находящуюся в *PSP* программы и имеющую размер 20 байтов. Новая таблица создается в свободной памяти за пределами программы и для ее успешного выполнения требуется, чтобы в системе был свободный блок памяти соответствующего размера. Поскольку максимальное число открытых файлов лимитируется не только размером *JFT*, но также и числом блоков описания файлов в системной таблице файлов *SFT*, наряду с расширением *JFT* требуется также расширить *SFT* с помощью директивы файла *config*.sys FILES=.

При вызове:	AH = 67*h*
	BX = требуемое число дескрипторов
При возврате:	В *PSP* текущей программы записан адрес новой *JFT*

INT 21*h*, функция 68*h* — *сброс буферов DOS в файл*. Выполняет принудительное обновление файла на диске. Все данные из буферов *DOS* записываются в файл. Обновляется запись каталога.

При вызове:	AH = 68*h*
	BX = дескриптор
При ошибке:	CF = 1, AX = код ошибки

INT 21*h*, функция 69*h* — *получение или установка серийного номера тома.* Читает или записывает метку тома и серийный номер тома для данного диска.

При вызове: AH = 69*h*
AL = подфункция:
00*h* — получить серийный номер
01*h* — установить серийный номер
BL = дисковод (0 — текущий, 1 — *A*: и т. д.)
DS:BX → буфер размером 32 байта
При возврате: AL = 00*h* — в буфер помещается копия содержимого расширенного блока параметров *BIOS* (*BPB*) на диске
AL = 01*h* — в расширенный *BPB* на диске копируется информация из буфера
При ошибке: CF = 1, AX = код ошибки.

Формат буфера:

Смещение	Число байт	Содержимое
00*h*	2	0
02*h*	4	Серийный номер диска
06*h*	11	Метка тома или "NONAME", если она отсутствует
11*h*	8	Тип файловой системы — "*FAT*12" или "*FAT*16"

INT 25*h* — *абсолютное чтение с диска.* Позволяет прочитать в память с диска один или группу секторов с заданным начальным относительным номером. Секторы нумеруются от 0 от начала логического (не физического) диска. Таким образом, загрузочный сектор данного логического диска имеет номер 0, следующий за ним на диске первый сектор первой копии *FAT* — номер 1 и т. д.

После выполнения прерываний INT 25*h* и INT 26*h* в стеке задачи остается слово, содержащее значение регистра флагов. Если это слово не удалить, то нарушится дальнейший ход программы.

При вызове: AL = номер дисковода (0 — *A*:, 1 — *B*: и т. д.)
CX = число читаемых секторов
DX = относительный номер первого читаемого сектора
DS:BX = адрес буфера
При ошибке: CF = 1, AX = код ошибки

AH	*Тип ошибки*	AL	*Тип ошибки*
01*h*	Неправильная команда	00*h*	Ошибка защиты записи
02*h*	Неправильная адресная метка	01*h*	Неизвестное устройство
04*h*	Запрошенный сектор не найден	02*h*	Дисковод не готов
08*h*	Ошибка прямого доступа к памяти	03*h*	Неизвестная команда
10*h*	Ошибка данных (неправильная контрольная сумма)	04*h*	Ошибка данных (неправильная контрольная сумма)
20*h*	Ошибка контроллера	06*h*	Ошибка позиционирования
40*h*	Ошибка позиционирования	07*h*	Неизвестный тип носителя
		08*h*	Сектор не найден

INT 26*h* — *абсолютная запись на диск.* Позволяет записать из памяти на диск один или группу секторов с заданным начальным относительным номером. Секторы нумеруются от 0 от начала логического (не физического) диска. Таким образом, загрузочный сектор данного логического диска имеет номер 0, следующий за ним на диске первый сектор первой копии *FAT* — номер 1 и т. д.

После выполнения прерываний INT 25*h* и INT 26*h* в стеке задачи остается слово, содержащее значение регистра флагов. Если это слово не удалить, то нарушится дальнейший ход программы.

При вызове:	AL = номер дисковода (0 — *A*:, 1 — *B*: и т. д.)
	CX = число читаемых секторов
	DX = относительный номер первого читаемого сектора
	DS:BX = адрес буфера
При ошибке:	CF = 1, AX = код ошибки (см. предыдущую команду)

INT 2F*h* — *мультиплексное прерывание.* Прерывание предназначено для организации связи между процессами и, в частности, для обмена информацией с системными и прикладными резидентными программами. Для пользователя зарезервированы функции C0*h* ... FF*h*.

При вызове:	AH = функция
	AL = подфункция
	Другие регистры используются по мере необходимости
При возврате:	AL = 0, если программа не установлена, и ее можно установить
	AL = 1, если программа не установлена, и ее нельзя установить
	AL = FF*h*, если программа установлена
При ошибке:	CF = 1, AX = код ошибки

4.6. Арифметический сопроцессор 8087

Арифметический сопроцессор 8087 (*NDCP — Numeric Data Coprocessor*) фирмы *Intel* (отечественный аналог 1810ВМ87) используется для проектирования МП-систем на базе МП 8086/8088/80186, повышая эффективность вычислений с плавающей точкой в среднем в 100 раз по сравнению с их программной эмуляцией (табл. 4.25). Отдельные команды *NDCP* 8087 заменяют сотни команд МП 8086, так как выполнение арифметических операций сложения, вычитания, умножения, деления и вычисление тригонометрических, логарифмических и показательных функций реализовано на аппаратном уровне. Сопроцессор позволяет существенно повысить эффективность вычислений в научных, навигационных и военных приложениях. Используемые в *NDCP* представления чисел с плавающей точкой совместимы со стандартом *IEEE Floating Point Standard* 754.

Производится *NDCP* по высококачественной *n*-канальной технологии *HMOS* III с частотами тактового сигнала 5 МГц (8087), 8 МГц (8087-2) и 10 МГЦ (8087-1). Максимальный ток потребления $I_{CC\ max}$ равен 475 мА. Максимальная рассеиваемая мощность составляет 3,0 Вт. Выходные сигналы *NDCP* характеризуются параметрами: $V_{OL\ max}$ = 0,45 В при I_{OL} = 2,5 мА и $V_{OH\ min}$ = 2,4 В при I_{OH} = –400 мкА.

Большинство контактов *NDCP* 8087 (рис. 4.31) имеют то же назначение, что и в МП 8086 (см. рис. 4.1). Для подключения сопроцессора 8087 к микропроцессору 8086 не требуется никакой внешней логики (все одноименные контакты соединяются между собой).

Таблица 4.25. **Время выполнения численных команд и их эмуляции на МП 8086**

Команды с плавающей точкой (*Floating Point Instruction*)	Время выполнения, мкс	
	8086/8087 (8 МГц)	8086 *Emulation*
Сложение/Вычитание (*Add/Subtract*)	10,6	1000
Умножение с одинарной точностью (*Multiply single precision*)	11,9	1000
Умножение с расширенной точностью (*Multiply extended precision*)	16,9	1312,5
Деление (*Divide*)	24,4	2000
Сравнение (*Compare*)	5,6	812,5
Загрузка в *NDCP* операнда *Long Real* (*Load double precision*)	6,3	1062,5
Запись в память операнда *Long Real* (*Store double precision*)	13,1	750
Квадратный корень (*Square Root*)	22,5	12250
Тангенс (*Tangent*)	56,3	8125
Возведение в степень (*Exponentiation*)	62,5	10687,5

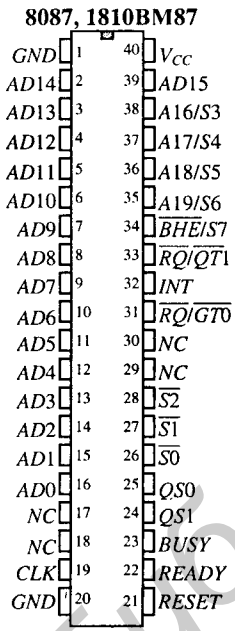


Рис. 4.31. *NDCP*

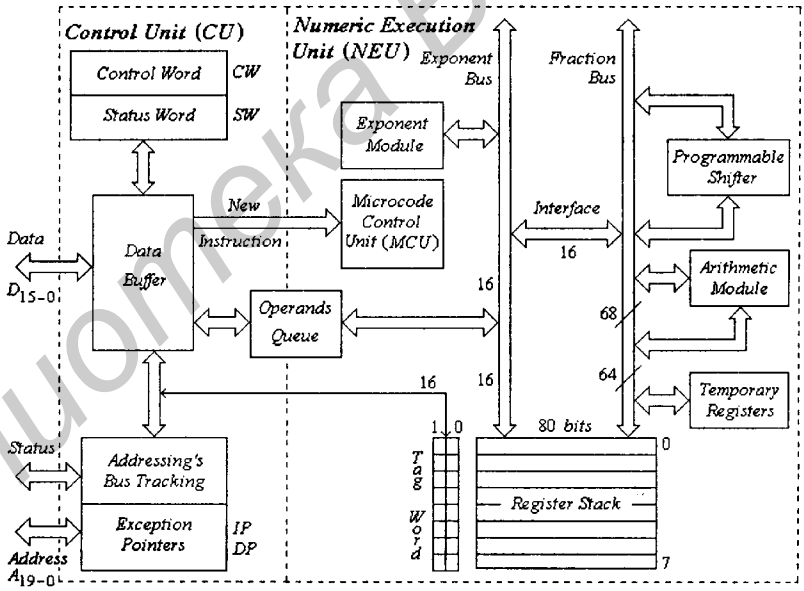


Рис. 4.32. Структурная схема арифметического сопроцессора 8087

Структурная схема *NDCP* 8087. Взаимодействие МП и *NDCP* было описано в § 4.1 (см. рис. 4.15) и предполагается читателю уже известным. На рис. 4.32 изображена структурная схема *NDCP* без указания конкретных сигналов управления, адреса и данных, показанных на рис. 4.31 (*NC — No Connection* — контакт не используется).

NDCP состоит из двух относительно независимых частей: численного исполнительного устройства *NEU* (*Numeric Execution Unit*) и устройства управления *CU* (*Control Unit*) — шинного интерфейса. Передача данных между этими устройствами организована в виде очереди операндов (*Operands Queue*), что обеспечивает достаточную независимость их параллельной работы.

В исполнительном устройстве *NEU* реализуются основные функции по обработке чисел с плавающей точкой. Оно содержит регистровый стек (*Register Stack*), состоящий из восьми 80-разрядных регистров, 16-разрядный регистр тэгов (*Tag Word*), регистры временного хранения данных (*Temporary Registers*), 68-разрядные арифметический модуль (*Arithmetic Module*) и программируемый сдвигатель (*Programmable Shifter*), модуль обработки порядка чисел (*Exponent Module*) и микропрограммное устройство управления *MCU* (*Microcode Control Unit*).

Устройство управления *CU* принимает и декодирует команды (инструкции) программы, принадлежащие *NDCP*, читает и записывает операнды в память, выполняет команды управления *NDCP* (запись и чтение из памяти слова управления и слова состояния, запрет и разрешение запросов прерываний и др.), а также поддерживает синхронизацию с МП 8086, в то время как исполнительное устройство *NEU* занято выполнением численных команд (обработкой чисел с плавающей точкой). Устройство управления *CU* содержит буфер данных (*Data Buffer*), регистр слова управления *CW* (*Control Word*), регистр слова состояния *SW* (*Status Word*), блок отслеживания адресации шины (*Addressing's Bus Tracking*) и указатель особых случаев *EP* (*Exception Pointers*). Указатель *EP*, состоит из указателя инструкции *IP* (*Instruction Pointer*) и указателя данных *DP* (*Data Pointer*).

Одноименные контакты МП 8086 и *NDCP* 8087 имеют почти одинаковое назначение и в МП-системе 8086 + 8087 должны быть соединены. Назначение сигналов *NDCP*:

AD_{15-0} (*Address Data Bus*) — мультиплексная шина адреса-данных. Если шинами управляет *NDCP*, то по двунаправленным линиям AD_{15-0} с разделением во времени передаются младшие 16 разрядов адреса памяти A_{15-0} и данные D_{15-0}. В первом такте (T_1) цикла шины выдается адрес A_{15-0}, который необходимо зафиксировать во внешнем адресном регистре (в том же самом, который использует МП), а затем (такты T_2, T_3, T_W, T_4) принимаются или передаются данные D_{15-0}. Фиксация старшего байта адреса A_{15-8} не требуется в МП-системах 8088 + 8087 — в этом случае *NDCP* выдает значения A_{15-8} в течение тактов T_{1-4}. Если шинами управляет МП, то линии AD_{15-0} являются входами — *NDCP* отслеживает управление шинами микропроцессором 8086. Для упрощения восприятия структурной схемы *NDCP* (см. рис. 4.32) в ней показаны независимые шины адреса и данных;

A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3 (*Address/Status*) — мультиплексные выходные сигналы адреса-состояния. Если шинами управляет *NDCP*, то в первом такте (T_1) на эти линии выдаются значения старших четырех разрядов адреса памяти A_{19-16}, а в остальных тактах цикла шины (T_2, T_3, T_W, T_4) — значения сигналов состояния $S_6 = 1$, $S_4 = 1$, $S_3 = 1$ (зарезервированы) и $S_5 = 0$. Когда шинами управляет МП, линии A_{19-16}/S_{6-3} являются входами — *NDCP* отслеживает управление шинами микропроцессором 8086;

$\overline{BHE}/S_7$ (*Bus High Enable/Status*) — разрешение старшей части шины/состояние. Если шинами управляет *NDCP*, то в первом такте (T_1) на эту линию выводится значение сигнала $\overline{BHE}$, а в остальных тактах цикла шины (T_2, T_3, T_W, T_4) — сигнал состояния $S_7 = 0$. Сигнал $\overline{BHE}$ совместно с сигналом A_0 используется для адресации памяти (см. табл. 4.3). Значения сигналов $\overline{BHE}$ и A_{19-16}, как и сигналов A_{15-0}, необходимо фиксировать во внешнем регистре. Если шинами управляет МП, то линия $\overline{BHE}/S_7$ является входом — *NDCP* отслеживает управление шинами микропроцессором 8086;

S_{2-0} (*Status*) — сигналы состояния, которые активны в тактах T_4, T_1 и T_2 каждого цикла шины и переходят в пассивное состояние (111) в течение такта T_3 или T_W при значении сигнала *READY* = 1. Сигналы состояния S_{2-0} подаются на контроллер шин 8288 (1810ВГ88), который формирует полный набор системных сигналов управления памятью. Любое изменение сигналов S_{2-0} в такте T_4 определяет начало следующего цикла шины, а переход их в пассивное состояние (111) указывает конец цикла шины. Если шинами управляет *NDCP*, то линии S_{2-0} являются вы-

ходами, определяющими тип цикла шины (табл. 4.26). Если шинами управляет МП, то линии $S_{2\text{-}0}$ являются входами — *NDCP* отслеживает управление шинами микропроцессором 8086;

Таблица 4.26. **Системные сигналы управления**

$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	Состояние МП	Сигналы 8288	Примечание
0	×	×	*Unused*	*None*	Состояния не используются
1	0	0	*Unused*	*None*	Состояние не используется
1	0	1	*Read Memory*	$\overline{MRDC}$	Чтение памяти
1	1	0	*Write Memory*	$\overline{AMWC}$, $\overline{MWTC}$	Запись в память
1	1	1	*Passive (no bus cycle)*	*None*	Пассивное состояние

$\overline{RQ}/\overline{GT}_0$ (*Request/Grant*) — двунаправленная линия запроса/предоставления шин, обеспечивающая бесконфликтное управление шинами микропроцессором 8086 и арифметическим сопроцессором 8087. Управление запросом и предоставлением шин производится последовательностью из трех импульсов низкого уровня, длительность которого равна одному периоду тактового сигнала *CLK* (см. рис. 4.9). Контакт $\overline{RQ}/\overline{GT}_0$ *NDCP* соединяется с контактом $\overline{RQ}/\overline{GT}_0$ или $\overline{RQ}/\overline{GT}_1$ МП. При запросе шин от *NDCP* микропроцессор предоставит их в конце текущего цикла шины, переведя свои шины в *Z*-состояние;

$\overline{RQ}/\overline{GT}_1$ (*Request/Grant*) — двунаправленная линия запроса/предоставления шин, используемая для подключения другого ведущего устройства и передачи его сигналов запроса и предоставления шин на/от МП через контакт $\overline{RQ}/\overline{GT}_0$ *NDCP*. В качестве такого ведущего устройства может быть подключен процессор ввода-вывода 8089 фирмы *Intel* (см. рис. 4.58) [12, 13]. К линии $\overline{RQ}/\overline{GT}_1$ подключен внутренний нагрузочный резистор для задания высокого уровня (если контакт $\overline{RQ}/\overline{GT}_1$ не используется, то его можно оставлять не подключенным через резистор к источнику питания);

$QS_{1\text{-}0}$ (*Queue Status*) — сигналы состояния шестибайтовой очереди команд (см. табл. 4.5), поступающие от МП 8086 и позволяющие *NDCP* 8087 прослеживать очередь команд МП (синхронизировать свою очередь команд с очередью команд МП).

INT (*Interrupt*) — сигнал запроса прерывания, подаваемый на контроллер прерываний 8259*A*. Появление запроса прерывания (*INT* = 1) указывает, что при выполнении численной команды (*Numeric Instruction*) произошел незамаскированный особый случай (*Exception*), когда запросы прерывания для *NDCP* были разрешены (разряд *M* = 1 в слове управления *CW* — см. рис. 4.40);

BUSY — сигнал занятости, подаваемый на вход $\overline{TEST}$ МП. Значение сигнала *BUSY* автоматически устанавливается в 1, когда исполнительное устройство *NEU* начинает выполнять численную команду. Команда WAIT, вставленная перед командой *NDCP*, инициирует проверку микропроцессором 8086 значения сигнала на входе $\overline{TEST}$. Пока значение $\overline{TEST}$ = 1, МП будет находиться в состоянии ожидания. По завершении текущей численной команды исполнительное устройство *NEU* переводит значение сигнала *BUSY* в 0 и МП выходит из состояния ожидания. Этим обеспечивается синхронизация работы МП и *NDCP* (МП не должен пропускать в *NDCP* его команды быстрее, чем *NDCP* может их выполнить). При возникновении незамаскированного особого случая сигнал *BUSY* остается активным (1), пока особый случай не будет сброшен;

READY — сигнал готовности, временно приостанавливающий работу *NDCP* при значении *READY* = 0. Этот сигнал используется, только если быстродействия памяти недостаточно для синхронной работы с *NDCP*. Сигнал готовности *RDY* от памяти поступает на генератор 8284

(см. рис. 4.13), который формирует общий для МП-системы 8086 + 8087 сигнал готовности *READY*. Значение сигнала *RDY* = 1 означает, что память завершила передачу данных;

RESET — сигнал сброса. Значение *RESET* = 1 длительностью не мене четырех тактов сигнала *CLK* заставляет *NDCP* немедленно закончить выполняемые операции. Сигнал сброса синхронизирован с сигналом *CLK* внутри *NDCP*;

CLK (*Clock*) — тактовый сигнал от внешнего генератора общий для МП и *NDCP* (частота сигнала 5, 8 и 10 МГц в зависимости от модели МП и *NDCP*). Оптимальная скважность сигнала *CLK* равна 3.

Устройство управления *CU*. В устройстве управления *CU* имеется очередь команд, аналогичная очереди команд МП. Команды, выбираемые из памяти микропроцессором 8086, представляют собой некоторую последовательность (смесь) команд МП 8086 и *NDCP* 8087 в едином потоке команд. *NDCP* распознает цикл выборки команды по сигналам состояния S_{2-0}, и при появлении байта или слова команды на шине адреса-данных AD_{15-0} устройство управления *CU* подключается к ней параллельно с МП и помещает команду в свою очередь команд. Устройство управления *CU*, анализируя значение сигнала BHE/S_7 сразу после сброса (*RESET* = 1), автоматически определяет, с каким типом МП (8086/80186 или 8088/80188) оно работает. В соответствии с типом МП устанавливается длина очереди: 6 или 4 байта. Контролируя сигналы состояния QS_1 и QS_0 очереди МП, устройство управления *CU* получает и декодирует команды из очереди синхронно с МП. Оба процессора (МП и *NDCP*) работают со своими очередями команд параллельно, но выполняют только свои команды.

Старшие пять разрядов D_{7-3} первого байта всех команд *NDCP* одинаковы (код 11011 — см. табл. 4.11). Эти разряды определяют команду ESC переключения с МП на *NDCP* — мнемоника ESC в системе команд МП заменяет собой мнемоники всех команд *NDCP*, но в ассемблерных программах не употребляется (вместо нее указывается одна из команд *NDCP*, мнемоника которых всегда начинается с буквы F — *Float*; см. § 4.7). Команды ESC устройство управления *CU* выполняет либо самостоятельно, либо передает в численное исполнительное устройство *NEU* (*New Instruction* на рис. 4.32), а команды МП игнорируются. Микропрограммное устройство управления *MCU* (см. рис. 4.32) принимает от устройства управления *CU* численные команды, декодирует их и генерирует последовательность микрокоманд общей длительностью от 10 до 1100 тактов.

МП различает два типа команд ESC. Если команда ESC не требует обращения к памяти, то МП просто переходит к выполнению следующей команды, находящейся в очереди. Если команда ESC требует обращения к памяти (чтение операнда из памяти или запись операнда в память), то МП вычисляет 20-разрядный физический адрес A_{19-0}, а затем выполняет "фиктивное чтение" ("*dummy read*") операнда из памяти при любом допустимом режиме адресации, указанном в команде ESC (этот операнд МП игнорирует). Устройство управления *CU* перехватывает этот физический адрес A_{19-0}, появляющийся на линиях AD_{15-0} и A_{19-16}/S_{6-3} при "фиктивном чтении", и помещает его в указатель данных *DP* (см. рис. 4.32).

Если принятая *NDCP* команда ESC требует чтения операнда из памяти, то устройство управления *CU* принимает слово операнда D_{15-0}, появляющееся на линиях AD_{15-0} при "фиктивном чтении". Принятого слова достаточно только в случае, если в команде указан операнд типа *Word Integer* (см. рис. 4.29). Если длина операнда превышает одно слово, то устройство управления *CU* сразу сигналом $\overline{RQ/GT_0}$ запрашивает у МП доступ к шине и читает из памяти остальную часть операнда в последовательных циклах шины, используя перехваченный физический адрес операнда A_{19-0}.

Если принятая *NDCP* команда ESC требует записи операнда в память, то устройство управления *CU* игнорирует слово операнда D_{15-0}, появляющееся на линиях AD_{15-0} в цикле "фик-

тивного чтения". Когда *NDCP* будет готов записать операнд в память, он по линии $\overline{RQ}/\overline{GT}_0$ посылает в МП запрос доступа к шине и записывает операнд в память, используя перехваченный физический адрес операнда A_{19-0}.

Программная модель МП-системы 8086 + 8087. С точки зрения программиста *NDCP* 8087 можно считать просто расширением МП 8086 — увеличено число программно доступных регистров и допустимых форматов чисел, а также расширена система команд. Форматы чисел *NDCP* были описаны в § 4.4 (см. рис. 4.29), а с системой команд рекомендуется бегло ознакомиться по табл. 4.32 (§ 4.7) уже сейчас, так как в этом параграфе будут использоваться команды *NDCP* (или по мере необходимости следует обращаться к табл. 4.32). Взаимодействие МП и *NDCP* на аппаратном уровне невидимо (прозрачно) для программ, как если бы был создан новый микропроцессор, содержащий на одном кристалле МП 8086 и *NDCP* 8087.

Программная модель МП-системы 8086 + 8087 содержит как регистры МП 8086, так и регистры *NDCP* 8087 (рис. 4.33), которые доступны программисту на уровне машинных команд. Основу программной модели *NDCP* составляет *регистровый стек* из восьми 80-разрядных регистров *R*7 ÷ *R*0. В этих регистрах хранятся числа, представленные во временном вещественном формате, и все вычисления выполняются в этом формате.

Состояние каждого регистра стека *R*7 ÷ *R*0 характеризуется двухразрядным *тэгом* (признаком):

00 — регистр содержит действительное число (конечное ненулевое число — *Valid*),

01 — регистр содержит истинный нуль (*Zero*),

10 — регистр содержит специальное число (*Special* — не-число, бесконечность, денормализованное число и др.),

11 — регистр не содержит данных (*Empty* — пустой).

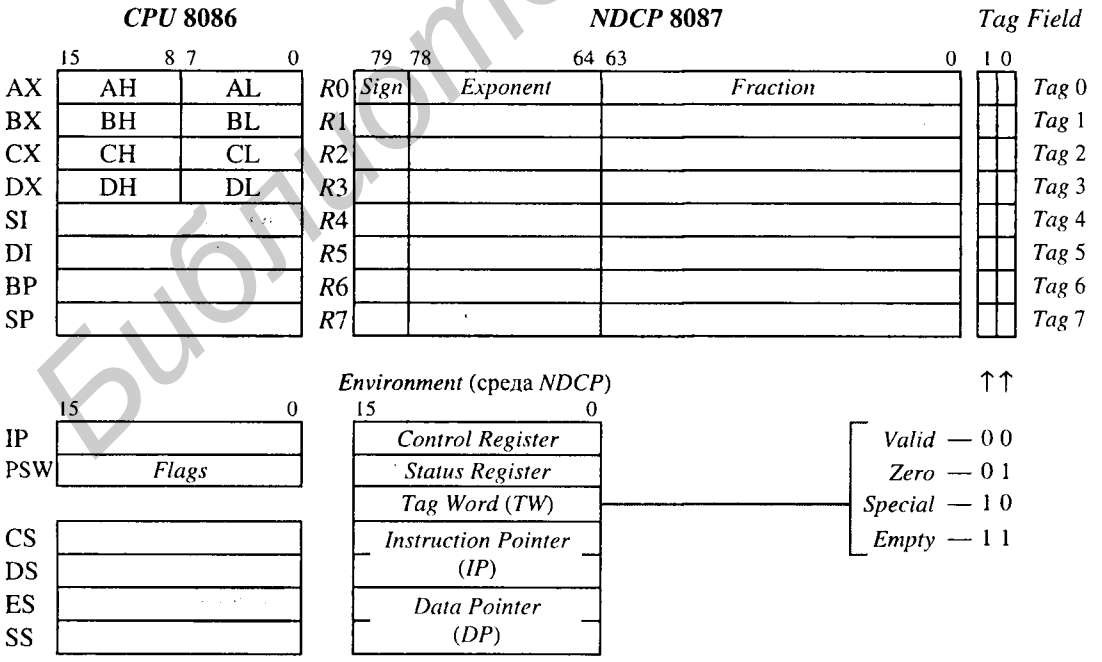


Рис. 4.33. Программная модель МП-системы 8086 + 8087

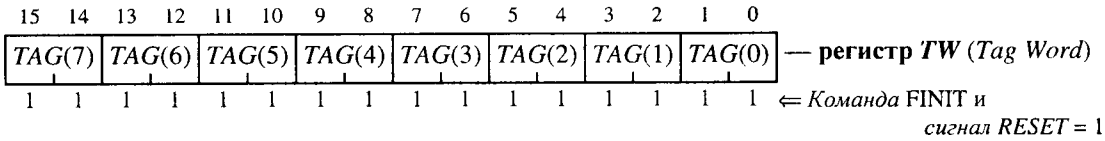


Рис. 4.34. Формат слова тэгов

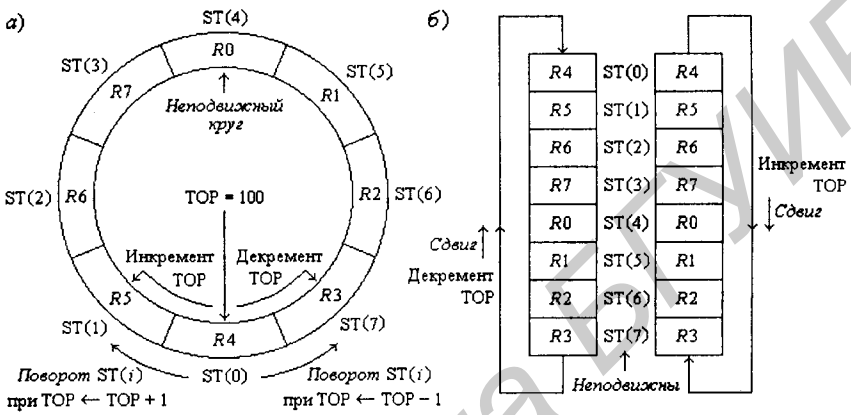


Рис. 4.35. Принцип работы стека

При загрузке в регистр операнда (*push* — включение) в тэг записывается двухразрядный код, не равный 11 (регистр занят), а при извлечении операнда из регистра (*pop*) в тэг записывается код 11 (регистр освобожден). Сопроцессор контролирует операции включения и извлечения операндов из стека — попытки загрузки операнда в занятый регистр и извлечения операнда из пустого регистра фиксируются в слове состояния *SW* как особые случаи недействительных операций (*IE* ← 1 — см. рис. 4.36).

Восемь двухразрядных *тэгов* составляют 16-разрядный регистр тэгов *TW* (*Tag Word* — слово тэгов; рис. 4.34). Регистр тэгов *TW* позволяет *NDCP* быстрее обнаруживать особые случаи и эффективнее обрабатывать специальные численные значения, включая и нуль. При аппаратном сбросе (сигнал *RESET* = 1) или при выполнении команды FINIT (см. табл. 4.32) инициализации *NDCP* все восемь тэгов отмечаются как пустые.

Адресуются регистры стека *R*7 ÷ *R*0 трехразрядным полем TOP (*Top of Stack Pointer* — указатель вершины стека) в слове состояния *SW* (см. рис. 4.36). Стек имеет круговую (кольцевую) организацию (рис. 4.35, *а*) — если поле TOP = 000 (адресуется регистр *R*0) и производится его декремент, то новое значение TOP будет равно 111 (адресуется регистр *R*7), а если поле TOP = 111 и выполняется его инкремент, то новое значение TOP будет равно 000.

Текущее значение поля TOP всегда указывает текущую вершину стека ST(0), которая используется в большинстве команд *NDCP* в качестве единственного или одного из двух операндов. Все остальные операнды ST(*i*) находятся ниже вершины стека ST(0) (рис. 4.35, *б* — тот же стек, что и на рис. 4.35, *а*, но регистры расположены в виде стопки). Операнды ST(*i*) адресуются относительно вершины стека ST(0) — адрес *i* операнда ST(*i*) отсчитывается относительно адреса *i* = 0. Операнды ST(*i*), или *относительные регистры*, также используются в командах *NDCP*, но нет команд, содержащих операнды *R*× (× = 0 ÷ 7) с прямой адресацией регистров. Не зная предысторию использования стека, ничего нельзя сказать о том, какой из физических ре-

гистров *R*× в данный момент находится в вершине стека. Контроль правильного использования (адресации) стека должен осуществлять программист.

Для интерпретации содержимого регистров стека *R*7 ÷ *R*0 программист может использовать слово тэгов *TW*. Так как размещение тэгов регистров в слове тэгов *TW* соответствует физическим регистрам *R*7 ÷ *R*0, то для ассоциирования их тэгов с относительными регистрами ST(7) ÷ ST(0) необходимо привлекать поле TOP из слова состояния *SW*.

Итак, программист работает относительно текущей вершины стека ST(0) и не употребляет абсолютных номеров (имен) регистров *R*×. Принятые в *NDCP* организация стека и адресация регистров *R*× упрощают программирование с использованием подпрограмм. Через стек удобно передавать подпрограмме параметры и осуществлять доступ к ним в процессе ее выполнения, а также обращаться к результатам вычислений после завершения подпрограммы.

При выполнении операции *включения* операнда в стек (*push*) осуществляется декремент поля TOP (рис. 4.35), и адресуемые данные загружаются в регистр, соответствующий новой вершине стека ST(0), причем производится изменение содержимого поля тэга этого регистра с 11 на 00, 01 или 10 в зависимости от типа загружаемого числа. Без промежуточных извлечений из стека подряд можно сделать не более восьми включений в стек. В противном случае происходит *переполнение стека* (*stack overflow*), фиксируемое в слове состояния *SW* как особый случай недействительной операции (*IE* ← 1 — см. рис. 4.36).

При операции *извлечения* из стека (*pop*) в получатель, которым чаще всего является память, передается содержимое текущей вершины стека ST(0) и содержимое поля тэга этого регистра изменятся с 00, 01 или 10 на 11, а затем производится инкремент поля TOP (рис. 4.35), назначая новую вершину стека ST(0). Без промежуточных включений в стек подряд можно сделать не более восьми извлечений из стека. В противном случае происходит *антипереполнение стека* (*stack underflow*), также фиксируемое в слове состояния *SW* как особый случай недействительной операции (*IE* ← 1).

И вообще, любая попытка загрузки операнда в занятый регистр (*Tag* ≠ 11) и извлечения операнда из пустого регистра (*Tag* = 11) фиксируется в слове состояния *SW* как особый случай недействительной операции (*IE* ← 1).

Кодирование чисел в вещественных форматах. Кроме нормализованных чисел трех вещественных форматов (см. рис. 4.29) *NDCP* может загружать из памяти, создавать, сохранять в памяти и обрабатывать числа другого типа (табл. 4.27). Для кодирования нормализованных (основных) чисел минимальное (00 ... 00) и максимальное (11 ... 11) значения смещенного порядка *E* = *exp* + *bias* не используются — эти порядки отведены для представления специальных значений: –0 и +0 (истинных нулей), –∞ и +∞ (бесконечностей), денормализованных чисел, не-чисел (*NAN — not a Number*) и *неопределенности*, или *NAN*-неопределенности (частного случая не-числа).

Примеры кодов некоторых чисел в трех вещественных форматах:

	Short Real	*Long Real*	*Temporary Real*
–0:	8000 0000	8000 0000 0000 0000	8000 0000 0000 0000 0000
+0:	0000 0000	0000 0000 0000 0000	0000 0000 0000 0000 0000
–∞	FF80 0000	FFF0 0000 0000 0000	FFFF 8000 0000 0000 0000
+∞	7F80 0000	7FF0 0000 0000 0000	7FFF 8000 0000 0000 0000
неопределенность:	FFC0 0000	FFF8 0000 0000 0000	FFFF C000 0000 0000 0000

Нуль. Число 0 (истинный нуль) во всех вещественных форматах имеет нулевой смещенный порядок 00 ... 00, нулевую мантиссу 0.00 ... 00 и положительный или отрицательный знак

(S = 0 или 1), но при выполнении операций знак нуля игнорируется. Так, квадратный корень $\sqrt{-0} = 0$, а не неопределенности.

Ненормализованные числа. Старший разряд мантиссы F нормализованных (*normalized*) чисел f_0 = 1 (явный разряд в формате *Temporary Real* и неявный в форматах *Short/Long Real*). Для ненормализованных (*unnormalized*) чисел разряд f_0 = 0, а диапазон смещенных порядков такой же, что и для нормализованных чисел. Ненормализованные числа существуют только во временном вещественном формате (в регистрах стека).

При попытке сохранения ненормализованного числа *Temporary Real* в памяти в форматах *Short/Long Real* в слове состояния *SW* фиксируется особый случай недействительной операции (*IE* ← 1 — см. рис. 4.36), и в память записывается код *NAN*-неопределенности FFC0 0000*r* или FFF8 0000 0000 0000*r* (ненормализованные числа в форматах *Short/Long Real* не существуют).

Таблица 4.27. **Кодирование чисел в вещественных форматах**

Тип численного значения	Знак S	Порядок E ..... e_0	Мантисса F $f_0 f_1$ .....
Нормализованное число (*Tag* = 00):			
минимальное *Short/Long Real*	×	00 ... 01	∴ 00 ... 00
максимальное *Short/Long Real*	×	11 ... 10	∴ 11 ... 11
минимальное *Temporary Real*	×	00 ... 01	1 00 ... 00
максимальное *Temporary Real*	×	11 ... 10	1 11 ... 11
Нуль (*Tag* = 01):			
Short/Long Real	×	00 ... 00	∴ 00 ... 00
Temporary Real	×	00 ... 00	0 00 ... 00
Ненормализованное число (*Tag* = 10):			
минимальное *Temporary Real*	×	00 ... 01	0 00 ... 00
максимальное *Temporary Real*	×	11... 10	0 11 ... 11
Денормализованное число (*Tag* = 10):			
минимальное *Short/Long Real*	×	00 ... 00	∴ 00 ... 01
максимальное *Short/Long Real*	×	00 ... 00	∴ 11 ... 11
минимальное *Temporary Real*	×	00 ... 00	0 00 ... 01
максимальное *Temporary Real*	×	00 ... 00	1 11 ... 11
Бесконечность (*Tag* = 10):			
Short/Long Real	×	11 ... 11	∴ 00 ... 00
Temporary Real	×	11 ... 11	1 00 ... 00
Не-числа (*Tag* = 10):			
минимальное *Short/Long Real*	×	11 ... 11	∴ 00 ... 01
максимальное *Short/Long Real*	×	11 ... 11	∴ 11 ... 11
неопределенность *Short/Long Real*	1	11 ... 11	∴ 10 ... 00
минимальное *Temporary Real*	×	11 ... 11	1 00 ... 01
максимальное *Temporary Real*	×	11 ... 11	1 11 ... 11
неопределенность *Temporary Real*	1	11 ... 11	1 10 ... 00

Примечание: × — 0 (знак плюс) и 1 (знак минус), ∴ — неявный (скрытый) разряд, $Se_7 \dots e_1 e_0 f_1 \dots f_{23}$ — *Short Real*, $Se_{10} \dots e_1 e_0 f_1 \dots f_{52}$ — *Long Real*, $Se_{14} \dots e_1 e_0 f_0 . f_1 \dots f_{63}$ — *Temporary Real*.

Денормализованные числа. При выполнении команд *NDCP* могут возникать столь малые числа, что их смещенный порядок $E = exp + bias$ должен быть отрицательным при значении разряда $f_0 = 1$. Это означает, что такие числа невозможно представить в нормализованной форме — для этого требуются значения $E \geq 1$ и $f_0 = 1$. Такая ситуация называется *исчезновением порядка* или *антипереполнением*. При антипереполнении *NDCP* сдвигает мантиссу вправо с одновременным инкрементом порядка до тех пор, пока он не будет равным нулю — при сдвигах в старшие разряды мантиссы поступают нули. Такие числа с нулевым смещенным порядком ($E = exp + bias = 0$) и ненулевой мантиссой называются *денормализованными*. Понятно, что старшие нули в мантиссе приводят к потере значимости, поэтому расширение диапазона в сторону малых чисел сопровождается потерей точности.

При загрузке из памяти в регистр стека денормализованного числа в форматах *Short/Long Real* оно автоматически нормализуется сдвигом мантиссы влево и декрементом порядка (в слове состояния *SW* фиксируется особый случай денормализованного операнда: $DE \leftarrow 1$ — см. рис. 4.36). В результате получается эквивалентное *нормализованное число* в формате *Temporary Real* — проблем в использовании денормализованных чисел возникнуть не может. Более того, в *NDCP* допускается использование денормализованных чисел в формате *Temporary Real*.

Сохранение в форматах *Short/Long Real* ранее загруженного в регистр стека денормализованного числа, но уже нормализованного, не изменяет его значения, т. е. в память будет записано исходное денормализованное число.

При сохранении в памяти в форматах *Short/Long Real* денормализованного числа в формате *Temporary Real*, находящегося в регистре стека, в слове состояния *SW* фиксируются особые случаи антипереполнения ($UE \leftarrow 1$) и потери точности ($PE \leftarrow 1$) — см. рис. 4.36, и в память записывается истинный нуль.

Расширение допустимых значений на денормализованные числа называется *плавным антипереполнением*. В большинстве прикладных программ денормализованные числа встречаются редко, так как очень малыми обычно бывают промежуточные, а не окончательные результаты, представляемые в форматах *Short/Long Real*. Так как внутри *NDCP* все операции производятся над числами, представленными в формате *Temporary Real*, то его огромный диапазон делает антипереполнения маловероятными. Действительно, трудно представить себе числа, имеющие какой-то практический смысл, большие, чем 10^{+4932} или меньшие, чем 10^{-4932}. При нехватке регистров стека промежуточные результаты можно сохранять в памяти также в формате *Temporary Real*.

Бесконечность. Во всех вещественных форматах допускаются $+\infty$ (положительная бесконечность) и $-\infty$ (отрицательная бесконечность), значение которых (по модулю) больше любого вещественного числа, представимого в конкретном вещественном формате. Бесконечности имеют смещенный порядок 11 ... 11 и мантиссу 1.00 ... 00. Бесконечности допускаются в качестве операндов в большинстве арифметических операций *NDCP* и обрабатываются по соответствующим правилам.

Не-числа. Все коды, за исключением кода бесконечности, имеющие смещенный порядок 11 ... 11, относятся к классу не-чисел (*NAN*-числа — *not a Number*). Результатом любой операции с не-числом является не-число, т. е. не-числа распространяются в вычислениях, передаются в окончательный результат и могут содержать информацию о месте возникновения ошибки.

Как и нормализованные числа, не-числа можно загружать из памяти в регистры стека и сохранять в памяти в любом вещественном формате.

Неопределенность. Под вещественной неопределенностью понимается специальный код, который относится к классу не-чисел (*NAN*-неопределенность), и выдается как результат операции над такими операндами, при которых никакой осмысленный ответ невозможен. Примерами могут служить извлечение квадратного корня из отрицательного числа, умножение ∞ на 0

и др. Во всех вещественных форматах неопределенность имеет отрицательный знак, смещенный порядок 11 ... 11 и мантиссу 1.100 ... 00.

Кодирование целых чисел. Для представления целых двоичных чисел используется дополнительный код (табл. 4.28: $n = 14$ для чисел типа *Word Integer*, $n = 30$ для чисел типа *Short Integer* и $n = 62$ для чисел типа *Long Integer*). Для представления неопределенности взят код наибольшего по модулю отрицательного числа (-2^{15}, -2^{31} и -2^{63} в соответствующих форматах), т. е. этот код имеет двойную интерпретацию. Целая двоичная неопределенность возникает только в одной недействительной операции — запоминание не-числа в целом формате.

Для представления целых десятичных упакованных чисел (Packed BCD) используется прямой код (табл. 4.29). В этом формате представимы целые числа из диапазона $\pm(10^{18} - 1)$.

Использования целой двоичной и десятичной неопределенностей следует избегать, так как *NDCP* считает двоичную неопределенность максимальным отрицательным числом, т. е. она не распространяется в вычислениях на другие результаты, а при загрузке десятичной неопределенности в регистр стека его содержимое оказывается неопределенным.

Таблица 4.28. **Кодирование целых двоичных чисел**

Тип численного значения	$S\ I_n I_{n-1} I_{n-2} \ldots I_2 I_1 I_0$
Целое положительное число:	
максимальное	0 111 ... 111
минимальное (+1)	0 000 ... 001
Нуль	0 000 ... 000
Целое отрицательное число:	
максимальное (–1)	1 111 ... 111
минимальное	1 000 ... 001
Неопределенность	1 000 ... 000

Таблица 4.29. **Кодирование целых десятичных упакованных чисел**

Тип численного значения	S - - -	- - - -	D_{17}	D_{16}	D_{15}	D_{14}	...	D_1	D_0
Целое положительное число:									
максимальное ($+10^{18} - 1$)	0000	0000	1001	1001	1001	1001	...	1001	1001
минимальное (+1)	0000	0000	0000	0000	0000	0000	...	0000	0001
Нуль (+0)	0000	0000	0000	0000	0000	0000	...	0000	0000
Нуль (–0)	1000	0000	0000	0000	0000	0000	...	0000	0000
Целое отрицательное число:									
максимальное (–1)	1000	0000	0000	0000	0000	0000	...	0000	0001
минимальное ($-10^{18} + 1$)	1000	0000	1001	1001	1001	1001	...	1001	1001
Неопределенность	1111	1111	1111	1111	xxxx	xxxx	...	xxxx	xxxx

Слово состояния *SW*. В слове состояния *SW* (*Status Word*; рис. 4.36) младшие шесть разрядов отведены для регистрации особых случаев (*Exception Flags* — флаги особых случаев), а в слове управления *CW* (*Control Word*; см. рис. 4.40) эти же разряды используются для их индивидуального маскирования. Особые случаи регистрируются в слове состояния *SW* независимо от состояния маски, но разряд *IR* запроса прерывания устанавливается в состояние 1 только при возникновении какого-либо незамаскированного особого случая. Фактическое генерирование запроса прерывания (значения сигнала *INT* = 1 — см. рис. 4.31), вызывающего *подпрограмму обработки особого случая*, зависит от состояния разряда *M* в слове управления *CW* (*M* — общее маскирование для всех особых случаев).

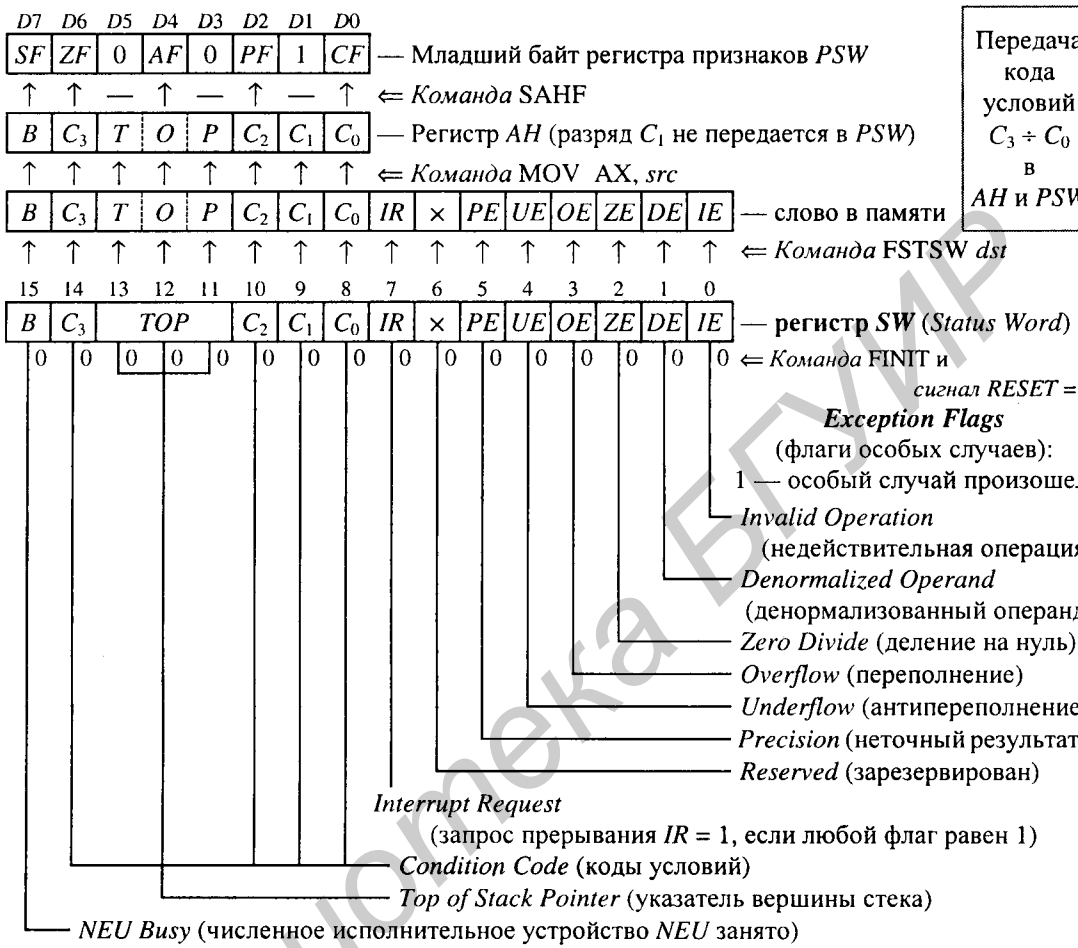


Рис. 4.36. Слово состояния *SW* сопроцессора 8087

Если особый случай замаскирован, *NDCP* исполняет встроенную процедуру *маскированной реакции* без вызова подпрограммы обработки особого случая и выполнение программы продолжается. В большинстве особых случаев *NDCP* выдает наиболее естественный численный результат, который может использоваться в дальнейших вычислениях. Только в особом случае недействительной операции сопроцессор 8087 в качестве результата выполнения команды выдает не-число. Поэтому в большинстве прикладных программ рекомендуется маскировать все особые случаи, за исключением особого случая недействительной операции.

В процессе выполнения команд *NDCP* особые случаи будут накапливаться, поэтому предусмотрена команда FCLEX сброса всех зарегистрированных особых случаев. Флаги особых случаев слова состояния *SW* имеют назначение:

IE — регистрация недействительных операций ($-\infty + \infty$, $\infty \times 0$, $\pm\infty \times \pm\infty$, $0 / 0$, $\pm\infty / \pm\infty$, $\sqrt{-x}$ при $x < 0$, любые операции с не-числами, загрузка операнда в занятый и извлечение операнда из пустого регистра стека и др.). Маскированная реакция *NDCP* — выдача не-числа, если им является один из операндов, не-числа с большей мантиссой, когда не-числами являются оба

операнда и кода *NAN*-неопределенности в остальных случаях. Из этого видно, что после возникновения не-числа, например *NAN*-неопределенности, оно не исчезает, а будет распространяться в последующих вычислениях. Особый случай недействительной операции свидетельствует, как правило, об ошибке в программе;

DE — регистрация использования в команде денормализованного операнда или получения денормализованного результата. Маскированная реакция *NDCP* — продолжается нормальная работа;

ZE — регистрация деления ненулевого конечного числа (±*N*) на нуль (±0). Маскированная реакция *NDCP* — выдача кода бесконечности с соответствующим знаком;

OE — регистрация переполнения (порядок истинного результата больше, чем максимальный допустимый порядок получателя). Маскированная реакция *NDCP* — выдача кода бесконечности;

UE — регистрация антипереполнения (порядок истинного результата слишком мал для соответствующих форматов вещественных чисел). Маскированная реакция *NDCP* — денормализация результата (сдвиг мантиссы вправо) пока порядок выходит за допустимый диапазон (плавное антипереполнение);

PE — регистрация потери точности (результат выполнения команды не может быть точно представлен в формате получателя). Этот особый случай возникает довольно часто и указывает лишь на то, что происходит некоторая, обычно приемлемая, потеря точности. Флаг *PE* предназначен для применений, в которых требуется исключительно точное выполнение операций. Маскированная реакция *NDCP* — производится округление результата согласно заданному в слове управления *CW* режиму округления и продолжается нормальная работа.

Обнаружение особых случаев (установка значения 1 флагов) недействительной операции, деления на нуль и денормализованного операнда происходит до выполнения операции, а остальные особые случаи регистрируются только после вычисления истинного результата и сохранения его в регистре стека или в памяти.

Разряды $C_3 \div C_0$ фиксируют код условия, характеризующий некоторые особенности результата выполнения команд сравнения (FCOM), проверки (FTST), анализа (FXAM) и команды вычисления частичного остатка (FPREM).

Три разряда TOP представляют собой указатель вершины стека. При выполнении многих команд *NDCP* поле TOP автоматически модифицируется (TOP ← TOP ± 1). В описаниях команд и комментариях к программам для указателя TOP будем использовать обозначение *Top*.

Флаг занятости *B* устанавливается в состояние 1, когда численное операционное устройство *NEU* занято выполнением текущей команды. Состояние этого разряда выведено в *NDCP* как сигнал *BUSY* (см. рис. 4.31), информирующий МП о необходимости приостановки дальнейших выборок команд из очереди команд до завершения выполнением текущей команды *NDCP* (до тех пор, пока не будет выдано значение сигнала *BUSY* = 0).

Слово состояния *SW* командой FSTSW *dst* можно записать в память, а затем проанализировать программой МП коды условий $C_3 \div C_0$. Все разряды слова состояния *SW* устанавливается в 0 при инициализации *NDCP* командой FINIT или значением сигнала *RESET* = 1.

Код условия $C_3 \div C_0$ фиксирует особенности результата, получаемого при выполнении некоторых команд *NDCP* (табл. 4.30). При выполнении команд сравнения двух чисел (*Compare*) и одного числа с нулем (*Test*) код условия $C_3 \div C_0$ используется для реализации условных переходов на основании отношений чисел “>”, “<” и “=”. Самостоятельно *NDCP* не может влиять на ход выполнения программы, поэтому для организации условных переходов по результатам выполнения указанных команд приходится сначала передавать код условия $C_3 \div C_0$ в память, затем из памяти в регистр *AH*, а из регистра *AH* в регистр признаков *PSW* (см. рис. 4.36).

Таблица 4.30. **Интерпретация кода условия**

Тип инструкции	C_3	C_2	C_1	C_0	Интерпретация кода условия
Compare: FCOM *src* FCOMP *src* FCOMPP *Test*: FTST	0	0	×	0	ST(0) > *src* (*Compare*), ST(0) > 0 (*Test*)
	0	0	×	1	ST(0) < *src* (*Compare*), ST(0) < 0 (*Test*)
	1	0	×	0	ST(0) = *src* (*Compare*), ST(0) = 0 (*Test*)
	1	1	×	1	Не сравнимы (*Compare* и *Test*)
Remainder: FPREM	Q_1	0	Q_0	Q_2	Полное приведение с получением младших трех разрядов $Q_2Q_1Q_0$ частного (табл. 4.31) — вычисление завершено
	×	1	×	×	Неполное приведение — вычисление не завершено
Examine: FXAM	0	0	0	0	Положительное ненормализованное
	0	0	0	1	Положительное не-число
	0	0	1	0	Отрицательное ненормализованное
	0	0	1	1	Отрицательное не-число
	0	1	0	0	Положительное нормализованное
	0	1	0	1	+∞ (положительная бесконечность)
	0	1	1	0	Отрицательное нормализованное
	0	1	1	1	–∞ (отрицательная бесконечность)
	1	0	0	0	+0 (положительный нуль)
	1	0	0	1	Регистр пустой
	1	0	1	0	–0 (отрицательный нуль)
	1	0	1	1	Регистр пустой
	1	1	0	0	Положительное денормализованное
	1	1	0	1	Регистр пустой
	1	1	1	0	Отрицательное денормализованное
	1	1	1	1	Регистр пустой

Таблица 4.31. **Интерпретация кода условия после команды FPREM**

Диапазон делимого	Q_2	Q_1	Q_0
Dividend < 2 × *Modulus*	C_3*	C_1*	Q_0
Dividend < 4 × *Modulus*	C_3*	Q_1	Q_0
Dividend ≥ 4 × *Modulus*	Q_2	Q_1	Q_0

Примечание: * на предыдущее значение разряда команда не воздействовала.

После передачи кода условия $C_3 \div C_0$ в регистр признаков *PSW* можно выполнять команды некоторых условных переходов по флагам $ZF = C_3$, $PF = C_2$ и $CF = C_0$. Разряд C_1 кода условия передать в регистр признаков *PSW* невозможно, но его и не нужно использовать — C_1 = × (неопределенное значение) для команд сравнения (*Compare*) и проверки (*Test*).

Пример 1 (иллюстрация реализации условных переходов):

```
                ∴                   ; В сегменте данных
num_1   dd      –9.0                ; C1100000h
num_2   dd      +7.5                ; 40F00000h
buff    dw      ?                   ; Резервирование слова памяти
                ∴                   ; В сегменте кода
        FLD     num_1               ; Top ← Top – 1, ST(0) ← M(num_1) = C1100000h
        FCOM    num_2               ; Оценка разности ST(0) – M(num_2) = M(num_1) – M(num_2)
```

```
        FSTSW  buff        ; M(buff) ← SW (Status Word)
        MOV    AH, byte ptr buff + 1  ; AH ← старший байт SW
        SAHF               ; Младший байт PSW ← AH
        JP     Noncom      ; Переход, если числа M(num_1) и M(num_2) несравнимы
        JE     Equal       ; Переход, если M(num_1) = M(num_2)
        JB     Below       ; Переход, если M(num_1) < M(num_2)
          ∴                ; Продолжение программы для случая M(num_1) > M(num_2)
Equal:    ∴                ; Продолжение программы при равенстве чисел
          ∴
Below:    ∴                ; Продолжение программы для случая M(num_1) < M(num_2)
          ∴
Noncom:   ∴                ; Программа принятия решения при несравнимых числах
```

Команда вычисления частичного остатка по некоторому модулю FPREM производит последовательные вычитания заданного модуля (*Modulus*) из делимого (*Dividend*) до тех пор, пока остаток не будет меньше модуля. Если делимое значительно больше модуля, то команда FPREM будет выполняться долго, а МП в это время будет находиться в состоянии ожидания и не сможет обслуживать запросы прерываний от внешних устройств. Поэтому по команде FPREM производится не более 64 вычитаний, а значит, по одной команде FPREM вычисление может быть не завершено. Определить завершение вычисления остатка или перейти к повторению команды FPREM можно на основании значения флага $PF = C_2$, передав, конечно, сначала код условия $C_3 \div C_0$ в регистр признаков *PSW*.

Следует отметить, что флаги условий *ZF* и *CF* имеют одинаковый смысл как для МП, так и для *NDCP*: флаг нуля $ZF = 1$ при равенстве сравниваемых чисел, а флаг переноса $CF = 1$ только при вычитании из меньшего числа большего числа. Флаг же паритета *PF* имеет другой смысл — $PF = C_2 = 1$ при несравнимых операндах и неполном вычислении остатка. Из табл. 4.13 следует, что для условных переходов по результатам команд *NDCP* типа *Compare* и *Test*, можно использовать команды условных переходов JZ/JE и JNZ/JNE (по флагу $ZF = C_3$), JB/JNAE/JC и JNB/JAE/JNC (по флагу $CF = C_0$), JP/JPE и JNP/JPO (по флагу $PF = C_2$), JBE/JNA и JNBE/JA (по флагам $CF = C_0$ или $ZF = C_3$).

Пример 2 (иллюстрация вычисления частичного остатка):

```
num_1   dd     44762.0  ; 472EDA00h                  В сегменте данных
num_2   dd     653.0    ; 44234000h
buff    dw     ?        ; Резервирование слова памяти
          ∴              ; В сегменте кода
        FLD    num_2    ; Top ← Top – 1, ST(0) ← M(num_2) — Modulus
        FLD    num_1    ; Top ← Top – 1, ST(0) ← M(num_1) — Dividend
Rem:    FPREM           ; M(num_1) – q × M(num_2) — частичный остаток
        FSTSW  buff     ; M(buff) ← SW (Status Word)
        MOV    AX, buff ; AX ← M(buff) = SW
        SAHF            ; В младший байт PSW ← AH (флаг PF = C2)
        JP     Rem      ; Переход, если вычисление остатка не завершено
        FIST   buff     ; M(buff) ← остаток
        MOV    DX, buff ; DX ← 0166h = 358d — остаток
```

Команда анализа числа (*Examine*) FXAM выдает четырехразрядный код $C_3 \div C_0$, характеризующий число, находящееся в вершине стека ST(0).

Пример 3 (загрузка и анализ чисел вещественных форматов):

```
nor_q   dq   792.33          ; 4088C2A3D70A3D70h                    В сегменте данных
mz_d    dd   –0.0            ; 80000000h — отрицательный нуль
in_d    dd   7F800000r       ; 7F800000h — +∞ (+INF)
unn_t   dt   3FFF7000000000000001r ; Положительное ненормализованное число
den_d   dd   00000001h               ; Минимальное положит. денормализованное число
den_q   dq   0000000000000001h       ; Минимальное положит. денормализованное число
den_t   dt   0000002FFFFFFFFFFFFFh ; Положительное денормализованное число
nan_d   dd   7F800001r               ; 7F800001h — положительное не-число (+NAN)
        FINIT            ; Инициализация NDCP 8087                  В сегменте кода
        FLD   nor_q      ; Top ← Top – 1, ST(0) ← M(nor_q) = 4008 C615 1EB8 51EB 8000h
        FXAM             ; C3C2C1C0 = 0100, Tag = 00 (Valid)
        FLD   mz_d       ; Top ← Top – 1, ST(0) ← M(mz_d) = 80000000h
        FXAM             ; C3C2C1C0 = 1010, Tag = 01 (Zero)
        FLD   in_d       ; Top ← Top – 1, ST(0) ← M(in_d) = 7F800000h
        FXAM             ; C3C2C1C0 = 0101, Tag = 10 (Spec.)
        FLD   unn_t      ; Top ← Top – 1, ST(0) ← M(unn_t) = 3FFF7000000000000001h
        FXAM             ; C3C2C1C0 = 0000, Tag = 10 (Spec.)
        FLD   den_d      ; Top ← Top – 1, DE ← 1, ST(0) ← M(den_d) = 00000001h
        FXAM             ; C3C2C1C0 = 0100, Tag = 00 (Valid)
        FCLEX            ; Сброс флагов особых случаев
        FLD   den_q      ; Top ← Top – 1, DE ← 1, ST(0) ← M(den_q) = 0000000000000001h
        FXAM             ; C3C2C1C0 = 0100, Tag = 00 (Valid)
        FCLEX            ; Сброс флагов особых случаев
        FLD   den_t      ; Top ← Top – 1, ST(0) ← M(den_t) = 0000002FFFFFFFFFFFFFh
        FXAM             ; C3C2C1C0 = 1100, Tag = 10 (Spec.)
        FLD   nan_d      ; Top ← Top – 1, ID ← 1, ST(0) ← M(nan_d) = 7F800001h
        FXAM             ; C3C2C1C0 = 0001, Tag = 10 (Spec.)
; Содержимое окна NDCP 8087 отладчика на этот момент времени показано на рис. 4.37
        FCLEX            ; Сброс флагов особых случаев
        FLD   mz_d       ; Top ← Top – 1, ID ← 1, ST(0) ← –NAN = FFFF C000 0000 0000h
        FXAM             ; C3C2C1C0 = 0011, Tag = 10 (Spec.)
```

Tag	ST(*i*)	Значение операнда	Содержимое стека					CW	SW		ie	de	cc
Spec.	ST(0)	+NAN	7FFF	C000	0100	0000	0000	*im* = 1	*ie* = 1	*nan_d*	1	0	1
Spec.	ST(1)	0	0000	002F	FFFF	FFFF	FFFF	*dm* = 1	*de* = 0	*den_t*	0	0	C
Valid	ST(2)	4.94065 … e–324	3BCD	8000	0000	0000	0000	*zm* = 1	*ze* = 0	*den_q*	0	1	4
Valid	ST(3)	–1.40129 … e–45	BF6A	8000	0000	0000	0000	*om* = 1	*oe* = 0	*den_d*	0	1	4
Spec.	ST(4)	0.875	3FFF	7000	0000	0000	0001	*um* = 1	*ue* = 0	*unn_t*	0	0	0
Spec.	ST(5)	+INF	7FFF	8000	0000	0000	0000	*pm* = 1	*pe* = 0	*in_d*	0	0	5
Zero	ST(6)	–0	8000	0000	0000	0000	0000	*iem* = 0	*ir* = 0	*mz_d*	0	0	A
Valid	ST(7)	792.32999999 …	4008	C615	1EB8	51EB	8000	*pc* = 3	*cc* = 1	*nor_q*	0	0	4
								rc = 0	*st* = 0				
								ic = 0					

Рис. 4.37. Окно *NDCP* отладчика

На рис. 4.37 показана информация, выводимая отладчиком для *NDCP* 8087 (*iem* = *M*, *cc* = $C_3C_2C_1C_0$, *st* = TOP). Справа от окна отладчика приведены значения флагов и код условия, которые изменяются при загрузке в стек чисел и их анализе командой FXAM. После выполнения команды FINIT в левом столбце окна отладчика появляются слова *Empty* — все регистры пустые (*Tag* = 11). Последняя команда производит загрузку в уже занятый регистр стека, поэтому загружается не число –0, а не-число *–NAN* (неопределенность), и устанавливается значение флага недействительной операции *IE* = 1.

Пример 4 (минимальные и максимальные значения положительных нормализованных чисел):

```
        ∴                                   ; В сегменте данных
d0      dd      00800000h                   ; Минимальное число в формате Short Real
d1      dd      7F7FFFFFr                   ; Максимальное число в формате Short Real
q0      dq      0010000000000000h           ; Минимальное число в формате Long Real
q1      dq      7FEFFFFFFFFFFFFFr           ; Максимальное число в формате Long Real
t0      dt      00018000000000000000h       ; Минимальное число в формате Temporary Real
t1      dt      7FFEFFFFFFFFFFFFFFFFr       ; Максимальное число в формате Temporary Real
        ∴               ; В сегменте кода
        FINIT           ; Инициализация NDCP 8087
        FLD     d0      ; Top ← Top – 1, ST(0) ← M(d0)
        FLD     d1      ; Top ← Top – 1, ST(0) ← M(d1)
        FLD     q0      ; Top ← Top – 1, ST(0) ← M(q0)
        FLD     q1      ; Top ← Top – 1, ST(0) ← M(q1)
        FLD     t0      ; Top ← Top – 1, ST(0) ← M(t0)
        FLD     t1      ; Top ← Top – 1, ST(0) ← M(t1)
```

На рис. 4.38 показано окно *NDCP* отладчика после выполнения последней команды FLD загрузки чисел в регистры стека (два регистра стека, отмеченные признаком *Empty*, остались свободными).

Tag	ST(*i*)	← *Значение операнда* →	← *Содержимое стека* →					*CW*	*SW*	
Valid	ST(0)	1.1897314953572318e+4932	7FFE	FFFF	FFFF	FFFF	FFFF	*im* = 1	*ie* = 0	*t*1
Valid	ST(1)	3.3621031431120935e–4932	0001	8000	0000	0000	0000	*dm* = 1	*de* = 0	*t*0
Valid	ST(2)	1.7976931348623157e+308	43FE	FFFF	FFFF	FFFF	F800	*zm* = 1	*ze* = 0	*q*1
Valid	ST(3)	2.2250738585072014e–308	3C01	8000	0000	0000	0000	*om* = 1	*oe* = 0	*q*0
Valid	ST(4)	3.4028234663852886e+38	407E	FFFF	FF00	0000	0000	*um* = 1	*ue* = 0	*d*1
Valid	ST(5)	1.1754943508222875e–38	3F81	8000	0000	0000	0000	*pm* = 1	*pe* = 0	*d*0
Empty	ST(6)		7FFF	C000	0100	0000	0000	*iem* = 0	*ir* = 0	
Empty	ST(7)		0000	002F	FFFF	FFFF	FFFF	*pc* = 3	*cc* = 0	
								rc = 0	*st* = 2	
								ic = 0		

Рис. 4.38. Окно *NDCP* отладчика

Пример 5 (маскированные реакции *NDCP* на особые случаи):

```
nr      dq      792.33  ; 4088C2A3D70A3D70h                    В сегменте данных
dn      dt      0000FFFFFFFFFFFFFFFFh   ; Денормализованное число
buf_d   dd      'DD'    ; 00004444h — резервирование памяти
buf_q   dq      'DQ'    ; 0000000000004451h — резервирование памяти
          ∴             ; В сегменте кода
        FINIT           ; Инициализация NDCP
        FLD     nr      ; ST(0) ← M(nr)
        FLD     nr      ; ST(0) ← M(nr)
        FMUL    nr      ; PE ← 1, ST(0) ← M(nr) × M(nr)
        FCLEX           ; Сброс флагов особых случаев
        FST     buf_d   ; PE ← 1, M(buf_d) ← 4919 44ADh
        FCLEX           ; Сброс флагов особых случаев
        FST     buf_q   ; PE ← 1, M(buf_q) ← 4123 2895 A865 94AEh
        FCLEX           ; Сброс флагов особых случаев
        FLD     dn      ; ST(0) ← M(dn)
        FLD     dn      ; ST(0) ← M(dn)
        FMUL    nr      ; DE ← 1, PE ← 1, ST(0) ← M(dn) × M(nr)
        FCLEX           ; Сброс флагов особых случаев
        FST     buf_d   ; UE ← 1, PE ← 1, M(buf_d) ← 0 Short Real
        FCLEX           ; Сброс флагов особых случаев
        FST     buf_q   ; UE ← 1, PE ← 1, M(buf_q) ← 0 Long Real
```

Tag	ST(*i*)	← *Значение операнда* →	← *Содержимое стека* →					*CW*	*SW*	
Valid	ST(0)	5.3277903667 … e–4929	000B	C615	1EB8	51EB	7FFF	*im* = 1	*ie* = 0	*dn*×*nr*
Spec.	ST(1)	0	0000	FFFF	FFFF	FFFF	FFFF	*dm* = 1	*de* = 0	*dn*
Valid	ST(2)	627786.82889999988	4012	9944	AD43	2CA5	728C	*zm* = 1	*ze* = 0	*nr*×*nr*
Valid	ST(3)	792.32999999999993	4008	C615	1EB8	51EB	8000	*om* = 1	*oe* = 0	*nr*
Empty	ST(4)		0000	0000	0000	0000	0000	*um* = 1	*ue* = 1	
Empty	ST(5)		0000	0000	0000	0000	0000	*pm* = 1	*pe* = 1	
Empty	ST(6)		8000	0000	0000	0000	0000	*iem* = 0	*ir* = 0	
Empty	ST(7)		4008	C615	1EB8	51EB	8000	*pc* = 3	*cc* = 0	
								rc = 0	*st* = 4	
								ic = 0		

Рис. 4.39. Окно *NDCP* отладчика

На рис. 4.39 показано окно *NDCP* отладчика после выполнения последней команды FST сохранения чисел в памяти.

Слово управления *CW*. Формат слова управления *CW* (*Control Word*) показан на рис. 4.40. Для каждой задачи программист может выбрать один из нескольких вариантов обработки численных данных в *NDCP*. Можно задать индивидуальное маскирование тех или иных особых случаев (*Exception Masks* — шесть младших разрядов слова управления *CW*), разрешить (*M* = 0) или запретить (*M* = 1) запросы прерывания при возникновении особых случаев, выбрать точность вычислений (разряды *PC*) и способ округления (разряды *RC*), а также задать интерпретацию бесконечности (разряд *IC*).

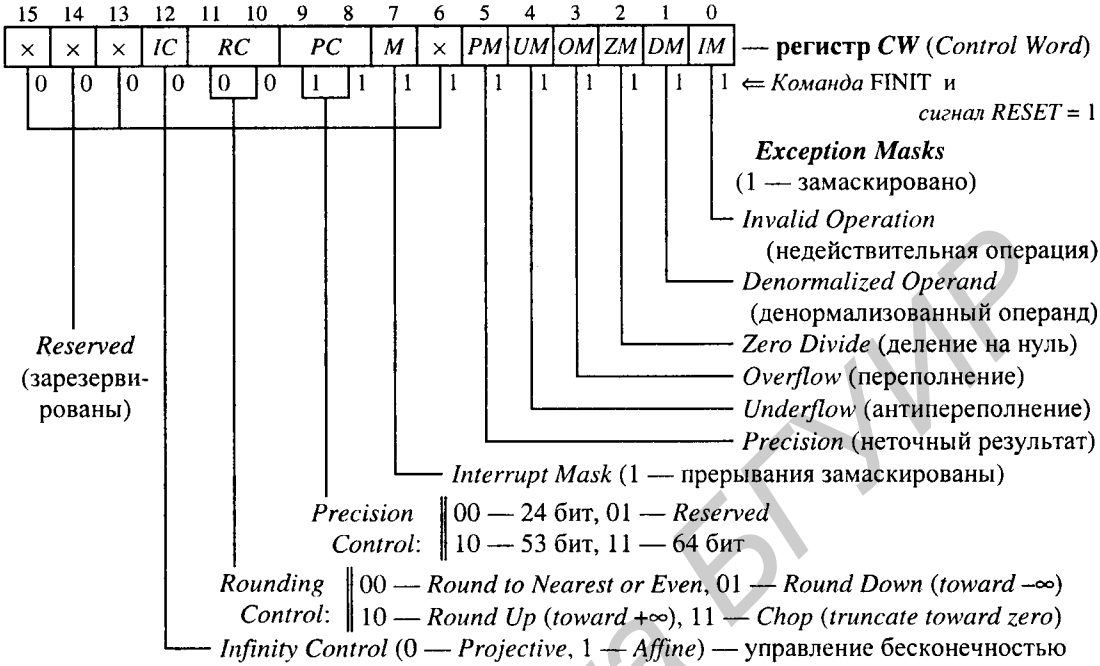


Рис. 4.40. Слово управления *CW* сопроцессора

Управление точностью. Двухразрядное поле управления точностью *PC* определяет точность вычислений в 24 бита (*PC* = 00), 53 бита (*PC* = 10) или 64 бита (*PC* = 11). По умолчанию устанавливается режим с максимальной точностью в 64 бита. Остальные режимы предусмотрены для эмуляции работы процессоров с 24- и 53-разрядной мантиссой. В этих режимах результат каждой операции округляется до указанного размера. Задание пониженной точности уничтожает достоинства временного вещественного формата по достижению максимальной точности, а производительность *NDCP* не увеличивается.

Управление округлением. Двухразрядное поле управления округлением *RC* определяет один из четырех возможных вариантов округления результатов операций *NDCP*:

RC = 00 — округление к ближайшему или четному числу,
RC = 01 — округление вниз (к −∞),
RC = 10 — округление вверх (к +∞),
RC = 11 — отбрасывание (усечение к нулю).

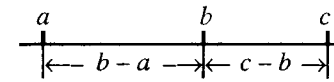


Рис. 4.41. Округление

Если результат *b* операции точно не представим в формате *NDCP*, то существуют представимые в этом формате числа *a* и *c*, наиболее близкие к *b*, такие, что $a < b < c$. Округление заключается в замене *b* на *a* или *c* одним из способов (рис. 4.41):

$b = a$ при $|b - a| < |c - b|$; $b = c$ при $|b - a| > |c - b|$; $b = a$ при $|b - a| = |c - b|$, если *a* четное, или $b = c$ при $|b - a| = |c - b|$, если *c* четное (*RC* = 00);

$b = a$ — округление вниз (*RC* = 01); $b = c$ — округление вверх (*RC* = 10);

$b = a$ при $a > 0$, $b = c$ при $c < 0$ (*RC* = 11).

Округление только вниз или только вверх при $|b - a| = |c - b|$ привело бы к появлению смещения результата. Чтобы исключить смещение, округление осуществляется к четному чис-

лу *a* или *c*, т. е. к числу, младший разряд которого содержит нуль. Следовательно, в среднем результаты округляются и вверх и вниз. Такой способ округления называется *несмещенным округлением к ближайшему четному*. Для реализации округления в *NDCP* имеются три дополнительных младших разряда мантисс.

Округление вверх и округление вниз, называемые *направленным округлением*, применяются в интервальной арифметике в целях получения достоверного результата независимо от ошибок округления. Верхняя и нижняя границы интервала значений результата находятся путем выполнения вычислений два раза: сначала с округлением вверх, а затем — с округлением вниз.

Режим округления с усечением к нулю применяется в целочисленной арифметике.

Округление производится как при выполнении арифметических операций, так и при записи в память, если формат получателя не позволяет точно представить истинный результат. Округление вносит погрешность, не превышающую единицы младшего разряда сохраняемого результата. По умолчанию принимается режим округления к ближайшему, обеспечивающий наиболее точную и статистически несмещенную оценку результатов.

Управление бесконечностью. Разряд *IC* используется для управления замыканием числового пространства в бесконечности: может быть задано или аффинное замыкание — $\pm\infty$ (*IC* = 1), или проективное замыкание — ∞ (*IC* = 0). Проективная бесконечность не имеет знака, т. е. два специальных значения "плюс бесконечность" и "минус бесконечность" обрабатываются как одно и то же значение "бесконечность", не имеющее знака. Аффинная бесконечность имеет знак плюс и минус. По умолчанию устанавливается проективная бесконечность.

Слово управления *CW* командой FSTCW *dst* можно записать в память, а командой FLDCW *src* загрузить из памяти в регистр *CW*. Кроме того, содержимое регистра слова управления *CW* устанавливается при инициализации *NDCP* командой FINIT или значением сигнала *RESET* = 1 и может быть оставлено по умолчанию.

Указатель особых случаев. Указатель особых случаев содержит указатель инструкции *IP* и указатель данных *DP* (см. рис. 4.33), в которые устройство управления *CU* помещает текущие физические адреса команды и операнда, если он имеется, а также 11-разрядный код операции выполняемой команды (рис. 4.42). Содержимое указателя особых случаев позволяет точно определить команду и операнд, вызвавшие особый случай, и установить причину его возникновения. Информация, содержащаяся в четырех 16-разрядных регистрах указателя особых случаев, предназначена только для процедур обработки особых случаев — такие процедуры обычно входят в состав системного программного обеспечения.

Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
+0	0	0	0	*IC*	*RC*		*PC*		*M*	0	*PM*	*UM*	*OM*	*ZM*	*DM*	*IM*	*Control Word*
+2	*B*	C_3	*TOP*			C_2	C_1	C_0	*IR*	0	*PE*	*UE*	*OE*	*ZE*	*DE*	*IE*	*Status Word*
+4	*TAG*(7)		*TAG*(6)		*TAG*(5)		*TAG*(4)		*TAG*(3)		*TAG*(2)		*TAG*(1)		*TAG*(0)		*Tag Word*
+6	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	*Instruction*
+8	A_{19}	A_{18}	A_{17}	A_{16}	0	*Instruction Opcode* (11-разрядный код операции)											*Pointer* (*IP*)
+10	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	*Data*
+12	A_{19}	A_{18}	A_{17}	A_{16}	0	0	0	0	0	0	0	0	0	0	0	0	*Pointer* (*DP*)

Рис. 4.42. Формат среды *NDCP* 8087 и ее сохранение в памяти

Среда и полное состояние *NDCP*. Содержимое регистров слова управления *CW*, слова состояния *SW*, слова тэгов *TW*, указателя инструкции *IP* и указателя данных *DP* называется *средой NDCP* (*Environment* — 7 слов; рис. 4.42).

Среда *NDCP* и содержимое регистрового стека называется *полным состоянием NDCP* (47 слов; рис. 4.43). Имеются команды сохранения в памяти и загрузки из памяти среды и полного состояния: FSTENV *dst* и FLDENV *src* — сохранение в памяти и загрузка из памяти среды, FSAVE *dst* и FRSTOR *src* — сохранение в памяти и загрузка из памяти полного состояния.

Команда FSTENV *dst* используется в основном в подпрограммах обработки незамаскированных особых случаев, так как предоставляет детальную информацию о причине, вызвавшей особый случай.

Команды FSAVE *dst* и FRSTOR *src* необходимы в подпрограммах обработки запросов прерываний от внешних устройств, в которых используются команды *NDCP*. Поскольку при вызове подпрограммы происходит переключение на решение другой задачи, то текущее состояние следует сохранить в памяти, а при возврате из подпрограммы восстановить его. Для изучения арифметического сопроцессора 8087 дополнительно можно привлечь литературу [13, 20, 21].

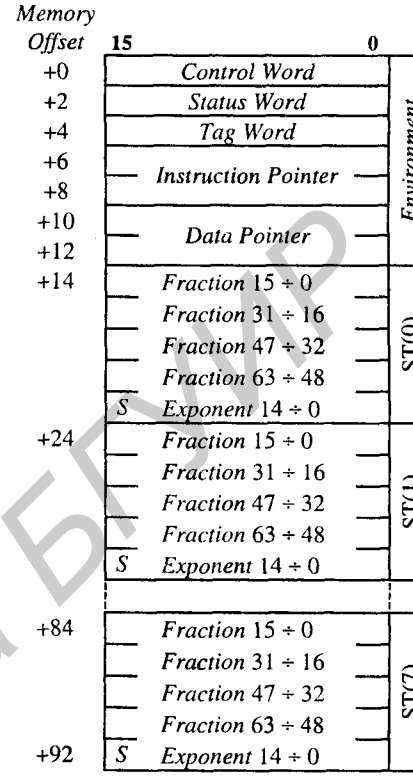


Рис. 4.43. Формат полного состояния

4.7. Система команд арифметического сопроцессора 8087

По назначению команды *NDCP* разделяются на 6 групп: команды передачи данных (*Data Transfer*), команды сравнения (*Comparison*), команды загрузки констант (*Constants*), арифметические команды (*Arithmetic*), трансцендентные команды (*Transcendental*) и команды управления сопроцессором (*Processor Control*).

Форматы команд *NDCP* 8087. Ниже приведены форматы всех команд *NDCP*. Если команда способна адресовать операнд в памяти (в команде есть поле *r/m*), то она может содержать еще один или два байта смещения (*disp*8 или *disp*16). Режимы адресации данных в памяти такие же, что и в МП 8086, т. е. задаются полями *mod* и *r/m* в соответствии с табл. 4.9. Только для значения поля *mod* = 11 поле *r/m* = ST(*i*), т. е. адресуются не регистры МП, а регистры регистрового стека *NDCP*.

Время выполнения команд указано в числе тактов сигнала *CLK*. Если команда адресует операнд в памяти, то к указанному числу тактов следует добавить число тактов *ea*, затрачиваемое на вычисление эффективного адреса *EA*, которое зависит от используемого в команде режима адресации данных (см. § 4.2). Минимальная длина команд *NDCP* составляет два байта, и первый байт всех команд содержит код $D_{7\text{-}3}$ = 11011 = ESCAPE — признак команд *NDCP*.

	Первый байт								Второй байт								Число тактов (без *ea*)			
Data Transfer	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	*Short Real*	*Short Integer*	*Long Real*	*Word Integer*
FLD — *Load* (загрузка)	1	1	0	1	1											*MF* =	**00**	**01**	**10**	**11**
ST(0) ← *Integer/Real Mem*	ESCAPE			*MF*		1			*mod*			0	0	0		*r/m*	38÷56	52÷60	40÷60	46÷54
ST(0) ← *Long Integer Mem*	ESCAPE			1	1	1			*mod*			1	0	1		*r/m*	60 ÷ 68			
ST(0) ← *Temp Real Mem*	ESCAPE			0	1	1			*mod*			1	0	1		*r/m*	53 ÷ 65			
ST(0) ← *Packed BCD Mem*	ESCAPE			1	1	1			*mod*			1	0	0		*r/m*	290 ÷ 310			
ST(0) ← ST(*i*)	ESCAPE			0	0	1			1	1	0	0	0			ST(*i*)	17 ÷ 22			
FST — *Store* (запоминание)																				
Integer/Real Mem ← ST(0)	ESCAPE			*MF*		1			*mod*			0	1	0		*r/m*	84÷90	82÷92	96÷104	80÷90
ST(*i*) ← ST(0)	ESCAPE			1	0	1			1	1	0	1	0			ST(*i*)	15 ÷ 22			
FSTP — *Store and Pop* (запоминание и извлечение операнда)																				
Integer/Real Mem ← ST(0)	ESCAPE			*MF*		1			*mod*			0	1	1		*r/m*	86÷92	84÷94	96÷106	82÷92
Long Integer Mem ← ST(0)	ESCAPE			1	1	1			*mod*			1	1	1		*r/m*	94 ÷ 106			
Temp Real Mem ← ST(0)	ESCAPE			0	1	1			*mod*			1	1	1		*r/m*	52 ÷ 58			
Packed BCD Mem ← ST(0)	ESCAPE			1	1	1			*mod*			1	1	0		*r/m*	520 ÷ 540			
ST(*i*) ← ST(0)	ESCAPE			1	0	1			1	1	0	1	1			ST(*i*)	17 ÷ 24			
FXCH — *Exchange* (обмен) ST(*i*) *and* ST(0)	ESCAPE			0	0	1			1	1	0	0	1			ST(*i*)	10 ÷ 15			
Comparison																				
FCOM — *Compare* (сравнение)																				
Integer/Real Mem to ST(0)	ESCAPE			*MF*		0			*mod*			0	1	0		*r/m*	60÷70	78÷91	65÷75	72÷86
ST(*i*) *to* ST(0)	ESCAPE			0	0	0			1	1	0	1	0			ST(*i*)	40 ÷ 50			
FCOMP — *Compare and Pop* (сравнение и извлечение операнда)																				
Integer/Real Mem to ST(0)	ESCAPE			*MF*		0			*mod*			0	1	1		*r/m*	63÷73	80÷93	67÷77	74÷88
ST(*i*) *to* ST(0)	ESCAPE			0	0	0			1	1	0	1	1			ST(*i*)	45 ÷ 52			
FCOMPP — *Compare* ST(1) *to* ST(0) *and Pop Twice* (извлечение дважды)	ESCAPE			1	1	0			1	1	0	1	1	0	0	1	45 ÷ 55			
FTST — *Test* (проверка) ST(0)	ESCAPE			0	0	1			1	1	1	0	0	1	0	0	38 ÷ 48			
FXAM — *Examine* (анализ) ST(0)	ESCAPE			0	0	1			1	1	1	0	0	1	0	1	12 ÷ 23			

	Первый байт	Второй байт	Число тактов (без *ea*)			
Constants	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	*Short Real*	*Short Integer*	*Long Real*	*Word Integer*
	1 1 0 1 1	*MF* =	**00**	**01**	**10**	**11**
FLDZ — ST(0) ← +0.0	ESCAPE 0 0 1	1 1 1 0 1 1 1 0	11 + 17			
FLD1 — ST(0) ← +1.0	ESCAPE 0 0 1	1 1 1 0 1 0 0 0	15 + 21			
FLDPI — ST(0) ← π	ESCAPE 0 0 1	1 1 1 0 1 0 1 1	16 + 22			
FLDL2T — ST(0) ← $\log_2 10$	ESCAPE 0 0 1	1 1 1 0 1 0 0 1	16 + 22			
FLDL2E — ST(0) ← $\log_2 e$	ESCAPE 0 0 1	1 1 1 0 1 0 1 0	15 + 21			
FLDLG2 — ST(0) ← $\log_{10} 2$	ESCAPE 0 0 1	1 1 1 0 1 1 0 0	18 + 24			
FLDLN2 — ST(0) ← $\log_e 2$	ESCAPE 0 0 1	1 1 1 0 1 1 0 1	17 + 23			
Arithmetic						
FADD — *Addition* (сложение)						
Integer/Real Mem with ST(0)	ESCAPE *MF* 0	*mod* 0 0 0 *r/m*	90+120	108+143	95+125	102+137
ST(*i*) *and* ST(0)	ESCAPE *d P* 0	1 1 0 0 0 ST(*i*)	70 + 100			
FSUB — *Subtraction* (вычитание)						
Integer/Real Mem with ST(0)	ESCAPE *MF* 0	*mod* 1 0 *R* *r/m*	90+120	108+143	95+125	102+137
ST(*i*) *and* ST(0)	ESCAPE *d P* 0	1 1 1 0 *R* ST(*i*)	70 + 100			
FMUL — *Multiplication* (умножение)						
Integer/Real Mem with ST(0)	ESCAPE *MF* 0	*mod* 0 0 1 *r/m*	110+125	130+144	112+168	124+138
ST(*i*) *and* ST(0)	ESCAPE *d P* 0	1 1 0 0 1 ST(*i*)	90 + 145			
FDIV — *Division* (деление)						
Integer/Real Mem with ST(0)	ESCAPE *MF* 0	*mod* 1 1 *R* *r/m*	215÷225	230+243	220+238	224+238
ST(*i*) *and* ST(0)	ESCAPE *d P* 0	1 1 1 1 *R* ST(*i*)	193 + 203			
FSQRT — *Square Root* (квадратный корень)						
ST(0) ← $\sqrt{ST(0)}$	ESCAPE 0 0 1	1 1 1 1 1 0 1 0	180 + 186			
FSCALE — *Scale* (масштабирование)						
ST(0) ← ST(0) × $2^{ST(1)}$	ESCAPE 0 0 1	1 1 1 1 1 1 0 1	32 + 38			
FPREM — *Partial Remainder* (частичный остаток)						
ST(0) ← ST(0) – *q* × ST(1)	ESCAPE 0 0 1	1 1 1 1 1 0 0 0	15 + 190			
FRNDINT — *Round* (округление до целого)						
ST(0) *to Integer*	ESCAPE 0 0 1	1 1 1 1 1 1 0 0	16 + 50			

FXTRACT — *Extract Components* (извлечение порядка *exp* и мантиссы *F*)

of ST(0) | ESCAPE | 0 0 1 | 1 1 1 1 0 1 0 0 | 27 ÷ 56

FABS — *Absolute Value* (абсолютное значение)

ST(0) ← |ST(0)| | ESCAPE | 0 0 1 | 1 1 1 0 0 0 0 1 | 10 ÷ 17

FCHS — *Change Sign* (изменение знака)

ST(0) ← –ST(0) | ESCAPE | 0 0 1 | 1 1 1 0 0 0 0 0 | 10 ÷ 17

	Первый байт								Второй байт								Число тактов
Transcendental									7	6	5	4	3	2	1	0	↓
FPTAN — *Partial Tangent* (частичный тангенс)	1	1	0	1	1												
of ST(0); $0 \le ST(0) \le \pi/4$	ESCAPE			0	0	1			1	1	1	1	0	0	1	0	30 ÷ 540
FPATAN — *Partial Arctangent* (частичный арктангенс)																	
of ST(0), ST(1); $0 \le ST(0) < ST(1) < +\infty$	ESCAPE			0	0	1			1	1	1	1	0	0	1	1	250 ÷ 800
F2XM1 — $ST(0) \leftarrow 2^{ST(0)} - 1;\ 0 \le ST(0) \le 2^{-1}$	ESCAPE			0	0	1			1	1	1	1	0	0	0	0	310 ÷ 630
FYL2X — $ST(1) \leftarrow ST(1) \times \log_2 ST(0)$	ESCAPE			0	0	1			1	1	1	1	0	0	0	1	900 ÷ 1100
FYL2XP1 — $ST(1) \times \log_2 [ST(0) + 1]$	ESCAPE			0	0	1			1	1	1	1	1	0	0	1	700 ÷ 1000

Processor Control

FINIT — *Initialized* 8087 (инициализация *NDCP*)

CW ← 03FF*h*, SW ← 0000*h*, TW ← FFFF*h* | ESCAPE | 0 1 1 | 1 1 1 0 0 0 1 1 | 2 ÷ 8

FENI — *Enable Interrupts* (разрешение запросов прерываний)

$(M)_{CW} \leftarrow 0$ (разряд *M* в слове управления) | ESCAPE | 0 1 1 | 1 1 1 0 0 0 0 0 | 2 ÷ 8

FDISI — *Disable Interrupts* (запрет запросов прерываний)

$(M)_{CW} \leftarrow 1$ (разряд *M* в слове управления) | ESCAPE | 0 1 1 | 1 1 1 0 0 0 0 1 | 2 ÷ 8

FLDCW ***src*** — *Load Control Word* (загрузка слова управления)

CW ← *src* | ESCAPE | 0 0 1 | *mod* | 1 0 1 | *r/m* | 7 ÷ 14

FSTCW ***dst*** — *Store Control Word* (запоминание слова управления)

dst ← CW | ESCAPE | 0 0 1 | *mod* | 1 1 1 | *r/m* | 12 ÷ 18

FSTSW ***dst*** — *Store Status Word* (запоминание слова состояния)

dst ← SW | ESCAPE | 1 0 1 | *mod* | 1 1 1 | *r/m* | 12 ÷ 18

FCLEX — *Clear Exceptions* (сброс особых случаев в слове состояния)

$(B/IR/PE/UE/OE/ZE/DE/IE)_{SW} \leftarrow 0$ | ESCAPE | 0 1 1 | 1 1 1 0 0 0 1 0 | 2 ÷ 8

FSTENV ***dst*** — *Store Environment* (запоминание семи слов среды *NDCP*)

dst ← *Environment* | ESCAPE | 0 0 1 | *mod* | 1 1 0 | *r/m* | 40 ÷ 50

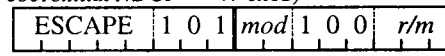
FLDENV ***src*** — *Load Environment* (загрузка семи слов среды *NDCP*)

Environment ← *src* | ESCAPE | 0 0 1 | *mod* | 1 0 0 | *r/m* | 35 ÷ 45

FSAVE ***dst*** — *Save State* (запоминание полного состояния *NDCP* — 47 слов)

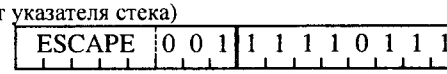
dst ← *State* | ESCAPE | 1 0 1 | *mod* | 1 1 0 | *r/m* | 197 ÷ 207

FRSTOR *src* — *Restore State* (загрузка полного состояния *NDCP* — 47 слов)
State ← *src*



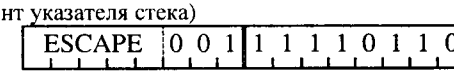
197 ÷ 207

FINCSTP — *Increment Stack Pointer* (инкремент указателя стека)
Top ← *Top* + 1



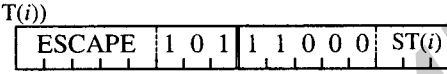
6 ÷ 12

FDECSTP — *Decrement Stack Pointer* (декремент указателя стека)
Top ← *Top* – 1



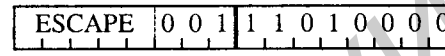
6 ÷ 12

FFREE — *Free* ST(*i*) (освобождение регистра ST(*i*))
Tag(*i*) ← 11



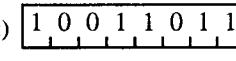
9 ÷ 16

FNOP — *No Operation* (пустая операция)



10 ÷ 16

FWAIT — *CPU Wait for* 8087 (команда МП 8086 — перевод МП в состояние ожидания)
(команды FWAIT и WAIT — синонимы)



$3 + 5 \times n$ (n — число тактов ожидания значения сигнала *BUSY* = 0)

Двухразрядное поле *MF* (*Memory Format* — формат памяти) в первом байте некоторых команд задает четыре формата чисел, поэтому в ассемблере основная мнемоника этих команд (например, FLD *src*) используется с операндами *src* в памяти, представляющими собой вещественные (*Real*) числа. Для указания операндов *src* в памяти, являющихся целыми (*Integer*) и упакованными десятичными (*Packed BCD*) числами в основную мнемонику после буквы F добавляются индикаторы I и B соответственно (например, FILD *src* и FBLD *src*). Во всех командах, оперирующих операндами из памяти, вторым (неявным) операндом является операнд, находящийся в текущей вершине регистрового стека. Этот операнд обозначается как ST(0).

Основные арифметические команды содержат одноразрядные индикаторы *d* (*Destination* — получатель), *P* (*Pop* — извлечение из стека) и *R* (*Reverse* — поменять местами операнды при выполнении вычитания и деления, если индикатор *d* = 1), каждый из которых задает по две разновидности операций:

d = 0 — получателем является регистр ST(0), находящийся в текущей вершине стека;

d = 1 — получателем является регистр ST(*i*), находящийся ниже вершины стека;

P = 0 — операнд не извлекается из стека, т. е. регистр ST(0) не освобождается (*Tag* ≠ 11);

P = 1 — операнд извлекается из стека, т. е. регистр ST(0) освобождается (*Tag* = 11); в конце мнемоники этих команд указывается буква P;

R = 0 — выполняются операции *dst* ← *dst* – *src*, *dst* ← *dst* / *src* (при использовании основной мнемоники команды);

R = 1 — выполняются операции *dst* ← *src* – *dst*, *dst* ← *src* / *dst* (при добавлении буквы *R* в конце мнемоники команды; если команды должны выполнять и извлечение из стека, то к основной мнемонике команды добавляются буквы RP).

Система команд *NDCP* 8087. Команды *NDCP* 8087 (табл. 4.32) имеют такие же форматы, что и команды МП 8086:

COP, COP *src*, COP *dst*, COP *dst*, *src*,

где COP (*Opcode*) — мнемоника кода операции, *src* — операнд-источник, *dst* — операнд-получатель, а в качестве операндов обоих типов могут указываться только регистры стека ST(0), ST(*i*) и переменные, определенные в памяти. Многие команды в коде операции адресуют один или два неявных операнда: регистры стека ST(0) и ST(1), регистры слова состояния и слова управления, среду *NDCP* (*Environment*), полное состояние *NDCP* и др. Некоторые одноопе-рандные и двухоперандные команды, в которых не указано ни одного операнда (команды типа

СОР), выполняют операции над содержимым вершины стека ST(0) или над содержимым ST(0) и ST(1) соответственно.

В командах можно указывать ST вместо ST(0), что далее и будет использовано. При выполнении команды может производиться инкремент или декремент указателя вершины стека *Top* (TOP в *SW* на рис. 4.36) с загрузкой операнда в новую вершину стека, а значит, для описания такой команды требуется использовать исходную и следующую вершины стека. Поэтому, как и в командах, для исходной вершины стека будет использоваться мнемоника ST, а для следующей (новой) вершины стека — **ST(0)**, выделенная полужирным шрифтом. Относительный адрес *i* операндов ST(*i*) в командах определяется относительно исходной вершины стека. Аналогичные обозначения будут использоваться и для тэгов: *Tag* — тэг регистра, находящегося в исходной вершине стека, *Tag*(0) — тэг регистра, находящегося в следующей вершине стека.

Таблица 4.32. **Система команд арифметического сопроцессора 8087**

Группа команд передачи данных		
FLD	*Top* ← *Top* – 1, **ST(0)** ← ST(1) — копирование регистра 1, *Tag*(0) ← ≠11	
FLD ST(*i*)	*Top* ← *Top* – 1, **ST(0)** ← ST(*i*) — копирование регистра *i* = 0 ÷ 7, *Tag*(0) ← ≠11	
FLD *src*	*Top* ← *Top* – 1, **ST(0)** ← *M*(*src*) — *Short/Long/Temporary Real*, *Tag*(0) ← ≠11	
FILD *src*	*Top* ← *Top* – 1, **ST(0)** ← *M*(*src*) — *Word/Short/Long Integer*, *Tag*(0) ← ≠11	
FBLD *src*	*Top* ← *Top* – 1, **ST(0)** ← *M*(*src*) — *Packed BCD*, *Tag*(0) ← ≠11	
FST	ST(1) ← ST, *Tag*(1) ← ≠11	*Нет команд*:
FST ST(*i*)	ST(*i*) ← ST, *i* = 0 ÷ 7, *Tag*(*i*) ← ≠11	*M*(*dst*) ← ST; *dst* — *Temporary Real*
FST *dst*	*M*(*dst*) ← ST; *dst* — *Short/Long Real*	*M*(*dst*) ← ST; *dst* — *Long Integer*
FIST *dst*	*M*(*dst*) ← ST; *dst* — *Word/Short Integer*	*M*(*dst*) ← ST; *dst* — *Packed BCD*
FSTP	ST(1) ← ST, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FSTP ST(*i*)	ST(*i*) ← ST, *i* = 0 ÷ 7, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FSTP *dst*	*M*(*dst*) ← ST; *dst* — *Short/Long/Temp. Real*, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FISTP *dst*	*M*(*dst*) ← ST; *dst* — *Word/Short/Long Integer*, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FBSTP *dst*	*M*(*dst*) ← ST; *dst* — *Packed BCD*, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FXCH	ST ↔ ST(1) — обмен содержимого регистров	
FXCH ST(*i*)	ST ↔ ST(*i*), *i* = 0 ÷ 7 — обмен содержимого регистров	
Группа команд сравнения (см. табл. 4.30 в § 4.6)		
FCOM	Сравнение ST с ST(1)	*Нет команд*:
FCOM ST(*i*)	Сравнение ST с ST(*i*), *i* = 0 ÷ 7	для *src Temporary Real*
FCOM *src*	Сравнение ST с *M*(*src*) — *Short/Long Real*	для *src Long Integer*
FICOM *src*	Сравнение ST с *M*(*src*) — *Word/Short Integer*	для *src Packed BCD*
FCOMP	Сравнение ST с ST(1), *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FCOMP ST(*i*)	Сравнение ST с ST(*i*), *i* = 0 ÷ 7, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FCOMP *src*	Сравнение ST с *M*(*src*) — *Short/Long Real*, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FICOMP *src*	Сравнение ST с *M*(*src*) — *Word/Short Integer*, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FCOMPP	Сравнение ST с ST(1), *Tag*(0) ← 11, *Tag*(1) ← 11, *Top* ← *Top* + 2, **ST(0)**	
FTST	Сравнение ST с числом 0	
FXAM	Анализ ST	
FUCOM, FUCOMP, FUCOMPP — для *NDCP* 80387 и выше (неупорядоченные сравнения)		

Продолжение табл. 4.32

Группа команд загрузки констант		
FLDZ	*Top* ← *Top* – 1, **ST(0)** ← +0.0, *Tag*(0) ← 01	
FLD1	*Top* ← *Top* – 1, **ST(0)** ← +1.0, *Tag*(0) ← 00	
FLDPI	*Top* ← *Top* – 1, **ST(0)** ← π, *Tag*(0) ← 00	
FLDL2T	*Top* ← *Top* – 1, **ST(0)** ← $\log_2 10$, *Tag*(0) ← 00	
FLDL2E	*Top* ← *Top* – 1, **ST(0)** ← $\log_2 e$, *Tag*(0) ← 00	
FLDLG2	*Top* ← *Top* – 1, **ST(0)** ← $\log_{10} 2$, *Tag*(0) ← 00	
FLDLN2	*Top* ← *Top* – 1, **ST(0)** ← $\log_e 2$, *Tag*(0) ← 00	
Группа арифметических команд		
FADD ST, ST(*i*) FADD ST(*i*), ST FADD *src* FIADD *src*	ST ← ST + ST(*i*), *i* = 0 ÷ 7 ST(*i*) ← ST(*i*) + ST, *i* = 0 ÷ 7 ST ← ST + *M*(*src*) — *Short/Long Real* ST ← ST + *M*(*src*) — *Word/Short Integer*	*Нет команд*: для *src Temporary Real* для *src Long Integer* для *src Packed BCD*
FADDP FADDP ST(*i*)	ST(1) ← ST(1) + ST, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)** ST(*i*) ← ST(*i*) + ST, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FSUB ST, ST(*i*) FSUB ST(*i*), ST FSUB *src* FISUB *src*	ST ← ST – ST(*i*), *i* = 0 ÷ 7 ST(*i*) ← ST(*i*) – ST, *i* = 0 ÷ 7 ST ← ST – *M*(*src*) — *Short/Long Real* ST ← ST – *M*(*src*) — *Word/Short Integer*	
FSUBR ST, ST(*i*) FSUBR ST(*i*), ST FSUBR *src* FISUBR *src*	ST ← ST(*i*) – ST, *i* = 0 ÷ 7 ST(*i*) ← ST – ST(*i*), *i* = 0 ÷ 7 ST ← *M*(*src*) – ST; *src* — *Short/Long Real* ST ← *M*(*src*) – ST; *src* — *Word/Short Integer*	
FSUBP FSUBP ST(*i*) FSUBRP FSUBRP ST(*i*)	ST(1) ← ST(1) – ST, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)** ST(*i*) ← ST(*i*) – ST, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)** ST(1) ← ST – ST(1), *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)** ST(*i*) ← ST – ST(*i*), *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FMUL ST, ST(*i*) FMUL ST(*i*), ST FMUL *src* FIMUL *src*	ST ← ST × ST(*i*), *i* = 0 ÷ 7 ST(*i*) ← ST(*i*) × ST, *i* = 0 ÷ 7 ST ← ST × *M*(*src*) — *Short/Long Real* ST ← ST × *M*(*src*) — *Word/Short Integer*	
FMULP FMULP ST(*i*)	ST(1) ← ST(1) × ST, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)** ST(*i*) ← ST(*i*) × ST, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	
FDIV ST, ST(*i*) FDIV ST(*i*), ST FDIV *src* FIDIV *src*	ST ← ST / ST(*i*), *i* = 0 ÷ 7 ST(*i*) ← ST(*i*) / ST, *i* = 0 ÷ 7 ST ← ST / *M*(*src*) — *Short/Long Real* ST ← ST / *M*(*src*) — *Word/Short Integer*	
FDIVR ST, ST(*i*) FDIVR ST(*i*), ST FDIVR *src* FIDIVR *src*	ST ← ST(*i*) / ST, *i* = 0 ÷ 7 ST(*i*) ← ST / ST(*i*), *i* = 0 ÷ 7 ST ← *M*(*src*) / ST; *src* — *Short/Long Real* ST ← *M*(*src*) / ST; *src* — *Word/Short Integer*	
FDIVP FDIVP ST(*i*)	ST(1) ← ST(1) / ST, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)** ST(*i*) ← ST(*i*) / ST, *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**	

Продолжение табл. 4.32

FDIVRP	ST(1) ← ST / ST(1), *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**
FDIVRP ST(*i*)	ST(*i*) ← ST / ST(*i*), *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**
FSQRT	ST ← $\sqrt{ST}$; $-0 \le ST \le +\infty$, $\sqrt{-0} = -0$
FSCALE	ST ← ST × $2^{ST(1)}$ — масштабирование; $-2^{15} \le ST(1) < +2^{15}$ — целое число
FPREM	ST ← ST – *q* × ST(1) — частичный остаток, *q* — целое число ($q \le 64$)
FRNDINT	ST ← округленное до целого число ST
FXTRACT	ST ← *exp* — истинный порядок ST, *Top* ← *Top* – 1, **ST(0)** ← *fraction* — мантисса ST, *Tag*(0) ← 00, ST(1) = *exp*
FABS	ST ← |ST| — абсолютное значение ST
FCHS	ST ← –ST — изменение знака ST
FPREM1 — 80387 и выше; ST ← ST – *Q* × ST(1), где *Q* — округление ST/ST(1) до целого	

Группа трансцендентных команд

FPTAN	*Partial Tangent* (частичный тангенс) *of* ST; $0 \le ST \le \pi/4$ ST ← *Y*, *Top* ← *Top* – 1, **ST(0)** ← *X*, *Tag*(0) ← 00, ST(1) = *Y* ⇒ tg ST = *Y* / *X*
FPATAN	*Partial Arctangent* (частичный арктангенс) *of* ST, ST(1); $0 \le ST(1) < ST < +\infty$ ST(1) ← arctg [ST(1) / ST], *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)** = arctg [ST(1) / ST]
F2XM1	ST ← $2^{ST} - 1$; $0 \le ST \le 2^{-1}$
FYL2X	ST(1) ← ST(1) · $\log_2$ ST, *Tag* ← 11, *Top* ← *Top*+1, **ST(0)**; $0 < ST < \infty$, $-\infty < ST(1) < +\infty$
FYL2XP1	ST(1) ← ST(1) · $\log_2$ (ST + 1), *Tag* ← 11, *Top* ← *Top* + 1, **ST(0)**; $0 \le \|ST\| < (2 - \sqrt{2})/2$, $-\infty < ST(1) < \infty$
FCOS, FSIN, FSINCOS — для *NDCP* 80387 и выше (вычисление COS, SIN, SIN и COS ST)	

Группа команд управления NDCP [1]

FINIT/FNINIT	CW ← 03FF*h*, SW ← 0000*h*, TW ← FFFF*h* — инициализация *NDCP*
FENI/FNENI	$(M)_{CW}$ ← 0 — разрешение запросов прерываний (разряд *M* в *CW*)
FDISI/FNDISI	$(M)_{CW}$ ← 1 — запрет запросов прерываний (разряд *M* в *CW*)
FLDCW *src* [2]	*CW* ← *M*(*src*) — переменная типа WORD
FSTCW/FNSTCW *dst*	*M*(*dst*) ← *CW*; *dst* — переменная типа WORD
FSTSW/FNSTSW *dst*	*M*(*dst*) ← *SW*; *dst* — переменная типа WORD
FCLEX/FNCLEX	$(B/IR/PE/UE/OE/ZE/DE/IE)_{SW}$ ← 0 — сброс особых случаев в *SW*
FSTENV/FNSTENV *dst*	*M*(*dst*) ← *Environment* — 7 слов среды *NDCP*
FLDENV *src*	*Environment* ← *M*(*src*) — 7 слов среды *NDCP*
FSAVE/FNSAVE *dst*	*M*(*dst*) ← *State* — 47 слов полного состояния *NDCP*
FRSTOR *src*	*State* ← *M*(*src*) — 47 слов полного состояния *NDCP*
FINCSTP [2]	*Top* ← *Top* + 1 ⇒ **ST(0)** — новая вершина стека
FDECSTP [2]	*Top* ← *Top* – 1 ⇒ **ST(0)** — новая вершина стека
FFREE [2]	*Tag*(1) ← 11 — освобождение регистра ST(1)
FFREE ST(*i*) [2]	*Tag*(*i*) ← 11 — освобождение регистра ST(*i*)
FNOP [2]	Пустая операция
FWAIT/WAIT	Команда МП 8086 — перевод МП в состояние ожидания
FSETPM (установка защищенного режима), FSTSW AX (AX ← *SW*) — *NDCP* 80287 и выше	

Примечание: [1] тип переменных *src* и *dst* для среды и полного состояния *NDCP* ассемблер не контролирует; [2] перед командой ассемблер не вводит команду ожидания FWAIT.

Для безошибочного использования регистров стека, следует запомнить *правила*:

если команда (кроме команды управления FINCSTP) производит инкремент указателя вершины стека (*Top* ← *Top* + 1), то исходная вершина ST освобождается (*Tag* ← 11), что автоматически обеспечивает свободный регистр для следующей команды, производящей загрузку операнда в стек с автоматическим декрементом указателя вершины стека (*Top* ← *Top* – 1);

после выполнения команды, производящей инкремент указателя вершины стека, в новой вершине **ST(0)** будет находиться содержимое исходного регистра ST(1), в регистре ST(1) — содержимое исходного регистра ST(2) и т. д. со смещением адресов *i* всех регистров ST(*i*);

если команда (кроме команды управления FDECSTP) производит декремент указателя вершины стека (*Top* ← *Top* – 1), то операнд загружается в новую вершину **ST(0)**, а в его тэг *Tag*(0) записывается 00 (нормализованное число), 01 (истинный нуль) или 10 (специальное значение);

после выполнения такой команды в новом регистре ST(1) будет находиться содержимое исходной вершины стека ST, в новом регистре ST(2) — содержимое исходного регистра ST(1) и т. д. со смещением адресов *i* всех регистров ST(*i*).

Многие команды в табл. 4.32 описаны достаточно полно и не требуют дополнительных пояснений. В некоторых дальнейших примерах, иллюстрирующих использование команд, будет приводиться окно отладчика для *NDCP* 8087, отображающее состояние регистрового стека после выполнения последней команды (без разрядов *RC* и *IC* в слове управления *CW* и разряда *CC* в слове состояния *SW*).

Команды передачи данных (*Data Transfer*). Эти команды обеспечивают передачу данных между регистрами стека, а также между вершиной стека и памятью. При выполнении команды загрузки F×LD числа из памяти, представленного в любом формате, в регистр стека оно автоматически преобразуется во временный вещественный формат (*Temporary Real*) с сохранением специальных значений (нуль, бесконечность и не-число). При выполнении команды сохранения F×ST× числа в памяти производится обратное преобразование форматов чисел.

Команды загрузки F×LD *src*. Эти команды производят сначала декремент указателя стека (*Top* ← *Top* – 1), а затем запись числа из регистра ST(*i*) или из памяти в новую вершину ST(0), т. е. осуществляют включение данных в стек. Загрузить число можно только в пустой регистр (*Tag* = 11), в противном случае в слове состояния *SW* фиксируется особый случай недействительной операции (*IE* ← 1), а в регистр записывается код *–NAN* = FFFF C000 0000 0000 0000*h* (код неопределенности), если особый случай недействительной операции замаскирован. Декремент указателя стека и запись кода *–NAN* не производится, если особый случай недействительной операции не замаскирован.

Команды сохранения F×ST× *dst*. Эти команды производят запись содержимого вершины стека ST в регистр ST(*i*) или в память в формате, определяемом типом операнда-получателя *dst*. При сохранении командами F×ST× содержимого пустого регистра в слове состояния *SW* фиксируется особый случай недействительной операции, и в получатель *dst* записывается код неопределенности (*–NAN*) в формате, указанном типом операнда-получателя *dst*, если особый случай недействительной операции замаскирован. Запись в получатель *dst* кода *–NAN* не производится, если особый случай недействительной операции не замаскирован. Команды сохранения F×STP содержимого регистра стека с извлечением операнда из стека (*Tag* ← 11), выполняют инкремент указателя стека (*Top* ← *Top* + 1) с назначением новой вершины стека ST(0). Команды сохранения F×ST без извлечения операнда из стека тэга не изменяют и инкремент указателя стека не производят. Сохранять данные в *занятом регистре* можно командами FST и FST ST(*i*) — копирование ST в ST(1) и ST(*i*).

При сохранении содержимого ST в памяти в вещественных форматах (команды FST×) мантисса округляется в соответствии с полем управления округлением *RC* в слове управления *CW* и длиной мантиссы получателя *dst*, а порядок корректируется с учетом длины и смещения порядка получателя. При сохранении ST в памяти в целых двоичных форматах (команды FIST×) округление ST производится до целого числа в соответствии с полем управления округлением *RC* в слове управления *CW*. При наличии в ST отрицательного нуля он будет записан как положительный нуль. При сохранении ST в памяти в упакованном десятичном формате (команда FBSTP) округление реализуется сложением ST с числом 0.5 и отбрасыванием разрядов дробной части. Другой способ округления можно задать командой FRNDINT.

Если в процессе округления содержимого ST число изменяется, в слове состояния *SW* фиксируется особый случай потери точности (*PE* ← 1). Если округленное число слишком велико для формата получателя, то в слове состояния *SW* фиксируются особый случай недействительной операции (*IE* ← 1) при сохранении в целых двоичных и упакованном десятичном форматах и особые случаи переполнения и потери точности (*OE* ← 1, *PE* ← 1) при сохранении в вещественных форматах. При этом в память записывается код неопределенности при сохранении в целых двоичных и упакованном десятичном форматах и код +∞ или –∞ при сохранении в вещественных форматах (маскированная реакция *NDCP*).

Если округленное число не является нулем, но меньше порога антипереполнения, то в память записывается 0, а в слове состояния *SW* фиксируются особый случай недействительной операции (*IE* ← 1) при сохранении в целых двоичных и упакованном десятичном форматах и особые случаи антипереполнения и потери точности (*OE* ← 1, *PE* ← 1) при сохранении в вещественных форматах (маскированная реакция).

Пример 1 (включение и извлечение операндов из регистрового стека):

```
          ∴                          ; В сегменте данных
          even                       ; Выравнивание на границу слова
n_Wi      dw    7E2Bh                ; 7E2Bh = 32299d — Word Integer
n_Pb      dt    –987654321001234567  ; 8098765432100123456 7h — Packed BCD
n_Sr      dd    792.33               ; 4446151Fh — Short Real
n_Lr      dq    792.33               ; 4088C2A3D70A3D70h — Long Real
n_Tr      dt    792.33               ; 4008C6151EB851EB851Fh — Temporary Real
buf_w     dw    'WW'                 ; 5757h — резервирование памяти
buf_d     dd    'DD'                 ; 00004444h — резервирование памяти
buf_q     dq    'QQ'                 ; 0000000000005151h — резервирование памяти
          ∴                          ; В сегменте кода
          FINIT                      ; Инициализация NDCP (все регистры стека освобождаются)
          FILD  n_Wi                 ; Top ← Top – 1, ST(0) ← M(n_Wi), Tag(0) ← 00
          FBLD  n_Pb                 ; Top ← Top – 1, ST(0) ← M(n_Pb), Tag(0) ← 00
          FLD                        ; Top ← Top – 1, ST(0) ← ST(1) = M(n_Wi), Tag(0) ← 00
          FLD   n_Sr                 ; Top ← Top – 1, ST(0) ← M(n_Sr)
          FSTP  buf_q                ; M(buf_d) ← ST(0), Tag(0) ← 11, Top ← Top + 1, ST(0)
          FLD   n_Lr                 ; Top ← Top – 1, ST(0) ← M(n_Lr), Tag(0) ← 00
          FST   buf_d                ; M(buf_d) ← ST(0), PE ← 1
          FLD   n_Tr                 ; Top ← Top – 1, ST(0) ← M(n_Tr), Tag(0) ← 00
          FIST  buf_w                ; M(buf_w) ← ST(0) Short Integer = 0318h = 792d, PE ← 1
          FXCH  ST(3)                ; ST ↔ ST(3)
          FLD   ST                   ; Top ← Top – 1, ST(0) ← ST = M(n_Pb), Tag(0) ← 00
```

```
FLD   ST(4)   ; Top ← Top – 1, ST(0) ← ST(4) = M(n_Tr), Tag(0) ← 00
FXCH  ST      ; ST ↔ ST — пустая операция
FSTP  ST(2)   ; ST(2) ← ST, Tag ← 11, Top ← Top + 1, ST(0)
```

Valid	ST(0)	–9.8765432100123457e+17	C03A	DB4D	A5F4	4288	0870	*im* = 1	*ie* = 0	*n_Pb*
Valid	ST(1)	792.33	4008	C615	1EB8	51EB	851F	*dm* = 1	*de* = 0	*n_Tr*
Valid	ST(2)	792.32999999999993	4008	C615	1EB8	51EB	8000	*zm* = 1	*ze* = 0	*n_Lr*
Valid	ST(3)	32299	400D	FC56	0000	0000	0000	*om* = 1	*oe* = 0	*n_Wi*
Valid	ST(4)	792.33	4008	C615	1EB8	51EB	851F	*um* = 1	*ue* = 0	*n_Tr*
Valid	ST(5)	32299	400D	FC56	0000	0000	0000	*pm* = 1	*pe* = 1	*n_Wi*
Empty	ST(6)		0000	0000	0000	0000	0000	*iem* = 0	*ir* = 0	
Empty	ST(7)		4008	C615	1EB8	51EB	851F	*pc* = 3	*st* = 2	

Директива EVEN дает указание ассемблеру назначить переменной *n_Wi* четный адрес — выравнивание по границе слов повышает производительность системы, так как слово с четным адресом передается за один цикл шины, а слово с нечетным адресом за два цикла шины. Поскольку длина переменных для *NDCP* во всех форматах кратна длине слова, можно указать единственную директиву EVEN перед несколькими переменными. Последовательность изменений состояний стека отображена в табл. 4.33.

Таблица 4.33. **Последовательность изменений состояния стека**

Стек	FILD *n_Wi*	FBLD *n_Pb*	FLD	FLD *n_Sr*	FSTP *buf_q*	FLD *n_Lr*	FLD *n_Tr*	FXCH ST(3)	FLD ST	FLD ST(4)	FSTP ST(2)
ST(0)	*n_Wi*	*n_Pb*	*n_Wi*	*n_Sr*	*n_Wi*	*n_Lr*	*n_Tr*	*n_Pb*	*n_Pb*	*n_Tr*	*n_Pb*
ST(1)	*Empty*	*n_Wi*	*n_Pb*	*n_Wi*	*n_Pb*	*n_Wi*	*n_Lr*	*n_Lr*	*n_Pb*	*n_Pb*	*n_Tr*
ST(2)	*Empty*	*Empty*	*n_Wi*	*n_Pb*	*n_Wi*	*n_Pb*	*n_Wi*	*n_Wi*	*n_Lr*	*n_Pb*	*n_Lr*
ST(3)	*Empty*	*Empty*	*Empty*	*n_Wi*	*Empty*	*n_Wi*	*n_Pb*	*n_Tr*	*n_Wi*	*n_Lr*	*n_Wi*
ST(4)	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*n_Wi*	*n_Wi*	*n_Tr*	*n_Wi*	*n_Tr*
ST(5)	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*n_Wi*	*n_Tr*	*n_Wi*
ST(6)	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*n_Wi*	*Empty*
ST(7)	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*

Задача 1. Вычислить квадратный корень из содержимого регистра ST(3). *Решение*:

```
FXCH   ST(3)   ; Передача ST(3) в вершину стека
FSQRT          ; Вычисление квадратного корня
FXCH   ST(3)   ; Замена аргумента результатом
```

При обмене содержимого вершины стека и пустого регистра ST(*i*) в вершину стека записывается код неопределенности (*–NAN*), а в слове состояния *SW* фиксируется особый случай недействительной операции (*IE* ← 1), если этот особый случай замаскирован словом управления *CW*. Если обмен выполнить между двумя пустыми регистрами, то в оба регистра будет записан код неопределенности (*IE* ← 1).

Команды сравнения (*Comparison*). Данные команды предназначены для сравнения числа, находящегося в вершине стека, с другим числом (FxCOMxx) или с числом 0 (FTST) и анализа числа, находящегося в вершине стека (FXAM). Результат выполнения этих команд фиксируется в коде условия $C_3 \div C_0$ слова состояния *SW* (см. табл. 4.30 в § 4.6). В частности,

код условия $C_3C_2C_1C_0$ для команд F×COM××:

00×0 (ST > *src*), 00×1 (ST < *src*),
10×0 (**ST** = *src*), 11×1 (не сравнимы);

код условия $C_3C_2C_1C_0$ для команды FTST:

00×0 (ST > 0), 00×1 (ST < 0),
10×0 (ST = 0), 11×1 (не сравнимы).

Пример 2 (сравнение чисел):

```
        ∴                   ; В сегменте данных
      even                  ; Выравнивание на границу слова
n_Wi  dw      7E2Bh         ; 7E2Bh = 32299d — Word Integer
n_Si  dd      3943A0BEh     ; 3943A0BEh = 960733374d — Short Integer
n_Sr  dd      0FF800000r    ; 0FF800000h — -∞ Short Real
n_Lr  dq      79265.133     ; 40F35A1220C49BA6h — Long Real
        ∴                   ; В сегменте кода
      FINIT                 ; Инициализация NDCP (все регистры стека освобождаются)
      FLD     n_Lr          ; Top ← Top – 1, ST(0) ← M(n_Lr), Tag(0) ← 00
      FLD     n_Sr          ; Top ← Top – 1, ST(0) ← M(n_Sr) , Tag(0) ← 00
      FILD    n_Si          ; Top ← Top – 1, ST(0) ← M(n_Si), Tag(0) ← 00
      FILD    n_Wi          ; Top ← Top – 1, ST(0) ← M(n_Wi), Tag(0) ← 00
      FCOM                  ; Сравнение ST = M(n_Wi) с ST(1) = M(n_Si) ⇒ C3C2C1C0 = 0001
      FCOM    ST(2)         ; Сравнение ST = M(n_Wi) с ST(2) = M(n_Sr) ⇒ C3C2C1C0 = 0000
      FCOM    n_Lr          ; Сравнение ST = M(n_Wi) с M(n_Lr) ⇒ C3C2C1C0 = 0001
      FTST                  ; Сравнение ST = M(n_Wi) с числом 0 ⇒ C3C2C1C0 = 0000
      FICOMP  n_Wi          ; Сравнение ST = M(n_Wi) с M(n_Wi) ⇒ C3C2C1C0 = 1000,
                            ;     Tag ← 11, Top ← Top + 1, ST(0)
      FCOMPP                ; Сравнение ST = M(n_Si) с ST(1) = M(n_Sr) ⇒ C3C2C1C0 = 0000,
                            ;     Tag ← 11, Top ← Top + 1, Tag ← 11, Top ← Top + 1, ST(0)
      FCOM                  ; Сравнение ST = M(n_Lr) с ST(1) = Empty ⇒ C3C2C1C0 = 1101, IE ← 1
      FCOMP   n_Si          ; Сравнение ST = M(n_Lr) с M(n_Si) ⇒ C3C2C1C0 = 0000,
                            ;     Tag ← 11, Top ← Top + 1, ST(0)
```

Последовательность изменений состояний стека при выполнении команд отображена в табл. 4.34. Использование команд F×COM×× и FTST для организации условных переходов было рассмотрено в *примере* 1 § 4.6 (с. 476), а анализ чисел с помощью команды FXAM — в *примере* 3 § 4.6 (с. 478).

Таблица 4.34. **Последовательность изменений состояния стека**

Стек	FLD *n_Lr*	FLD *n_Sr*	FILD *n_Si*	FILD *n_Wi*	FICOMP *n_Wi*	FCOMPP	FCOMP *n_Si*
ST(0)	*n_Lr*	*n_Sr*	*n_Si*	*n_Wi*	*n_Si*	*n_Lr*	*Empty*
ST(1)	*Empty*	*n_Lr*	*n_Sr*	*n_Si*	*n_Sr*	*Empty*	*Empty*
ST(2)	*Empty*	*Empty*	*n_Lr*	*n_Sr*	*n_Lr*	*Empty*	*Empty*
ST(3)	*Empty*	*Empty*	*Empty*	*n_Lr*	*Empty*	*Empty*	*Empty*
ST(4)	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
ST(5)	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
ST(6)	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*
ST(7)	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*	*Empty*

Команды загрузки констант (*Constants*). Эти команды загружают константы +0.0; +1.0, π, $\log_2 10$, $\log_2 e$, $\log_{10} 2$ и $\log_e 2$, представленные в формате *Temporary Real*, из внутреннего постоянного запоминающего устройства в свободные регистры стека (*Tag* = 11) с предварительным декрементом указателя стека (*Top* ← *Top* – 1), как и при выполнении команд загрузки FLD. Данные константы часто встречаются в вычислительных алгоритмах (точность представления констант — 19 десятичных цифр).

Пример 3 (загрузка констант):

```
        ∴               ; В сегменте кода
FINIT                   ; Инициализация NDCP (все регистры стека освобождаются)
FLDZ                    ; Top ← Top – 1, ST(0) ← +0.0, Tag(0) ← 01
FLD1                    ; Top ← Top – 1, ST(0) ← +1.0, Tag(0) ← 00
FLDPI                   ; Top ← Top – 1, ST(0) ← π, Tag(0) ← 00
FLDL2T                  ; Top ← Top – 1, ST(0) ← log2 10, Tag(0) ← 00
FLDL2E                  ; Top ← Top – 1, ST(0) ← log2 e, Tag(0) ← 00
FLDLG2                  ; Top ← Top – 1, ST(0) ← log10 2, Tag(0) ← 00
FLDLN2                  ; Top ← Top – 1, ST(0) ← loge 2, Tag(0) ← 00
```

Valid	ST(0)	0.69314718055994531	3FFE	B172	17F7	D1CF	79AC	*im* = 1	*ie* = 0	$\log_e 2$
Valid	ST(1)	0.3010299956639812	3FFD	9A20	9A84	FBCF	F799	*dm* = 1	*de* = 0	$\log_{10} 2$
Valid	ST(2)	1.4426950408889634	3FFF	B8AA	3B29	5C17	F0BC	*zm* = 1	*ze* = 0	$\log_2 e$
Valid	ST(3)	3.3219280948873623	4000	D49A	784B	CD1B	8AFE	*om* = 1	*oe* = 0	$\log_2 10$
Valid	ST(4)	3.1415926535897932	4000	C90F	DAA2	2168	C235	*um* = 1	*ue* = 0	π
Valid	ST(5)	1	3FFF	8000	0000	0000	0000	*pm* = 1	*pe* = 0	+1.0
Zero	ST(6)	0	0000	0000	0000	0000	0000	*iem* = 0	*ir* = 0	+0.0
Empty	ST(7)		0000	0000	0000	0000	0000	*pc* = 3	*st* = 1	

Арифметические команды (*Arithmetic*). Команды сложения F×ADD×, вычитания F×SUB××, умножения F×MUL×, деления F×DIV××, извлечения квадратного корня FSQRT, вычисления абсолютного значения FABS и изменения знака FCHS достаточно полно описаны в табл. 4.32 и не требуют дополнительных пояснений.

Задача 2. Найти корни x_1 и x_2 квадратного уравнения $ax^2 + bx + c = 0$. Для вычисления корней x_1 и x_2 естественно использовать известную формулу:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

(решение существует, если дискриминант $b^2 - 4ac \geq 0$). *Решение*:

```
        ∴                   ; В сегменте данных
        even                ; Выравнивание на границу слова
n_a     dq      +29.33      ; 403D547AE147AE14h = a — Long Real
n_b     dq      –86.74      ; C055AF5C28F5C28Fh = b — Long Real
n_c     dq      +55.37      ; 404BAF5C28F5C28Fh = c — Long Real
buf_x1  dq      'X1'        ; 0000000000005831h
buf_x2  dq      'X2'        ; 0000000000005832h
buf_sw  dw      'SW'        ; 5357h
```

```
        ∴               ; В сегменте кода
        FINIT           ; Инициализация NDCP
        FLD     n_a     ; Top ← Top – 1, ST(0) ← M(n_a) = a, Tag(0) ← 00
        FADD    ST, ST  ; ST ← a + a = 2a
        FST     ST(1)   ; ST = 2a, ST(1) ← 2a, Tag(1) ← 00
        FADD    ST, ST  ; ST ← 2a + 2a = 4a, ST(1) = 2a
        FMUL    n_c     ; ST ← 4a × M(n_c) = 4ac, ST(1) = 2a
        FLD     n_b     ; Top ← Top – 1, ST(0) ← b, Tag(0) ← 00, ST(1) = 4ac, ST(2) = 2a
        FST     ST(3)   ; ST = b, ST(1) = 4ac, ST(2) = 2a, ST(3) ← b, Tag(3) ← 00
        FMUL    ST, ST  ; ST ← M(n_b) × M(n_b) = b × b, ST(1) = 4ac, ST(2) = 2a, ST(3) ← b
        FSUBRP          ; ST(1) ← ST – ST(1) = b^2 – 4ac = d,
                        ;     Top ← Top + 1, ST(0) = d, ST(1) = 2a, ST(2) = b
        FTST            ; Проверка дискриминанта: d < 0?
        FSTSW   buf_sw  ; Запись в буфер памяти кода условия C3C2C1C0
        MOV     AH, byte ptr buf_sw + 1 ; AH ← C3C2C1C0
        SAHF            ; Младший байт PSW ← C3C2C1C0
        JB      Error   ; Переход, если d < 0 (отрицательный дискриминант)
        FSQRT           ; ST ← r = квадратный корень из d, ST(1) = 2a, ST(2) = b
        FST     ST(3)   ; ST = r, ST(1) = 2a, ST(2) = b, ST(3) ← r, Tag(3) ← 00
        FSUB    ST, ST(2) ; ST ← –b + r, ST(1) = 2a, ST(2) = b, ST(3) = r
        FDIV    ST, ST(1) ; ST ← x1 = (–b + r) / 2a, ST(1) = 2a, ST(2) = b, ST(3) = r
        FSTP    buf_x1  ; M(buf_x1) ← x1, Top ← Top + 1, ST(0) = 2a, ST(1) = b, ST(2) = r
        FXCH    ST(1)   ; ST ← b, ST(1) ← 2a, ST(2) = r
        FCHS            ; ST ← –b, ST(1) = 2a, ST(2) = r
        FSUB    ST, ST(2) ; ST ← –b – r, ST(1) = 2a, ST(2) = r
        FDIVRP          ; ST(1) ← x2 = (–b – r) / 2a, Top ← Top + 1, ST(0) = x2, ST(1) = r
        FSTP    buf_x2  ; M(buf_x2) ← x2, Top ← Top + 1, ST(0) = r
        FFREE   ST      ; Очистка вершины стека: Tag ← 11
        ∴               ; x1 = 2.0252236525954229, x2 = 0.93215786803192112
Error:  ∴
```

Подстановка в уравнение корней x_1 и x_2 дает результаты:

–0,00000000000000007575658904247 и –0,00000000000000087408670305914

Для уменьшения времени выполнения программы следует чаще использовать операнды, находящиеся в регистрах стека, без лишних обращений к памяти. При ограниченном числе регистров в стеке решение сложных задач потребует от программиста определенных усилий.

Команда масштабирования FSCALE. Эта команда используется для изменения порядка числа, находящегося в регистре ST. Команда FSCALE производит сложение содержимого регистра ST(1), интерпретируемого как целое двоичное число со знаком, с полем смещенного порядка числа ST:

$$ST \leftarrow ST \times 2^{ST(1)},\ -2^{15} \le ST(1) < +2^{15} \text{ — целое число (со знаком),}$$

т. е. команда FSCALE выполняет быстрое умножение при ST(l) > 0 и деление при ST(1) < 0 содержимого вершины стека (регистра ST) на целую степень числа 2. Если масштабный коэффициент ST(1) не является целым числом, но находится в указанном диапазоне и больше по абсолютному значению числа 1, то принимается ближайшее целое, меньшее по абсолютному

значению (производится усечение к нулю). Когда число в ST(1) находится вне допустимого диапазона или является правильной дробью, команда FSCALE дает непредсказуемый результат без фиксации особого случая. Поэтому рекомендуется задавать масштабный коэффициент в виде целого слова.

Команда вычисления частичного остатка FPREM. Эта команда выполняет точное деление чисел по заданному модулю с помощью последовательных его вычитаний из делимого:

$$ST \leftarrow ST - q \times ST(1),$$

где q — целое число ($q \le 64$), ST(1) — делитель (модуль), ST — частичный остаток, т. е. команда FPREM вычисляет остаток по модулю ST(1) содержимого вершины стека (регистра ST). Знак остатка равен знаку числа ST. Команда FPREM предназначена для приведения аргумента периодических трансцендентных функций в диапазон, допустимый в соответствующих командах *NDCP*. Например, команда FPTAN вычисления частичного тангенса требует, чтобы аргумент находился в диапазоне $0 \le ST \le \pi/4$. Использовав число $\pi/4$ в качестве модуля, можно уменьшить значение аргумента до допустимого значения. Для повышения точности вычисления функций необходимо иметь точный результат (без округления), а это и обеспечивает команда FPREM — производятся последовательные вычитания модуля из делимого до тех пор, пока не будет получен результат, меньший по абсолютному значению модуля.

Если делимое намного больше модуля, то для вычисления остатка может потребоваться значительное время, поэтому введено ограничение на число последовательных вычитаний: $q \le 64$. Это позволяет МП быстрее реагировать на запросы прерываний от внешних устройств. Программа вычисления окончательного остатка была приведена в *примере* 2 § 4.6 (с. 477). Кроме остатка в регистре ST команда FPREM формирует в коде условия слова состояния *SW* три младших разряда частного: $C_0C_3C_1 = Q_2Q_1Q_0$ (см. табл. 4.30 и 4.31). Это позволяет определить расположение исходного угла в одном из восьми октантов. Точные значения младших разрядов частного получаются при выполнении командой FPREM не более 62 вычитаний.

Команда округления FRNDINT. Эта команда производит округление числа в регистре ST до целого числа. Режим округления задается полем *RC* в слове управления *CW*.

Команда выделения истинного порядка и мантиссы FXTRACT. Эта команда преобразует число с плавающей точкой, находящееся в регистре ST, в два числа, представляющих собой истинный порядок (*exp*) и мантиссу (*fraction*):

$$ST \leftarrow exp \text{ — истинный порядок ST}, \; Top \leftarrow Top - 1,$$

$$\mathbf{ST(0)} \leftarrow fraction \text{ — мантисса ST}, \; ST(1) = exp,$$

где ST — исходная вершина стека, **ST(0)** — следующая вершина стека.

После выполнения команды FXTRACT в регистре ST(0) будет получено вещественное число, знак и мантисса которого равны знаку и мантиссе исходного числа, а истинный порядок равен нулю (смещенный порядок равен 16383). В регистр же ST(1) будет записан порядок исходного операнда, выраженный в формате временного вещественного числа.

Пример 4 (выделение истинного порядка и мантиссы):

```
          ∴             ; В сегменте данных
n_a     dq            –73929.33e–14  ; BE0966E483E410BFh — Long Real
buf_f   dt            'FF'   ; 00000000000000004646h
buf_e   dt            'EE'   ; 00000000000000004545h
          ∴             ; В сегменте кода
        FINIT           ; Инициализация NDCP (все регистры стека освобождаются)
```

```
    FLD       n_a    ; Top ← Top – 1, ST(0) ← M(n_a), Tag(0) ← 00
    FXTRACT          ; ST ← exp = C003 F800 0000 0000 0000 = –31d, Top ← Top – 1,
                     ;     ST(0) ← fraction = BFFF CB37 241F 2085 F800, ST(1) = exp
    FSTP      buf_f  ; M(buf_f) ← fraction (Temporary Real) = –1.5876202728259583
    FBSTP     buf_e  ; M(buf_e) ← exp = 8000 0000 0000 0000 0031 (Packed BCD) = –31d
```

После сохранения в памяти четыре младших слова *M*(*buf_f*) содержат мантиссу (*fraction*), а знак мантиссы (*S*) находится в старшем разряде старшего (пятого) слова. Так как истинный порядок всегда является целым числом, то его можно сохранять в упакованном десятичном формате (*Packed BCD*).

Если сразу после команды FXTRACT выполнить команду FSCALE, то будет получено исходное число. Команда FXTRACT используется для вычисления значений показательных функций, преобразования чисел из временного вещественного формата в десятичный формат с плавающей точкой при выводе чисел на дисплей и принтер и др.

Трансцендентные команды (*Transcendental*). Эти команды выполняют расчет основных параметров, необходимых для вычисления тригонометрических, обратных тригонометрических, логарифмических и показательных функций. Операнды должны быть нормализованными числами и находиться в допустимом для каждой из команд диапазоне. Соблюдение выполнения этих требований возлагается на программиста.

Команда вычисления частичного тангенса FPTAN. Все тригонометрические функции угла α или $\alpha/2$ выражаются через отношение двух величин $Y/X = Z = \mathrm{tg}\,(\alpha/2)$ по известным формулам:

$$\sin\alpha = (2\cdot Z)/(1+Z^2),\ \cos\alpha = (1-Z^2)/(1+Z^2),\ \mathrm{tg}\,(\alpha/2) = Z,\ \mathrm{ctg}\,(\alpha/2) = 1/Z,$$
$$\sec\alpha = (1+Z^2)/(1-Z^2),\ \mathrm{cosec}\,\alpha = (1+Z^2)/(2\cdot Z),$$

поэтому в *NDCP* достаточно реализовать вычисление только одной функции $\mathrm{tg}\,(\alpha/2)$, а вычисление остальных функций можно выполнить с помощью арифметических команд *NDCP*.

Команда FPTAN по значению угла $\alpha/2 = \mathrm{ST}$, заданному в радианах, генерирует числа X и Y:

$$\mathrm{ST} \leftarrow Y = \mathrm{tg}\,(\alpha/2),\ Top \leftarrow Top - 1,\ \mathbf{ST(0)} \leftarrow X = 1,\ Tag(0) \leftarrow 00,\ \mathrm{ST}(1) = Y = \mathrm{tg}\,(\alpha/2),$$

такие, что $\mathrm{tg}\,(\alpha/2) = Y/X$ (ST — исходная вершина стека, **ST(0)** — следующая вершина стека). На допустимые значения углов $\alpha/2$ накладывается ограничение $0 \le \mathrm{ST} \le \pi/4$.

Задача 3. Вычислить значение $\mathrm{ctg}\,\pi/3$. Так как число $\pi/3 > \pi/4$, то сначала следует вычислить значение $\mathrm{tg}\,\pi/6 = Y/1$, затем значения $\sin\pi/3 = (2\cdot Y)/(1+Y^2)$ и $\cos\pi/3 = (1-Y^2)/(1+Y^2)$, и, наконец, $\mathrm{ctg}\,\pi/3 = (\cos\pi/3)/(\sin\pi/3)$. *Решение*:

```
        ⋮                ; В сегменте данных
num dw        3 * 2      ; 0006h
        ⋮                ; В сегменте кода
    FINIT                ; Инициализация NDCP (все регистры стека освобождаются)
    FLDPI                ; Top ← Top – 1, ST(0) ← π, Tag(0) ← 00
    FDIV     num         ; ST ← π/6
    FPTAN                ; ST ← Y, Top ← Top – 1, ST(0) ← X = 1, Tag(0) ← 00, ST(1) = Y = tg (π/6)
    FLD1                 ; Top ← Top – 1, ST(0) ← 1, Tag(0) ← 00, ST(1) = 1, ST(2) = Y
    FADD     ST, ST      ; ST ← 2, ST(1) = 1, ST(2) = Y
    FMUL     ST, ST(2)   ; ST ← 2 × Y, ST(1) = 1, ST(2) = Y
    FXCH     ST(2)       ; ST ← Y, ST(1) = 1, ST(2) ← 2 × Y
```

```
FMUL   ST, ST     ; ST ← Y², ST(1) = 1, ST(2) = 2 × Y
FST    ST(3)      ; ST = Y², ST(1) = 1, ST(2) = 2 × Y, ST(3) ← Y², Tag(3) ← 00
FADD   ST, ST(1)  ; ST ← 1 + Y², ST(1) = 1, ST(2) = 2 × Y, ST(3) = Y²
FDIV   ST(2), ST  ; ST = 1 + Y², ST(1) = 1, ST(2) ← sin π/3, ST(3) = Y²
FXCH   ST(3)      ; ST ← Y², ST(1) = 1, ST(2) = sin π/3, ST(3) ← 1 + Y²
FSUBP             ; ST(1) ← 1 - Y², Tag(0) ← 11, Top ← Top + 1, ST(0) = 1 - Y²,
                  ;      ST(1) = sin π/3, ST(2) = 1 + Y²
FDIV   ST, ST(2)  ; ST ← cos π/3, ST(1) = sin π/3, ST(2) = 1 + Y²
FDIVRP            ; ST ← ctg π/3 = 0,57735026918962576, Tag(0) ← 11, Top ← Top + 1,
                  ;      ST(0) = ctg π/3, ST(1) = 1 + Y²
```

Если последнюю команду FDIVRP заменить командой FDIVP, то будет вычислено значение tg π/3.

Команда вычисления частичного арктангенса FPATAN. Все обратные тригонометрические функции выражаются через функцию arctg (*Y*/*X*) по известным формулам:

$$\arcsin z = \operatorname{arctg}(Y/X) = \operatorname{arctg}\frac{z}{\sqrt{(1-z)\cdot(1+z)}}, \quad \arccos z = 2\cdot\operatorname{arctg}(Y/X) = 2\cdot\operatorname{arctg}\frac{\sqrt{1-z}}{\sqrt{1+z}},$$

$$\operatorname{arctg} z = \operatorname{arctg}(Y/X) = \operatorname{arctg}(z/1), \qquad \operatorname{arcctg} z = \operatorname{arctg}(Y/X) = \operatorname{arctg}(1/z),$$

$$\operatorname{arcsec} z = \operatorname{arctg}(Y/X) = \operatorname{arctg}\frac{1}{\sqrt{(z-1)\cdot(z+1)}}, \quad \operatorname{arccosec} z = 2\cdot\operatorname{arctg}(Y/X) = 2\cdot\operatorname{arctg}\frac{\sqrt{z-1}}{\sqrt{z+1}},$$

где z — аргумент обратных тригонометрических функций. Поэтому в *NDCP* достаточно реализовать вычисление только одной функции arctg z = arctg (*Y*/*X*). Значения же *Y* и *X* для соответствующих функций достаточно просто вычисляются с помощью арифметических команд *NDCP* по вышеприведенным формулам.

Команда FPATAN по заданному значению функции tg $\alpha = z$ выполняет вычисление в радианах угла α = arctg z = arctg (*Y*/*X*):

$$\mathrm{ST}(1) \leftarrow \operatorname{arctg}(Y/X),\ Tag \leftarrow 11,\ Top \leftarrow Top + 1,\ \mathbf{ST(0)} = \operatorname{arctg}(Y/X),$$

где X = ST и Y = ST(1) — исходные операнды (удаляются). Для исходных операндов должно удовлетворяться условие $0 \le \mathrm{ST}(1) < \mathrm{ST}(0) < +\infty$.

Если в *задаче* 3 немедленно после команды FPTAN выполнить команду FPATAN, то будет получено значение угла $\pi/6 \approx 0.52359877559829887$ = 3FFE 860A 91C1 6B9B 2C23 — *Temporary Real*.

Задача 4. Вычислить значение угла α по значению функции $\cos\alpha = z = 1/\sqrt{2}$. Для функции arccos z значения Y и X определяются соотношениями $Y = \sqrt{1-z}$ и $X = \sqrt{1+z}$, которые и следует вычислить до выполнения команды FPATAN. *Решение*:

```
∴                 ; В сегменте кода
FINIT             ; Инициализация NDCP (все регистры стека освобождаются)
FLD1              ; Top ← Top - 1, ST(0) ← +1.0, Tag(0) ← 00
FADD   ST, ST     ; ST ← 2
FSQRT             ; ST ← √2 = 0.70710678118654752
FLD1              ; Top ← Top - 1, ST(0) ← +1.0, Tag(0) ← 00, ST(1) = √2
```

```
FDIVR   ST(1), ST ; ST = +1.0, ST(1) ← 1 / √2 = z
FSUB    ST, ST(1) ; ST ← 1 – z, ST(1) = z
FSQRT             ; ST ← √(1 – z) = Y, ST(1) = z
FLD1              ; Top ← Top – 1, ST(0) ← +1.0, Tag(0) ← 00, ST(1) = Y, ST(2) = z
FADD    ST, ST(2) ; ST ← 1 + z, ST(1) = Y, ST(2) = z
FSQRT             ; ST ← √(1 + z) = X, ST(1) = Y, ST(2) = z
FPATAN    ; ST(1) ← arctg (Y/X), Tag ← 11, Top ← Top + 1, ST(0) = arctg (Y/X), ST(1) = z
FADD    ST, ST    ; ST ← 2 · arctg (Y/X) = arccos z = 0.7853981633974831 = π/4, ST(1) = z
```

Задача 5. Вычислить значения функции sin *x* для углов 0 ÷ 90° и вывести на экран монитора один ее период с учетом свойств симметрии функций sin *x* и cos *x* в диапазоне углов 0 ÷ 360° [22]. *Решение*:

```
s_seg   segment STACK     ; Сегмент стека
        db      16 dup ('Stack SP')  ; Размер стека равен 128 байт
s_seg   ends
d_seg   segment           ; Сегмент данных
angle   dw      0         ; Начальное значение угла
temp    dw      ?         ; Вспомогательная ячейка
cnst    dd      180.0     ; Константа преобразования
scale   dw      100       ; Коэффициент масштабирования
pos     dw      0
shft    db      0
buf     dw      91 dup (?) ; Массив результатов
colr    db      1         ; Задание цвета кривой
d_seg   ends              ; Конец сегмента данных
c_seg   segment           ; Сегмент кода
        assume  CS : c_seg, DS : d_seg, SS : s_seg
Start:  mov     ax, d_seg ; Инициализация сегментного регистра DS
        mov     ds, ax
        LEA     BX, buf   ; Загрузка адреса буфера buf
        MOV     SI, 0     ; Индекс для значений синуса
L3:     MOV     AX, angle ; Выборка значения угла
        CMP     AX, 45    ; 45° — граничное условие
        JG      L1        ; Если угол > 45°, то переход
        JMP     L2        ; Если нет, то продолжение
L1:     NEG     AX        ; AX ← 0 – AX
        ADD     AX, 90    ; AX ← AX + 90
L2:     MOV     temp, AX  ; M(temp) ← AX
        FINIT             ; Инициализация NDCP
        FLDPI             ; Запись числа π в регистр стека
        FLD     cnst      ; Запись числа cnst в регистр стека
        FDIV              ; Деление числа cnst на π
        FILD    temp      ; Загрузка округленного целочисленного значения угла
        FMUL              ; Умножение на предыдущее число
        FPTAN             ; Вычисление тангенса произведения
        MOV     AX, angle ; Проверка значения угла
```

```
        CMP     AX, 45      ; Если ≤ 45°, то синус
        JG      COS         ; Если > 45°, то косинус
        JMP     SIN
COS:    FXCH    ST(1)       ; ST(0) ↔ ST(1), если функция косинуса
SIN:    FMUL    ST, ST
        FXCH    ST(1)       ; ST(0) ↔ ST(1)
        FLD     ST          ; Начало вычислений синуса или косинуса
        FMUL    ST, ST
        FADD    ST, ST(2)
        FSQRT
        FDIVP   ST(1), ST
        FIMUL   scale
        FISTP   buf [SI]    ; Запись результата из регистра стека в память
        ADD     SI, 2       ; Переход к следующему значению синуса
        ADD     angle, 1    ; Увеличение значения угла
        CMP     angle, 90   ; Цикл для значений угла 0 ÷ 90°
        JLE     L3
; Вывод точек рассчитанной кривой на экран графического монитора
        MOV     AH, 0       ; Установка графического режима экрана
        MOV     AL, 12h     ; AL = 12h — 640 × 480 точек, 16 цветов (VGA)
        INT     10h         ; INT 10h, функция BIOS 00h (установка видеорежима)
        LEA     BX, buf     ; Загрузка адреса массива buf
L8:     MOV     SI, 0       ; Начало массива
        MOV     AX, pos     ; Пересылка значения угла в регистр AL
        CMP     AX, 180     ; Угол > 180°?
        JLE     L4          ; Если нет, то первый или второй квадрант
        SUB     AX, 180     ; Коррекция значения угла, если оно больше или равно 180°
L4:     CMP     AX, 90      ; Угол > 90°?
        JLE     L5          ; Если да, то квадрант 2
        NEG     AX          ; Изменение знака
        ADD     AX, 180     ; Коррекция значения угла, если оно больше или равно 90°
L5:     ADD     SI, AX      ; Суммарное смещение в регистре AX
        SHL     SI, 1       ; Индекс для отсчета слов (×2)
        MOV     AL, byte ptr buf [SI] ; Выборка значения и запись в массив buf
        CMP     pos, 180    ; Если угол > 180°, то сделать смещение на экране
        JGE     L6
        NEG     AL          ; Иначе изменить знак
        ADD     AL, 100     ; Коррекция значения прибавлением 100
        JMP     L7          ; Смещение точки на экране
L6:     ADD     AL, 99
L7:     MOV     shft, AL    ; Запись смещения в ячейку shft
        MOV     AH, 0Ch     ; Вывод изображения точки
        MOV     AL, colr    ; AL — номер цветового регистра от 0 до 15
        MOV     CX, pos     ; CX — графический столбец
        ADD     CX, 140
        MOV     DH, 1       ; DX — графическая строка
        MOV     DL, shft
```

```
        INT     10h         ; INT 10h, функция BIOS 0Ch (запись пиксела)
        ADD     pos, 1      ; Переход к следующему углу
        CMP     pos, 360    ; Выведены 360 точек?
        JLE     L8          ; Если нет, то повторить
; Ожидание нажатия клавиши для возврата в DOS и
;         переключения монитора в текстовый режим
        MOV     AH, 7       ; Чтение регистра клавиатуры с ожиданием ввода
        INT     21h         ; INT 21h, функция DOS 07h
        MOV     pos, 0
        MOV     shft, 0
        INC     colr        ; Изменение цвета кривой
        AND     colr, 0Fh
        JNZ     color
        MOV     AH, 0       ; Установка видеорежима монитора
        MOV     AL, 3       ; AL = 03 — текстовый режим 80 × 25 знаков, 16 цветов
        INT     10h         ; INT 10h, функция BIOS 00h (установка видеорежима)
        mov     ah, 4Ch     ; Выход из программы и возврат управления в DOS
        int     21h         ; INT 21h, функция DOS 4Ch
c_seg   ends
        end     Start
```

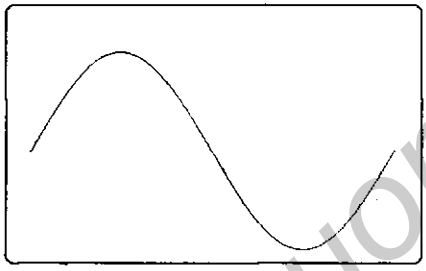


Рис. 4.44. Экран монитора при выполнении программы *задачи* 5

Программа выполняет вычисления в вещественном формате, но для построения изображения на экране монитора используются целые числа. Для этого вещественные числа округляются до целых и пересылаются в память. Изображение на экране монитора функции sin *x* для углов 0 ÷ 360° показано на рис. 4.44.

Для вывода изображения на экран монитора использованы функции *BIOS*:

INT 10*h*, функция *BIOS* 00*h* — установка графического и текстового видеорежимов,

INT 10*h*, функция *BIOS* 0C*h* — запись в видеобуфер точки заданного цвета (регистр *AL*) в указанную графическую позицию (регистры *CX* и *DX*).

Цвет каждого пиксела определяется числом, загруженным в регистр *AL*:

0 — черный,	4 — красный,	8 — серый,	12 — розовый,
1 — синий,	5 — фиолетовый,	9 — голубой,	13 — светло-фиолетовый,
2 — зеленый,	6 — коричневый,	10 — салатовый,	14 — желтый,
3 — бирюзовый,	7 — белый,	11 — светло-бирюзовый,	15 — ярко-белый.

Программа транслируется в *exe*-файл, который выполняется на любом персональном компьютере *IBM PC*.

Команда F2XM1. Эта команда выполняет операцию

$$ST \leftarrow 2^{ST} - 1,\ 0 \le ST \le 2^{-1},$$

где ST — содержимое регистра, находящегося в вершине стека. Вычисление $2^{ST} - 1$ вместо 2^{ST} позволяет получать точные нормализованные результаты даже при значениях ST близких

к нулю (например, если $2^{ST} = 1.0000003579$, то $2^{ST} - 1 = 0.0000003579$ и при нормализации добавляются младшие разряды мантиссы, которые в противном случае были бы потеряны). Используя известные формулы

$$10^X = 2^{X \log_2 10} \text{ при } 0 \le X \le 0.5 / \log_2 10 = 0.15051 \text{ и}$$

$$e^X = 2^{X \log_2 e} \text{ при } 0 \le X \le 0.5 / \log_2 e = 0.34657,$$

можно вычислять показательные функции для малых значений аргументов — константы $\log_2 10$ и $\log_2 e$ имеются во внутреннем постоянном запоминающем устройстве *NDCP*.

Задача 6. Вычислить значение показательной функции $10^{\pm X}$ как при $0 \le |U| \le 0.5$, так и при $|U| > 0.5$, где $U = X \times \log_2 10$. Так как при значениях $|U| > 0.5$ использовать команду F2XM1 нельзя, то вычисления можно выполнить по формуле

$$10^X = 2^U = 2^{FU \cdot 2^{EU}} = (2^{FU/4})^{2^{EU+2}} = [(2^{FU/4})^2]^{2^{EU+1}},$$

где *FU* — мантисса числа *U*, *EU* — истинный порядок числа *U*, причем всегда $FU/4 < 0.5$. *Решение*:

```
                ∴                       ; В сегменте данных
cnst    dd      0.5                     ; Константа 0.5
n_X     dq      23.3                    ; Число X (можно задавать числа любого знака)
buf_E   dw      'EE'                    ; 4545h
buf_F   dw      'FF'                    ; 4646h
                ∴                       ; В сегменте кода
        FINIT                           ; Инициализация NDCP (все регистры стека освобождаются)
        FLDL2T                          ; Top ← Top – 1, ST(0) ← log2 10, Tag(0) ← 00
        FMUL    n_X                     ; ST ← log2 10 × M(n_X) = ±X × log2 10 = ±U
        FIST    buf_F                   ; Сохранение в памяти знака числа ±U
        FABS                            ; ST ← |±U| = U
        FCOM    cnst                    ; Сравнение U с числом 0.5
        FSTSW   buf_E                   ; M(buf_E) ← SW — слово состояния NDCP
        MOV     AX, buf_E               ; AX ← M(buf_E) = SW
        SAHF                            ; ; Младший байт PSW ← старший байт SW
        JBE     Below                   ; Переход, если U ≤ 0.5
; Вычисление значения 10^X при U > 0.5
        FXTRACT         ; ST ← EU, Top ← Top – 1, ST(0) ← FU, Tag(0) ← 00, ST(1) = EU
        FXCH    ST(1)   ; ST ← EU, ST(1) ← FU
        FISTP   buf_E   ; M(buf_E) ← EU, Tag ← 11, Top ← Top + 1, ST(0) = FU
        FLD1            ; Top ← Top – 1, ST(0) ← 1, Tag(0) ← 00, ST(1) = FU
        FADD    ST, ST  ; ST ← 2, ST(1) = FU
        FDIV    ST(1), ST ; ST = 2, ST(1) ← FU/2
        FDIVP           ; ST(1) ← FU/4, Top ← Top + 1, ST(0) = FU/4
        F2XM1           ; ST ← 2^ST – 1 = 2^FU/4 – 1
        FLD1            ; Top ← Top – 1, ST(0) ← 1, Tag(0) ← 00, ST(1) = 2^FU/4 – 1
        FADDP           ; ST(1) ← 2^FU/4, Top ← Top + 1, ST(0) = 2^FU/4
        FST     ST(1)   ; ST = 2^FU/4, ST(1) ← 2^FU/4
        MOV     CX, buf_E ; CX ← EU
```

```
        MOV     AX, 2          ; AX ← 2
        SAL     AX, CL         ; AX ← AX × 2^CL = 2^(EU+1)
        FMUL    ST, ST(1)      ; ST ← (2^FU/4)^2, ST(1) = 2^FU/4
        FST     ST(1)          ; ST = (10^FU/4)^2, ST(1) ← (10^FU/4)^2
        DEC     AX             ; AX ← AX – 1
Exp:    FMUL    ST, ST(1)      ; ST ← ST × (2^FU/4)^2, ST(1) = (2^FU/4)^2
        DEC     AX             ; AX ← AX – 1
        JNZ     Exp
        JMP     short Sign     ; ST ← 10^+X, ST(1) = (2^FU/4)^2
; Вычисление значения 10^X при U ≤ 0.5
Below:  F2XM1                  ; ST ← 2^U – 1 = 10^+X – 1
        FLD1                   ; Top ← Top – 1, ST(0) ← 1, Tag(0) ← 00, ST(1) = 10^+X – 1
        FADDP                  ; ST(1) ← 10^+X, Tag ← 11, Top ← Top + 1, ST(0) = 10^+X
Sign:   MOV     AX, buf_F      ; AX ← M(buf_F)
        AND     AH, 80h        ; Анализ знака числа ±U
        JZ      FIN
        FLD1        ; Top ← Top – 1, ST(0) ← 1, ST(0) ← 1, Tag(0) ← 00, ST(1) = 10^+X
        FDIV    ST, ST(1)      ; ST ← 1 / 10^+X = 10^–X, ST(1) = 10^+X, ST(2) = (2^FU/4)^2
FIN:    ∴
```

Если команду FLDL2T заменить командой FLDL2E, то вышеприведенная программа будет вычислять значение показательной функции $e^{\pm X}$. Если же команду FLDL2T заменить командой FLD1, то будет вычисляться значение показательной функции $2^{\pm X}$. Примеры вычислений:

$10^{+23.3} = 1{,}9952623149688829e{+}23$, $10^{-23.3} = 5{,}0118723362727147e{-}24$;

$e^{+23.3} = 13154108760{,}016079$, $e^{-23.3} = 7{,}6021874096073512e{-}11$;

$2^{+23.3} = 10327587{,}874940477$, $2^{-23.3} = 9{,}682803110554637e{-}08$.

Вышеприведенный алгоритм вычисления показательных функций весьма сложен и неэффективен. Для вычисления показательных функций в любом диапазоне значений аргумента X можно использовать и другие алгоритмы, в частности, алгоритм, основанный на применении команды округления FRNDINT. В этом случае следует последовательно выполнить действия:

1) округлить число X до ближайшего целого Y (команда FRNDINT в режиме округления по умолчанию — RC = 00 в слове управления CW; см. рис. 4.40) и вычислить разность $Z = X - Y$, что обеспечивает выполнение условия $|Z| \le 1/2$; следовательно, будет выполняться соотношение

$$2^X = 2^Y \cdot 2^Z, \text{ где } Y \text{ — целое, } |Z| \le 1/2;$$

2) вычислить значение $2^Z - 1$ при $Z \ge 0$ или $2^{-Z} - 1$ при $Z < 0$ (команда F2XM1); во втором случае значение $2^Z - 1$ находится по формуле

$$2^Z - 1 = -(2^{-Z} - 1)/2^{-Z};$$

3) вычислить значение $2^Z = 2^Z - 1 + 1$, а затем значение функции $2^X = 2^Y \cdot 2^Z$ (команда FSCALE).

Задача 7. Написать программу для решения *задачи* 6 (вычисление значения показательной функции $10^{\pm X}$ как при $0 \le |U| \le 0.5$, так и при $|U| > 0.5$, $U = X \times \log_2 10$), с использованием вышеописанного алгоритма. *Решение*:

```
                ∴                       ; В сегменте данных
n_X     dq      23.3                    ; Число X (можно задавать числа любого знака)
buf_F   dw      'FF'                    ; 4646h
                ∴                       ; В сегменте кода
        FINIT                           ; Инициализация NDCP (все регистры стека освобождаются)
        FLDL2T                          ; Top ← Top – 1, ST(0) ← log2 10, Tag(0) ← 00
        FMUL    n_X                     ; ST ← log2 10 × M(n_X) = X × log2 10 = U
        FST                             ; ST(1) ← ST = U, Tag(1) ← 00, ST = U
        FRNDINT                         ; ST ← Y — округленное до целого число ST = U, ST(1) = U
        FXCH                            ; ST ↔ ST(1), ST = U, ST(1) = Y
        FSUB    ST, ST(1)               ; ST ← ST – ST(1) = U – Y = Z, ST(1) = Y
        FTST                            ; Сравнение ST = Z с 0
        FSTSW   buf_F                   ; M(buf_F) ← SW — слово состояния NDCP
        MOV     AX, buf_F               ; AX ← M(buf_F) = SW
        SAHF                            ; Младший байт PSW ← старший байт SW
        JB      Below                   ; Переход, если Z < 0
        F2XM1                           ; ST ← 2^ST – 1 = 2^Z – 1, ST(1) = Y
        JMP     L1
Below:  FCHS                            ; ST ← –ST = –Z, ST(1) = Y
        F2XM1                           ; ST ← 2^ST – 1 = 2^–Z – 1, ST(1) = Y
        FLD1                            ; Top ← Top – 1, ST(0) ← 1, Tag(0) ← 00,
                                        ;     ST = 1, ST(1) = 2^–Z – 1, ST(2) = Y
        FADD    ST, ST(1)               ; ST ← ST + ST(1) = 2^–Z, ST(1) = 2^–Z – 1, ST(2) = Y
        FDIVP                           ; ST(1) ← ST(1) / ST = (2^–Z – 1) / 2^–Z, Top ← Top + 1,
                                        ;     ST = (2^–Z – 1) / 2^–Z = 2^Z – 1, ST(1) = Y
        FCHS                            ; ST ← –ST = –(2^Z – 1), ST(1) = Y
L1:     FLD1                            ; Top ← Top – 1, ST(0) ← 1, Tag(0) ← 00,
                                        ;     ST = 1, ST(1) = 2^Z – 1 или –(2^Z – 1), ST(2) = Y
        FADDP                           ; ST(1) ← ST(1) + ST = 2^Z или –(2^Z – 2), Top ← Top + 1,
                                        ;     ST = 2^Z или –(2^Z – 2), ST(1) = Y
        FSCALE                          ; ST ← ST × 2^ST(1) = 10^X, ST(1) = Y.  Конец вычислений
```

Эта программа значительно проще предыдущей и выполняется за существенно меньшее время. *Задача* 6 иллюстрирует, сколь важен правильный выбор вычислительного алгоритма для увеличения производительности при выполнении вычислений с плавающей точкой.

Команда FYL2X. Эта команда выполняет операции над двумя числами X и Y:

$$ST(1) \leftarrow Y \cdot \log_2 X = ST(1) \cdot \log_2 ST,\ Tag \leftarrow 11,\ Top \leftarrow Top + 1,\ \mathbf{ST(0)} = Y \cdot \log_2 X,$$

где X = ST и Y = ST(1) — исходные операнды (удаляются), ST и **ST(0)** — исходная и следующая вершины стека. Для исходных операндов должны удовлетворяться условия: $0 < ST < \infty$ и $-\infty < ST(1) < +\infty$.

Используя известные формулы

$$X^Y = 2^{Y \log_2 X} \text{ при } 0 \le Y \cdot \log_2 X \le 0.5 \text{ и } \log_n X = \log_2 X / \log_2 n,$$

с помощью команды FYL2X можно вычислять показательные функции X^Y и логарифмы чисел по любому основанию. В частности $\log_{10} X = \log_2 X / \log_2 10$, где $\log_2 10$ — константа во внутреннем постоянном запоминающем устройстве *NDCP*.

Задача 8. Написать программу вычисления значений показательной функции $X^{\pm Y}$ при любых значениях модуля $|\pm Y \times \log_2 X|$ (как при значениях $0 \le |\pm Y \times \log_2 X| \le 0.5$, так и при значениях $|\pm Y \times \log_2 X| > 0.5$). *Решение*:

```
          ∴                     ; В сегменте данных
 n_X    dq        17.53         ; Число X (только положительные числа)
 n_Y    dq        –13.47        ; Число Y (числа любого знака)
 buf_F  dw        'FF'          ; 4646h
          ∴                     ; В сегменте кода
        FINIT                   ; Инициализация NDCP (все регистры стека освобождаются)
        FLD       n_Y           ; Top ← Top – 1, ST(0) ← M(n_Y) = Y, Tag(0) ← 00
        FLD       n_X           ; Top ← Top – 1, ST(0) ← M(n_X) = X, Tag(0) ← 00, ST(1) = Y
        FYL2X                   ; ST(1) ← Y × log2 X = U, Top ← Top + 1, ST(0) = U
; Далее повторяется программа задачи 7
        FST                     ; ST(1) ← ST = U, Tag(1) ← 00, ST = U
        FRNDINT                 ; ST ← Y — округленное до целого число ST = U, ST(1) = U
        FXCH                    ; ST ↔ ST(1), ST = U, ST(1) = Y
          ∴                     ;  и т. д. (программа задачи 7)
```

Примеры вычислений:

$$17.53^{+13.47} = 56720808123666196,\ \ 17.53^{-13.47} = 1.7630214256111064\text{e}{-17},$$

Задача 9. Вычислить значение десятичного логарифма $\log_{10} X = \log_2 X / \log_2 10$. *Решение*:

```
        ∴                       ; В сегменте данных
n_X   dq        79583.243       ; Число X (только положительные числа)
        ∴                       ; В сегменте кода
      FINIT                     ; Инициализация NDCP (все регистры стека освобождаются)
      FLD1                      ; Top ← Top – 1, ST(0) ← 1, Tag(0) ← 00
      FLD       n_X             ; Top ← Top – 1, ST(0) ← M(n_X) = X, Tag(0) ← 00, ST(1) = 1
      FYL2X                     ; ST(1) ← log2 X, Top ← Top + 1, ST(0) = log2 X
      FLDL2T                    ; Top ← Top – 1, ST(0) ← log2 10, Tag(0) ← 00, ST(1) = log2 X
      FDIVP                     ; ST(1) ← log2 X / log2 10 = log10 X, Top ← Top + 1,
                                ;       ST(0) = log10 X = 4.9008216325774979
```

Команда FYL2XP1. Эта команда выполняет операции над двумя числами X и Y:

$$\text{ST}(1) \leftarrow Y \cdot \log_2(X+1) = \text{ST}(1) \cdot \log_2(\text{ST}+1),$$

$$Tag \leftarrow 11,\ \ \text{TOP} \leftarrow \text{TOP}+1,\ \mathbf{ST(0)} = Y \cdot \log_2(X+1),$$

где $X = \text{ST}$ и $Y = \text{ST}(1)$ — исходные операнды (удаляются), ST и **ST(0)** — исходная и следующая вершины стека. Для исходных операндов должны удовлетворяться условия

$$0 \le |\text{ST}| < (2-\sqrt{2})/2,\ -\infty < \text{ST}(1) < +\infty.$$

Команда FYL2XP1 обеспечивает большую точность по сравнению с командой FYL2X при вычислении логарифмов чисел, близких к 1 — задать в нормализованном малом числе ε можно больше значащих разрядов, чем в нормализованном числе 1 + ε.

Для вычисления гиперболических функций используются формулы [23]:

$$\mathrm{sh}\,(x)=\frac{e^{x}-e^{-x}}{2}=\frac{\mathrm{sign}\,(x)}{2}\left[\left(e^{|x|}-1\right)+\frac{e^{|x|}-1}{e^{|x|}}\right]$$ — синус гиперболический,

$$\mathrm{ch}\,(x)=\frac{e^{x}+e^{-x}}{2}=\frac{1}{2}\left(e^{|x|}+\frac{1}{e^{|x|}}\right)$$ — косинус гиперболический,

$$\mathrm{th}\,(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=\mathrm{sign}\,(x)-\frac{e^{2|x|}-1}{e^{2|x|}+1}$$ — тангес гиперболический,

$$\mathrm{cth}\,(x)=\frac{e^{x}+e^{-x}}{e^{x}-e^{-x}}=\frac{1}{\mathrm{th}\,(x)}$$ — котангес гиперболический,

$$\mathrm{csch}\,(x)=\frac{2}{e^{x}-e^{-x}}=\frac{1}{\mathrm{sh}\,(x)}$$ — косеканс гиперболический,

$$\mathrm{sch}\,(x)=\frac{2}{e^{x}+e^{-x}}=\frac{1}{\mathrm{ch}\,(x)}$$ — секанс гиперболический,

где sign (*Signum*) — знак аргумента *x*.

Для вычисления обратных гиперболических функций (ареафункций) используются формулы [23]:

$$\mathrm{arsh}\,(x)=[\mathrm{sign}\,(x)]\cdot(\log_e 2)\cdot[\log_2(1+z)],\ \text{где}\ z=|x|+\frac{|x|}{\frac{1}{|x|}+\sqrt{1+\left(\frac{1}{|x|}\right)^2}};$$

$$\mathrm{arch}\,(x)=(\log_e 2)\cdot[\log_2(1+z)],\ \text{где}\ z=x-1+\sqrt{(x-1)\cdot(x+1)}\ \text{и}\ x\geq 1;$$

$$\mathrm{arth}\,(x)=[\mathrm{sign}\,(x)]\cdot(\log_e 2)\cdot[\log_2(1+z)],\ \text{где}\ z=\frac{2|x|}{1-|x|}\ \text{и}\ -1<x<1;$$

$$\mathrm{arcth}\,(x)=\mathrm{arth}\,(1/x);\quad \mathrm{arcsch}\,(x)=\mathrm{arsh}\,(1/x);\quad \mathrm{arsch}\,(x)=\mathrm{arch}\,(1/x).$$

Для вычисления некоторых из этих функций требуется использовать команду FYL2XP1.

Команды управления сопроцессором (*Processor Control*). Эти команды ориентированы в основном на выполнение системных операций. С их помощью можно произвести инициализацию *NDCP* и задать его режим работы (например, выбрать режимы округления и управления бесконечностью), передать в память код условия результата команд сравнения FCOM и проверки FTST для дальнейшего его использования в МП и др.

Инструкции WAIT/FWAIT в исходный текст вводить не следует — транслятор их добавляет автоматически. Перед шестью командами управления, отмеченных звездочкой (см. табл. 4.32), ассемблер не вводит команд ожидания WAIT, так как по ним производятся элементарные внутренние операции *NDCP*. Восемь команд управления имеют альтернативные мнемоники F××× и FN×××. Второй тип команд используется для указания ассемблеру не вводить перед ними команд ожидания WAIT.

Команда FINIT/FNINIT. Эта команда выполняет операции

$$\mathrm{CW}\leftarrow 03\mathrm{FF}h,\ \mathrm{SW}\leftarrow 0000h,\ \mathrm{TW}\leftarrow \mathrm{FFFF}h$$ — инициализация *NDCP*,

которые эквивалентны воздействию значения сигнала RESET = 1 за исключением того, что команда FINIT/FNINIT не влияет на синхронизацию выборки команд программы из памяти.

Команды FENI/FNENI и FDISI/FNDISI. Эти команды выполняют операции над разрядом *M* в слове управления *CW*:

$(M)_{CW} \leftarrow 0$ — разрешение запросов прерываний (команда FENI/FNENI),

$(M)_{CW} \leftarrow 1$ — запрет запросов прерываний (команда FDISI/FNDISI),

т. е. управляют разрешением и запретом вызова процедур обработки особых случаев.

Команды FSTCW/FNSTCW *dst* и FLDCW *src*. Эти команды выполняют операции:

$M(dst) \leftarrow CW$; $M(dst)$ — переменная типа WORD,

$CW \leftarrow M(src)$ — переменная типа WORD.

Команда FSTCW *dst* осуществляет передачу в память текущего слова управления *CW*, делая его доступным для МП 8086 — с помощью логических команд можно изменить слово управления *CW*, а затем командой FLDCW загрузить его в регистр *CW* с целью задания нового режима работы *NDCP*.

Если в слове состояния *SW* какой-либо незамаскированный флаг особого случая установлен в 1, то загрузка нового слова управления *CW* при значении разряда $M = 0$ приводит к генерированию запроса прерывания (сигнал *INT* ⎍ — см. рис. 4.31) перед выполнением следующей команды. Поэтому перед загрузкой нового слова управления *CW* рекомендуется сбрасывать флаги особых случаев в слове состояния.

Команда FSTSW/FNSTSW *dst*. Эта команда выполняет операцию

$M(dst) \leftarrow SW$; $M(dst)$ — переменная типа WORD.

Команда FSTSW *dst* осуществляет передачу в память текущего слова состояния *SW*, делая его доступным для МП 8086 с целью выполнения ветвления программы с помощью команд условных переходов (см. *задачу* 6 на с. 503 — команда FCOM и далее) или для обработки замаскированных особых случаев (без использования прерываний).

Команда FCLEX/FNCLEX. Эта команда выполняет операцию

$(B/IR/PE/UE/OE/ZE/DE/IE)_{SW} \leftarrow 0$ — сброс особых случаев в слове состояния *SW*.

Если какие-либо особые случаи не замаскированы, то процедура обработки особых случаев перед возвратом в прерванную программу должна выполнить команду FCLEX/FNCLEX — в противном случае сразу же будет сформирован новый запрос прерывания.

Команды FSTENV/FNSTENV *dst* и FLDENV *src*. Эти команды выполняют операции:

$M(dst) \leftarrow Environment$ — 7 слов среды *NDCP*,

$Environment \leftarrow M(src)$ — 7 слов среды *NDCP*.

Команда FSTENV/FNSTENV делает доступной для МП 8086 наиболее важную информацию о состоянии *NDCP* (см. рис. 4.42). Эта информация используется для обработки незамаскированных особых случаев в вызываемых по прерыванию процедурах обработки особых случаев. При выполнении команды FSTENV/FNSTENV маски всех особых случаев в слове управления *CW* устанавливаются в единицу, а разряд *M* остается неизменным. Среда *NDCP* обычно сохраняется в стеке МП 8086, как и полное состояние *NDCP* (см. *пример* 5).

Команда FLDENV *src* осуществляет загрузку среды *NDCP* из памяти. Если в слове управления *CW* разряд $M = 0$, то загрузка среды с незамаскированным особым случаем вызовет немедленный запрос прерывания.

Команды FSAVE/FNSAVE *dst* *и* FRSTOR *src* . Эти команда выполняют операции:

M(*dst*) ← *State* — 47 слов полного состояния *NDCP* (команда FSAVE/FNSAVE *dst*),

State ← *M*(*src*) — 47 слов полного состояния *NDCP* (команда FRSTOR *src*),

т. е. производят сохранение в памяти и загрузку из памяти полного состояния *NDCP* (94 байта) в формате, приведенном на рис. 4.43. При выполнении команды FSAVE/FNSAVE автоматически производится инициализация *NDCP*, аналогичная воздействию команды FINIT. Альтернативную мнемонику FNSAVE использовать нельзя, если разрешены прерывания в МП 8086.

Команды FSAVE/FNSAVE и FRSTOR обычно используется в тех случаях, когда *NDCP* необходимо переключить на выполнение другой задачи, например, процедуры обработки прерывания с применением команд *NDCP* по запросу прерывания от внешнего устройства. В таких случаях для сохранения полного состояния *NDCP* удобно использовать стек МП 8086, адресуя его с помощью указателя базы BP (см. табл. 4.9).

Сопроцессор по завершении выполнения команды FRSTOR немедленно реагирует на новое состояние — выдает значение сигнала *INT* = 1, если какой-либо флаг особых случаев в загруженном слове состояния не замаскирован.

Пример 5 (оформление процедуры обработки внешнего прерывания INT *type* при имитации прерывания командой CALL):

```
s_seg  segment  STACK      ; Сегмент стека
       dw       64 dup (?) ; Резервирование 128 байт для стека
s_seg  ends
d_seg  segment             ; Сегмент данных
n_X    dq       79583.243  ; Число X
n_Y    dq       3943.1938  ; Число Y
n_Z    dq       1936.4339  ; Число Z
buf_P  dq       'PP'       ; 0000000000005050h
d_seg  ends                ; Конец сегмента данных
c_seg  segment             ; Сегмент кода
       assume   CS : c_seg, DS : d_seg, SS : s_seg, ES : d_seg
start: mov      ax, d_seg  ; Инициализация сегментного регистра DS
       mov      ds, ax
       mov      es, ax     ; ES = DS
       FINIT               ; Инициализация NDCP (все регистры стека освобождаются)
       FLD1                ; TOP ← TOP – 1, ST(0) ← 1, Tag(0) ← 00
       FLD      n_X        ; TOP ← TOP – 1, ST(0) ← M(n_X) = X, Tag(0) ← 00, ST(1) = 1
       FYL2X               ; ST(1) ← log2 X, TOP ← TOP + 1, ST(0) = log2 X
       CALL     INTR       ; Имитация вызова процедуры обработки внешнего прерывания INT type
       FLDL2T              ; TOP ← TOP – 1, ST(0) ← log2 10, Tag(0) ← 00, ST(1) = log2 X
       FDIVP               ; ST(1) ← log2 X / log2 10 = log10 X, TOP ← TOP + 1,
                           ;     ST(0) = log10 X = 4.9008216325774979
         ∴                 ; Продолжение основной программы
       JMP      FIN        ; Конец основной программы
INTR   proc     near       ; Имитируемая процедура обработки внешнего прерывания INT type
       PUSH     AX         ; Сохранение состояния основной программы. Предполагается, что
       PUSH     BX         ; в основной программе и процедуре обработки внешнего прерывания
       PUSH     CX         ; эти регистры используются
```

```
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    BP
        SUB     SP, 94   ; SP ← SP – 94 (резервирование пространства в стеке)
        MOV     BP, SP   ; BP ← SP (адрес памяти для сохранения состояния NDCP)
        FSAVE   [BP]     ; Сохранение в стеке полного состояния NDCP и инициализация NDCP
        FLD     n_Y      ; TOP ← TOP – 1, ST(0) ← M(n_Y), Tag(0) ← 00
        FMUL    n_X      ; ST ← M(n_Y) × M(n_Z) = 7635734.14858982
        FST     buf_P    ; M(buf_P) ← M(n_Y) × M(n_Z)
          ∴              ; Продолжение процедуры обработки внешнего прерывания INT type
        MOV     BP, SP   ; BP ← SP (адрес полного состояния NDCP)
        FRSTOR  [BP]     ; Восстановление полного состояния NDCP
        ADD     SP, 94   ; SP ← SP – 94 (освобождение пространства в стеке)
        POP     BP       ; Восстановление состояния основной программы
        POP     DI
        POP     SI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        RET              ; Возврат в основную программу
INTR    endp
FIN:    mov     ah, 4Ch  ; Выход из программы (передача управления операционной системе)
        int     21h      ; INT 21h — функция DOS 4Ch
c_seg   ends
        end     start
```

Команда FINCSTP и FDECSTP. Эти команды выполняют операции:

$$TOP \leftarrow TOP + 1 \Rightarrow \mathbf{ST(0)} \text{ — новая вершина стека (команда FINCSTP),}$$

$$TOP \leftarrow TOP - 1 \Rightarrow \mathbf{ST(0)} \text{ — новая вершина стека (команда FDECSTP),}$$

т. е. данные команды не изменяют ни содержимого регистров стека, ни содержимого регистра тэгов *TW*, а только циклически сдвигают адреса *i* регистров ST(*i*) в ту или другую сторону, назначая новую вершину стека.

Команда FFREE и FFREE ST(*i*). Эти команды выполняют операции:

$Tag(1) \leftarrow 11$ — освобождение регистра ST(1) при выполнении команды FFREE,

$Tag(i) \leftarrow 11$ — освобождение регистра ST(*i*) при выполнении команды FFREE ST(*i*),

т. е. данные команды записывают в тэг указанного регистра код, отмечающий его в качестве свободного регистра (*Empty*), не изменяя его содержимого.

Команда FNOP. Эта команда не производит никаких операций.

Команда FWAIT/WAIT. Эта команда переводит МП 8086 в состояние ожидания завершения выполнения сопроцессором 8087 следующей за командой FWAIT/WAIT команды *NDCP* — ожидание значения сигнала *BUSY* = 0.

Система команд *NDCP* 80287. Данный *NDCP* предназначен для совместной работы с МП 80286. Его система команд отличается от системы команд *NDCP* 8087 только двумя добавочными двухбайтовыми командами, относящимися к группе команд управления:

FSETPM (*Set Protected Mode* — установка защищенного режима),

FSTSW AX (*Store Status Word* — передача в регистр AX слова состояния *SW*).

Машинные коды этих команд равны DB*h* (первый байт), E4*h* (второй байт) и DF*h* (первый байт), E0*h* (второй байт) соответственно. Команда FSTSW AX выполняет операцию AX ← *SW* и используется для непосредственной передачи кода условия в МП 80286 с целью дальнейшей реализации условных переходов. Команда установки защищенного режима FSETPM переводит *NDCP* 80287 в виртуальный режим. После аппаратного сброса значением сигнала RESET = 1 *NDCP* 80287, как и МП 80286, переводится в реальный режим. Для *NDCP* 80287 различие реального и защищенного режимов выражается лишь в формате указателя особых случаев: для реального режима формат такой же, что и для *NDCP* 8087 (см. рис. 4.42), а для защищенного режима указатели инструкции и данных содержат смещение и селектор [23].

Перед командами *NDCP* 80287 ассемблер не вводит автоматически команд ожидания WAIT, так как предусмотрена параллельная работа МП 80286 и *NDCP* 80287. Когда МП 80286 встречает команду *NDCP* 80287, он ожидает, пока *NDCP* не закончит выполнение текущей команды, чтобы позволить *NDCP* начать выполнение новой команды. Затем МП сразу же переходит к выполнению следующих своих команд программы (если они есть), до тех пор, пока не встретится новая команда *NDCP*. Пока *NDCP* выполняет свою "медленную" команду, МП способен выполнить большое количество своих команд, если выполнение не будет прервано появлением очередной команды *NDCP*. Такая организация взаимодействия (синхронизации) МП 80286 и *NDCP* 80287 позволяет существенно повысить производительность МП-системы.

Программист должен самостоятельно вводить команды ожидания WAIT в тех случаях, когда операнд, пересылаемый сопроцессором в память, будет востребован микропроцессором до завершения пересылки. В противном случае МП прочитает из памяти неверное значение операнда.

Пример 6 (синхронизация работы МП 80286 и *NDCP* 80287):

```
        ∴                       ; В сегменте данных
num_X   dd      993493.8391     ; 49728D5Dh -- Short Real
buf     dd      'DD'            ; 00004444h
        ∴                       ; В сегменте кода
        FLD     num_X           ; Top ← Top – 1, ST(0) ← M(num_X), Tag(0) ← 00
        FSQRT                   ; ST ← √993493.8391 = 996.74159765708585
        FRNDINT     ; ST ← 997d = 000003E5h (округление результата до целого числа)
        FIST    buf             ; M(buf) ← ST = 000003E5h (передача операнда в память)
        MOV     DX, 3943h       ; DX ← 3943h — адрес порта вывода (команды МП 80286)
        OUT     DX, AL          ; I/O(DX) ← AL
        WAIT        ; Синхронизация (МП будет использовать результат вычислений NDCP)
        MOV     AX, word ptr buf    ; AX ← 03E5h = 997d
        MOV     DX, word ptr buf + 2  ; DX ← 0000h
        ∴
```

Команда WAIT переводит МП 80286 в состояние ожидания завершения передачи операнда в память сопроцессором 80287.

4.8. Генератор тактовых сигналов 8284 и контроллер шин 8288

Для построения МП-системы на МП 8086/8087 необходимы дополнительные ИС — генератор тактовых сигналов 8284 (отечественный аналог 1810ГФ84) или 8284*A* и контроллер шин 8288 (отечественный аналог 1810ВГ88), генерирующий системные сигналы управления [24].

Генераторы тактовых сигналов 8284 и 8284*A*. Эти интегральные схемы (рис. 4.45) изготовляются по ТТЛШ-технологии и обеспечивают генерацию тактового сигнала при использовании кварцевых резонаторов, имеющих резонансные частоты:

$$12\ \text{МГц} \le f_{OSC} \le 25\ \text{МГц (8284)},\quad 12\ \text{МГц} \le f_{OSC} \le 30\ \text{МГц (8284}A\text{)}.$$

Фирмой *Intel* выпущены также функциональные аналоги генератора 8284*A* по технологии *CHMOS*: 82*C*84*A* и 82*C*84*A*–5 (2,4 МГц $\le f_{OSC} \le$ 25 МГц и 6 МГц $\le f_{OSC} \le$ 15 МГц). Основные параметры генераторов приведены в табл. 4.35. Максимальная рассеиваемая мощность составляет 1 Вт.

Структурная схема генератора 8284 изображена на рис. 4.46, *а*. Генератор содержит узлы:

кварцевый генератор *XTAL OSC*;

синхронизируемые внешним сигналом *CSYNC* делители частоты с коэффициентами деления 3 (триггеры Q_0 и Q_1) и 2 (триггер Q_2) — рис. 4.46, *б*;

схемы формирования тактового сигнала $\overline{H}$ для делителей частоты и сигнала готовности *RDYIN*, выполненные на ЛЭ И–ИЛИ–НЕ;

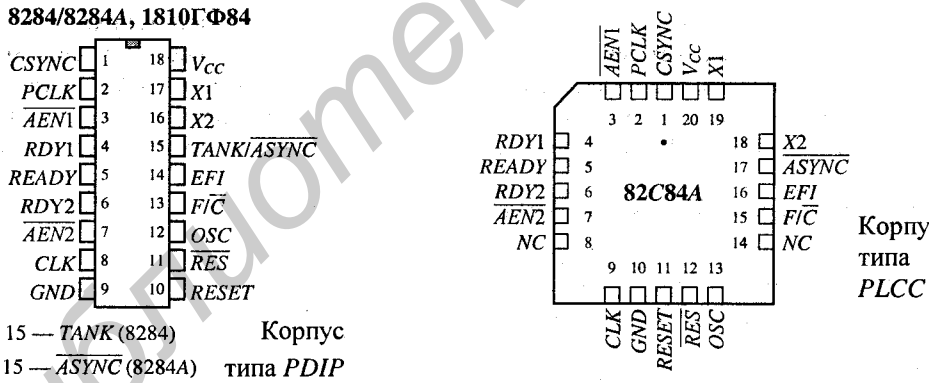


Рис. 4.45. Генераторы тактовых сигналов

Таблица 4.35. **Основные параметры генераторов**

БИС	$I_{CC\ max}$, мА	$V_{OH\ min}/I_{OH}$, В/мА	$V_{OL\ max}/I_{OL}$, В/мА	$V_{OH\ min}/I_{OH}$, В/мА (*CLK*)	$V_{OL\ max}/I_{OL}$, В/мА (*CLK*)	F_{CLK}, МГц
8284	140	2,4/–1	0,45/5	4/–1	0,45/5	5
8284*A*	170	2,4/–1	0,45/5	4/–1	0,45/5	8
8284*A*–1	170	2,4/–1	0,45/5	4/–1	0,45/5	10
82*C*84*A*	40	(V_{CC} – 0,4)/–2,5	0,4/2,5	(V_{CC} – 0,4)/–4	0,4/4	8
82*C*84*A*–5	10	3/–2,5	0,4/2,5	3/–2,5	0,4/4	5

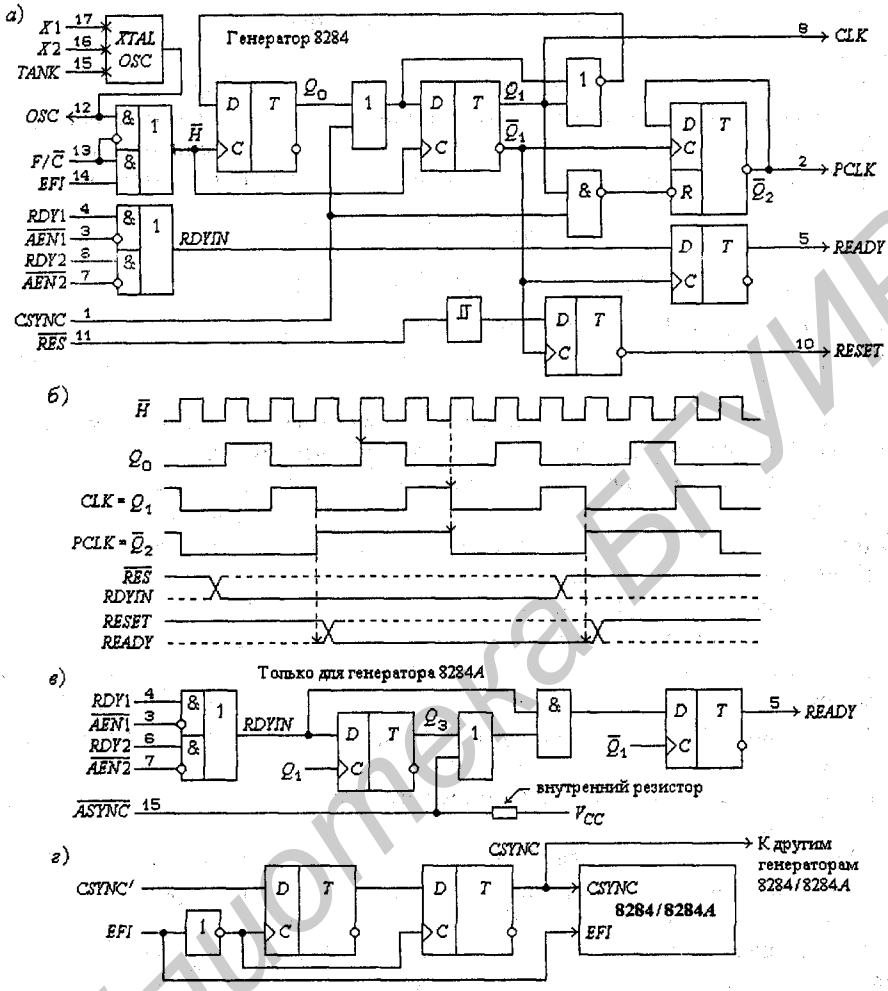


Рис. 4.46. Генераторы тактовых сигналов 8284/8284*A*

схемы привязки временного положения входных сигналов готовности *RDYIN* и сброса $\overline{RES}$ к тактовому сигналу *CLK*, выполненные на синхронных *D*-триггерах (рис. 4.46, *б*).

Генератор 8284*A* отличается от генератора 8284 только схемой формирования сигнала готовности *READY* (рис. 4.46, *в*) и отсутствием входа *TANK*, предназначенного для задания условий генерации тактового сигнала на субгармониках кварцевого резонатора. Сигналы генераторов имеют назначение:

X_1, X_2 (*Crystal In*) — контакты для подключения кварцевого резонатора (кристалла). Его резонансная частота f_{OSC} должна быть в три раза больше желательной частоты синхронизации МП — частоты выходного сигнала генератора *CLK*;

OSC (*Oscillator*) — выходной сигнал генератора *XTAL OSC*. Его частота равна f_{OSC};

CLK (*Processor Clock*) — выходной сигнал генератора, используемый для синхронизации МП и всех устройств, которые подключаются непосредственно к локальной шине МП. Частота этого сигнала равна $f_{CLK} = f_{OSC} / 3$, скважность равна 3 (рис. 4.46, *б*), а уровень логической 1

равен 4,5 В при напряжении питания V_{CC} = 5 В, что требуется для надежного управления МОП-устройствами;

EFI (*External Frequency In*) — входной сигнал внешней синхронизации, который может быть использован для задания частоты синхронизации вместо кварцевого резонатора;

$F/\overline{C}$ (*Frequency/Crystal Select*) — сигнал выбора источника синхронизации (кварцевого резонатора или сигнала внешней синхронизации *EFI*) в соответствии с выражением:

$$\overline{H} = \overline{OSC \cdot F/\overline{C}} \vee EFI \cdot F/\overline{C}$$

(мультиплексная функция: $F/\overline{C} = 0 \Rightarrow \overline{H} = \overline{OSC}$, $F/\overline{C} = 1 \Rightarrow \overline{H} = EFI$);

PCLK (*Peripheral Clock*) — выходной сигнал ТТЛ-уровней, предназначенный для синхронизации периферийных устройств. Частота сигнала равна $f_{PCLK} = f_{CLK} / 2$, а скважность равна 1/2 (рис. 4.46, *б*);

$\overline{AEN_1}$, $\overline{AEN_2}$ (*Address Enable*) — сигналы разрешения доступа к двум шинам данных, имеющие низкий активный уровень. Эти сигналы используются для управления готовностью МП в мультипроцессорных конфигурациях МП-системы с несколькими ведущими системной шины (*Multi-Master System Bus*) [12, 13, 24] — на вход $\overline{AEN_1}$ подается сигнал $\overline{AEN}$ от арбитра шин 8289 (см. рис. 4.49, *б*; *BA* — *Bus Arbiter*) для перевода МП в состояние ожидания доступа к системной шине ($\overline{AEN}$ = 1), которая может быть занята другим МП для пересылок данных. В не мультипроцессорных конфигурациях следует положить $\overline{AEN_i} = 0$;

RDY_1, RDY_2 (*Bus Ready*) — входные сигналы готовности передачи данных, поступающие от устройств, подключенных к двум шинам данных (см. рис. 4.62). Генератор из сигналов RDY_i и $\overline{AEN_i}$ формирует внутренний общий сигнал готовности $RDYIN = RDY_1 AEN_1 \vee RDY_2 AEN_2$;

READY — выходной сигнал готовности передачи данных, представляющий собой привязанный к тактовому сигналу *CLK* общий сигнал готовности *RDYIN* (рис. 4.46, *б*). Этот сигнал подается на одноименный вход МП для перевода его значением *READY* = 0 в состояние ожидания значения *READY* = 1;

$\overline{RES}$ (*Reset In*) — входной сигнал сброса, имеющий низкий активный уровень. Этот сигнал подается на триггер Шмитта для формирования сигнала с крутыми фронтами из медленно изменяющегося сигнала, создаваемого *RC*-цепью при включении питания;

RESET — выходной сигнал сброса МП (высокого активного уровня), представляющий собой привязанный к тактовому сигналу *CLK* входной сигнал сброса $\overline{RES}$ (рис. 4.46, *б*);

CSYNC (*Clock Synchronization*) — сигнал синхронизации тактовых сигналов нескольких генераторов 8284/8284*A*, позволяющий обеспечить синфазность их работы (значением сигнала *CSYNC* = 1 делители частоты всех генераторов сбрасываются в нулевое состояние, а при переходе сигнала *CSYNC* с 1 на 0 делители частоты начинают работать с одного и того же начального состояния). Сигнал *CSYNC* следует привязать к сигналу внешней синхронизации *EFI* (рис. 4.46, *г*). При использовании внутреннего генератора вход *CSYNC* необходимо заземлить (подать логический 0);

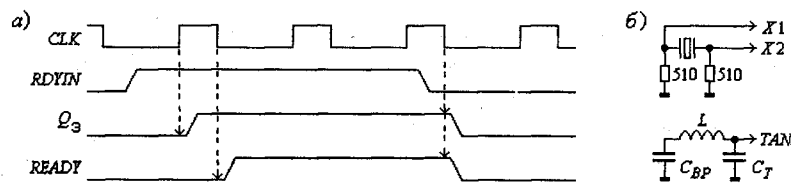


Рис. 4.47. Управление сигналом готовности и частотой генератора

$\overline{ASYNC}$ (*Ready Synchronization Select*) — входной сигнал, задающий способ временно́й привязки сигнала готовности *RDYIN* к тактовому сигналу *CLK* (только для генератора 8284*A*). При значении сигнала $\overline{ASYNC}$ = 1 получается такой же способ привязки, что и в генераторе 8284. При значении же сигнала $\overline{ASYNC}$ = 0 получается двухступенчатая схема временной привязки (рис. 4.47, *а*);

TANK — контакт для подключения внешнего параллельного колебательного *LC*-контура (рис. 4.47, *б*), настраиваемого на субгармонику кварцевого резонатора

$$f = 1/2\pi\sqrt{LC_T}\ ,\ C_{BP} >> C_T \text{ (только для генератора 8284).}$$

Контроллер шин 8288 (1810ВГ88). Эта ИС изготовляется по ТТЛШ-технологии и обеспечивает генерацию сигналов управления при частоте $f_{CLK} \le 10$ МГц тактового сигнала *CLK* (рис. 4.48). Фирмой *Intel* выпущен также функциональный аналог контроллера шин 8288 по *CHMOS*-технологии — 82*C*88. Основные параметры контроллеров шин приведены в табл. 4.36 (P_{DIS} — *Power Dissipation* — рассеиваемая мощность).

Контроллер шин 8288 используется для построения средних и больших МП-систем, обеспечивая генерацию всех необходимых сигналов управления памятью и внешними устройствами (см. рис. 4.13). Структурная схема контроллера шин изображена на рис. 4.49, *а*. Контроллер содержит узлы:

Status Decoder — дешифратор сигналов состояния МП $\overline{S}_{2-0}$, идентифицирующих операцию, выполняемую микропроцессором в текущем цикле шины;

Control Logic — устройство управления;

Control Signal Generator — генератор *сигналов управления* адресным регистром, приемопередатчиком данных и контроллерами прерываний (вспомогательные сигналы управления);

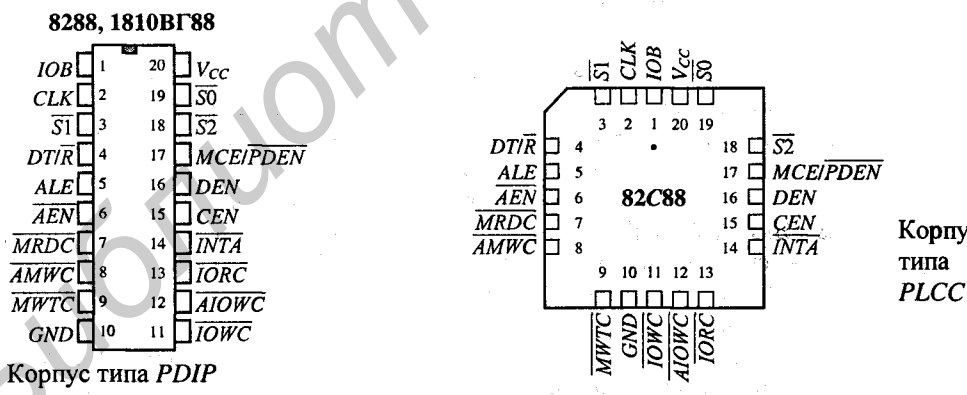


Рис. 4.48. Контроллеры шин

Таблица 4.36. **Основные параметры контроллеров шин**

БИС	$I_{CC\ max}$, мА	P_{DIS}, Вт	Сигналы команд		Сигналы управления		F_{CLK}, МГц
			$V_{OH\ min}/I_{OH}$, В/мА	$V_{OL\ max}/I_{OL}$, В/мА	$V_{OH\ min}/I_{OH}$, В/мА	$V_{OL\ max}/I_{OL}$, В/мА	
8288	230	1,5	2,4/–5	0,5/32	2,4/–1	0,5/16	10
82*C*88	8	1,0	3/–8	0,5/20	3/–4	0,4/8	8

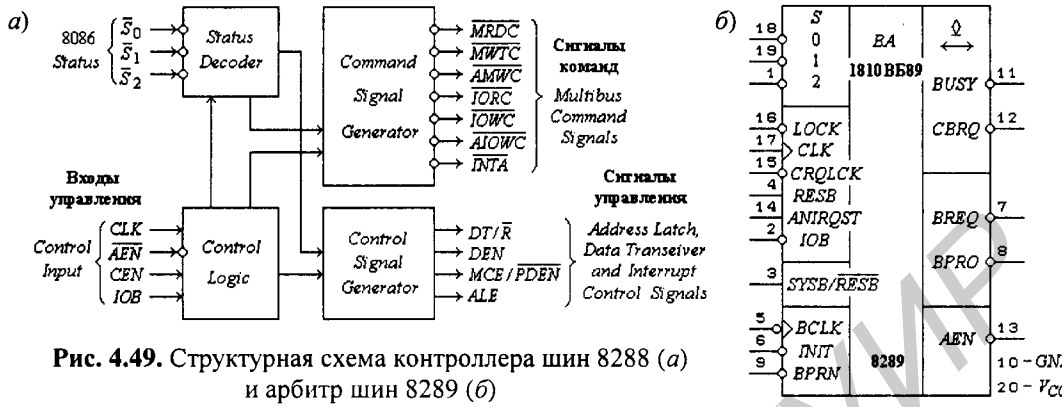


Рис. 4.49. Структурная схема контроллера шин 8288 (*а*) и арбитр шин 8289 (*б*)

Таблица 4.37. **Сигналы состояния МП 8086**

$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	Состояние МП	Сигналы ИС 8288
0	0	0	*Interrupt Acknowledge* (подтверждение прерывания)	$\overline{INTA} = 0$
0	0	1	*Read I/O Port* (ввод)	$\overline{IORC} = 0$
0	1	0	*Write I/O Port* (вывод)	$\overline{IOWC} = 0$ и $\overline{AIOWC} = 0$
0	1	1	*Halt* (останов)	*None*
1	0	0	*Code Access* (выборка команды)	$\overline{MRDC} = 0$
1	0	1	*Read Memory* (чтение памяти)	$\overline{MRDC} = 0$
1	1	0	*Write Memory* (запись в память)	$\overline{MWTC} = 0$ и $\overline{AMWC} = 0$
1	1	1	*Passive* (пассивное состояние)	*None*

Command Signal Generator — генератор *сигналов команд* (системных сигналов управления), обеспечивающих чтение и запись данных в память, ввод и вывод данных во внешние устройства, а также чтение типа прерывания *type* из контроллера прерываний 8259*A*.

Контроллер шин выполнен с перестраиваемой конфигурацией для задания сигналом *IOB* двух режимов работы: режим шины ввода-вывода (*I/O Bus Mode* при *IOB* = 1) и режим системной шины (*System Bus Mode* при *IOB* = 0). Сигналы контроллера шин имеют назначение:

$\overline{S}_{2\text{–}0}$ (*Status*) — сигналы состояния, поступающие от МП или *NDCP* и идентифицирующие тип цикла шины (табл. 4.37). Контроллер шин декодирует эти сигналы для генерации активных уровней определенных сигналов управления и сигналов команд;

CLK (*Clock*) — сигнал, подаваемый от генератора 8284/8284*A*. С помощью этого сигнала задается временно́е положение в цикле шины активных уровней сигналов управления и сигналов команд (рис. 4.50);

ALE (*Address Latch Enable*) — сигнал управления, используемый для фиксации адреса в асинхронных потенциальных регистрах памяти (см. рис. 4.13). Такие регистры называются иначе прозрачными фиксаторами типа *D* (*transparent D type latches*) — любые изменения адреса поступают на выходы регистров при значении сигнала *ALE* = 1, а фиксация адреса происходит при изменении сигнала *ALE* с 1 на 0;

DEN (*Data Enable*) — сигнал включения приемопередатчиков для передачи данных в ту или другую сторону (от МП или в МП);

DT/$\overline{R}$ (*Data Transmit/Receive*) — сигнал управления направлением передачи данных приемопередатчиками (см. рис. 4.13). Если сигнал *DT*/$\overline{R}$ = 1, то передача данных производится от

МП в память или внешние устройства, если же $DT/\overline{R}$ = 0, то передача данных происходит от памяти или внешних устройств в МП;

IOB (*Input/Output Bus Mode*) — входной сигнал управления, задающий режим работы контроллера шин: *IOB* = 1 — режим шины ввода-вывода (*I/O Bus Mode*), *IOB* = 0 — режим системной шины (*System Bus Mode*);

$\overline{AEN}$ (*Address Enable*) — входной сигнал разрешения доступа к шинам в мультипроцессорных системах (на этот вход подается выходной сигнал $\overline{AEN}$ арбитра шин 8289, изображенного на рис. 4.49, *б*). Значение сигнала доступа к шинам $\overline{AEN}$ = 1 переводит выходы всех сигналов команд в *Z*-состояние при значении сигнала управления *IOB* ≡ 0 (режим системной шины). Если же установлен режим шины ввода-вывода (*IOB* ≡ 1), то значение сигнала $\overline{AEN}$ = 1 переводит в *Z*-состояние только выходы сигналов команд чтения и записи данных в память. Разрешение выходов сигналов команд (снятие *Z*-состояния) происходит не позднее, чем через 115 нс после того, как сигнал $\overline{AEN}$ изменится с 1 на 0;

CEN (*Command Enable*) — сигнал управления, переводящий значением *CEN* = 0 все сигналы команд, а также сигналы управления *DEN* и $\overline{PDEN}$ в неактивное состояние (сигнал *DEN* — в состояние 0, а остальные сигналы — в состояние 1). Сигнал *CEN* используется в мультипроцессорных системах для разделения управления резидентной и системной памятью (см. рис. 4.62);

$\overline{MRDC}$ (*Memory Read Command*) — сигнал чтения данных из памяти (рис. 4.50);

$\overline{MWTC}$ (*Memory Write Command*) — сигнал записи данных в память;

$\overline{AMWC}$ (*Advanced Memory Write Command*) — сигнал записи данных в память с ранним ее информированием о выполнении операции записи в текущем цикле шины (рис. 4.50), например, для генерации памятью сигнала готовности. Временны́е характеристики этого сигнала такие же, что и сигнала чтения $\overline{MRDC}$;

$\overline{IORC}$ (*I/O Read Command*) — сигнал ввода (чтения) данных из внешнего устройства;

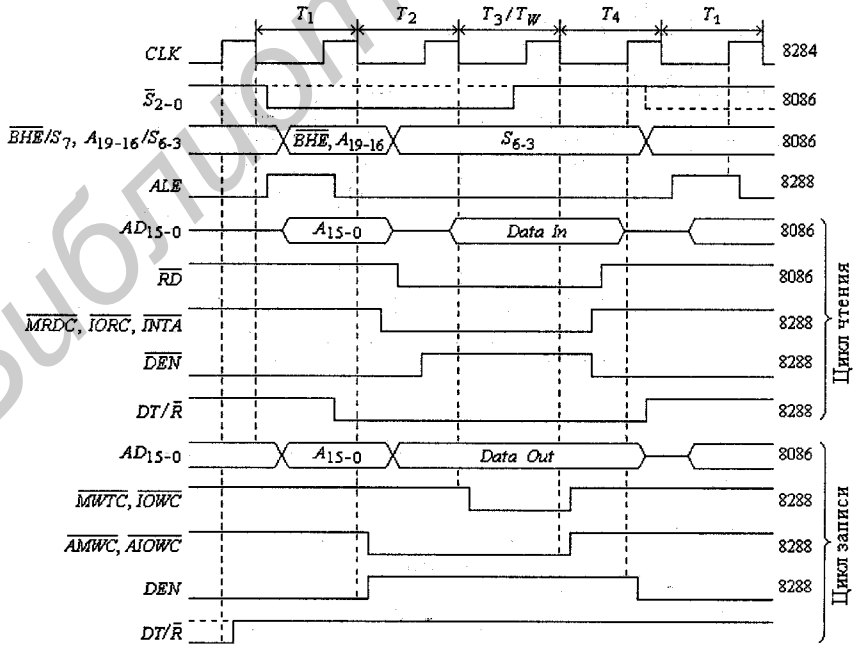


Рис. 4.50. Временны́е диаграммы сигналов контроллера шин 8288

$\overline{IOWC}$ (*I/O Write Command*) — сигнал вывода (записи) данных во внешнее устройство;

$\overline{AIOWC}$ (*Advanced I/O Write Command*) — сигнал вывода (записи) данных во внешнее устройство с ранним его информированием о выполнении операции вывода в текущем цикле шины (рис. 4.50), например, для генерации внешним устройством сигнала готовности. Временны́е характеристики этого сигнала такие же, что и сигнала ввода $\overline{IORC}$;

$\overline{INTA}$ (*Interrupt Acknowledge*) — сигнал подтверждения прерывания, выдаваемый в ответ на запрос прерывания внешнего устройства (рис. 4.50). Для аппаратной поддержки ввода-вывода по прерыванию обычно используется контроллер прерываний 8259*A*, поэтому сигнал $\overline{INTA}$ производит чтение байта типа прерывания *type* из этого контроллера;

MCE/$\overline{PDEN}$ (*Master Cascade Enable / Peripheral Data Enable*) — назначение этого сигнала программируется сигналом *IOB*: в режиме системной шины (*IOB* ≡ 0) генерируется сигнал *MCE*, а в режиме шины ввода-вывода (*IOB* ≡ 1) — сигнал $\overline{PDEN}$. В режиме системной шины значение сигнала *MCE* = 1 генерируется только в циклах подтверждения прерывания для передачи через шину данных адреса каскадирования из ведущего контроллера прерываний в регистр адреса с фиксацией его сигналом *ALE* (рис. 4.51). В режиме шины ввода-вывода значение сигнала $\overline{PDEN}$ = 0 генерируется при выполнении каждой операции ввода и вывода. Этот сигнал используется для включения приемопередатчика шины ввода-вывода (см. рис. 4.63).

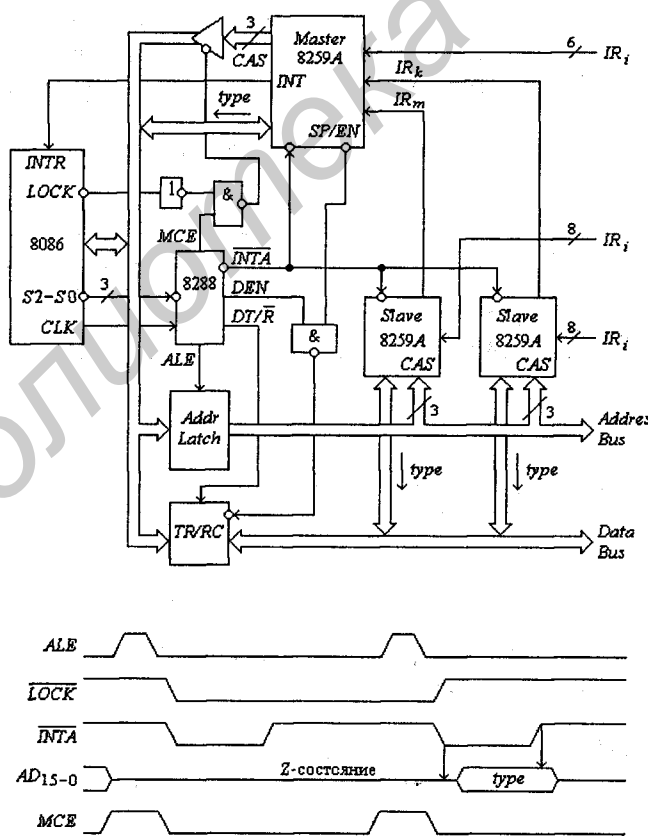


Рис. 4.51. Каскадирование контроллеров прерывания 8259*A*

Режим шины ввода-вывода. Задается этот режим в мультипроцессорных системах значением сигнала *IOB* = 1 (см. рис. 4.63). В режиме шины ввода-вывода сигналы команд ввода-вывода $\overline{IORC}$, $\overline{IOWC}$, $\overline{AIOWC}$ и $\overline{INTA}$ не зависят от значения сигнала $\overline{AEN}$, поступающего от арбитра шин 8289. Когда МП выдает код состояния $\overline{S}_2\overline{S}_1\overline{S}_0$ = 000, 001 или 010 (см. табл. 4.37), контроллер шин 8288 генерирует активные уровни какого-либо сигнала команды ввода ($\overline{IORC}$ или $\overline{INTA}$) или вывода ($\overline{IOWC}$ и $\overline{AIOWC}$) и сигналов $\overline{PDEN}$ и *DT*/$\overline{R}$ для управления приемопередатчиком шины ввода-вывода. Доступ к системной шине (доступ к памяти) МП получает только по значению сигнала $\overline{AEN}$ = 0. Таким образом, режим шины ввода-вывода позволяет одному контроллеру шин 8288 обслуживать две внешние шины — вторая шина связана с памятью и ее приемопередатчик включается сигналом *DEN*. Режим шины ввода-вывода целесообразно использовать в тех случаях, когда какой-либо МП, входящий в состав мультипроцессорной системы, должен обслуживать только свои (не общие) внешние устройства.

Режим системной шины. Задается этот режим значением сигнала *IOB* = 0. Разрешение выходов сигналов команд происходит не позднее, чем через 115 нс после того, как сигнал $\overline{AEN}$ изменится с 1 на 0. В режиме системной шины логика арбитража шины (ИС 8289) значением сигнала $\overline{AEN}$ = 0 должна сообщить контроллеру шины, когда шина свободна для использования. Этот режим применяется в мультипроцессорных системах, когда и память и внешние устройства разделяются более чем одним МП (имеется только системная шина — см. рис. 4.66).

Принципиальная схема микроконтроллера. На рис. 4.52 изображена принципиальная схема микроконтроллера специализированного назначения, построенного на основе МП 8086 и *NDCP* 8087 и обладающего мощными вычислительными возможностями. Микроконтроллер содержит *EPROM* размером 64 Кбайта программной памяти и памяти констант (две БИС 27256 по 32К × 8 бит) и *SRAM* размером 12 Кбайта оперативной памяти данных (шесть БИС 537РУ10 по 2К × 8 бит): *EPROM* расположено по адресам 00000*h* ÷ 0FFFF*h*, а *SRAM* — по адресам 10000*h* ÷ 12FFF*h* (табл. 4.38), причем для *SRAM* по адресам 12000*h* ÷ 12FFF*h* (4 Кбайта) используется резервное питание от аккумулятора для сохранения оперативных данных при выключении основного питания.

Поскольку в микроконтроллере используется не все адресное пространство памяти (1 Мбайт), то старшие разряды адреса A_{19} и A_{18} можно не подавать на адресные дешифраторы памяти (A_{19} = ×, A_{18} = ×). *EPROM* селектируется значением сигнала $\overline{CS} = \overline{A}_{17}A_{16} = 0$, т. е. обращение к *EPROM* производится как при A_{17} = 1, так и при A_{16} = 0 (табл. 4.38), а значит в диапазоне адресов FFFF0*h* ÷ FFFFF*h* обращение будет производиться к *EPROM* (FFFF0*h* — стартовый адрес МП 8086, устанавливаемый значением сигнала системного сброса *RESET* = 1 при включении питания). Старшие 16 байт *EPROM* должны содержать команду безусловного перехода JMP *addr* в область адресов 00400*h* ≤ *addr* < 0FFF0*h* (адреса *EPROM* 00000*h* ÷ 003FF*h* предназначены для векторов прерываний команд INT *type*).

Сигналы управления записью данных в *SRAM* определяются выражениями:

$$\overline{WEL} = \overline{AMWC} \vee AB_0 = \overline{AMWC \cdot \overline{AB_0}}, \quad \overline{WEH} = \overline{AMWC} \vee \overline{BHEB} = \overline{AMWC \vee BHEB},$$

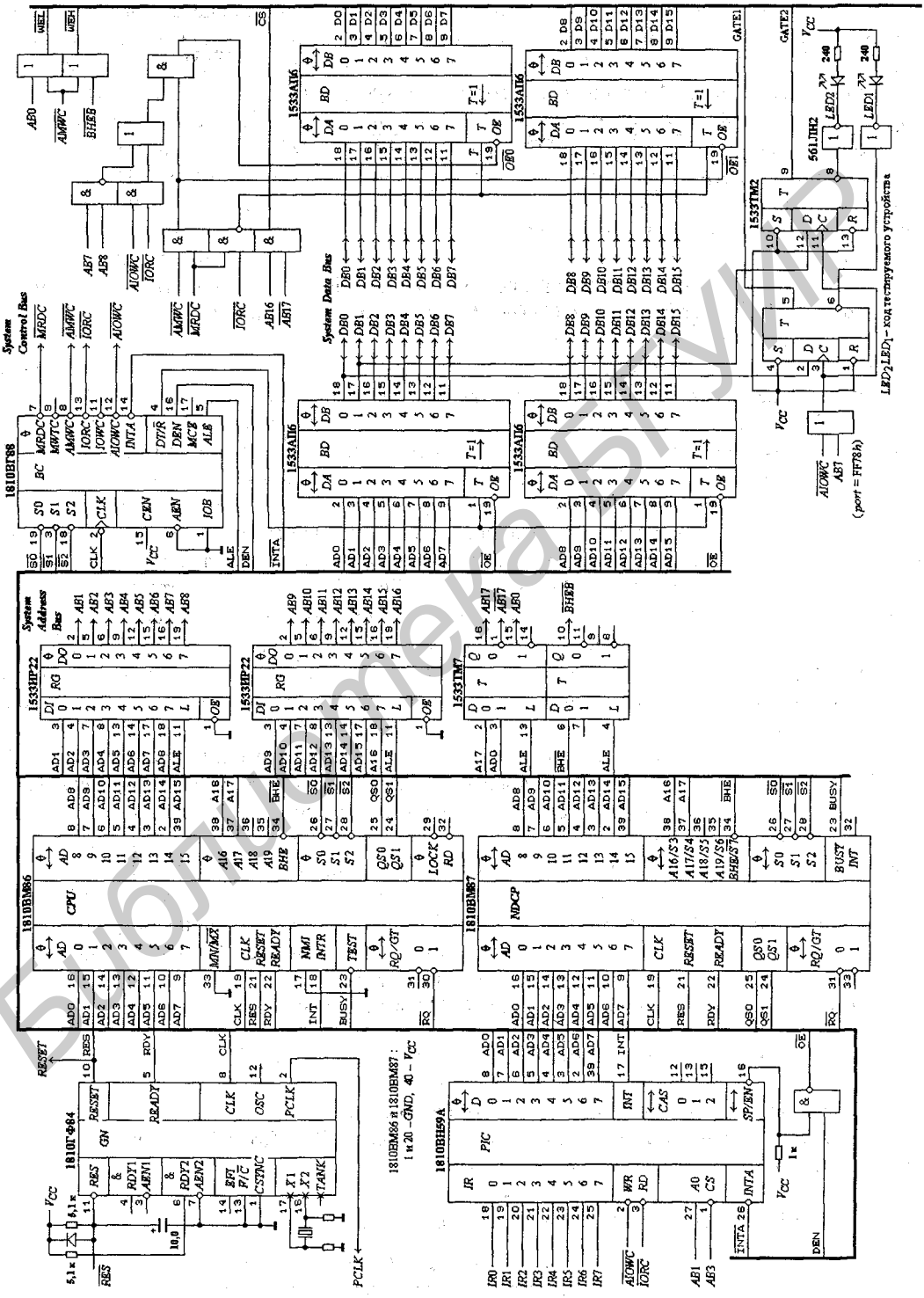
что обеспечивает запись слов и любого одного байта слова.

Сигналы управления *T* и $\overline{OE}$ приемопередатчиков 1533АП6, обслуживающих память и внешние устройства, описываются соотношениями:

$$T = \overline{\overline{MRDC} \cdot \overline{IORC}} = MRDC \vee IORC, \quad \overline{OE}_1 = \overline{\overline{MRDC} \cdot \overline{AMWC}} = \overline{MRDC \vee AMWC},$$

$$\overline{OE}_0 = (\overline{\overline{IORC} \cdot \overline{AIOWC}} \vee \overline{AB_8 \cdot AB_7}) \cdot \overline{AMWC} \cdot \overline{MRDC} =$$

$$= \overline{(IORC \vee AIOWC) \cdot AB_8 \cdot AB_7 \vee AMWC \vee MRDC}.$$



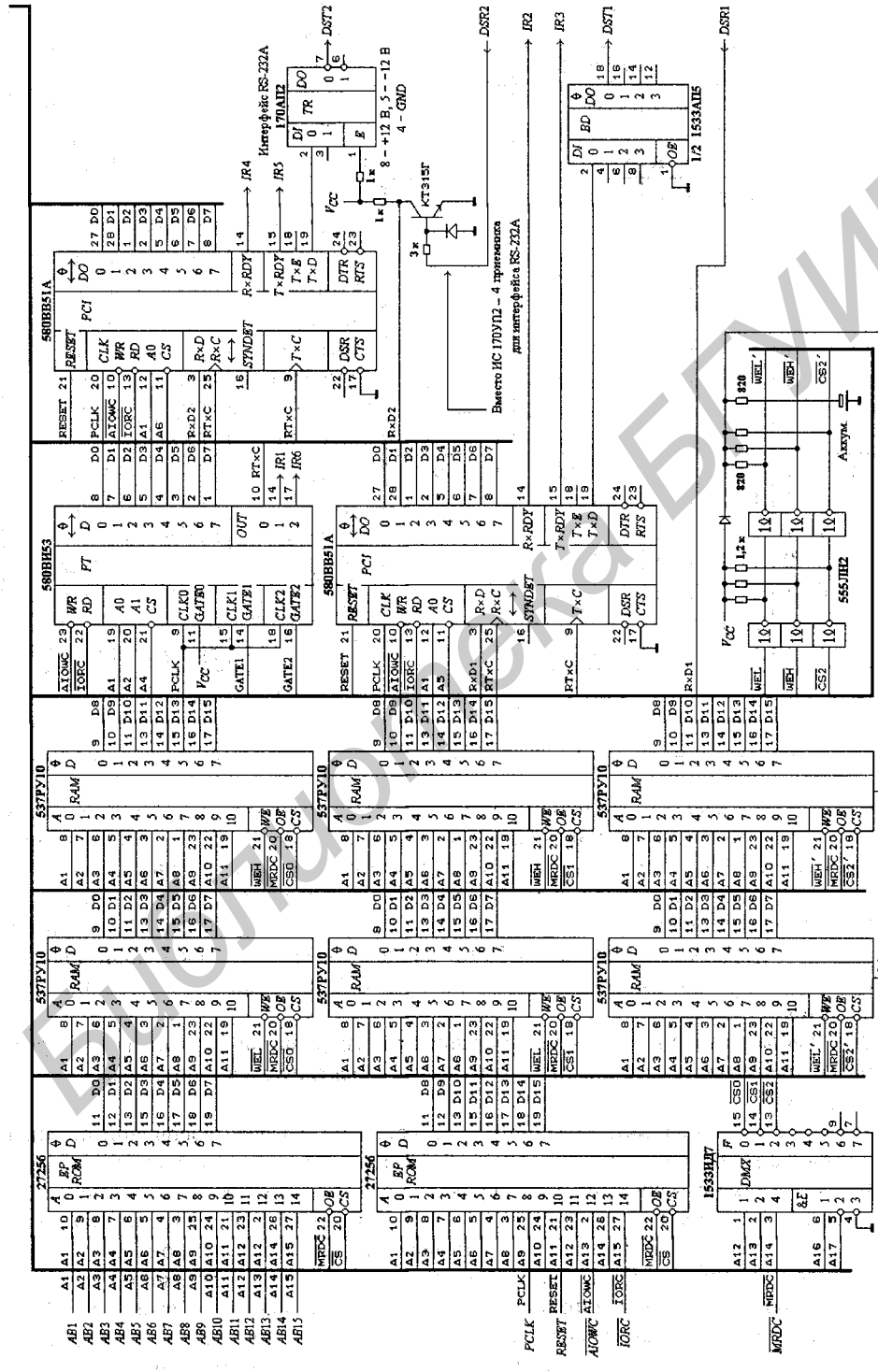


Рис. 4.52. Принципиальная схема микроконтроллера

Таблица 4.38. **Адресация памяти и внешних устройств**

Разряды адреса																		*Memory and I/O*	*Address and Port*
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
–	–	0	0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	*EPROM*	00000*h* ÷ 0FFFF*h* [1]
–	–	1	0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	27256	20000*h* ÷ 2FFFF*h* [1]
–	–	1	1	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	64K × 8 бит	F0000*h* ÷ FFFFF*h* [2]
–	–	0	1	–	0	0	0	×	×	×	×	×	×	×	×	×	×	×	×	*RAM*	10000*h* ÷ 10FFF*h* [3]
–	–	0	1	–	0	0	1	×	×	×	×	×	×	×	×	×	×	×	×	537РУ10	11000*h* ÷ 11FFF*h* [3]
–	–	0	1	–	0	1	0	×	×	×	×	×	×	×	×	×	×	×	×	12 К × 8 бит	12000*h* ÷ 12FFF*h* [3]
–	–	–	–	1	1	1	1	1	1	1	1	1	1	1	1	0	–	×	–	1810ВН59А	FFF0*h*, FFF2*h*
–	–	–	–	1	1	1	1	1	1	1	1	1	1	1	0	1	×	×	–	580ВИ53	FFE8*h* ÷ FFEE*h* [4]
–	–	–	–	1	1	1	1	1	1	1	1	1	1	0	1	1	–	×	–	580ВВ51*a*	FFD8*h*, FFDA*h*
–	–	–	–	1	1	1	1	1	1	1	1	1	0	1	1	1	–	×	–	580ВВ51*b*	FFB8*h*, FFBA*h*
–	–	–	–	1	1	1	1	1	1	1	1	0	1	1	1	1	–	–	–	571ХЛ5*a*	FF78*h* (рис. 4.53)
–	–	–	–	1	1	1	1	1	1	1	0	1	1	1	1	1	–	–	–	571ХЛ5*b*	FEF8*h* (рис. 4.53)
–	–	–	–	1	1	1	1	1	1	0	1	1	1	1	1	1	×	×	–	Зарезервировано для расширения пространства ввода-вывода	FDF8*h* ÷ FDFE*h* [4]
–	–	–	–	1	1	1	1	1	0	1	1	1	1	1	1	1	×	×	–		FBF8*h* ÷ FBFE*h* [4]
–	–	–	–	1	1	1	1	0	1	1	1	1	1	1	1	1	×	×	–		F7F8*h* ÷ F7FE*h* [4]
–	–	–	–	1	1	1	0	1	1	1	1	1	1	1	1	1	×	×	–		EFF8*h* ÷ EFFE*h* [4]
–	–	–	–	1	1	0	1	1	1	1	1	1	1	1	1	1	×	×	–		DFF8*h* ÷ DFFE*h* [4]
–	–	–	–	1	0	1	1	1	1	1	1	1	1	1	1	1	×	×	–		BFF8*h* ÷ BFFE*h* [4]
–	–	–	–	0	1	1	1	1	1	1	1	1	1	1	1	1	×	×	–		7FF8*h* ÷ 7FFE*h* [4]

Примечание: знак "–" — разряд адреса не используется при дешифрации адреса, знак "×" — разряд адреса имеет значения 0 и 1; [1] при $A_{19} = A_{18} = 0$, [2] при $A_{19} = A_{18} = 1$, [3] при $A_{19} = A_{18} = A_{15} = 0$, [4] только четные адреса портов.

Из второго выражения следует, что приемопередатчик старшего байта данных включается только при чтении и записи данных в память (значение сигнала $\overline{OE}_1 = 0$). Из последнего выражения следует, что значение сигнала $\overline{OE}_0 = 0$ при каждом чтении и записи данных в память, а также при вводе и выводе данных, но при условии, что значения адресных сигналов $AB_8 = 1$ и $AB_7 = 1$, т. е. если нет обращения к внешнему устройству, изображенному на рис. 4.53. Сигнал $\overline{OE}_0$ управляет приемопередатчиком младшего байта данных — ввод и вывод производится только байтами по четным адресам портов (разряд адреса AB_0 для адресации внешних устройств не используется).

Адреса портов ввода-вывода ввиду их малочисленности выбраны так, что внешний адресный дешифратор не требуется — все внешние устройства селектируются одиночными адресными разрядами A_k ($k = 3 \ldots 15$).

Тестирование и инициализация микроконтроллера. При каждом включении питания необходимо убедиться в работоспособности микроконтроллера. Для этого первым делом микроконтроллер должен выполнить программу тестирования основных своих узлов. Только если тестирование пройдет успешно, выполняется программа инициализации микроконтроллера. В обязательном порядке должно тестироваться оперативное запоминающее устройство (*SRAM*), так как при его отказе нормальная работа невозможна вследствие непредсказуемости результатов. Программы тестирования и инициализации микроконтроллера располагаются в *EPROM*, в работе которого, конечно, не должно быть отказов. Тестирование других узлов (например, интерфейсных БИС) производится по усмотрению разработчика микроконтроллера.

Задача 1. Написать программу элементарного тестирования статической оперативной памяти (*SRAM*), таймера 580ВИ53, контроллера прерываний 1810ВН59*A* и *NDCP* 1810ВМ87 для схемы, изображенной на рис. 4.52. Для индикации тестируемого устройства использовать два светоизлучающих диода (СИД), управляемых сигналами

$LED_2LED_1 = 00b$ (↑↑ ↑↑) — тестирование *SRAM*,
$LED_2LED_1 = 10b$ (× ↑↑) — тестирование таймера 580ВИ53,
$LED_2LED_1 = 01b$ (↑↑ ×) — тестирование контроллера прерываний 1810ВН59*A*,
$LED_2LED_1 = 11b$ (× ×) — тестирование *NDCP* 1810ВМ87,
$LED_2LED_1 = 00b$ (↑↑ ↑↑) — тестирование успешно закончено,

где LED_2 и LED_1 — сигналы управления СИД (см. рис. 4.52), × — СИД погашен, ↑↑ — СИД излучает свет. Выполнить инициализацию контроллера прерываний с заданием базового типа прерывания *type* = D0*h*. *Решение* (по существу, проверяется целостность связей между МП и внешней средой):

```
d_seg   segment          ; Сегмент данных (12K × 8 бит)
S_Ram   equ    3000h     ; S_Ram = 3000h — размер SRAM (Size Ram)
data    dw     S_Ram/2 dup (?)  ; Оперативная память и стек
d_seg   ends
c_seg   segment          ; Сегмент кода
        assume CS: c_seg, DS: d_seg
pLED    equ    0FF78h    ; pLED — адрес триггеров управления индикатором (два СИД)
p53_0   equ    0FFE8h    ; p53_0 — адрес канала 0 таймера 8253
p53_3   equ    0FFEEh    ; p53_3 — адрес регистра CW таймера 8253
p59_0   equ    0FFF0h    ; p59_0 — адрес порта 0 контроллера прерываний 8259A
p59_1   equ    0FFF2h    ; p59_1 — адрес порта 1 контроллера прерываний 8259A
cnt1    equ    1Dh       ; cnt1 и cnt2 — задание времени индикации
cnt2    equ    4FFFh     ;     (примерно 2 секунды при f_CLK = 5 МГц)
        org    200h      ; 0000h ÷ 01FFh — область EPROM для векторов прерывания (type)
delay   proc   near      ; Процедура задания времени индикации тестов
        MOV    AL, cnt1  ; cnt1 и cnt2 — задание времени индикации
PD2:    MOV    CX, cnt2  ;     (процедуру delay использовать нельзя пока не протестирована
PD1:    LOOP   PD1       ;     оперативная память и не назначен стек)
        DEC    AL
        JNZ    PD2
        RET
delay   endp
; 1. Тестирование SRAM 12K × 8 бит (шесть БИС 537РУ10)
RAM1:   MOV    DX, pLED  ; DX ← pLED = FF78h — адрес порта СИД
        MOV    AL, 0     ; Индикация тестирования SRAM 537РУ10
        OUT    DX, AL    ; I/O(pLED) ← 00h, LED2LED1 ← 00b (↑↑ ↑↑ — СИД)
        MOV    AL, cnt1  ; cnt1 и cnt2 — задание времени наблюдения состояния СИД
RAM3:   MOV    CX, cnt2
RAM2:   LOOP   RAM2
        DEC    AL
        JNZ    RAM3
        MOV    CX, S_Ram/2  ; CX ← S_Ram/2 = 1800h.        Начало тестирования SRAM
```

```
        MOV   BX, 0
RAM6:   MOV   AX, 5555h ; AX ← 0101 0101 0101 0101 = 5555h
RAM4:   MOV   [BX], AX
        CMP   AX, [BX]
        JNZ   RAM4      ; Переход, если возникла ошибка
        NOT   AX        ; AX ← 1010 1010 1010 1010 = AAAAh
RAM5:   MOV   [BX], AX
        CMP   AX, [BX]
        JNZ   RAM5      ; Переход, если возникла ошибка
        INC   BX
        INC   BX
        DEC   CX
        JNZ   RAM6      ;                                  Конец тестирования
        MOV   SP, S_Ram ; SP ← S_Ram = 3000h (теперь можно использовать процедуру delay)
; 2. Тестирование таймера 580ВИ53
        MOV   AL, 2
        OUT   DX, AL    ; I/O(pLED) ← 02h, LED2LED1 ← 10b (× ↑↑ — СИД)
        CALL  delay
        MOV   DX, p53_3 ; DX ← p53_3 = FFEEh                Начало тестирования таймера
        MOV   AL, 36h   ; AL ← CW0 = 36h — слово управления канала 0 (см. рис. 3.28)
        OUT   DX, AL    ; I/O(p53_3) ← CW0 = 0011 0110 (два байта, режим 3, двоичный счет)
        MOV   DX, p53_0 ; DX ← p53_0 = FFE8h
PIT1:   MOV   AL, 55h   ; AL ← 0101 0101 = 55h
        OUT   DX, AL    ; I/O(p53_0) ← 55h — запись младшего байта модуля пересчета
        OUT   DX, AL    ; I/O(p53_0) ← 55h — запись старшего байта модуля пересчета
        IN    AL, DX    ; AL ← I/O(p53_0) — чтение младшего байта состояния счетчика 0
        IN    AL, DX    ; AL ← I/O(p53_0) — чтение старшего байта состояния счетчика 0
        CMP   AL, 55h   ; AL – 55h
        JNZ   PIT1      ; Переход, если возникла ошибка
PIT2:   MOV   AL, 0AAh  ; AL ← 1010 1010 = AAh
        OUT   DX, AL    ; I/O(p53_0) ← AAh — запись младшего байта модуля пересчета
        OUT   DX, AL    ; I/O(p53_0) ← AAh — запись старшего байта модуля пересчета
        IN    AL, DX    ; AL ← I/O(p53_0) — чтение младшего байта состояния счетчика 0
        IN    AL, DX    ; AL ← I/O(p53_0) — чтение старшего байта состояния счетчика 0
        CMP   AL, 0AAh  ; AL – AAh
        JNZ   PIT2      ; Переход, если возникла ошибка                Конец тестирования
; Инициализация канала 0 таймера 580ВИ53
        MOV   AL, 9     ; AL ← 09h (Modulo = 5 МГц / 9600 бод = 520,833 ≈ 521d = 209h)
        OUT   DX, AL    ; I/O(p53_0) ← 09h — запись младшего байта модуля пересчета
        MOV   AL, 2     ; AL ← 02h
        OUT   DX, AL    ; I/O(p53_0) ← 02h — запись старшего байта модуля пересчета
; 3. Тестирование контроллера прерываний 1810ВН59А
        MOV   DX, pLED  ; DX ← pLED = FF78h — адрес порта СИД
        MOV   AL, 1
        OUT   DX, AL    ; I/O(pLED) ← 01h, LED2LED1 ← 01b (↑↑ × — СИД)
        CALL  delay
```

```
; Инициализация контроллера прерываний 1810ВН59А
        MOV   DX, p59_0 ; DX ← p59_0 = FFF0h — адрес порта БИС 1810ВН59А (A1 = 0)
        MOV   AL, 13h   ; AL ← ICW1 = 13h = 0001 0011 — один контроллер, ICW4 есть,
        OUT   DX, AL    ; I/O(p59_0) ← ICW1,     запрос прерываний по фронту (см. рис. 3.49)
        MOV   DX, p59_1 ; DX ← p59_1 = FFF2h — адрес порта БИС 1810ВН59А (A1 = 1)
        MOV   AL, 0D0h  ; AL ← ICW2 = D0h ⇒ type = 1101 0×××  = D0h … D7h (см. рис. 3.50)
        OUT   DX, AL    ; I/O(p59_1) ← ICW2
        MOV   AL, 0Dh   ; AL ← ICW4 = 0Dh = 0000 1101 (см. рис. 3.53)
        OUT   DX, AL    ; I/O(p59_1) ← ICW4
PIC1:   MOV   AL, 7Fh   ; AL ← OCW1 = 3Fh                  Начало тестирования контроллера
        OUT   DX, AL    ; I/O(p59_1) ← OCW1 = 3Fh ⇒ IMR = 0111 1111 (см. рис. 3.57)
        IN    AL, DX    ; AL ← IMR — Interrupt Mask Request (чтение маски)
        CMP   AL, 7Fh   ; AL – 7Fh — сравнение масок
        JNZ   PIC1      ; Переход, если возникла ошибка
PIC2:   MOV   AL, 80h   ; AL ← OCW1 = 80h
        OUT   DX, AL    ; I/O(p59_1) ← OCW1 = 80h ⇒ IMR = 1000 0000
        IN    AL, DX    ; AL ← IMR — чтение маски
        CMP   AL, 80h   ; AL – 80h — сравнение масок
        JNZ   PIC2      ; Переход, если возникла ошибка          Конец тестирования
; 4. Тестирование NDCP 1810ВМ87
        MOV   DX, pLED  ; DX ← pLED = FF78h
        MOV   AL, 3     ; Индикация тестирования NDCP 8087
        OUT   DX, AL    ; I/O(pLED) ← 03h, LED2LED1 ← 11b (× × — СИД)
        CALL  delay
NDCP:   MOV   data, 0           ; M(data+2, data) ← 40B0 0000h — код Short Real числа 5.5
        MOV   data + 2, 40B0h
        FLD   dword ptr data    ; Top ← Top – 1, ST(0) ← M(data) = 40B0 0000h
        FSQRT           ; ST ← 2.3452078799117148 — квадратный корень из числа 5.5
        FRNDINT         ; ST ← 0002h — целая часть числа ST
        FIST  data      ; M(data) ← ST = 0002h
        CMP   data, 2   ; M(data) – 0002h
        JNZ   NDCP      ; Переход, если возникла ошибка          Конец тестирования
        MOV   DX, pLED  ; DX ← pLED = FF78h
        MOV   AL, 0     ; Тестирование прошло успешно
        OUT   DX, AL    ; I/O(pLED) ← 00h, LED2LED1 ← 00b (↑↑ ↑↑ — СИД)
        CALL  delay
        STI             ; Флаг IF ← 1 — прерывания разрешены
        mov   ah, 4Ch   ; Выход из программы
        int   21h       ; INT 21h, функция DOS 4Ch
        org   0FFF0h
Reset:  MOV   AX, d_seg ; Точка входа в программу по значению сигнала RESET = 1
        MOV   DS, AX    ; Инициализация сегментного регистра данных DS
        MOV   SS, AX    ; Инициализация сегментного регистра стека SS
        JMP   RAM1
c_seg   ends
        end   Reset
```

Тестирование статической оперативной памяти (*SRAM*) микроконтроллера основано на записи во все ячейки памяти слов данных с последующим их чтением и сравнением полученного при чтении слова с исходным. Для обнаружения обрывов и замыканий линий связи проверку следует выполнять при записи как прямых, так и инверсных слов данных (в программе *задачи* 1 использованы слова данных 5555*h* и AAAA*h*).

Для тестирования интерфейсных БИС необходимо определить регистры, доступные для записи и последующего чтения из них байт данных. Если таймер 580ВИ53 запрограммировать в режим работы *Mode* 3 (делитель частоты) с заданием модуля пересчета двумя байтами, то старший байт модуля пересчета можно будет прочитать из регистра хранения текущего состояния счетчика OL_M (см. рис. 3.27), так как за время тестирования таймера может измениться содержимое только регистра OL_L. Конечно, при этом следует задать достаточно большое значение младшего байта модуля пересчета (для тестирования таймера использованы прямой и инверсный байты данных 55*h* и AA*h*).

Для тестирования работоспособности контроллера прерываний 1810ВН59*А* в составе микроконтроллера в программе *задачи* 1 выполняется запись и чтение маски прерываний в регистр *IMR* (см. рис. 3.42). После успешного завершения теста вход запроса прерываний IR_7 остается замаскированным.

Принцип тестирования связных интерфейсов 580ВВ51А был подробно рассмотрен в § 3.7 (см. рис. 3.93 и соответствующую ему программу). Этот же принцип тестирования может быть использован и в рассматриваемом микроконтроллере — предварительно следует поставить две перемычки для замыкания выхода *DST*1 с входом *DSR*1 и выхода *DST*2 с входом *DSR*2 (см. рис. 4.52). После тестирования перемычки необходимо снимать, что предоставляет определенные неудобства. Поэтому тестирование связных интерфейсов следует производить только в необходимых случаях, а не при каждом включении питания микроконтроллера.

Тестирование арифметического сопроцессора 1810ВМ87 (*NDCP*) в программе *задачи* 1 основано на вычислении квадратного корня из вещественного числа 5.5, выделении его целой части и сравнении полученного результата с заранее известным ответом. Для вещественного числа 5.5 использовано представление в формате *Short Real* (см. рис. 4.29). Поэтому необходимо вручную выполнить перевод числа 5.5 в формат *Short Real*:

$$5.5 = 101.1 = 1.\underline{011}e{+}2 \Rightarrow \text{смещенный порядок } E = exp + bias = 2 + 127 = 129d$$

скрытый разряд мантиссы f_0 ↑ ↑ *Fraction* (мантисса) ⇒

	S	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}
⇒	0	1	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
+		$E = 129d$								*Fraction*																						
	4				0				B				0				0				0				0				0			

(итак, 40B0 0000*h* — код *Short Real* числа 5.5). Целая часть квадратного корня из числа 5.5 равна 2 = 0000 0010 — использована для тестирования *NDCP*.

Контроллер клавиатуры и дисплея, изображенный на рис. 3.144 и 3.145, может быть подключен не только к МП-системе, выполненной на МП 8080 или 8085, но и к микроконтроллеру, показанному на рис. 4.52. Для этого сигналы МП 8080/8085 следует заменить соответствующими сигналами микроконтроллера (например, сигнал *CLK* на сигнал *PCLK*).

Обмен данными между микроконтроллером и персональным компьютером *PC*. На рис. 4.53 изображен приемопередатчик, позволяющий организовать связь микроконтроллера с персональным компьютером с помощью метода ввода-вывода с квитированием по прерыванию (см. § 2.4). Далее, для краткости, микроконтроллер будет называться микроЭВМ (или μЭВМ).

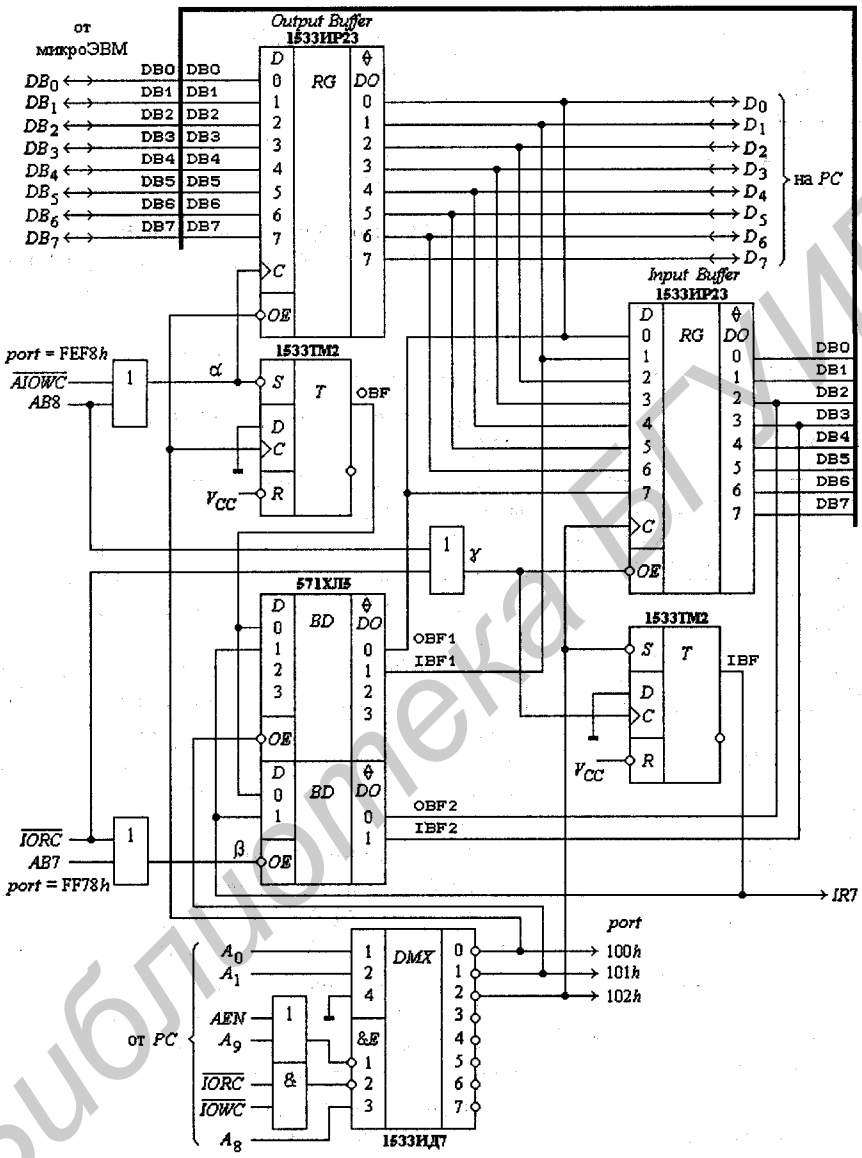


Рис. 4.53. Принципиальная схема приемопередатчика

Приемопередатчик содержит два буферных регистра памяти: выходной буфер 1533ИР23 (*Output Buffer*) для передачи данных из микроЭВМ в *IBM PC* (вывод) и входной буфер 1533ИР23 (*Input Buffer*) для передачи данных из *IBM PC* в микроЭВМ (ввод). Квитирование ввода-вывода осуществляется с помощью флагов, формируемых триггерами 1533ТМ2:

$$IBF_1 = \begin{cases} 0 - \text{указание } PC \text{ выдать байт данных,} \\ 1 - \text{входной буфер заполнен;} \end{cases} \qquad OBF_1 = \begin{cases} 0 - \text{выходной буфер пустой,} \\ 1 - \text{указание } PC \text{ принять байт данных;} \end{cases}$$

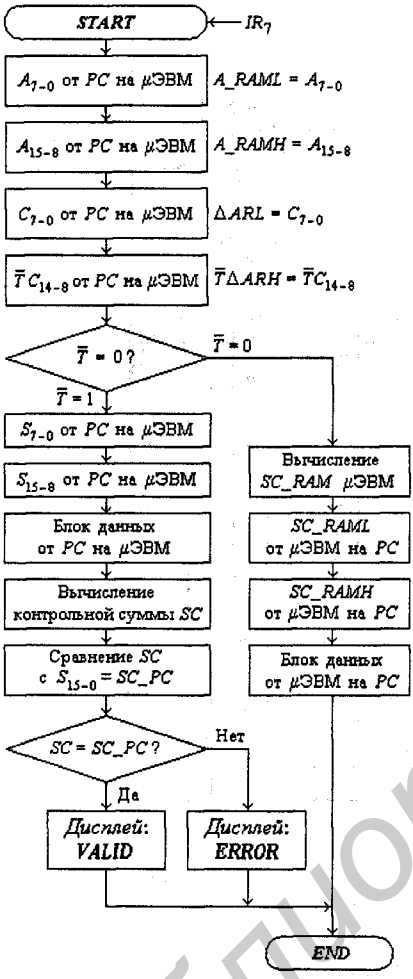


Рис. 4.54. Блок-схема алгоритма пересылки блока данных

$$IBF_2 = \begin{cases} 0 - \text{входной буфер пустой,} \\ 1 - \text{указание микроЭВМ принять данных;} \end{cases}$$

$$OBF_2 = \begin{cases} 0 - \text{указание микроЭВМ выдать байт данных,} \\ 1 - \text{выходной буфер заполнен.} \end{cases}$$

Блок-схема алгоритма обмена данными между микроЭВМ и *IBM PC* представлена на рис. 4.54:

$A\_RAML = A_{7-0}$ — младший байт начального адреса *SRAM* микроЭВМ;

$A\_RAMH = A_{15-8}$ — старший байт начального адреса *SRAM* микроЭВМ;

$A_{15-0} = A\_beg$ — начальный адрес *SRAM* микроЭВМ;

$\Delta ARL = C_{7-0}$ — младший байт размера блока пересылаемых данных;

$\Delta ARH = C_{14-8}$ — семь младших разрядов старшего байта размера блока пересылаемых данных;

$C_{14-0} = \Delta AR$ — размер блока пересылаемых данных (в байтах);

$\overline{T}$ — старший разряд старшего байта размера блока пересылаемых данных, задающий направление передачи данных ($\overline{T} = 0$ — передача от микроЭВМ на *PC*);

S_{7-0} — младший байт контрольной суммы *SC_PC* (*SC* — *Sum Control*) пересылаемого из *PC* в микроЭВМ блока данных;

S_{15-8} — старший байт контрольной суммы *SC_PC* пересылаемого из *PC* в микроЭВМ блока данных;

$S_{15-8} = SC\_PC$ — контрольная сумма пересылаемого из *PC* в микроЭВМ блока данных, которую вычисляет *PC* перед передачей блока данных;

SC — контрольная сумма принятого микроЭВМ от *PC* блока данных, которую вычисляет микроЭВМ после получения блока данных;

SC_RAM — контрольная сумма пересылаемого из микроЭВМ в *PC* блока данных, которую вычисляет микроЭВМ перед передачей блока данных;

SC_RAML — младший байт контрольной суммы пересылаемого из микроЭВМ в *PC* блока данных;

SC_RAMH — старший байт контрольной суммы пересылаемого из микроЭВМ в *PC* блока данных.

Инициатором обмена данными всегда является оператор, запускающий специальную программу *PC* (здесь не рассматривается), введя с помощью клавиатуры параметры обмена *A_beg*, ΔAR и $\overline{T}$. Пересылать в *SRAM* микроЭВМ можно и небольшие программы для проверки их выполнения с учетом аппаратной реализации микроЭВМ.

Задача 2. Написать для микроЭВМ программу обмена данными между микроЭВМ и персональным компьютером по блок-схеме алгоритма, приведенной на рис. 4.54. *Решение* (без учета вызова программы по запросу прерывания сигналом IR_7):

```
d_seg   segment              ; Сегмент данных (12K × 8 бит)
S_Ram   equ     3000h        ; S_Ram = 3000h — размер SRAM (Size Ram)
data    dw      S_Ram/2 dup (0EEDDh) ; Оперативная память и стек
d_seg   ends
c_seg   segment              ; Сегмент кода
        assume  CS: c_seg, DS: d_seg
pTRF    equ     0FF78h       ; pTRF = FF78h — порт флагов приемопередатчика
pTRD    equ     0FEF8h       ; pTRD = FEF8h — порт данных приемопередатчика
ack_in  proc                 ; Процедура ввода с квитированием байта данных
        MOV     DX, pTRF     ; DX ← pTRF — адрес порта флагов приемопередатчика
LIN:    IN      AL, DX       ; AL ← ××××IBF2OBF2×× — чтение флагов (см. рис. 4.53)
        AND     AL, 8        ; Выделение флага IBF2
        JZ      LIN
        MOV     DX, pTRD     ; DX ← pTRD — адрес порта данных приемопередатчика
        IN      AL, DX       ; AL ← Input Buffer (ИС 1533ИР23)
        RET
ack_in  endp
ack_out proc                 ; Процедура вывода с квитированием байта данных
        MOV     DX, pTRF     ; DX ← pTRF — адрес порта флагов приемопередатчика
LOUT:   IN      AL, DX       ; AL ← ××××IBF2OBF2×× — чтение флагов (см. рис. 4.53)
        AND     AL, 4        ; Выделение флага OBF2
        JNZ     LOUT
        MOV     DX, pTRD     ; DX ← pTRD — адрес порта данных приемопередатчика
        MOV     AL, AH       ; AL ← AH — выводимые данные
        OUT     DX, AL       ; Output Buffer ← AL
        RET
ack_out endp
sc      proc                 ; Процедура вычисления контрольной суммы блока данных
        MOV     DX, 0        ; DX ← 0
LSC:    MOV     AL, [BX]     ; Чтение из SRAM одного байта блока данных
        AND     AX, 0Fh      ; Выделение младшей тетрады байта
        ADD     DX, AX       ; DX ← DX + AX   (SC ← SC + тетрада)
        MOV     AL, [BX]     ; Повторное чтение из SRAM одного байта блока данных
        SHR     AL, 1        ; Преобразование старшей тетрады байта в младшую тетраду
        SHR     AL, 1
        SHR     AL, 1
        SHR     AL, 1
        ADD     DX, AX       ; DX ← DX + AX   (SC ← SC + тетрада)
        INC     BX
        LOOP    LSC          ; DX = SC — контрольная сумма
        RET
sc      endp
Start:  MOV     AX, d_seg    ; Первые четыре команды имеются в программе задачи 1
        MOV     DS, AX       ; Инициализация сегментного регистра данных DS
        MOV     SS, AX       ; Инициализация сегментного регистра стека SS
        MOV     SP, S_Ram    ; SP ← S_Ram = 3000h
        CALL    ack_in       ; МикроЭВМ ← A_RAML
```

```
        MOV   BL, AL      ; BL ← A_RAML
        CALL  ack_in      ; МикроЭВМ ← A_RAMH
        MOV   BH, AL      ; BH ← A_RAMH,  BX = A_beg
        PUSH  BX
        CALL  ack_in      ; МикроЭВМ ← ΔARL
        MOV   CL, AL      ; CL ← ΔARL
        CALL  ack_in      ; МикроЭВМ ← T̄ΔARH
        MOV   CH, AL      ; CH ← T̄ΔARH
        AND   CH, 7Fh     ; CX = ΔAR — размер блока пересылаемых данных
        PUSH  CX
        AND   AL, 80h     ; Флаг T̄ = 0 ⇒ PC ← микроЭВМ (направление передачи)
        JZ    LT0         ; Флаг T̄ = 1 ⇒ микроЭВМ ← PC
        CALL  ack_in  ; МикроЭВМ ← SC_PCL     Пересылка данных микроЭВМ ← PC
        MOV   AH, AL      ; AH ← AL
        CALL  ack_in      ; МикроЭВМ ← SC_PCH
        XCHG  AL, AH
        MOV   SI, AX      ; SI = SC_PC — контрольная сумма блока данных PC
LPC:    CALL  ack_in      ; МикроЭВМ ←  блок данных PC
        MOV   [BX], AL
        INC   BX
        LOOP  LPC
        POP   CX
        POP   BX
        CALL  SC          ; DX = SC — контрольная сумма принятого блока данных
        CMP   SI, DX      ; Сравнение контрольных сумм SC и SC_PC
        JNZ   ERROR
          ∴               ; Вывод на дисплей сообщения VALID
        JMP   short FIN
ERROR:    ∴               ; Вывод на дисплей сообщения ERROR
        JMP   short FIN
LT0:    CALL  SC          ; DX = SC_RAM       Пересылка данных PC ← микроЭВМ
        MOV   CX, DX      ; CX ← DX
        MOV   AH, CL      ; AH ← CL
        CALL  ack_out     ; PC ← SC_RAML микроЭВМ
        MOV   AH, CH      ; AH ← CH
        CALL  ack_out     ; PC ← SC_RAMH микроЭВМ
        POP   CX
        POP   BX
LMC:    MOV   AH, [BX]    ; Чтение из SRAM одного байта блока данных
        CALL  ack_out     ; Пересылка блока данных PC ← микроЭВМ
        INC   BX
        LOOP  LMC
FIN:
c_seg   ends
        end   Start
```

Конфигурации памяти. На рис. 4.55 представлена структурная схема памяти, содержащая ПЗУ 64К × 8 бит (*EPROM* 27256 по 32К × 8 бит — см. рис. 3.18) и ОЗУ 64К × 8 бит (*SRAM MT5C2568* по 32К × 8 бит — см. рис. 1.45). Управление записью в *SRAM* младшего D_{7-0} и старшего D_{15-8} байтов данных производится, как и положено, сигналами A_0 и $\overline{BHE}$.

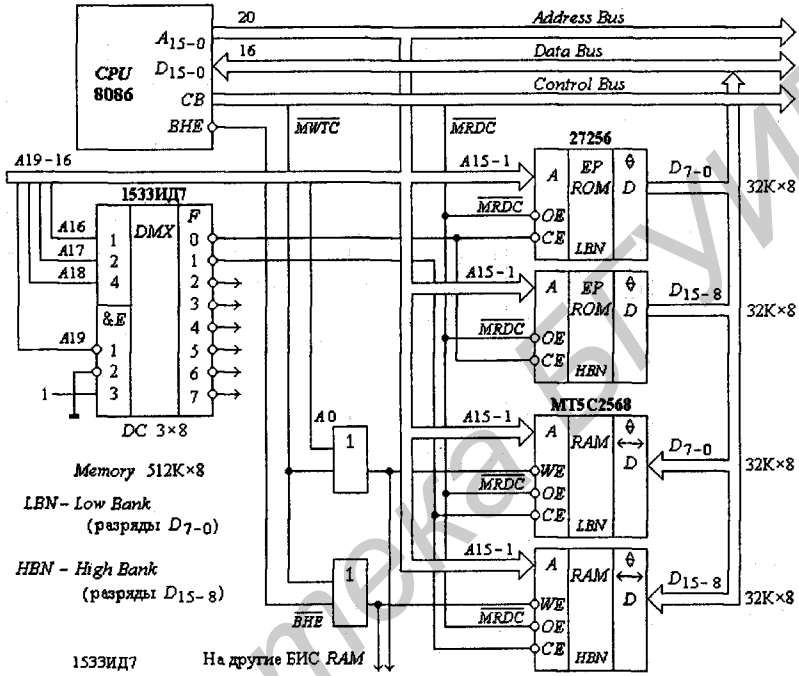


Рис. 4.55. Структурная схема памяти

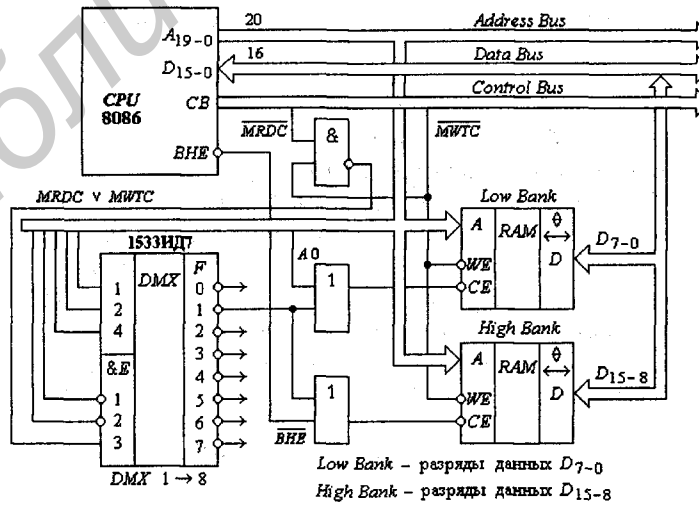


Рис. 4.56. Схема управления ОЗУ, не имеющего входа $\overline{OE}$

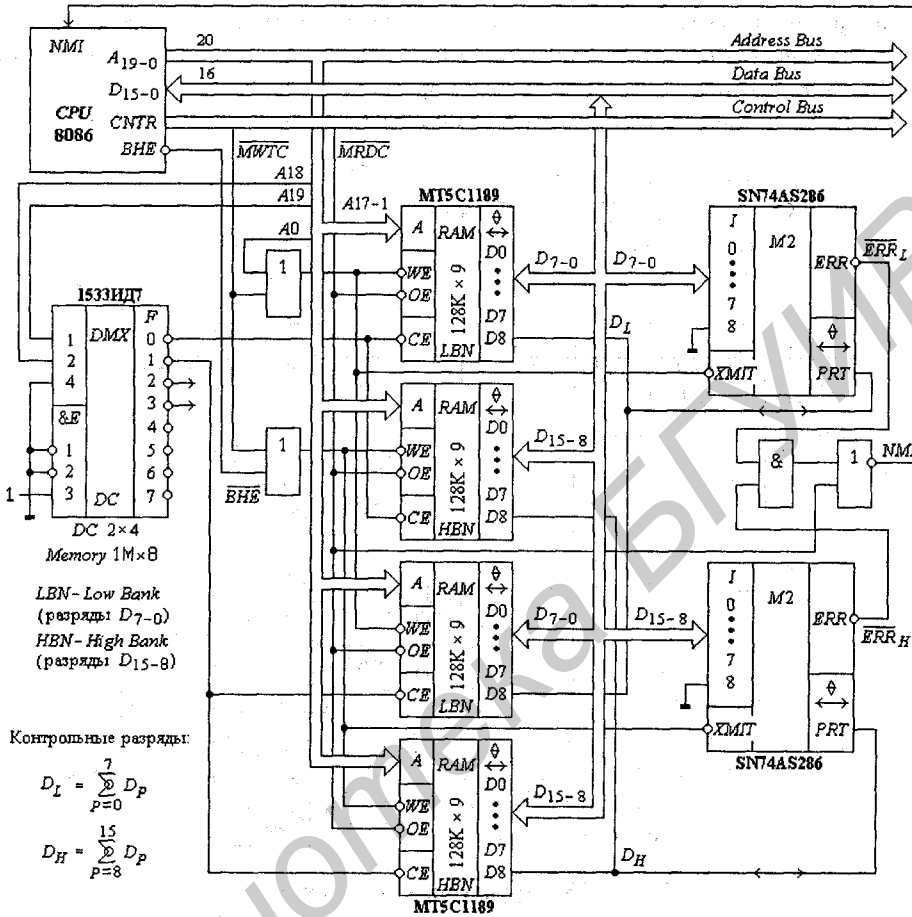


Рис. 4.57. Структурная схема ОЗУ с обнаружением однократных ошибок

Эта память может быть расширена до общего объема 512К × 8 бит за счет неиспользованных выходов дешифратора *DC* 3 × 8, выполненного на ИС 1533ИД7. Блок *CPU* 8086 должен быть выполнен по схеме, изображенной на рис. 4.13.

Если БИС *SRAM* не имеют входа $\overline{OE}$ для прямой подачи сигнала чтения $\overline{MRDC}$, то ОЗУ должно подключаться к системным шинам в соответствии с рис. 4.56 — селекция БИС по входам $\overline{CE}$ производится демультиплексированным сигналом $MRDC \vee MWTC$.

На рис. 4.57 изображена структурная схема ОЗУ объемом 512К × 8 бит с обнаружением однократных ошибок с помощью ИС контроля паритета *SN74AS286* (см. § 1.11). Здесь ОЗУ выполнено на четырех БИС *MT5C1189* по 128К × 9 бит каждая (см. рис. 1.45). Сигнал запроса немаскированного прерывания

$$NMI = \overline{ERR_L} \cdot \overline{ERR_H} \vee \overline{MRDC} = \overline{(ERR_L \vee ERR_H) \cdot MRDC} = \sqcup\!\sqcap$$

при обнаружении ошибки паритета в любом из байтов данных D_{7-0} или D_{15-8} в цикле чтения памяти.

4.9. Мультипроцессорные системы

Для увеличения производительности МП-систем используются их мультипроцессорные конфигурации. Все МП, входящие в мультипроцессорную систему (МПС), работают параллельно — на каждый МП возлагается решение некоторой части общей задачи. Для обмена данными между МП вся память (1 Мбайт) или ее часть разделяются (используются) несколькими однотипными или разных типов МП и *NDCP* (8086, 8088, 8089 и 8087).

Классификация мультипроцессорных систем. На практике принято различать *сопроцессорные*, *сильно связанные* (локальные) и *слабо связанные* (дистанционные) *конфигурации* МПС. На рис. 4.58 показана структурная схема МПС, иллюстрирующая принцип построения сопроцессорной и сильно связанной конфигураций (можно использовать только один МП 8086):

CPU 8086 и *NDCP* 8087 — сопроцессорная конфигурация (центральный процессор 8086 и арифметический сопроцессор 8087);

CPU 8086, *IOP*–1 8089 и *IOP*–2 8089 — сильно связанная конфигурация (центральный процессор 8086 и процессоры ввода-вывода 8089 [12, 13]; *IOP — I/O Processor*); в отличие от сопроцессора 8087 процессор ввода-вывода 8089 самостоятельно выбирает свои команды из памяти, т. е. *IOP* является независимым микропроцессором, который можно использовать и в слабо связанной конфигурации.

При использовании сопроцессорных и сильно связанных конфигураций главный (*host*) МП 8086 выполняет функцию приоритетного арбитража шин — предоставление шин сопроцессору 8087 и процессорам ввода-вывода 8089 осуществляется с помощью двунаправленных сигналов $\overline{RQ}/\overline{GT}$ (см. рис. 4.9) по их запросам системной шины $\overline{RQ} = \sqcup\!\!\sqcap$.

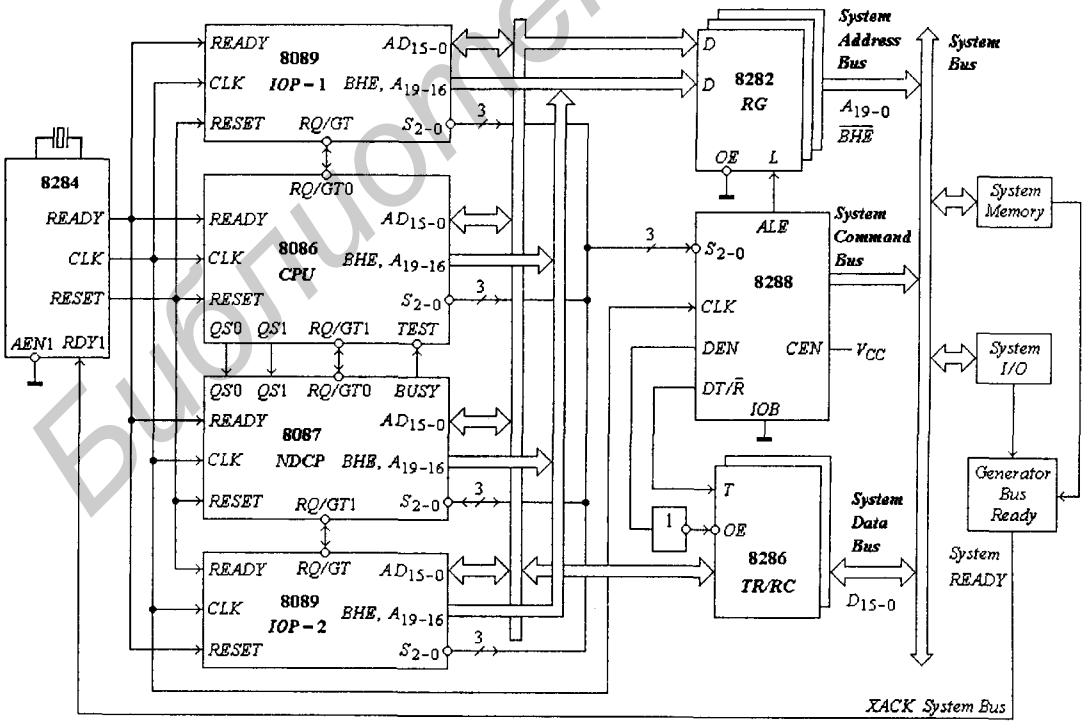


Рис. 4.58. Структурная схема МПС средней сложности

В сопроцессорных и сильно связанных конфигурациях МПС доступ ко всем ресурсам системы (память и устройства ввода-вывода) осуществляется через общую системную шину данных — используется общий генератор синхронизации 8284 и общая логика управления шинами: ИС 8288 (контроллер шин), ИС 8282 (функциональный аналог 8-разрядного асинхронного потенциального регистра памяти 1533ИР22) и ИС 8286 (функциональный аналог 8-разрядного приемопередатчика 1533АП6 [5]). В каждый момент времени к системной шине данных может обращаться только один МП или сопроцессор, поэтому возможности их параллельной работы ограничены (в частности, *CPU* 8086, *IOP*–1 8089 и *IOP*–2 8089 выборку команд своих программ должны производить из общей системной памяти, а пропускная способность системной шины данных ограничена).

В слабо связанных конфигурациях МПС (рис. 4.59; *Status* = $\overline{S}_{2-0}$, *A/D* = AD_{15-0}, A_{19-16}, $\overline{BHE}$) приоритетный арбитраж шин выполняется арбитрами шин 8289 (*Bus Arbiter* — рис. 4.60), связанных между собой шиной управления *Multi-Master Control Bus*, и используются независимые МП с возможностью предоставления им различных комбинаций из шин трех типов: общей для всех МП *системной шины* (*Multi-Master System Bus*), *резидентной шины* (собственной шины для любого МП) и *шины ввода-вывода* (также собственной шины для любого МП). К системной шине может обращаться каждый МП для управления общими (системными) ресурсами (*System Memory* и *System I/O*), в частности, для обмена данными с другими МП.

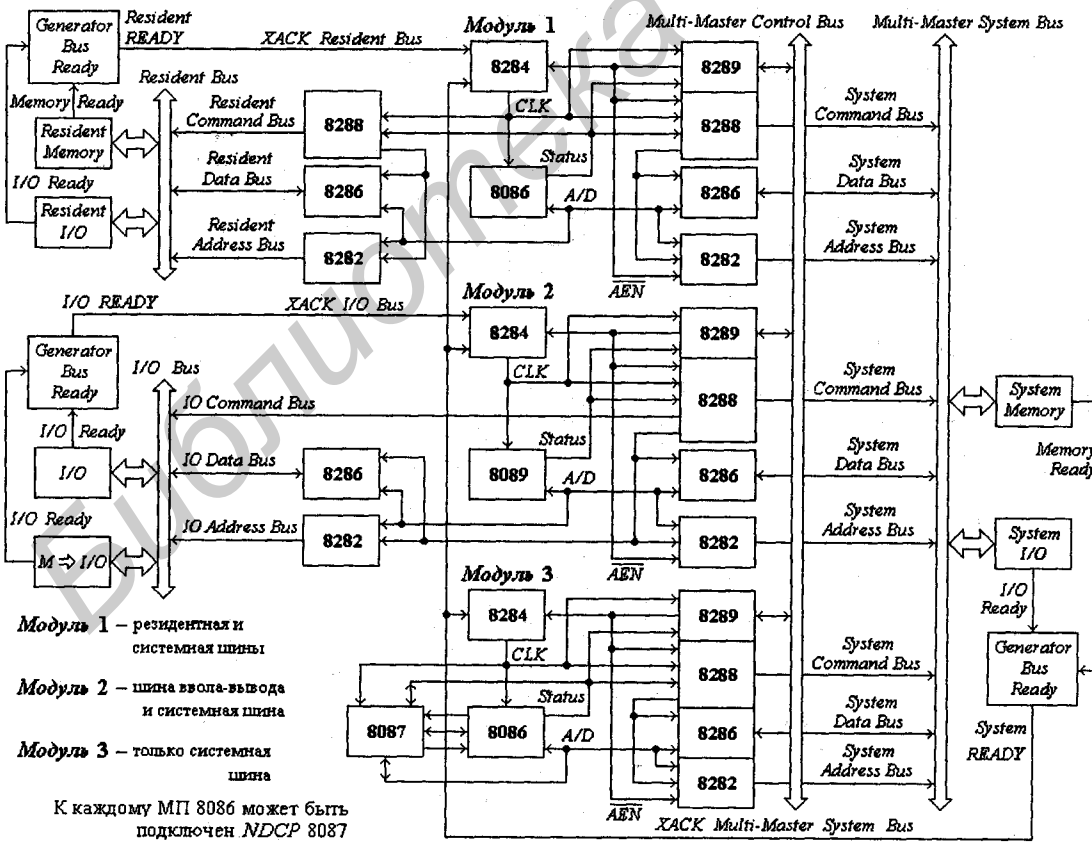


Рис. 4.59. Структурная схема большой МПС

По резидентной шине МП обращается к индивидуальным ресурсам (*Resident Memory* и *Resident I/O*), а по шине ввода-вывода — к индивидуальным устройствам ввода-вывода *I/O* и памяти *M*, отображенной на устройства ввода-вывода (*M* ⇒ *I/O* на рис. 4.59).

Арбитр шин 8289. Фирма *Intel* выпустила два варианта арбитров шин 8289 и 8289–1, различающихся быстродействием, определяемым частотами тактовых сигналов *CLK* и $\overline{BCLK}$:

$$f_{CLK} \le 8 \text{ МГц}, f_{BCLK} \le 10 \text{ МГц (8289) и } f_{CLK} \le 10 \text{ МГц}, f_{BCLK} \le 10 \text{ МГц (8289–1)}.$$

Контроллеры шин 8289 и 8289–1 изготовляются по ТТЛШ-технологии и характеризуются параметрами (рассеиваемая мощность равна 1,5 Вт):

$V_{OL} \le 0{,}45$ В при $I_{OL} = 20$ мА ($\overline{BUSY}$, $\overline{CBRQ}$), $I_{OL} = 16$ мА ($\overline{AEN}$) и $I_{OL} = 10$ мА ($\overline{BPRO}$, $\overline{BREQ}$);

V_{OH} — *Open Collector* ($\overline{BUSY}$, $\overline{CBRQ}$ — открытый коллекторный выход);

$V_{OH} \ge 2{,}4$ В при $I_{OH} = -0{,}4$ мА ($\overline{AEN}$, $\overline{BPRO}$, $\overline{BREQ}$).

Структурная схема арбитра шин изображена на рис. 4.61:

State Generator — генератор состояния МП (дешифратор сигналов состояния $\overline{S}_{2-0}$), идентифицирующий операцию, выполняемую микропроцессором в текущем цикле шины, для инициирования действий арбитра шин по захвату и освобождению системной шины;

Control — схема управления, предназначенная для задания режимов работы арбитра шин и его синхронизации сигналом *CLK* со стороны МП;

Arbitration — схема приоритетного арбитража, анализирующая сигналы управления шины *Multibus™ Command Signal* для предоставления микропроцессору доступа к системной шине;

Multibus Interface — интерфейс *Multibus*, осуществляющий взаимодействие всех арбитров шин МПС и синхронизирующий сигналом $\overline{BCLK}$ действия по захвату и освобождению микропроцессорами системной шины;

Local Bus Interface — интерфейс локальной шины, формирующий сигнал разрешения $\overline{AEN}$ доступа МП к системной шине ($\overline{AEN} = 0$) или к шине ввода-вывода ($\overline{AEN} = 1$).

Арбитр шин выполнен с аппаратно перестраиваемой сигналами $\overline{IOB}$ и *RESB* конфигурацией для задания четырех режимов его работы (см. табл. 4.39). Сигналы арбитра шин имеют назначение:

$\overline{S}_{2-0}$ (*Status* — состояние) — сигналы состояния, поступающие от МП 8086/8088/8089 или *NDCP* 8087 и идентифицирующие тип цикла шины (см. табл. 4.37). Состояние $\overline{S}_{2-0}$ декодируется арбитром шин для инициирования действий по захвату, удержанию и освобождению системной шины;

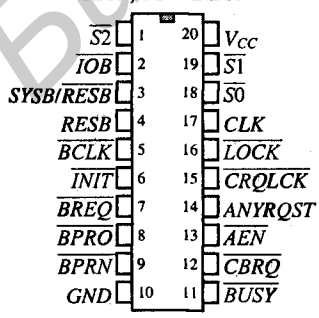


Рис. 4.60. Арбитр шин

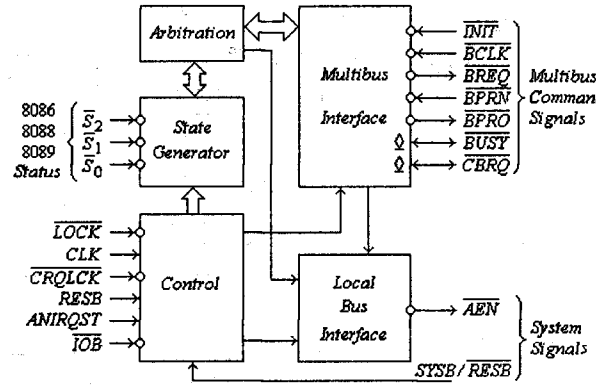


Рис. 4.61. Структурная схема арбитра шин

CLK (*Clock*) — тактовый сигнал от генератора 8284*A*, осуществляющий синхронизацию работы арбитра шин с действиями микропроцессора;

$\overline{LOCK}$ (*Lock* — блокировка) — сигнал запрета ($\overline{LOCK}$ = 0) освобождения системной шины любому другому арбитру независимо от его приоритета. На вход $\overline{LOCK}$ арбитра шин подается сигнал с выхода $\overline{LOCK}$ МП, активное значение которого (0) инициируется префиксом LOCK и сохраняется до конца выполнения следующей за ним команды. Значение сигнала $\overline{LOCK}$ = 0 также во время и между импульсами $\overline{INTA}$ = 0;

$\overline{CRQLCK}$ (*Common Request Lock* — блокировка общего запроса) — сигнал запрета освобождения системной шины при запросе ее по входу $\overline{CBRQ}$;

RESB (*Resident Bus* — резидентная шина) — сигнал аппаратного задания режима работы МП с резидентной и системной шинами (*RESB* = 1 — рис. 4.62). Управление выбором обращения к одной из этих шин производится сигналом *SYSB*/$\overline{RESB}$;

SYSB/$\overline{RESB}$ (*System Bus/Resident Bus* — системная шина/резидентная шина) — сигнал разрешения обращения к системной (*SYSB*/$\overline{RESB}$ = 1) или резидентной (*SYSB*/$\overline{RESB}$ = 0) шинам при значении сигнала режима *RESB* = 1. При задании значения *RESB* = 0 сигнал *SYSB*/$\overline{RESB}$ игнорируется. Сигнал *SYSB*/$\overline{RESB}$ формируется внешним адресным дешифратором *DC* (рис. 4.62), разделяющим адресное пространство памяти (1 Мбайт) на две части — системную память *SM* и резидентную память *RM*. Если в МПС несколько МП имеют резидентную шину, то общий объем памяти всей системы будет значительно больше 1 Мбайта;

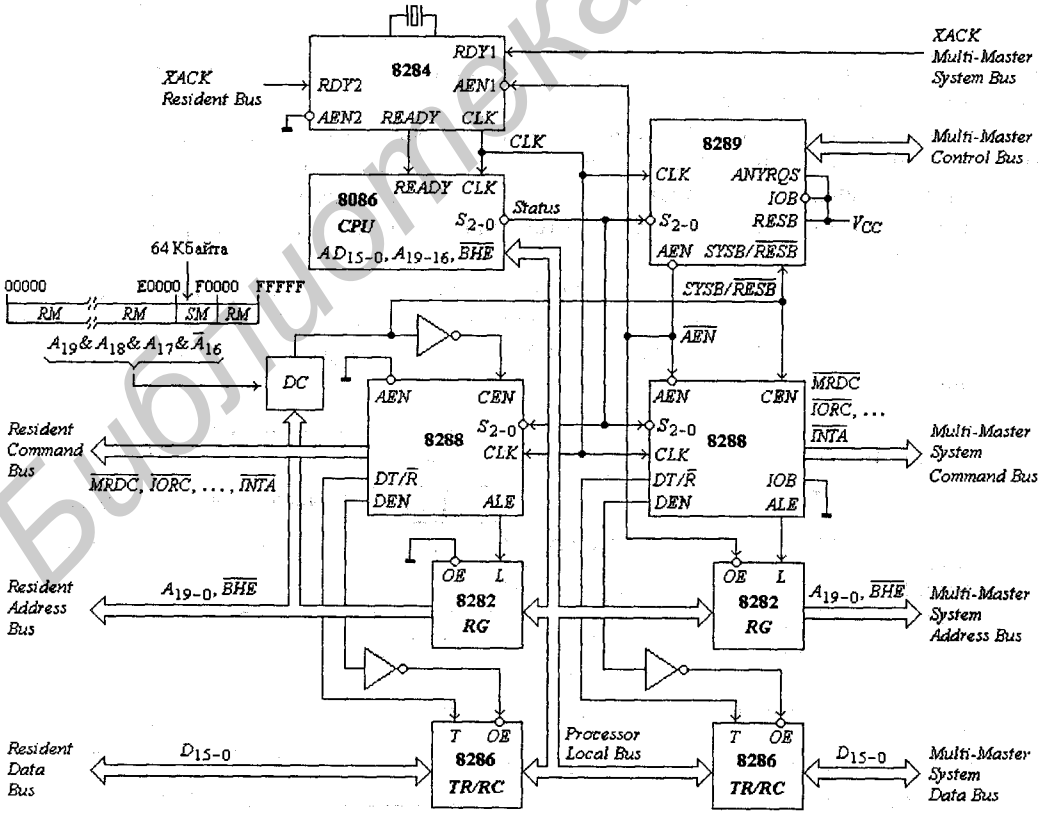


Рис. 4.62. Модуль 1 МПС, изображенной на рис. 4.59 (резидентная и системная шины)

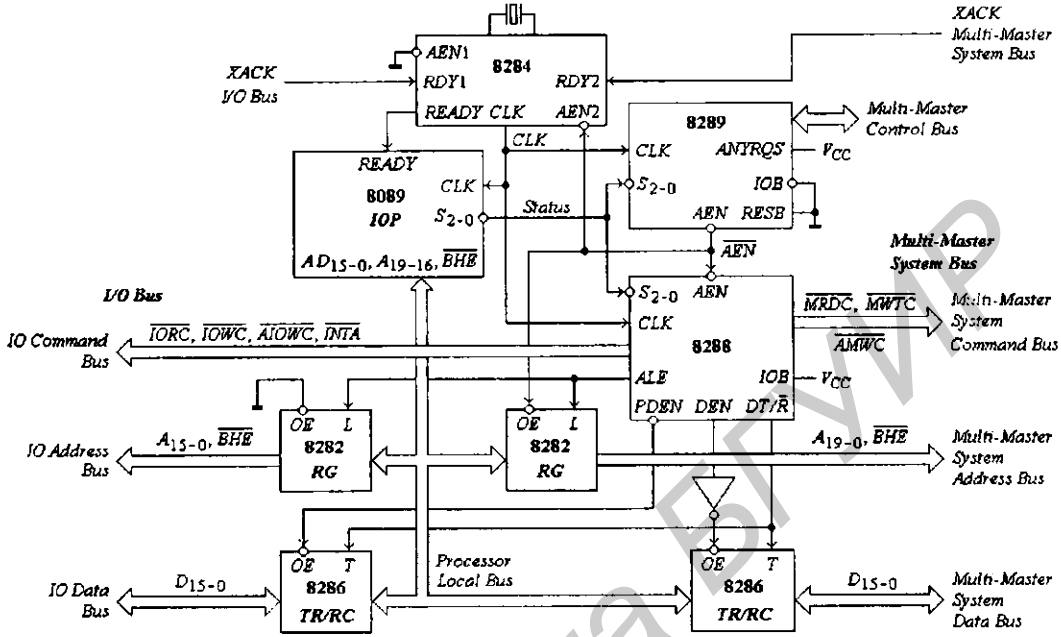


Рис. 4.63. Модуль 2 МПС, изображенной на рис. 4.59 (шина ввода-вывода и системная шина)

ANIRQST (*Any Request* — любой запрос) — сигнал разрешения освобождения системной шины при ее запросе любым арбитром с более низким приоритетом, как будто это был запрос системной шины от арбитра, имеющего более высокий приоритет. Значение сигнала $\overline{CBRQ} = 0$ при значении $ANYRQST = 1$ заставляет арбитр шины освободить шину по окончании текущего цикла передачи;

$\overline{IOB}$ (*IO Bus* — шина ввода-вывода) — сигнал аппаратного задания режима работы МП с шиной ввода-вывода и системной шиной ($\overline{IOB} = 0$ — рис. 4.63). Арбитр шин запрашивает системную шину только при необходимости обращения МП к памяти (см. табл. 4.39);

$\overline{AEN}$ (*Address Enable* — разрешение доступа) — сигнал разрешения доступа к системной шине, предназначенный для перевода МП в состояние ожидания освобождения ее другим МП. Этот сигнал предназначен для управления контроллером шин 8288, адресными регистрами 8282 и сигналом готовности *READY* генератора 8284*A* (рис. 4.62). Значение сигнала $\overline{AEN} = 1$ переводит в Z-состояние выходы сигналов команд ИС 8288 и адресные выходы ИС 8282;

$\overline{CBRQ}$ (*Common Bus Request* — общий запрос шины) — двунаправленный сигнал общего запроса шины с открытым коллекторным выходом (см. рис. 4.64 и 4.65). Как входной сигнал ($\overline{CBRQ} = 0$) сообщает арбитру шин, что имеются другие арбитры с более низким приоритетом, запрашивающие доступ к системной шине. Значение $\overline{CBRQ} = 0$ как выходного сигнала выдается арбитрами шин, которые не получили доступа к системной шине, но требуют его. Значение $\overline{CBRQ} = 1$ как входного сигнала сообщает арбитру шин, что запросов системной шины от других арбитров нет, и он может удерживать управление системной шиной и в пассивном состоянии ($\overline{S}_2\overline{S}_1\overline{S}_0 = 111$) для экономии времени, затрачиваемого на последующий захват шины;

$\overline{INIT}$ (*Initialize* — инициализация) — сигнал сброса всех арбитров шин. После подачи значения сигнала $\overline{INIT} = 0$ ни один из арбитров не имеет доступа к системной шине;

$\overline{BCLK}$ (*Bus Clock*) — тактовый сигнал синхронизации работы всех арбитров шин но захвату системной шины. В качестве сигнала $\overline{BCLK}$ можно использовать сигнал *CLK* одного из МП;

$\overline{BPRN}$ (*Bus Priority In*) — входной сигнал приоритетного разрешения доступа к шине (см. рис. 4.64 и 4.65). Значение сигнала $\overline{BPRN} = 0$ указывает арбитру шин, что он имеет самый высокий приоритет из всех арбитров, одновременно запросивших доступ к системной шине, и что он может захватить системную шину при следующем переходе сигнала $\overline{BCLK}$ с 1 на 0. Значение сигнала $\overline{BPRN} = 1$ сообщает арбитру шин, что он потерял приоритет по отношению к некоторым другим арбитрам;

$\overline{BPRO}$ (*Bus Priority Out*) — выходной сигнал приоритетного разрешения доступа к шине ($\overline{BPRO} = 0$) в схеме с последовательным включением арбитров шин (см. рис. 4.64);

$\overline{BREQ}$ (*Bus Request*) — сигнал запроса системной шины ($\overline{BREQ} = 0$) в схеме с параллельным включением арбитров шин (см. рис. 4.65);

$\overline{BUSY}$ (*Busy* — занятый) — двунаправленный сигнал занятости шины с открытым коллекторным выходом (см. рис. 4.64 и 4.65). Значение $\overline{BUSY} = 0$ как выходного сигнала выдается арбитром шин, получившим доступ к системной шине, для запрета доступа к системной шине остальным арбитрам. Закончив работу с системной шиной, арбитр выдает значение сигнала $\overline{BUSY} = 1$, позволяя другим арбитрам захватить системную шину.

Методы приоритетного арбитража. Можно использовать три метода арбитража, основанных на приоритетной концепции: последовательный метод, параллельный метод и параллельный метод с циклическим сдвигом приоритетов. В последовательной приоритетной цепочке (*daisy-chain*) арбитр шин AB_0 (рис. 4.64) имеет наибольший приоритет, и далее по цепочке приоритеты арбитров AB_m последовательно уменьшаются. Если арбитр AB_0 не запрашивает управление системной шиной, то он устанавливает значение сигнала $\overline{BPRO}_0 = 0$ для передачи разрешения запроса арбитру шин AB_1 (и т. д. по цепочке).

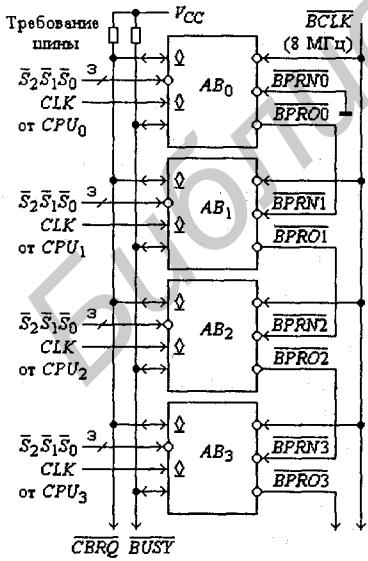


Рис. 4.64. Последовательное включение арбитров шин

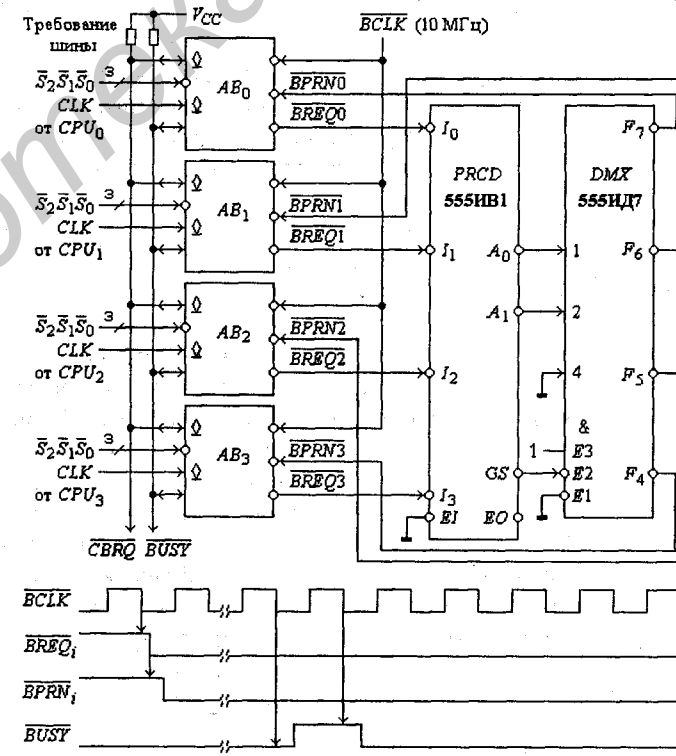


Рис. 4.65. Параллельное включение арбитров шин

От начала к концу цепочки задержки сигналов $\overline{BPRO_i}$ увеличиваются, поэтому при частоте 10 МГц сигнала $\overline{BCLK}$ можно последовательно включить не более трех арбитров (арбитраж должен быть выполнен за один период тактового сигнала $\overline{BCLK}$).

Для параллельного включения арбитров шин (рис. 4.65) требуются дополнительные ИС: 555ИВ1 (приоритетный шифратор 8 × 3 [5]) и 555ИД7 (демультиплексор 1 → 8), позволяющие организовать арбитраж до восьми независимых МП (задержки не накапливаются).

Режимы работы арбитров шин задаются аппаратно сигналами $\overline{IOB}$ и *RESB* (табл. 4.39) для настройки работы каждого МП с определенной комбинацией шин (системная шина имеется в любом режиме). Режим работы с резидентной шиной и шиной ввода-вывода ($\overline{IOB}$ *RESB* = 0 1) не имеет преимуществ перед режимом работы с резидентной шиной ($\overline{IOB}$ *RESB* = 1 1), поэтому он используется редко (аппаратная часть только усложняется из-за наличия трех шин).

Таблица 4.39. **Режимы работы арбитра шин 8289**

Команды	$\overline{S_2}\,\overline{S_1}\,\overline{S_0}$	$\overline{IOB}$ *RESB*					
		11 [1]		00 [2]	10 [3]	01 [4]	
		SB/$\overline{RB}$ = 1	*SB/*$\overline{RB}$ = 0	*SB/*$\overline{RB}$ = ×	*SB/*$\overline{RB}$ = ×	*SB/*$\overline{RB}$ = 1	*SB/*$\overline{RB}$ = 0
Команды ввода-вывода (*I/O Status*)	0 0 0	+	–	–	+	–	–
	0 0 1	+	–	–	+	–	–
	0 1 0	+	–	–	+	–	–
Состояние останова (*Halt Status*)	0 1 1	–	–	–	–	–	–
Команды обращения к памяти (*Memory Status*)	1 0 0	+	–	+	+	+	–
	1 0 1	+	–	+	+	+	–
	1 1 0	+	–	+	+	+	–
Пассивное состояние (*Idle*)	1 1 1	–	–	–	–	–	–

Примечание: *SB/*$\overline{RB}$ = *SYSB/*$\overline{RESB}$, "+" — арбитр запрашивает управление системной шиной, "–" — системная шина не запрашивается и может быть сдана другому арбитру, [1] резидентная шина (см. рис. 4.62), [2] шина ввода-вывода (см. рис. 4.63), [3] только системная шина (рис. 4.66), [4] резидентная шина и шина ввода-вывода.

Таблица 4.40. **Условия запроса и освобождения системной шины**

Режим	$\overline{IOB}$	*RESB*	Условия запроса системной шины	Условия освобождения системной шины
Резидентная шина	1	1	(*SYSB/*$\overline{RESB}$ = 1) & & *Active Status* $\overline{S_2}\,\overline{S_1}\,\overline{S_0}$	(*SYSB/*$\overline{RESB}$ = 0 ∨ *TI*) & *CBRQ* ∨ ∨ *HLT* ∨ *HPBRQ*
Шина ввода-вывода	0	0	*Memory Command*	(*I/O Status* ∨ *TI*) & *CBRQ* ∨ ∨ *HLT* ∨ *HPBRQ*
Только одна общая системная шина	1	0	*Active Status* $\overline{S_2}\,\overline{S_1}\,\overline{S_0}$	*HLT* ∨ *TI* & *CBRQ* ∨ *HPBRQ*
Резидентная шина и шина ввода-вывода	0	1	(*Memory Command*) & & (*SYSB/*$\overline{RESB}$ = 1)	(*I/O Status* ∨ *SYSB/*$\overline{RESB}$ = 0) & & *CBRQ* ∨ *HPBRQ* ∨ *HLT*

Примечание: *HPBRQ* — *Higher Priority Bus Request or* $\overline{BPRN}$ = 1, *TI* — *Processor Idle Status* ($\overline{S_2}\,\overline{S_1}\,\overline{S_0}$ = = 111), *HLT* — *Processor Halt Status* ($\overline{S_2}\,\overline{S_1}\,\overline{S_0}$ = 011).

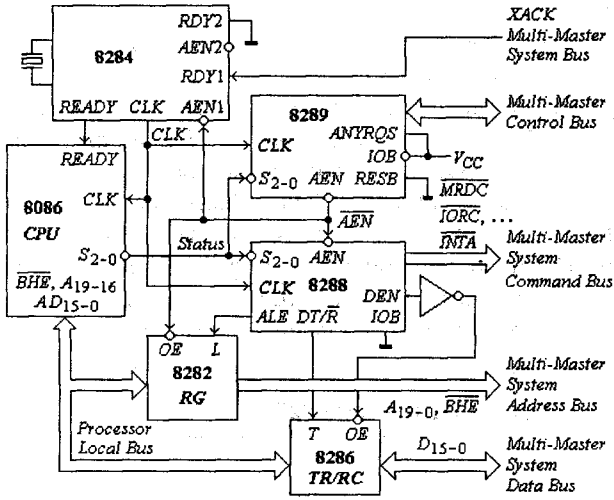


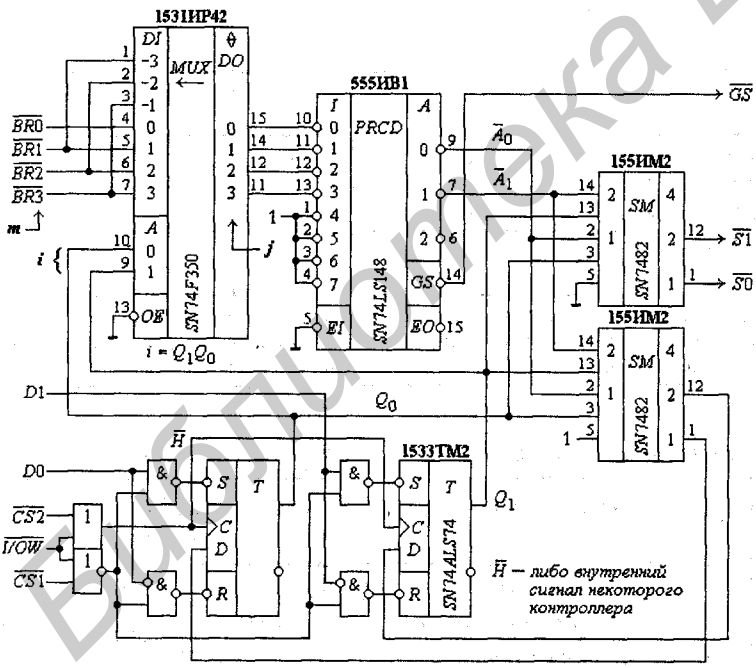
Рис. 4.66. Модуль 3 МПС, изображенной на рис. 4.59

Условия запроса и освобождения системной шины арбитрами шин 8289 в зависимости от режима их работы приведены в табл. 4.40.

Универсальный шифратор 4 × 2 с циклическим сдвигом приоритетов. Принципиальная схема такого шифратора показана на рис. 4.67:

$$i^+ = Q_1^+ Q_0^+ = \langle \overline{S}_1 \overline{S}_0 + 1 \rangle,$$

где $\overline{S}_1\overline{S}_0 = Q_1Q_0 + \overline{A}_1\overline{A}_0$, $\langle E \rangle$ — значение числа E по модулю 4. Если в схеме, изображенной на рис. 4.65, вместо ИС 555ИВ1 использовать эту схему, то будет реализован параллельный метод арбитража с циклическим сдвигом приоритетов.



i	Q_1 Q_0	j			
		3	2	1	0
0	0 0	3	2	1	0
1	0 1	2	1	0	3
2	1 0	1	0	3	2
3	1 1	0	3	2	1

↑ m

i	Q_1 Q_0	$\overline{A}_1\overline{A}_0$			
		00	01	10	11
0	0 0	00	01	10	11
1	0 1	01	10	11	00
2	1 0	10	11	00	01
3	1 1	11	00	01	10

↑ $\overline{S}_1\overline{S}_0$

m	$\overline{S}_1$ $\overline{S}_0$	Q_1^+ Q_0^+	i^+
0	1 1	0 0	0
1	1 0	1 1	3
2	0 1	1 0	2
3	0 0	0 1	1

Рис. 4.67. Универсальный приоритетный циклический шифратор 4 × 2

Для построения мультипроцессорных систем на базе МП 8080 и 8085 используются контроллеры шин 8218 и 8219, содержащие арбитр шин, подобный арбитру 8289.

Рассмотренные в данном учебном пособии вопросы должны помочь быстрому освоению основ проектирования как аппаратной (*hardware*), так и программной (*software*) частей микропроцессорных и мультипроцессорных систем.

ЛИТЕРАТУРА

1. **Коффрон Дж.** Технические средства микропроцессорных систем: Практический курс. / Пер. с англ. — М.: Мир, 1983. — 344 с.

2. **Микропроцессоры** и микропроцессорные комплекты интегральных микросхем: Справочник. В 2 т. / Под ред. В. А. Шахнова. — М.: Радио и связь, 1988. — Т. 1. — 368 с.

3. **Микро-ЭВМ** / Пер. с англ. Под ред. А. Дирксена. — М.: Энергоиздат, 1982. — 328 с.

4. **Токхайм Р.** Микропроцессоры: Курс и упражнения / Пер. с англ.; Под ред. В. Н. Грасевича. — М.: Энергоатомиздат, 1987.

5. **Пухальский Г. И., Новосельцева Т. Я.** Цифровые устройства: Учебное пособие для втузов. — СПб.: Политехника, 1996. — 885 с.

6. **Григорьев В. Л.** Программное обеспечение микропроцессорных систем. — М.: Энергоатомиздат, 1983. — 208 с.

7. **Гуртовцев А. Л., Гудыменко С. В.** Программы для микропроцессоров: Справ. пособие. — Минск: Высш. шк., 1989. — 352 с.

8. **Пухальский Г. И., Новосельцева Т. Я.** Проектирование дискретных устройств на интегральных микросхемах: Справочник. — М.: Радио и связь, 1990. — 304 с.

9. **Берлекэмп Э.** Алгебраическая теория кодирования / Пер. с англ. Под ред. И. И. Грушко. — М.: Мир, 1971. — 477 с.

10. **Левенталь Л.** Введение в микропроцессоры: Программное обеспечение, аппаратные средства, программирование / Пер. с англ. — М.: Энергоатомиздат, 1983. — 464 с.

11. **Рафикузаман М.** Микропроцессоры и машинное проектирование микропроцессорных систем. В 2-х кн. / пер. с англ. — М.: Мир, 1988. — Кн. 1. — 312 с. Кн. 2. — 288 с.

12. **Лю Ю-Чжен, Гибсон Г.** Микропроцессоры семейства 8086/8088. Архитектура, программирование и проектирование микрокомпьютерных систем / Пер. с англ. — М.: Радио и связь, 1987. — 512 с.

13. **Микропроцессорный** комплект К1810: структура, программирование, применение: Справочная книга / Ю. М. Казаринов, В. Н. Номоконов, Г. С. Подклетнов, Ф. В. Филиппов; Под ред. Ю. М. Казаринова. — М.: Высш. шк., 1990. — 269 с.

14. **Дао Л.** Программирование микропроцессора 8088 / Пер. с англ. — М.: Мир, 1988. — 358 с.

15. **Григорьев В. Л.** Программирование однокристальных микропроцессоров. — М.: Энергоатомиздат, 1987. — 288 с.

16. **Использование** *Turbo Assembler* при разработке программ / Сост. А. А. Чекатков. — Киев: Диалектика, 1995. — 288 с.

17. **Абель П.** Язык ассемблера для *IBM PC* и программирования / Пер. с англ. Ю. В. Сальникова. — М.: Высш. шк., 1992. — 447 с.

18. **Лямин Л. В.** Макроассемблер *MASM*. — М.: Радио и связь, 1994. — 320 с.

19. **Финогенов К. Г.** Самоучитель по системным функциям *MS-DOS*. — М.: Радио и связь; Энтроп, 1995. — 382 с.

20. **Григорьев В.** Л. Архитектура и программирование арифметического сопроцессора. — М.: Энергоатомиздат, 1991. — 208 с.

21. **Скэнлон Л.** Персональные ЭВМ *IBM* PC и XT. Программирование на языке ассемблера / Пер. с англ. — М.: Радио и связь. 1991. — 336 с.

22. **Паппас К., Марри У.** Микропроцессор 80386: Справочник / Пер. с англ. — М.: Радио и связь, 1993. — 320 с.

23. **Морс С. П., Алберт Д. Д.** Архитектура микропроцессора 80286: Пер. с англ. — М.: Радио и связь, 1990. — 304 с.

24. **Микропроцессоры** и микропроцессорные комплекты интегральных микросхем: Справочник. В 2 т. / Под ред. В. А. Ш а х н о в а. — М.: Радио и связь, 1988. — Т. 2. — 368 с.

ОГЛАВЛЕНИЕ

Глава 4. Микропроцессоры 8086/8088 и сопроцессор 8087