МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский ядерный университет «МИФИ»
Саровский физико-технический институт
Филиал федерального государственного автономного образовательного учреждения высшего образования
«Национальный исследовательский ядерный университет «МИФИ»
(СарФТИ НИЯУ МИФИ)

Факультет информационных технологий и электроники (ФИТЭ)

Кафедра вычислительных и информационных технологий (ВИТ)

В.А. Павлов

Учебно-методические материалы по дисциплинам «Архитектура ЭВМ», «Организация ЭВМ» и «ЭВМ и периферийные устройства»

для бакалавров
информационных технологий и электроники

УТВЕРЖДЕНЫ
на заседании кафедры

«___» _____2021г., протокол № ___
Заведующий кафедры

_____Холушкин В. С.

Председатель научно-методического совета
СарФТИ НИЯУ МИФИ

_____Ф.П. Скрыпник

Саров
2021

Учебно-методические материалы:

Организация микропроцессора (i8086) и сопроцессора (i8087), архитектура их команд.

Аннотация

Рассмотрена организация микропроцессора К1810ВМ86 (i 8086), арифметического сопроцессора К1810ВМ87 (i8087) и архитектура их команд. Приведена мнемоника их команд, используемая в ассемблере и отладчиках Debug и AFD, а также примеры их двоичной структурной организации и их шестнадцатеричные эквиваленты. В конце разделов сформулированы контрольные вопросы для самотестирования

В учебных материалах использована информация из первоисточников [1], [2], [3]и др.

Учебные материалы предоставляются студентам СарФТИ НИЯУ МИФИ в электронном виде.

Павлов Виктор Александрович

Оглавление

Предисловие

Микропроцессор i8086 и сопроцессор i8087 являются родоначальниками микропроцессоров шестнадцатеричной архитектуры i16 фирмы Intel с системой команд CISC (Complex Instruction Set Computer). В них были заложены базисные решения, которые в той или иной мере поддерживаются в микропроцессорах архитектуры i32 и i64. Программное обеспечение, разработанное для этих микропроцессоров, в рамках подрежимов работы микропроцессоров архитектуры i32 и i64 или в рамках соответствующих виртуальных машин может выполняться и модифицироваться на современной вычислительной технике.

В учебных материалах первый раздел посвящен рассмотрению структурной организации микропроцессора К1810ВМ86 (i8086), режимов работы, организации памяти и организации прерываний.

Во втором разделе рассмотрены форматы команд, способы адресации, дается общее описание команд, и раскрываются особенности выполнения отдельных команд.

В третьем разделе в рамках первого подраздела приводятся общие сведения о сопроцессоре К1810ВМ87 (i8087). Второй подраздел посвящен рассмотрению особенностей его структурной организации. В третьем подразделе рассмотрены форматы данных используемых в сопроцессоре. В четвертом подразделе дается достаточно подробное описание системы команд сопроцессора. В пятом подразделе кратко рассмотрены специальные случаи функционирования сопроцессора. В шестом подразделе приводится информация об особенностях использования мнемоники команд сопроцессора в рамках ассемблера и отладчиков Debug и AFD.

Данные учебные материалы предназначены для информационного обеспечения студентов СарФТИ в рамках дисциплин «Архитектура ЭВМ», «Организация ЭВМ», «ЭВМ и периферийные устройства» и других дисциплин, связанных с программированием на самом нижнем уровне (на уровне ассемблера и кодов команд).

В.А. Павлов.

1. Микропроцессор К1810ВМ86 (i8086)

1.1. Общие сведения

Микропроцессор (МП) К1810ВМ86 является аналогом микропроцессора i8086 фирмы Intel и входит в состав отечественного микропроцессорного комплекта (МПК) серии к 1810 (см. табл.1.1).

Таблица 1.1. Микропроцессорный комплект серии К1810

Тип БИС	Назначение	Технология
К1810ВМ86	Центральный процессор	n-МДП
К1810ВМ88	Центральный процессор с 8-битовой внешней шиной данных	n-МДП
К1810ВМ87	Арифметический сопроцессор	n-МДП
К1810ВМ89	Специализированный процессор ввода - вывода	n-МДП
К1810ГФ84	Генератор тактовых сигналов	ТТЛШ
К1810ВГ88	Системный контроллер	ТТЛШ
К1810ВБ89	Арбитр системной шины	ТТЛШ
К1810ВТ02	Контроллер динамической памяти (16К)	n-МДП
К1810ВТ03	Контроллер динамической памяти (64К)	n-МДП
К1810ВИ54	Интервальный таймер	n-МДП
К1810ВТ37	Усовершенствованный контроллер прямого доступа к памяти	n-МДП
К1810ВН59	Программируемый контроллер прерываний	n-МДП
К1810ИР82/83	Регистр-защелка	ТТЛШ
К1810ВА86/87	Шинный формирователь	ТТЛШ

В микропроцессорной системе (МПС), выполненной на базе МПК К1810, микропроцессор К1810ВМ86 используется в качестве центрального процессора осуществляющего обработку данных и общее управление системой в соответствии с заданной программой. Характерной особенностью МП К1810ВМ86 является возможность частичной реконфигурации аппаратной части для обеспечения работы в двух режимах — минимальном и максимальном.

В минимальном режиме МП формирует все сигналы для управления внутрисистемным интерфейсом МПС и используется для построения однопроцессорных контроллеров и микро ЭВМ. При этом для построения блока центрального процессора используется малое число дополнительных ИС (генератор К1810ГФ84, буферные регистры К1810ИР82/83 и шинные формирователи К1810ИР86/87). ***В максимальном режиме*** МП используется для построения многопроцессорных МПС, в которых сигналы управления шиной вырабатываются системным контроллером К1810ВГ88 на основании кода, сформированного МП.

Центральный процессор К1810ВМ88 (аналог i8088) в отличие от К1810ВМ86 имеет 8-разрядную внешнюю шину данных, что, в частности, дает возможность вводить его в системы, ранее разработанные для МП К580ВМ80 (аналог i8080).

Арифметический сопроцессор К1810ВМ87, называемый также сопроцессором числовых данных, подключается параллельно центральному процессору и выполняет предназначенные ему команды из общего с ЦП потока команд. Оба процессора фактически работают попеременно, причем сопроцессор является зависимым от центрального процессора.

Процессор ввода – вывода К1810ВМ89 и ЦП являются независимыми, и каждый из них выполняет собственный поток команд, что позволяет распараллелить выполнение программы.

К1810ВМ86 имеет встроенные средства для координации работы с арифметическим сопроцессором К1810ВМ87, процессором ввода – вывода К1810ВМ89, с программируемыми контроллерами прерывания и прямого доступа к памяти, контроллерами портов

ввода/вывода, контроллерами периферийных устройств и т.д. (см. табл.1.1) [1].

Микросхема К1810ВМ86 представляет собой однокристальный 16-битовый МП, выполненный по n-МОП-технологии. Кристалл микросхемы с геометрическими размерами 5,5×5,5 мм содержит около 29000 транзисторов. Схема выпускается в 40-выводном корпусе, потребляет 1,7 Вт от источника питания +5 В и синхронизируется однофазными импульсами с частотой повторения 2 - 5 МГц от внешнего тактового генератора. Основные операции обработки данных (сложение, вычитание, логические действия) типа регистр — регистр выполняются МП за три такта, что обеспечивает быстродействие $1{,}66\times10^6$ оп/с при периоде тактовых импульсов 200 нс. С максимальным быстродействием (за два такта) выполняются регистровые пересылки, а также некоторые однооперандные команды (например, сдвиг на один разряд, инкремент, декремент, управление флагами).

Микропроцессор К1810ВМ86 (далее обозначен для краткости ВМ86) содержит 14 16-битовых внутренних регистров и образует 16-битовую шину данных для связи с внешней памятью и портами ввода/вывода. Шина адреса имеет 20 линий, что позволяет непосредственно адресоваться к памяти емкостью до 1 Мбайт = 2^{20} = 1 048 576 байт. Пространство памяти разделяется на сегменты по 64 Кбайт, причем в любой момент времени МП может обращаться к ячейкам четырех сегментов, которые программно выбраны в качестве текущих. Сегментация памяти обеспечивает удобный механизм вычисления физических адресов и способствует модульному проектированию программного обеспечения, что упрощает программирование и отладку.

Для сокращения необходимого числа выводов БИС младшие 16 адресных линий мультиплексированы во времени с линиями данных и составляют единую шину адреса/данных (ШАД). Четыре старшие адресные линии аналогично мультиплексированы с линиями состояния. Чтобы сигналы этих линий можно было использовать в системе, их обязательно разделяют с помощью внешних схем, т. е. осуществляют демультиплексирование шин.

При выполнении операций ввода/вывода используются 8- или 16-битовые адреса, так что кроме доступа к основной памяти МП может обращаться к портам (регистрам ввода/вывода), суммарная емкость памяти которых составляет 64 Кбайт. В МП ВМ86 реализована многоуровневая система прерываний по вектору с числом векторов до 256. Адреса (вектора) подпрограмм прерывания занимают область емкостью 1 Кбайт, которая располагается в памяти, начиная с младших адресов. Предусмотрена также организация прямого доступа к памяти, при котором МП прекращает работу и переводит в третье состояние свои шины адреса, данных и управления.

Микропроцессор ВМ86 появился как результат совершенствования МП ВМ80, и архитектура обоих процессоров имеет много общего.

1.2. Структура микропроцессора

Укрупненная структурная схема МП ВМ86 (рис. 1.1) содержит две относительно независимые части: **операционное устройство**, реализующее заданные командой операции, и **устройство шинного интерфейса**, осуществляющее выборку команд из памяти, а также обращение к памяти и внешним устройствам для считывания операндов и записи результатов. Оба устройства могут работать параллельно, что обеспечивает совмещение во времени процессов выборки и исполнения команд. Это повышает быстродействие МП, так как операционное устройство, как правило, выполняет команды, коды которых уже находятся в МП, и поэтому такты выборки команды не включаются в ее цикл.

1.2.1. Назначение выводов МП ВМ86

Назначение выводов МП ВМ86 зависит от режима его работы (минимальный или максимальный). В связи с этим восемь выводов микропроцессора имеет двойное обозначение, причем обозначения в скобках соответствуют максимальному режиму. В табл. 1.2

приведено назначение выводов МП, являющихся общими для обоих режимов, в табл. 1.3.— назначение выводов, используемых только в минимальном режиме, а в табл. 1.4 — используемых только в максимальном режиме. Буквой z отмечены трехстабильные выходы, которые переводятся в третье (высокоомное) состояние при переходе МП в режим состояния захвата.

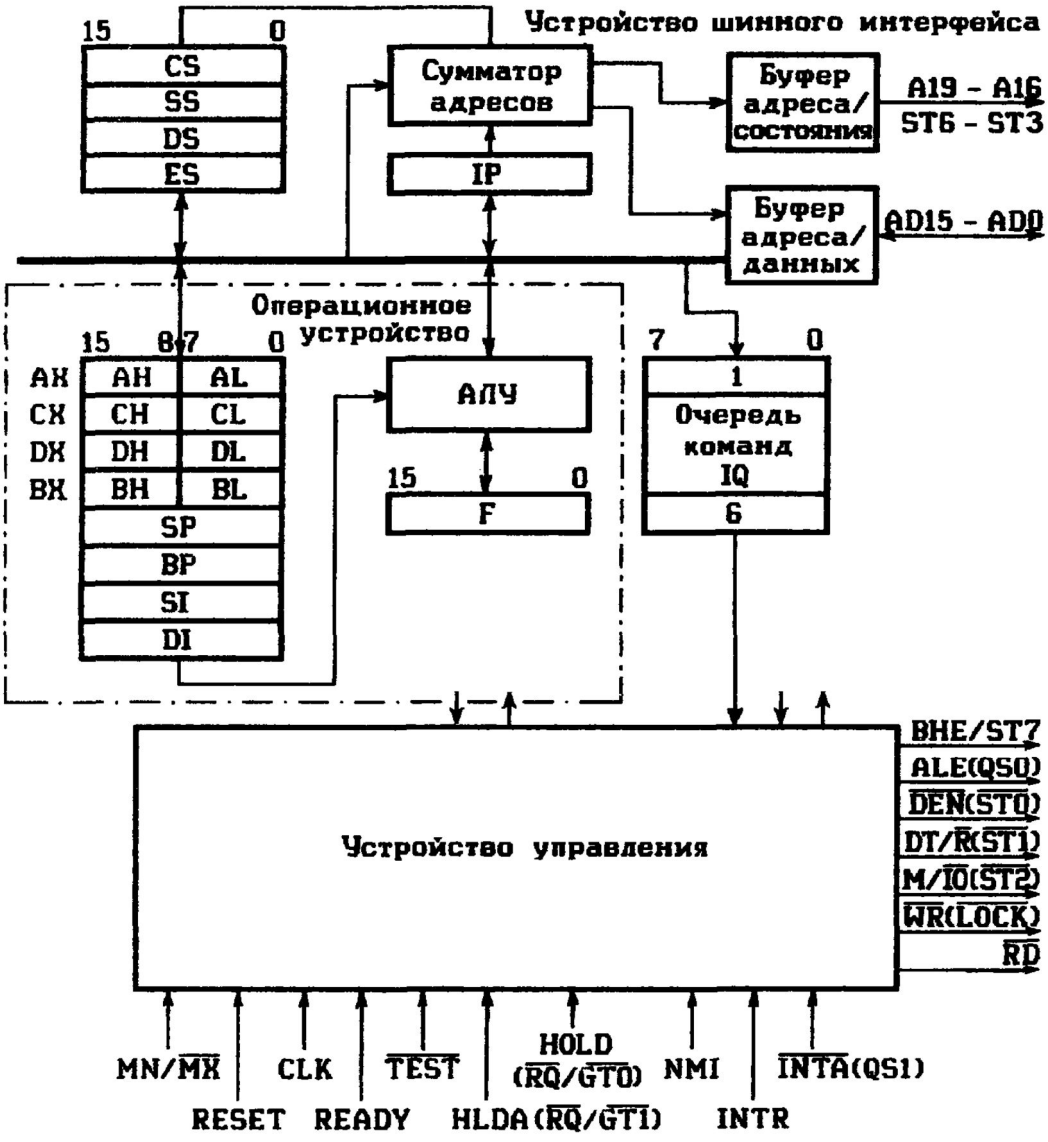

Рис. 1.1. Структурная схема МП ВМ86

Ниже более подробно, чем в таблицах рассмотрено функциональное назначение ряда сигналов микропроцессора. (Более подробную информацию можно найти в [1] и др.).

A19/S6 — A16/S3 — мультиплексные выходные линии адреса/состояния. В первом такте на эти линии выдаются старшие 4 бит адреса памяти, а при адресации ВУ — нули. В остальных тактах цикла шины МП выдает на эти линии сигналы состояния S6 — S3. Код на линиях S4, S3 определяет сегментный регистр, участвующий в формировании физического адреса памяти, т. е. указывает сегмент памяти, к которому производится обращение в текущем цикле (табл. 1.5). Следует отметить, что при обращении к ВУ, когда сегментные регистры не участвуют в формировании адреса, устанавливается значение S4 = 1, S3 = 0.

Сигналы S4, S3 могут использоваться для расширения адресного пространства системы. В этом случае отдельный банк памяти объемом 1 Мбайт выделяется каждому из четырех сегментов. К линиям S4, S3 подключается дешифратор, который выбирает

соответствующий банк памяти. Такой прием обеспечивает расширение адресной памяти до 4 Мбайт и защиту от ошибочной записи в сегмент, перекрывающийся с другими сегментами. Сигнал S5 соответствует состоянию флага разрешения прерываний IF: 0 — прерывания запрещены, 1 — прерывания разрешены. Сигнал S6 не используется и всегда равен нулю.

Таблица 1.2 Общие для обоих режимов сигналы МП

Обозначение	Назначение	Тип
AD15-AD0	Линии шины адреса/данных (ШАД)	Выход (z)
A16/S3 A17/S4 A18/S5 A19/S6	Линии адреса/состояния. В течение такта T1 содержат старшие биты адреса при обращении к памяти или ВУ, в течение T2, T3, TW и T4 - информацию о состоянии МП	Выход (z)
$\overline{BHE}$/S7	Разрешение старшего байта шины/состояние	Выход (z)
$\overline{RD}$	Чтение, строб, указывающий на то, что МП выполняет цикл чтения	Выход (z)
RDY	Готовность, подтверждение того, что адресованное устройство готово к взаимодействию с МП при передаче данных	Вход
INTR	Запрос прерывания, по которому МП переходит на подпрограмму обработки прерывания, если имеется разрешение	Вход
NMI	Немаскируемое прерывание, вызывает прерывание по фиксированному вектору (тип 2); не может быть запрещено внутренними средствами МП (программно)	Вход
$\overline{TEST}$	Входной сигнал, проверяемый командой WAIT, которая переводит МП в состояние ожидания, если $\overline{TEST}$ = 1	Вход
CLK,(CLC)	Тактовые импульсы, обеспечивающие синхронизацию работы МП	Вход
RESET (CLR)	Сброс, заставляет МП немедленно прекратить выполняемые действия и затем возобновить выполнение программы сначала	Вход
MN/$\overline{MX}$	Минимальный/максимальный, обеспечивает соответствующий режим работы МП	Вход

Таблица 1.3 Сигналы, используемые только в минимальном режиме

Обозначение	Назначение	Тип
$\overline{INTA}$	Подтверждение прерывания, стробирует чтение вектора (типа) прерывания	Выход
ALE (STB)	Разрешение регистра-защелки адреса, стробирует появление адресной информации в такте T1 на ШАД	Выход
$\overline{DEN}$ (DE)	Разрешение данных, стробирует появление данных на шине адреса/данных	Выход (z)
DT/$\overline{R}$ (OP/$\overline{IP}$)	Передача/прием данных, определяет направление пересылки данных по ШАД	Выход (z)
M/$\overline{IO}$	Обращение к ЗУ или ВУ в данном цикле шины	Выход (z)
$\overline{WR}$	Запись, строб, указывающий на то, что МП выполняет цикл записи	Выход (z)
HOLD	Запрос захвата, указывает на то, что некоторое устройство запрашивает шины МП	Вход
HLDA	Подтверждение захвата, указывает на то, что МП перевел свои шины адреса/данных, адреса/состояния и управления в z-состояние	Выход

— разрешение старшего байта. Формируется в первом такте цикла одновременно с адресной информацией. Активный сигнал нулевого уровня означает, что по старшей половине AD15 — AD8 шины адреса/данных передаются 8-битовые данные. Сигнал защелкивается во внешнем регистре адреса и используется как

Таблица 1.4. Сигналы, используемые только в максимальном режиме

Обозначение	Назначение	Тип
$\overline{S2}, \overline{S1}, \overline{S0}$ (ST2-ST0)	Линии состояния, характеризуют тип выполняемого цикла; необходимы для выработки управляющего сигнала	Выход (z)
$\overline{RQ}/\overline{GT0}$	Запрос/предоставление, используется для обмена сигналами между процессорами в многопроцессорной системе, для управления процедурой использования шин	Вход/выход
$\overline{RQ}/\overline{GT1}$		
$(\overline{RQ}/\overline{E0})$		
$(\overline{RQ}/\overline{E1})$		
$\overline{LOCK}$	Блокировка (занятость) шины, информирует другие процессоры и устройства о том, что они не должны запрашивать шину	Выход
QS1,QS0	Состояние очереди, указывает состояние внутренней 6-байтовой очереди команд МП	Выход

дополнительный адресный выход, определяющий доступ к старшему банку памяти либо к ВУ с байтовой организацией, подключенному к старшей половине шины AD. Совместное использование и младшей линии адреса A0 для дешифрации адресов позволяет осуществлять передачу слов или отдельных байтов по шине AD (табл. 1.6). Отметим, что после окончания сигнала на выход подается резервный сигнал состояния S7, не имеющий определенного значения.

Таблица 1.5

S4S3	Сегментный регистр
0 0	ES
0 1	SS
1 0	CS
1 1	DS

Таблица 1.6

$\overline{BHE}$	A0	Разрядность данных
0	0	Все слово (оба байта)
0	1	Старший байт D15-D8, нечетный адрес
1	0	Младший байт D7-D0, четный адрес
1	1	Нет обращения

— — сигналы состояния, обеспечивающие информацию о типе выполняемого цикла шины (табл. 1.7). Сигналы состояния подаются в контроллер шины, который дешифрует их и формирует расширенный набор управляющих сигналов. Если МП не инициирует цикл шины, то сигналы — устанавливаются в пассивное состояние 111. Отметим, что сигнал логически эквивалентен сигналу M/ , а —сигналу DT/ .

QS1 QS0 — состояние очереди. Идентифицирует состояние внутренней 6-байтовой очереди команд МП (табл. 1.8) и действует в течение такта синхронизации после выполнения операции над очередью. Сигналы QS1 — QS0 предназначены для сопроцессора, который воспринимает команды и операнды с помощью команды ESC. Сопроцессор

контролирует шину AD и фиксирует момент, когда из программной памяти выбирается предназначенная для него команда ESC, а затем следит за очередью команд и определяет момент, когда эта команда должна выполняться.

Таблица 1.7

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Тип цикла шины
0	0	0	Подтверждение прерывания
0	0	1	Чтение ВУ
0	1	0	Запись ВУ
0	1	1	Останов
1	0	0	Выборка команды
1	0	1	Чтение ЗУ
1	1	0	Запись ЗУ
1	1	1	Цикла шины нет

Таблица 1.8

QS1	QS0	Операция над очередью
0	0	Операции нет, в последнем такте не было выборки из очереди
0	1	Из очереди выбран первый байт команд
1	0	Очередь пуста, была опустошена командой передачи управления
1	1	Из очереди выбран следующий байт команды

1.2.2. Операционное устройство

Операционное устройство МП содержит группу *общих регистров*, *арифметико-логическое устройство (АЛУ)*, *регистр флагов F* и *блок (устройство) управления*.

Восемь 16-битовых ***регистров общего назначения*** участвуют во многих командах. В этих случаях регистры общего назначения кодируются трехбитовым кодом, который размещается в соответствующем поле (или полях) формата команды.

В соответствии с основным назначением рассматриваемых регистров выделяют регистры AX, BX, CX, DX, используемые, прежде всего для хранения данных, и регистры SP, BP, SI, DI, которые хранят главным образом адресную информацию. Особенностью регистров AX, BX, CX, DX является то, что они допускают раздельное использование их младших байтов AL, BL, CL, DL и старших байтов AH, BH, CH, DH. Тем самым обеспечивается возможность обработки, как слов, так и байтов и создаются необходимые условия для программной совместимости ВМ86 и ВМ80.

Все остальные регистры являются неделимыми и оперируют 16-битовыми словами, даже в случае использования только старшего или младшего байтов. Указательные регистры SP и BP хранят смещение адреса в пределах текущего стекового сегмента памяти, а индексные регистры SI и DI хранят смещение адреса соответственно в текущем сегменте данных и в текущем дополнительном сегменте. Однако при использовании этих регистров для адресации операндов возможна смена сегментов памяти (см. в табл. 1.10).

Кроме основных функций, соответствующих названию регистров, общие регистры выполняют специальные функции, указанные в табл. 1.9.

Арифметико-логическое устройство (АЛУ) содержит 16-битовый комбинационный сумматор, с помощью которого выполняются арифметические операции, наборы комбинационных схем для выполнения логических операций, схемы для операций сдвигов и десятичной коррекции, а также регистры для временного хранения операндов и результатов.

К АЛУ примыкает ***регистр флагов F*** (рис. 1.2, где × обозначает неопределенное состояние бита). Его младший байт FL полностью соответствует регистру флагов К580ВМ80, а старший байт FH содержит четыре флага, отсутствующие в К580ВМ80. Шесть *арифметических (статусных) флагов* фиксируют определенные признаки результата выполнения операции (арифметической, логической, сдвига или загрузки регистра). Значения этих флагов (кроме флага AF) используются для реализации условных переходов, изменяющих ход выполнения программы. Различные команды влияют на флаги по-разному.

Назначение арифметических (статусных) флагов.

CF — флаг переноса, фиксирует значение переноса (заема), возникающего при

сложении (вычитании) байтов или слов, а также значение выдвигаемого бита при сдвиге операнда.

PF — флаг четности (или паритета), фиксирует наличие четного числа единиц в младшем байте результата операции, может быть использован, например, для контроля правильности передачи данных.

Таблица 1.9. Специальные функции общих регистров

Регистр	Название	Специальная функция регистра
AX	Аккумулятор	Умножение, деление и ввод - вывод слов
AL	Аккумулятор (младший байт)	Умножение, деление и ввод - вывод байтов; преобразование байтов; десятичная арифметика
AH	Аккумулятор (старший байт)	Умножение и деление слов
BX	Базовый регистр	Адресация по базе; преобразование адресов
CX	Счетчик	Подсчет циклов; подсчет элементов цепочек
CL	Счетчик (младший байт)	Реализация параметрических сдвигов
DX	Регистр данных	Умножение и деление слов; косвенный ввод - вывод
SP	Указатель стека	Операции с использованием стека
BP	Указатель базы	Базовый регистр
SI	Индекс источника	Указатель цепочки-источника, индексный регистр
DI	Индекс приемника	Указатель цепочки-приемника, индексный регистр

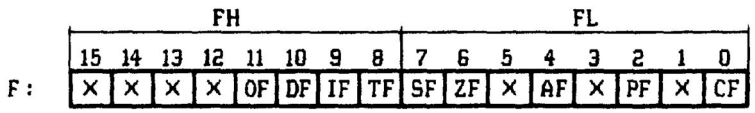

Рис. 1.2. Формат регистра флагов F

AF — флаг вспомогательного переноса, фиксирует перенос (заем) из младшей тетрады, т. е. из бита a_3, в старшую при сложении (вычитании), используется только для двоично-десятичной арифметики, которая оперирует исключительно младшими байтами.

ZF — флаг нуля, сигнализирует о получении нулевого результата операции.

SF — флаг знака, дублирует значение старшего бита результата, который при использовании дополнительного кода соответствует знаку числа.

OF — флаг переполнения, сигнализирует о потере старшего бита результата сложения или вычитания в связи с переполнением разрядной сетки при работе со знаковыми числами. При сложении этот флаг устанавливается в единицу, если происходит перенос в старший бит и нет переноса из старшего бита или имеется перенос из старшего бита, но отсутствует перенос в него; в противном случае флаг OF устанавливается в нуль. При вычитании он устанавливается в единицу, когда возникает заем из старшего бита, но заем в старший бит отсутствует либо имеется заем в старший бит, но отсутствует заем из него. Имеется специальная команда прерывания при переполнении, которая в указанных случаях генерирует программное прерывание.

Для управления некоторыми действиями МП предназначены *три дополнительных (управляющих) флага.*

DF — флаг направления, управляемый командами CLD и STD; определяет порядок обработки цепочек в соответствующих командах: от меньших адресов (DF = 0) или от больших (DF = 1).

IF — флаг разрешения прерываний, управляемый с помощью команд CLI и STI; при IF = 1 микропроцессор воспринимает (распознает) и соответственно реагирует на запрос прерывания по входу INTR; при IF = 0 прерывания по этому входу запрещаются (маскируются) и МП игнорирует поступающие запросы прерываний. Значение флага IF не влияет на восприятие внешних немаскируемых прерываний по входу NMI, а также внутренних (программных) прерываний, выполняемых по команде INT.

TF — флаг трассировки (прослеживания). При TF = 1 МП переходит в покомандный (пошаговый) режим работы, применяемый при отладке программ, когда автоматически генерируется сигнал внутреннего прерывания типа 1 (см. рис. 1.6, 1.7) после выполнения каждой команды с целью перехода к соответствующей подпрограмме, которая обычно обеспечивает индикацию содержимого внутренних регистров МП. Команды установки или сброса флага TF отсутствуют, так что управление этим флагом осуществляется опосредованно, путем пересылки содержимого регистра флагов F через стек в общий регистр, установки требуемого значения восьмого бита и обратной пересылки сформированного слова в регистр F.

Устройство управления (УУ) дешифрует команды, а также воспринимает и вырабатывает необходимые управляющие сигналы. В его состав входит блок микропрограммного управления, в котором на микрокомандном уровне реализовано управление выполнением некоторых функций операционным устройством и устройством шинного интерфейса

1.2.3. Устройство шинного интерфейса

Устройство шинного интерфейса (или просто шинный интерфейс) содержит блок *сегментных регистров*, *указатель команд*, *сумматор адресов*, *очередь команд* и *буферы*, обеспечивающие связь с шиной. Шинный интерфейс выполняет операции обмена между МП и памятью или портами ввода/вывода по запросам операционного устройства. Когда операционное устройство занято выполнением команды, шинный интерфейс самостоятельно инициирует опережающую выборку кодов очередных команд из памяти.

Очередь команд представляет собой набор байтовых регистров и выполняет роль регистра команд, в котором хранятся коды, выбранные из программной памяти. Длина очереди составляет 6 байт, что соответствует максимально длинному формату команд. Наличие очереди команд, а также способность операционного устройства и шинного интерфейса работать параллельно позволяют совместить во времени фазы выборки команды и выполнения заданной операции: пока одна команда исполняется в операционном устройстве, шинный интерфейс осуществляет выборку следующей команды. Таким образом достигаются высокая плотность загрузки шины и повышение скорости выполнения программы. Пример, иллюстрирующий реализацию описанного конвейерного принципа, дан на рис. 1.3, где TI обозначает холостые такты работы шины, когда очередь команд заполнена, а операционное устройство занято выполнением текущей команды и не запрашивает выполнения цикла шины.

Рис. 1.3. Пример конвейерного выполнения команд

Шинный интерфейс инициирует выборку следующего командного слова автоматически, как только в очереди освободятся два байта. Как правило, в очереди находится минимум один байт потока команд, так что операционное устройство не ожидает выборки команды. *Ясно, что опережающая выборка команд позволяет экономить время только*

при естественном порядке выполнения команд. Когда операционное устройство выполняет команду передачи управления (перехода) в программе, шинный интерфейс сбрасывает очередь, выбирает команду по новому адресу, передает ее в операционное устройство, а затем начинает заполнение (реинициализацию) очереди из следующих ячеек памяти. Эти действия предпринимаются в условных и безусловных переходах, вызовах подпрограмм, возвратах из подпрограмм и при обработке прерываний.

По мере необходимости операционное устройство считывает байт из очереди и выполняет предписанную командой операцию. При многобайтовых командах из очереди считываются и другие байты команды. В тех редких случаях, когда к моменту считывания очередь оказывается пустой, операционное устройство ожидает выборку очередного командного слова, которую инициирует шинный интерфейс. Если команда требует обращения к памяти или порту ввода/вывода, операционное устройство запрашивает шинный интерфейс на выполнение необходимого цикла шины для передачи данных. Когда шинный интерфейс не занят выборкой команды, он удовлетворяет запрос немедленно; в противном случае операционное устройство ожидает завершения текущего цикла шины. Со своей стороны, шинный интерфейс приостанавливает выборку команд во время обмена данными между операционным устройством и памятью или портами ввода/вывода.

Буфер шины адреса/данных (БАД) содержит 16 двунаправленных управляемых усилителей с тремя выходными состояниями и обеспечивает номинальную нагрузочную способность линий AD15 — AD0.

Буфер шины адреса/состояния (БАС) содержит четыре однонаправленных усилителя с тремя выходными состояниями и обеспечивает номинальную нагрузочную способность линий A19/S6 — A16/S3.

Сегментные регистры хранят базовые (начальные) адреса сегментов памяти: кодового сегмента CS, в котором содержится программа; стекового сегмента SS; сегмента данных DS; дополнительного сегмента ES, в котором обычно содержатся данные. Наличие сегментных регистров обусловлено разделением памяти на сегменты и используемым способом формирования адресов памяти. Хотя МП имеет 20-битовую шину физического адреса памяти, он оперирует 16-битовыми логическими адресами, состоящими из базового адреса сегмента и внутрисегментного смещения. Внутрисегментное смещение может быть вычислено в соответствии с указанным в команде способом адресации, может находиться в формате команды или содержаться в общем регистре. Физический адрес формируется путем суммирования смещения и содержимого соответствующего сегментного регистра, которое дополняется четырьмя нулевыми младшими разрядами.

Сумматор адресов осуществляет вычисление 20-битовых физических адресов.

Указатель команд IP хранит смещение следующей команды в текущем кодовом сегменте, т. е. указывает на следующую по порядку команду. Он является аналогом стандартного программного счетчика с той лишь разницей, что его содержимое определяет адрес команды лишь в совокупности с содержимым регистра CS; если же CS заполнен нулями, аналогия становится полной. Модификация IP осуществляется шинным интерфейсом так, что при обычной работе IP содержит смещение того командного слова, которое шинный интерфейс будет выбирать из памяти. Оно не совпадает со смещением очередной команды (находящейся в этот момент на выходе очереди команд), которую будет выполнять операционное устройство. Поэтому при запоминании содержимого IP в стеке, например при вызове подпрограмм, оно автоматически корректируется, чтобы адресовать следующую команду, которая будет выполняться. Эта особенность является следствием опережающей выборки команд, реализованной в BM86. Непосредственный доступ к IP имеют команды передачи управления.

Контрольные вопросы

1. Охарактеризуйте состав микропроцессорного комплекта К1810.

2. Чем отличаются минимальный и максимальный режимы работы микропроцессора К1810ВМ86?
3. Чем отличается МП ВМ88 от МП ВМ86?
4. Для чего используется сопроцессор ВМ87?
5. Охарактеризуйте микросхему К1810ВМ86.
6. Кратко опишите основные характеристики МП ВМ86.
7. Какая система прерываний поддерживается МП ВМ86?
8. Какие части содержит структура МП?
9. Охарактеризуйте общие для обоих режимов сигналы МП.
10. Охарактеризуйте сигналы, используемые только в минимальном режиме.
11. Охарактеризуйте сигналы, используемые только в максимальном режиме.
12. На что указывают комбинации сигналов S4,S3 и $\overline{BHE}, A0$?
13. На что указывают комбинации сигналов $\overline{S2}$ — $\overline{S0}$ и QS1, QS0?
14. Что входит в остав операционного устройства?
15. Охарактеризуйте регистры общего назначения.
16. Охарактеризуйте специальные функции общих регистров
17. Охарактеризуйте состав арифметико-логического устройства.
18. Охарактеризуйте формат регистра флагов.
19. Каково назначение арифметических (статусных) флагов?
20. Каково назначение управляющих флагов?
21. Что входит в состав устройства управления?
22. Каков состав и назначение компонентов устройства шинного интерфейса?
23. Как действует очередь команд?
24. Когда шинный интерфейс сбрасывает очередь команд?
25. Как осуществляется синхронизация параллельной работы операционного устройства и устройства шинного интерфейса?
26. Каково назначение БАД и БАС?
27. Каково назначение сегментных регистров?
28. Чем указатель команд IP отличается от счетчика команд?

1.3. Адресное пространство памяти и ввода - вывода

1.3.1.Размещение байтов и слов в памяти.

Память логически организована как одномерный массив байтов, каждый из которых имеет 20-битовый физический адрес в диапазоне 00000 — FFFFF. (Для записи адресов здесь и далее используется 16-ричная система счисления). Любые два смежных байта в памяти могут рассматриваться как 16-битовое слово. Младший байт слова имеет меньший адрес, а старший — больший. Такое размещение байтов слова используется также в ВМ80 и в большинстве микро ЭВМ и персональных компьютеров. Адресом слова считается адрес его младшего байта. Таким образом, 20-битовый адрес памяти может рассматриваться и как адрес байта, и как адрес слова.

Полная информация, необходимая для определения физического адреса, содержится в адресном объекте «***сегмент:смещение***», который называется *указателем адреса* и содержит адрес сегмента и внутрисегментное смещение. Для запоминания указателя адреса требуется два слова памяти, причем слово с меньшим адресом всегда содержит смещение, а слово с большим адресом — базовый адрес сегмента (рис. 1.4). Каждое слово хранится обычным образом, т. е. по принципу «младший байт — по меньшему адресу».

Команды, байты и слова данных можно свободно размещать по любому адресу, что позволяет экономить память благодаря ее плотной упаковке. Однако для экономии времени выполнения программы целесообразно размещать слова данных в памяти по **четным адресам**, так как МП передает такие слова за один цикл шины. Слово с четным адресом называется выравненным на границе слов. Слова с нечетными адресами (невыравненные)

также допустимы, но для их передачи требуются два цикла шины, что снижает производительность МП. (Каждый цикл имеет четыре обязательных такта Т.) Отметим, что шинный интерфейс инициирует необходимое для выборки слова число обращений к

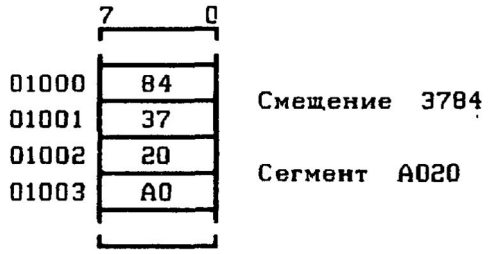

Рис. 1.4. Размещение слов в памяти

памяти автоматически, так что двукратное обращение к памяти не требует специального указания в программе. Особенно важно иметь выравненные слова для операций со стеком, так как в них участвуют только слова. Следовательно, ***указатель стека SP необходимо всегда инициализировать на четный адрес.***

Команды всегда выбираются словами по четным адресам, за исключением первой выборки после передачи управления по нечетному адресу, когда выбирается один байт. Поток команд разделяется на байты при заполнении очереди команд внутри МП, так что выравнивание команд не влияет на производительность и поэтому не используется.

1.3.2. Сегментация памяти и вычисление адресов.

Пространство памяти емкостью 1 Мбайт представляется как набор сегментов, определяемых программным путем. Сегмент состоит из смежных ячеек памяти и является независимой и отдельно адресуемой единицей памяти емкостью 64 Кбайт. Каждому сегменту программой назначается начальный (базовый) адрес, являющийся адресом первого байта сегмента в пространстве памяти. Начальные адреса четырех сегментов, выбранных в качестве текущих, записываются в сегментные регистры CS, DS, SS и ES, тем самым фиксируются текущие сегменты кода (программы), данных, стека и дополнительных данных. Для обращения к командам и данным, находящимся в других сегментах, необходимо изменять содержимое сегментных регистров, что позволяет использовать все пространство памяти емкостью 1 Мбайт. Сегментные регистры инициализируются в начале программы путем засылки в них соответствующих констант. *Частный случай загрузки всех сегментных регистров нулями приводит к организации памяти, характерной для ВМ80, т. е. фактически к отказу от сегментации памяти.*

В сегментном регистре хранится 16 старших битов 20-битового начального адреса сегмента. Четыре младших бита адреса принимаются равными нулю и дописываются справа к содержимому сегментного регистра при вычислении физических адресов ячеек памяти. Поэтому начальные адреса сегментов всегда кратны 16. Поскольку других ограничений на размещение сегментов в памяти нет, сегменты могут быть соседними (смежными), неперекрывающимися, частично или полностью перекрывающимися. ***Физическая ячейка памяти может принадлежать одному или нескольким сегментам.***

Физический адрес ячейки памяти представляет 20-битовое число в диапазоне 0 — FFFFF, которое однозначно определяет положение каждого байта в пространстве памяти емкостью 1 Мбайт. В начале каждого цикла шины, связанного с обращением к памяти, физический адрес выдается на шину адреса и сопровождается сигналом ALE. Так как МП ВМ86 является 16-битовым, то все операции при вычислении физического адреса производятся с 16-битовыми адресными объектами.

Логический адрес ячейки памяти состоит из двух 16-битовых беззнаковых значений: начального адреса сегмента, который называется также просто базой или сегментом, и внутрисегментного смещения, которое определяет расстояние от начала сегмента до этой ячейки. Для вычисления физического адреса база сегмента сдвигается влево на 4 бит и суммируется со смещением, как показано на рис. 1.5, где также приведены возможные

источники компонентов логического адреса (EA — ***эффективный адрес***, вычисляемый в соответствие с заданным способом адресации).

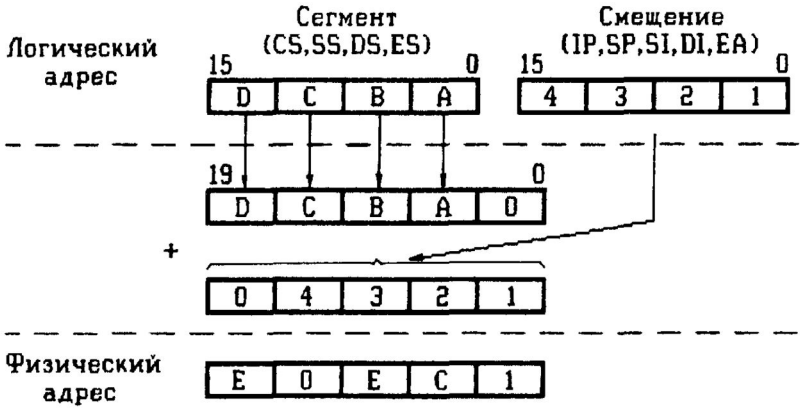

Рис. 1.5. Вычисление физического адреса

Перенос из старшего бита, который может возникнуть при суммировании, игнорируется. Это приводит к так называемой ***кольцевой организации памяти***, при которой за ячейкой с максимальным адресом FFFFF следует ячейка с нулевым адресом. ***Аналогичную кольцевую организацию имеет и каждый сегмент.***

Источники логического адреса для различных типов обращения к памяти приведены в табл. 1.10.

Таблица 1.10

Тип обращения к памяти	Сегмент (по умолчанию)	Вариант	Смещение
Выборка команды	CS	Нет	IP
Стековая операция	SS	Нет	SP
Переменная	DS	CS,SS,ES	EA
Цепочка-источник	DS	CS,SS,ES	SI
Цепочка-приемник	ES	Нет	DI
BP как базовый регистр	ES	CS,SS,DS	EA

Команды всегда выбираются из текущего сегмента кода в соответствии с логическим адресом CS:IP**.** Стековые команды всегда обращаются к текущему сегменту стека по адресу SS:SP. Если при вычислении адреса EA используется регистр BP, то обращение производится также к стековому сегменту. ***В последнем случае принцип стека «первый пришел — последний вышел» игнорируется и ячейки стекового сегмента рассматриваются как ОЗУ с произвольной выборкой,*** что обеспечивает большую гибкость в использовании этих ячеек.

Операнды, как правило, размещаются в текущем сегменте данных, и обращение к ним организуется по адресу DS:EA. Однако программист может заставить МП обратиться к переменной, находящейся в другом текущем сегменте***. При цепочных операциях считается, что цепочка-источник находится в текущем сегменте данных, а ее смещение задается регистром SI. Цепочка-получатель обязательно располагается в текущем дополнительном сегменте ES, а смещение берется из регистра DI. Команды обработки цепочек автоматически модифицируют содержимое индексных регистров SI и DI по мере продвижения по цепочке в направлении, соответствующем флагу DF.***

Смена сегментного регистра в соответствии с вариантами, указанными в табл. 1.10, осуществляется с помощью однобайтового префикса замены сегмента **001SR110**, который ставится перед первым байтом команды. Двухбитовое поле SR содержит код сегментного регистра, используемого для вычисления физического адреса в данной команде: 00 —

регистр ES; 01 — CS, 10 — SS; 11 — DS. В мнемонических обозначениях команд смена сегмента отражается следующим образом: MOV AX, CS:[BX] — пересылка в AX слова из кодового сегмента со смещением из BX; ADD ES:[ROW], BL — сложение байта из дополнительного сегмента (со смещением + ROW) с содержимым регистра BL и размещение результата в ОЗУ на место первого слагаемого.

Сегментная структура памяти обеспечивает возможность создания позиционно независимых или динамически перемещаемых программ, что необходимо в мультипрограммной среде для эффективного использования оперативной памяти. Чтобы обеспечить позиционную независимость, все смещения в программе должны задаваться относительно фиксированных значений, содержащихся в сегментных регистрах. Это позволяет произвольно перемещать программу в адресном пространстве памяти, изменяя только содержимое сегментных регистров.

Стек, как обычно, организуется в ОЗУ, и его положение определяется содержимым регистров SS и SP. Регистр SS хранит базовый адрес текущего сегмента стека, а регистр SP указывает на вершину стека, т. е. содержит смещение вершины стека в стековом сегменте. При каждом обращении к стеку пересылается одно слово, причем содержимое SP модифицируется автоматически: при записи (включении) в стек оно уменьшается на два, при чтении (извлечении) из стека — увеличивается на два.

При всех достоинствах принятой в BM86 организации памяти она имеет некоторый недостаток, заключающийся в трудности манипуляции физическими адресами при необходимости их программной обработки.

1.3.3. Организация ввода — вывода.

Ввод и вывод данных может осуществляться двумя способами: с использованием адресного пространства ввода вывода, и с использованием общего с памятью адресного пространства, т. е. с отображением на память.

При **первом способе** применяются специальные команды IN (ввод) и OUT (вывод), которые обеспечивают передачу данных между аккумуляторами AL или AX и адресуемыми портами. При выполнении этих команд вырабатывается сигнал $M/\overline{IO} = 0$, который идентифицирует выбор пространства ввода — вывода и в совокупности с сигналами $\overline{WR}$ и $\overline{RD}$ позволяет сформировать системные сигналы IOW и IOR для управления операциями записи данных в порт и чтения из порта.

Команды IN и OUT могут использовать прямую адресацию (по аналогии с одноименными командами BM80), когда адрес порта содержится в виде константы во втором байте команды, и косвенную адресацию, когда адрес располагается в регистре DX. В первом случае можно адресовать по 256 портов для ввода и вывода данных. Во втором обеспечивается адресное пространство до 64К 8-битовых портов или до 32К 16-битовых портов. Косвенная адресация позволяет вычислять адреса портов при выполнении программы и удобна при организации вычислительных циклов для обслуживания нескольких портов с помощью одной процедуры.

Восемь ячеек F8 — FF в пространстве ввода — вывода зарезервированы для системных целей, и использовать их в прикладных программах не рекомендуется.

При **втором способе** адреса портов размещаются в общем адресном пространстве, и обращение к ним не отличается от обращения к ячейкам памяти. Это повышает гибкость программирования, так как для ввода — вывода можно использовать любую команду с обращением к памяти при любом способе адресации. Так, команда MOV позволяет передавать данные между любым общим регистром или ячейкой памяти и портом ввода — вывода, а логические команды AND, OR, XOR и TEST позволяют манипулировать битами в регистре порта. При этом, однако, следует учитывать, что команды с обращением к памяти имеют больший формат и выполняются дольше, чем простые команды IN и OUT. Кроме того, несколько **усложняется дешифрирование 20-битового физического адреса порта** и сокращается число адресов, которые могут использоваться для ячеек памяти.

Микропроцессор может передавать по шине байт или слово в/из ВУ. Чтобы слово передавалось за один цикл шины, адрес ВУ должен быть четным. Адрес байтового ВУ может быть четным или нечетным, и соответственно порты этих внешних устройств подключаются к линиям младшего и старшего байта шины данных. Для раздельного обращения к этим портам дешифрирование адресов осуществляется с учетом сигналов на линиях $\overline{\text{BHE}}$ и A0.

Контрольные вопросы.

1. Как размещаются в памяти байты и слова?
2. Что понимается под указателем памяти?
3. Что понимается под выравненным на границе словом?
4. Почему указатель стека SP необходимо всегда инициализировать на четный адрес?
5. Почему выравнивание команд не влияет на производительность МП?
6. Как производится сегментация памяти?
7. Почему начальные адреса сегментов всегда кратны 16?
8. Может ли физическая ячейка памяти принадлежать нескольким сегментам?
9. Что понимается под физическим адресом?
10. Что понимается под логическим адресом?
11. Как вычисляется физический адрес на основании логического адреса?
12. Что понимается под эффективным адресом?
13. Что понимается под кольцевой организацией памяти и сегмента?
14. Когда к стековой памяти обращаются как к стеку, а когда как к ОЗУ с произвольной выборкой?
15. Каковы особенности размещения в памяти операндов и цепочек данных?
16. Как и где используется префикс замены сегмента?
17. Как сегментная структура памяти обеспечивает создание позиционно независимых или динамически перемещаемых программ?
18. Как организуется стек в ОЗУ?
19. Как организуется ввод/вывод с использованием команд IN и OUT?
20. Как организуется ввод/вывод с использованием адресного пространства памяти?

1.4. Организация прерываний

1.4.1. Общие сведения о системе прерываний.

Микропроцессор ВМ86 имеет эффективную систему прерываний, в которой каждому прерыванию поставлен в соответствие код (от 0 до 255), который идентифицирует тип (номер вектора) прерывания. Прерывания могут инициироваться внешними устройствами (внешние прерывания) или командами программных прерываний, а в некоторых ситуациях — автоматически самим МП (внутренние прерывания). Возможные источники прерываний показаны на рис. 1.6.

Прерывание заставляет МП временно прекратить выполнение текущей программы и перейти к выполнению подпрограммы обработки прерывания, которая считается более важной или срочной. Возобновление прерванной программы должно быть произведено так, будто прерывание отсутствовало. Для этого в стеке запоминается адрес возврата (CS и IP) и содержимое регистра флагов F, а также содержимое тех регистров, которые потребуются для выполнения подпрограммы обработки прерывания. Содержимое регистров CS, IP и F запоминается и восстанавливается автоматически, а для запоминания и последующего восстановления содержимого других регистров МП должны быть предусмотрены соответствующие команды в начале и конце подпрограммы обработки прерываний. Следует отметить, что в стек включается скорректированное содержимое указателя команд IP, соответствующее адресу команды, перед которой МП начал обслуживать прерывание.

Необходимость коррекции вызвана тем, что IP адресует команды с опережением из-за существования внутренней очереди команд.

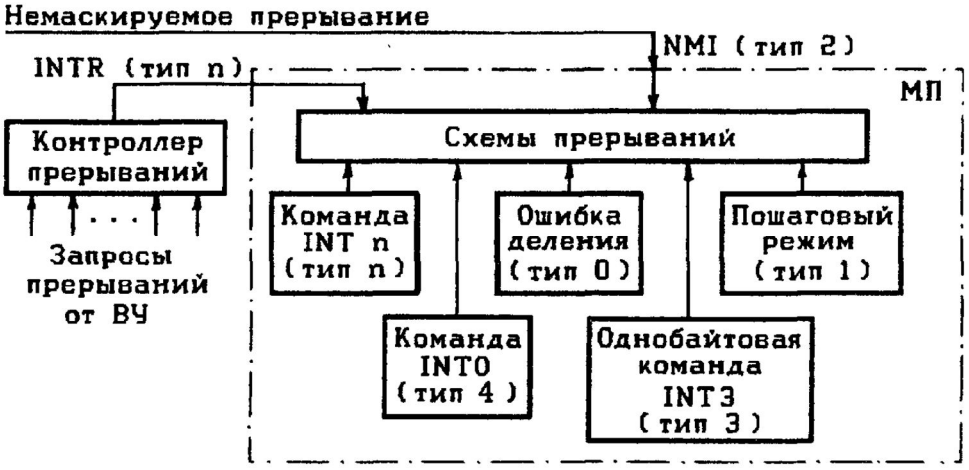

Рис. 1.6. Источники прерываний

1.4.2. Внешние прерывания.

Запросы на внешние прерывания поступают в МП по двум входам - INTR и NMI, а сами прерывания делятся соответственно на ***маскируемые*** и ***немаскируемые***. Запросы на *маскируемые прерывания* от ВУ обычно поступают на входы программируемого контроллера прерываний (ПКП) К1810ВН59А, который формирует сигнал, подаваемый на вход INTR микропроцессора. Отметим, что этот ПКП может использоваться как с МП ВМ86, так и с ВМ80, причем его работа существенно зависит от типа МП. При работе с ВМ80 контроллер в ответ на первый сигнал подтверждения прерывания выставляет на шину данных код команды CALL. В МП этот код инициирует еще два сигнала: и , по которым ПКП выдает два байта адреса подпрограммы. При работе с ВМ86 в ответ на сигнал ПКП не выдает данных в микропроцессор и буфер данных ПКП остается в высокоомном состоянии. По сигналу ПКП посылает в микропроцессор байт, определяющий тин (номер вектора) прерывания.

Когда устанавливается сигнал INTR = 1, действия МП зависят от состояния флага IF разрешения прерываний. Однако до завершения текущей команды МП, как правило, не предпринимает никаких действий.

Имеется несколько случаев, когда сигнал INTR распознается только при завершении следующей команды. Префиксы повторения, блокировки шины и замены сегмента считаются частью команды, поэтому прерывание между префиксом и командой не воспринимается. Команды пересылки в сегментный регистр MOV sr, src и извлечения из стека в сегментный регистр POP sr рассматриваются аналогично: прерывание не распознается до завершения следующей за ними команды. Это необходимо для правильной смены сегмента, когда осуществляется перезагрузка сегментного регистра и регистра, определяющего смещение в сегменте (например, регистров SS и SP).

Имеются два особых случая, когда запрос прерывания распознается во время выполнения команды, относящихся к цепочечной команде с повторением и к команде WAIT, которые могут выполняться в течение значительного времени. В этих случаях прерывания воспринимаются после любой законченной цепочечной операции (т. е. после очередной операции с элементом цепочки) или после цикла проверки сигнала на входе (каждый цикл проверки занимает время 5Т).

Если IF = 0, т. е. прерывания по входу INTR запрещены (замаскированы), МП игнорирует запрос прерывания и переходит к следующей команде. Микропроцессор не запоминает состояние сигнала INTR, поэтому этот сигнал должен оставаться активным,

пока прерывающее ВУ не получит сигнала подтверждения или само не снимет запрос. Если IF = 1, то МП распознает запрос прерывания и обрабатывает его. Состоянием флага IF программист может управлять с помощью команд STI (установка) и CLI (сброс). Кроме того, ПКП может осуществлять селективное маскирование запросов прерывания от отдельных устройств, если в контроллер послан соответствующий приказ.

Микропроцессор ВМ86 подтверждает запрос прерывания, выполняя два последовательных цикла . Если в этих циклах появляется запрос шипы по линии HOLD (в минимальном режиме) или = (в максимальном режиме), то он не воспринимается до завершения обоих циклов . В максимальном режиме МП генерирует в этих циклах сигнал блокировки шины , чтобы другие процессоры не пытались запрашивать шипу.

Запросы *на немаскируемое прерывание* поступают по входу NMI и обычно используются для прерывания работы МП при «катастрофических» событиях, требующих немедленной реакции, таких, как аварийное пропадание питания, обнаружение ошибки памяти и т. д. Вход NMI воспринимает переход сигнала от низкого уровня к высокому (положительный фронт), чтобы текущая программа не прерывалась от одного сигнала NMI = 1 несколько раз. Запросы NMI запоминаются в МП и имеют более высокий приоритет, чем прерывания по входу INTR. Обработка немаскируемого прерывания не зависит от состояния флага IF. Немаскируемому прерыванию присваивается фиксированный код типа 2, который автоматически формируется внутри МП. Поэтому в ответ на NMI циклы шины подтверждения прерывания не формируются, что ускоряет реакцию МП на запросы немаскируемых прерываний.

Внешнее прерывание может появиться в произвольный момент времени, т. е. асинхронно по отношению к действиям МП. Время реакции МП, определяющее запаздывание обслуживания прерывания, зависит от времени завершения текущей команды. Наибольшее запаздывание может произойти при выполнении команд умножения, деления и параметрического сдвига на много битов. Основные параметры различных видов прерываний даны в табл. 1.11, где прерывания перечислены в порядке убывания их приоритетов.

При анализе приоритетов необходимо учитывать маскируемость внешних прерываний по входу INTR, что может привести к перераспределению приоритетов. Если, например, одновременно возникают немаскируемое и маскируемое прерывания, МП начинает выполнение немаскируемого прерывания как имеющего высший приоритет и маскирует внешние прерывания сбросом флага IF. В этом случае запрос по входу INTR принимается лишь по окончании обслуживания немаскируемого прерывания. Однако обслуживание запроса маскируемого прерывания может быть разрешено при выполнении подпрограммы обслуживания любого прерывания путем установки флага IF.

Таблица 1.11. Параметры различных видов прерываний

Вид прерывания	Тип прерывания	Приоритет	Время вызова подпрограммы прерывания
По ошибке деления	0	1	50
По команде INT n	5 - 31	1	51
По команде INT0	4	1	53
По команде INT 3	3	1	52
По входу NMI	2	2	50
По входу INTR	32 - 255	3	61
По флагу TF	1	4	50

1.4.3. Внутренние прерывания

Внутренние прерывания характеризуются типом прерывания, который либо предопределен, либо содержится в коде команды, а также тем, что циклы шины подтверждения прерывания INTA не формируются и внутренние прерывания не могут быть запрещены (кроме пошагового прерывания).

Прерывание по ошибке деления (тип 0) генерируется микропроцессором сразу после выполнения команд деления DIV и IDIV, если формат частного превышает формат получателя или в случае деления на нуль.

Прерывание по переполнению (тип 4) генерируется по однобайтовой команде INT0, если установлен флаг OF.

Пошаговое прерывание (тип 1) вырабатывается автоматически при TF = 1 после выполнения каждой команды или пары команд, если первая команда изменяет содержимое сегментного регистра. Обычно это прерывание используется в программах отладки для реализации покомандного выполнения программы. При обработке прерывания МП включает в стек регистры F, CS и IP, а затем сбрасывает флаги IF и TF. Поэтому после вызова подпрограммы МП работает обычным образом, а не в пошаговом режиме. Подпрограмма обработки пошагового прерывания обычно осуществляет индикацию внутренних регистров МП и содержимого некоторых ячеек памяти. Когда подпрограмма завершается, из стека извлекаются прежние состояния флагов и МП снова переводится в пошаговый режим работы.

Как уже отмечалось, МП ВМ86 не имеет команд установки и сброса флага TF. Отсутствуют также команды, которые позволили бы организовать пересылки между старшим байтом регистра F и общим регистром МП. Состояние флага TF можно изменять, воздействуя на него после включения регистра F в стек. Для включения регистра F в стек и извлечения его из стека предусмотрены соответственно команды PUSHF и POPF. Значение TF = 1 устанавливается путем объединения по ИЛИ содержимого регистра с константой 0100, а TF = 0 — путем объединения по И с константой FEFF. Если установлено TF = 1, то первое пошаговое прерывание произойдет после выполнения команды, следующей за командой возврата из подпрограммы обработки пошагового прерывания.

В пошаговом режиме МП реагирует па внешние и внутренние прерывания. Обычным путем (с включением в стек регистров CS, IP и F) осуществляется переход на подпрограмму обработки возникшего прерывания. Однако до выполнения первой команды этой подпрограммы распознается пошаговое прерывание и управление передается подпрограмме обработки пошагового прерывания типа 1, после завершения которой МП возвращается к выполнению подпрограммы принятого ранее прерывания.

1.4.4. Прерывание, определяемое пользователем

Прерывание, определяемое пользователем при составлении программы, осуществляется по двухбайтовой команде INT n, в которой тип прерывания указывается во втором байте команды. Команда INT n вызывает требуемую подпрограмму, как и команда CALL, однако при переходе на подпрограмму команда INT n осуществляет запоминание не только адреса возврата (CS и IP), но и регистра флагов F. При этом выполняется межсегментный переход, причем адрес подпрограммы располагается не в формате команды или в произвольной ячейке памяти, а в специально сформированной таблице (таблица векторов прерываний).

Процедуры программных прерываний удобно применять в системах, где допускается динамическое перемещение программ (процедур) при их выполнении. Поскольку таблица указателей находится в фиксированной области памяти, процедуры могут вызывать друг друга через эту таблицу. Этим обеспечивается устойчивое взаимодействие, не зависящее от адресов процедур, если перемещение процедуры в памяти сопровождается соответствующей модификацией таблицы указателей.

Однобайтовая команда INT3 вызывает прерывание типа 3, которое определено как прерывание контрольной точки (точки разрыва). Контрольной точкой может быть любое место в программе, где нормальное ее выполнение прерывается, и производятся некоторые специальные действия. Контрольные точки обычно вводятся при отладке как средство индикации содержимого регистров, ячеек памяти и портов ввода в критических местах программы. Эту команду можно также использовать, чтобы вставить дополнительный фрагмент программы без ее повторной трансляции.

1.4.5. Процедура обслуживания прерываний.

Связь между кодом, определяющим тип прерывания, и подпрограммой (процедурой) обслуживания прерывания устанавливается с помощью таблицы указателей векторов прерываний (рис. 1.7). Полная таблица занимает 1 Кбайт памяти и содержит 256 элементов, расположенных по адресам 0 — 3FF. Каждый элемент n таблицы содержит два слова, определяющие начальный логический адрес подпрограммы. Слово с большим адресом содержит базовый адрес сегмента, а слово с меньшим адресом — смещение

Рис. 1.7. Таблица указателей векторов прерываний

подпрограммы от начала кодового сегмента. При переходе на подпрограмму смещение загружается в регистр IP, а адрес сегмента загружается в регистр CS. Так как размер каждого элемента таблицы составляет 4 Байт, МП вычисляет адрес (смещение) требуемого элемента путем умножения типа прерывания (номера вектора прерывания) на 4.

Необходимо отметить, что при обращении к таблице указателей сегментные регистры не используются. При формировании физического адреса вычисленное смещение складывается с нулем и ни один сегментный регистр не используется, на что указывают сигналы состояния S4 = S5 = 10. После установления нового содержимого регистров IP и CS микропроцессор выбирает код операции первой команды подпрограммы и затем выполняет обычные действия по заполнению очереди команд, выполнению команд и обмену данными.

Когда осуществляется переход на подпрограмму обслуживания прерывания, содержимое регистра F (вместе с содержимым регистров CS и IP) запоминается в стеке, флаг IF (а также флаг TF) сбрасывается. Тем самым автоматически запрещаются внешние прерывания по входу INTR, что нужно, например, для защиты начального участка подпрограммы, в течение которого осуществляется включение в стек внутренних регистров МП. Затем подпрограмма может разрешить внешние прерывания командой STI. Кроме того, она может

быть прервана запросом на входе NMI и внутренними прерываниями. Необходимо следить, чтобы в подпрограмме не возникало прерывание того типа, которое она обслуживает. Например, попытка деления на нуль в процедуре прерывания из-за ошибки деления приведет к бесконечным вызовам этой процедуры.

В конце подпрограммы восстанавливают содержимое регистров МП, которые были включены в стек в начале подпрограммы с целью сохранения данных, относящихся к прерванной программе. Этот участок подпрограммы следует защитить с помощью команды CLI от прерываний по входу INTR. Подпрограмма обработки прерывания должна заканчиваться командой возврата из прерывания IRET. Перед выполнением команды IRET стек должен быть в том состоянии, в котором он был сразу перед вызовом подпрограммы. Тогда эта команда извлекает три верхних слова из стека в регистры IP, CS и F, что обеспечивает возврат к команде, которая выполнялась бы в случае отсутствия прерывания.

Контрольные вопросы

1. В каком диапазоне лежат коды типов (номеров векторов) прерывания?
2. Как могут инициироваться прерывания?
3. Перечислите источники прерываний.
4. Как реагирует МП на прерывание?
5. Охарактеризуйте внешние прерывания.
6. Как взаимодействует МП ВМ80 и МП ВМ86 с контроллером прерывания в процессе подтверждения прерывания?
7. Когда МП реагирует на сигнал INTR = 1?
8. Охарактеризуйте случаи, когда сигнал INER распознается МП только при завершении выполнения следующей команды.
9. Охарактеризуйте особые случаи, когда запрос прерывания распознается в процессе выполнения команды.
10. Охарактеризуйте реакцию МП на запрос прерывания INER = 1 при IF = 0.
11. Какими командами можно управлять состоянием флага IF?
12. Как, в процессе подтверждения прерывания, МП ВМ86 реагирует на запрос шины другим процессором или сопроцессором в минимальном и максимальном режимах?
13. Как МП реагирует на запрос прерывания по входу NMI?
14. Чем определяется время задержки реакции МП на запрос прерывания?
15. Чем характеризуются внутренние прерывания?
16. Охарактеризуйте прерывание типа 0.
17. Охарактеризуйте прерывание типа 4.
18. Охарактеризуйте прерывание типа 1.
19. Как программно можно изменить состояние флага TF?
20. Как МП в пошаговом режиме реагирует на внешние и внутренние прерывания?
21. Как МП выполняет программные прерывания INT n?
22. Как МП выполняет программное прерывание INT3?
23. Охарактеризуйте таблицу указателей векторов прерываний.
24. Какие действия совершает МП при переходе к выполнению программы обработки прерывания?
25. Какие требования предъявляются к программе обработки прерывания?

2. Система команд микропроцессора К1810ВМ86 (i8086)

2.1. Форматы команды

Микропроцессор К1810ВМ86 относится к классу однокристальных с фиксированной системой команд. При рассмотрении системы команд удобно обращаться к программной модели МП, содержащей его функциональные узлы (регистры), доступные программисту (рис. 2.1). Общие регистры разбиты на две группы: 1) группа HL (универсальные регистры), состоящая из регистров AX, BX, CX, DX, которые предназначены, прежде всего, для хранения данных и допускают раздельную адресацию их старших (H) и младших (L) половин; 2) группа PI (регистры смещений), содержащая указательные регистры BP, SP и индексные регистры SI, DI, в которых обычно хранится адресная информация. Для общих и сегментных регистров указаны коды, используемые в форматах команд для их адресации. Регистр указатель команд IP и регистр флагов F адресуются в командах неявно. Более подробные характеристики регистров приведены в разделе 1.2 при описании операционного устройства МП.

Рис. 2.1. Программно-доступные регистры МП ВМВБ

Команды МП ВМ86 могут адресовать один или два операнда, причем двухоперандные команды являются, как правило, симметричными, так как результат операции может быть направлен на место любого из операндов. Однако в таких командах один из операндов должен обязательно располагаться в регистре, поскольку имеются команды типа регистр — регистр, регистр — память и память — регистр, но команды типа память — память отсутствуют (за исключением команды пересылки цепочки байт или слов).

В общем виде формат двухоперандной команды приведен на рис. 2.2, *а*, где штриховыми линиями обозначены необязательные байты команды.

Первый байт команды содержит код операции COP и два однобитовых поля: направления d и слова w. При d = l осуществляется передача операнда или результата операции в регистр, который определяется полем reg второго байта команды; при d = 0 — передача из указанного регистра. Поле w идентифицирует тип (разрядность) операндов: при w = l команда оперирует словом, при w = 0 — байтом.

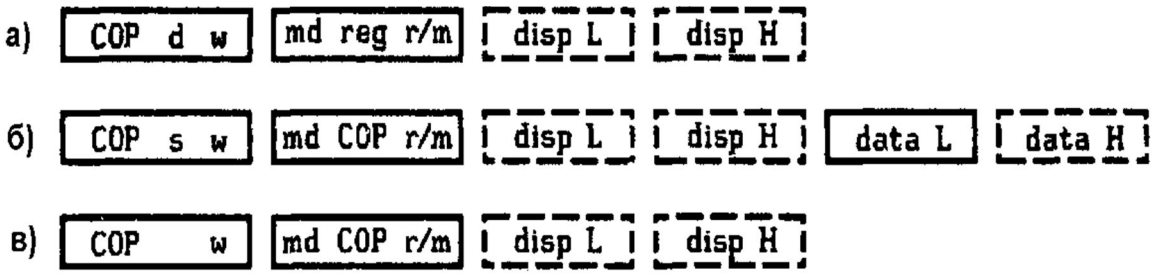

Рис. 2.2. Форматы двухоперандных команд

Второй байт, называемый постбайтом, определяет участвующие в операции регистры или регистр и ячейку памяти. Постбайт состоит из трех полей: md — режим, reg — регистр, r/m — регистр/память. Поле reg определяет операнд, который обязательно находится в регистре МП и условно считается вторым операндом. Поле r/m определяет операнд, который может находиться в регистре или памяти и условно считается первым. Способ кодирования внутренних регистров МП в полях reg и r/m представлен в табл. 2.1.

Таблица 2.1 Кодирование регистров в полях reg и r/m

reg, r/m	w=0	w=1	reg, r/m	w=0	w=1
000	AL	AX	100	AH	SP
001	CL	CX	101	CH	BP
010	DL	DX	110	DH	SI
011	BL	BX	111	BH	DI

Отметим, что поле reg используется для указания регистра только в двухоперандных командах. Если в команде один операнд, то он идентифицируется полем r/m, а поле reg используется для расширения кода операции.

Поле md показывает, как интерпретируется поле r/m для нахождения первого операнда: если md=11, то операнд содержится в регистре, в остальных случаях — в памяти. Когда адресуется память, поле md определяет вариант использования смещения disp, находящегося в третьем и четвертом байтах команды:

$$\mathrm{md} = \begin{cases} 00, & \mathrm{disp} = 0 \text{ — смещение отсутствует;} \\ 01, & \mathrm{disp} = \mathrm{disp\ L} \text{ — команда содержит 8-битовое смещение,} \\ & \text{которое расширяется со знаком до 16 бит;} \\ 10, & \mathrm{disp} = \mathrm{disp\ H,\ disp\ L} \text{ — команда содержит 16-битовое смещение.} \end{cases}$$

При md ≠ 11 реализуется косвенная адресация памяти, и поле r/m определяет правила формирования эффективного адреса EA операнда в соответствии с табл. 2.2, где disp означает смещение, заданное в формате команды (при md = 00 – отсутствует).

Приведенные в табл. 2.2 правила имеют одно исключение, позволяющее реализовать прямую (абсолютную) адресацию: если md=00 и r/m=110, то EA=disp H, disp L. Таким образом, имеется три варианта интерпретации поля md и восемь вариантов интерпретации поля r/m, что дает 24 варианта вычисления эффективного адреса EA. Суммарные сведения о постбайтовых режимах адресации ЦП ВМ86 приведены в табл. 2.3 и на рис. 2.3.

Рассмотренная интерпретация полей md и r/m справедлива для всех форматов команд с постбайтовой адресацией.

Подчеркнем смысловое различие двух случаев употребления термина “смещение”. Смещение disp, содержащееся в команде, интерпретируется как знаковое целое, которое участвует в вычислении эффективного адреса EA. С другой стороны, из-за сегментной

организации памяти весь эффективный адрес EA является смещением (offset) относительно базового адреса сегмента и интерпретируется как беззнаковое целое при вычислении физического адреса. В необходимых случаях во избежание ошибок будем называть offset смещением в сегменте.

Таблица 2.2. Правила формирования эффективного адреса EA

Поле r/m	Эффективный адрес EA	Адресация
000	BX+SI+disp	Базово-индексная
001	BX+DI+disp	
010	BP+SI+disp	
011	BP+DI+disp	
100	SI+disp	Индексная
101	DI+disp	
110	BP+disp	Базовая
111	BX+disp	

Таблица 2.3 Суммарные сведения о постбайтовых режимах

Поле	Поле md				
r/m	00	01	10	11	
				W = 0	W = 1
000	BX+SI	BX+SI+D8	BX+SI+D16	AL	AX
001	BX+DI	BX+DI+D8	BX+DI+D16	CL	CX
010	BP+SI	BP+SI+D8	BP+SI+D16	DL	DX
011	BP+DI	BP+DI+D8	BP+DI+D16	BL	BX
100	SI	SI+D8	SI+D16	AH	SP
101	DI	DI+D8	DI+D16	CH	BP
110	D16	BP+D8	BP+D16	DH	SI
111	BX	BX+D8	BX+D16	BH	DI

Примечание. D8 = disp L (однобайтовое смещение); D16 = disp H disp L (двухбайтовое смещение).

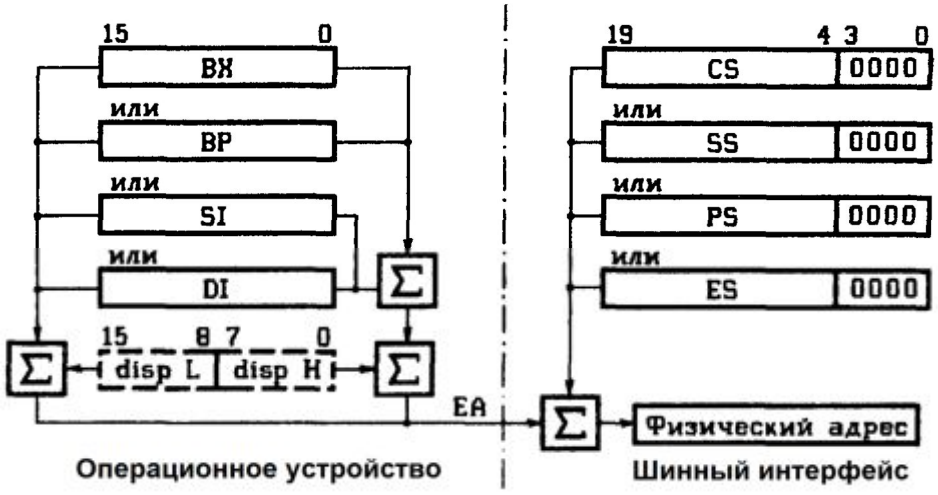

Рис. 2.3. Вычисление физического адреса памяти

Наиболее общий формат двухоперандной команды с непосредственным операндом

приведен на рис. 2.2,б. Необходимость адресации второго операнда отсутствует, и поле reg используется для расширения кода операции. Отсутствует также бит направления d, так как результат операции можно поместить только на место первого операнда. Место этого бита занимает бит s, который является признаком использования одного байта для задания непосредственного операнда при работе со словами.

Поля s и w интерпретируются следующим образом:

$$sw = \begin{cases} \times 0, & \text{один байт данных data L;} \\ 01, & \text{два байта данных data H, data L;} \\ 11, & \text{один байт данных, который автоматически расширяется со знаком до 16 бит.} \end{cases}$$

Типичный формат однооперандной команды (рис. 2.2,*в*) содержит поля, назначение которых уже рассмотрено.

Следует отметить, что постбайтовая адресация является весьма универсальной и позволяет адресовать как общие регистры, так и ячейки памяти с указанием любого варианта вычисления эффективного адреса ЕА. Однако при адресации только регистров или аккумулятора постбайт оказывается излишним, если трехбитовое поле reg разместить в первом байте команды или использовать неявную адресацию. Эта возможность реализуется в специальных (укороченных) форматах, которые содержат минимально необходимое число байтов. ***Специальные форматы*** предусмотрены для команд, выполняющих следующие часто используемые операции: включение содержимого регистра в стек, извлечение из стека в регистр, передача непосредственных данных в регистр, инкремент и декремент содержимого регистра, обмен содержимым аккумулятора АХ и регистра, пересылка между аккумулятором и ячейкой памяти с прямой адресацией, вычитание с участием содержимого аккумулятора и непосредственного операнда. В качестве примера на рис. 2.4 приведены стандартный и специальный форматы команды INC r. Программа-ассемблер выбирает более короткий формат команды автоматически.

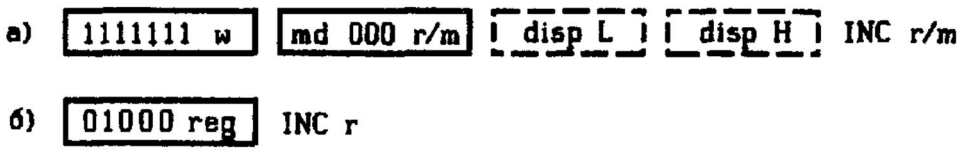

Рис. 2.4. Пример стандартного (а) и специального (б) форматов команды INC r

Всего можно выделить девять существенно различных форматов команд. В табл. 2.4 приведены форматы команд, не обязательные байты заключены в круглые скобки, а также указаны варианты форматов, различающихся назначением битов b_i первого байта B1 формата команды, в котором содержится код операции COP. (В данную таблицу не включены редко используемые двухбайтовые команды AAM и AAD, оба байта которых заняты кодом операции.)

2.2. Способы адресации

Команды ЦП ВМ86 реализуют весьма разнообразные способы адресации, что упрощает организацию и использование сложных структур данных, а также расширяет возможности отдельных команд и повышает гибкость их применения.

Регистровая адресация. Операнд находится в одном из общих регистров МП или в одном из сегментных регистров. Регистр может быть определен в байте кода операции или в постбайте, в котором выделены 3-битовые поля reg и r/m (при md = 11). Команды, оперирующие содержимым регистров, являются наиболее короткими и выполняются за наименьшее время, так как не требуют вычисления ЕА и выполнения цикла шины для обращения к памяти. Для многих команд с регистровой адресацией предусмотрены специальные укороченные форматы.

Таблица 2.4. Форматы команд

Формат	Вариант формата	Полный список команд
COP		CLC; CMC; STC; CLD; STD; CLI; STI; HLT; WAIT; NOP; RET; IRET; INT0; INT3; PUSHF; POPF; LAHF; SAHF; XLAT; CBW; CWD; AAA; DAA; AAS; DAS; LOCK; REP; REPNZ
	b0=w	CMPS; LODS; MOVS; STOS; SCAS; IN ac,DX; OUT DX,ac
	b2-b0=reg	INC r; DEC r; PUSH r; POP r; XCHG AX,r
	b2-b0=COP b4-b3=sr	PUSH sr; POP sr
COP md reg r/m (disp L) (disp H)		LEA; LDS; LES
	b0=w	TEST r,r; TEST r,m; XCHG r,r; XCHG r,m
	b0=w, b1=d	MOV r,r; MOV r,m; MOV m,r; ADD r,r; ADD r,m; ADD m,r; ADC r,r; ADC r,m; ADC m,r; SUB r,r; SUB r,m; SUB m,r; SBB r,r; SBB r,m; SBB m,r; CMP r,r; CMP r,m; CMP m,r; AND r,r; AND r,m; AND m,r; OR r,r; OR r,m; OR m,r; XOR r,r; XOR r,m; XOR m,r;
COP md COP r/m (disp L) (disp H)		PUSH r; PUSH m; POP r; POP m; JMP [r]; CALL [r]; JMP [m]; CALL [m]
	b0=w	INC m; DEC m; NEG; NOT; MUL; IMUL; DIV; IDIV
	b0=w, b1=v	RCL; RCR; ROL; ROR; SAL; SAR; SHL; SHR
COP md COP r/m (disp L) (disp H) data L (data H)	b0=w	MOV m,d; MOV r,d; AND m,d; AND r,d; OR m,d; OR r,d; XOR m,d; XOR r,d; TEST m,d; TEST r,d; CMP m,d; CMP r,d
	b0=w	ADD m,d; ADD r,d; ADC m,d; ADC r,d; SUB m,d; SUB r,d; SBB m,d; SBB r,d
COP data L (data H)		RET d
	acb0=w	ADD ac,d; ADC ac,d; SUB ac,d; SBB ac,d; CMP ac,d; AND ac,d; OR ac,d; XOR ac,d; TEST ac,d
	b2-b0=reg b3=w	MOV r,d
COP (disp L) (disp H)		JMP disp16; JMP disp8; Jcond disp8; CALL disp16;
	b0=w	MOV ac,m; MOV m,ac (при disp L, disp H)
COP md 0sr r/m (disp L) (disp H)		MOV sr,m; MOV sr,r; MOV m,sr; MOV r,sr;
COP addr8		IN addr8; OUT addr8
COP offset L offset H seg L seg H		JMP addr; CALL addr (тип FAR)

Непосредственная адресация. Непосредственными операндами являются константы длиной 8 или 16 бит, которые размещаются в последних байтах команды (младший байт следует первым). Доступ к таким операндам в ВМ86 осуществляется очень быстро, поскольку во время выполнения команды они находятся во внутренней очереди команд. Имеются команды, позволяющие манипулировать непосредственными операндами и содержимым общих регистров или ячеек памяти. Однако отсутствуют команды непосредственной загрузки сегментных регистров и включения константы в стек.

Поэтому эти операции осуществляются с помощью промежуточной загрузки общего регистра или ячейки памяти. Однако отсутствуют команды непосредственной загрузки сегментных регистров и включения константы в стек. Поэтому эти операции осуществляются с помощью промежуточной загрузки общего регистра или ячейки памяти.

Прямая адресация является простейшим способом адресации ячейки ЗУ, при котором эффективным адресом EA является содержимое байтов смещения disp команды.

В командах преобразования данных этот способ реализуется при использовании постбайта с полями md = 00 и r/m = 110. Для пересылок между аккумулятором и памятью предусмотрен укороченный формат.

Разновидностью этого способа является длинная прямая адресация, при которой в формате команды содержатся четыре байта, указывающие базовый адрес сегмента и сегментное смещение offset. В этом случае обеспечивается доступ к ячейке с любым логическим адресом, т. е. к произвольной ячейке во всем пространстве адресов 1 Мбайт, но длинная прямая адресация используется только в командах межсегментных переходов и вызовов подпрограмм. Другая разновидность прямой адресации применяется для указания портов ввода-вывода в двухбайтовых командах IN и OUT, второй байт которых содержит адрес (номер) порта.

Косвенная регистровая адресация. В командах преобразования данных эффективный адрес EA равен содержимому одного из регистров SI, DI, BX и BP при соответствующем кодировании полей md и reg постбайта: md = 00; r/m = 100, 101, 111 и md = 01, r/m = 110; disp L = 0. В командах безусловного перехода и вызова подпрограммы с регистровой косвенной адресацией допускается указание любого 16-битового общего регистра (при md = 11; r/m = 000,..., 111).

Данный способ адресации позволяет вычислять адреса во время выполнения программ, что часто требуется, например, для обращения к различным элементам таблиц данных. При модификации содержимого регистра одна и та же команда оперирует различными ячейками памяти, что позволяет организовать вычислительные циклы. Изменение содержимого регистра обычно осуществляется с помощью команд инкрементирования и декрементирования, а также других арифметических команд и команды загрузки эффективного адреса LEA. Разновидностью этого способа является косвенная адресация портов ввода - вывода с помощью содержимого регистра DX в однобайтовых командах IN и OUT.

Базовая адресация. Эффективный адрес операнда EA вычисляется путем суммирования содержимого базовых регистров BX или BP и смещения disp (при md = 01, 10 и r/m = 111, 110). При использовании BX происходит обращение к операнду в текущем сегменте данных, а при использовании BP - в текущем сегменте стека. Смещения, содержащиеся в команде, могут иметь длину 8 или 16 бит и интерпретируются как знаковые целые, представленные в дополнительном коде.

Базовая адресация обычно используется для подступа к элементам структур данных, когда смещение (номер) элемента известно на стадии разработки программы (при ее ассемблировании), а базовый (начальный) адрес структуры должен вычисляться при

выполнении программы. Модификация содержимого базового регистра позволяет обратиться к одноименному элементу различных структур данных.

Индексная адресация. Значение EA вычисляется как сумма смещения disp, находящегося в команде, и содержимого индексного регистра SI или DI (при md = 01, 10 и r/m = 100, 101). Данный способ обычно применяется для обращения к различным элементам одномерного массива (таблицы) данных, когда смещение определяет известный при ассемблировании начальный адрес массива, а индексный регистр, содержимое которого может модифицироваться при выполнении программы, определяет элемент массива. По существу индексная адресация в BM86 аналогична базовой.

Базовая индексная адресация. Эффективный адрес EA равен сумме содержимого базового регистра BX или BP, индексного регистра SI или DI и смещения disp, находящегося в команде (в частном случае смещение может отсутствовать). Этот способ реализуется при следующем кодировании полей постбайта: md ≠ 11; r/m = 000, 001, 010, 011 и обеспечивает наибольшую гибкость адресации, так как два компонента адреса можно определить и варьировать при выполнении программы. Это удобно при обращении к элементам матриц, т. е. к двумерным массивам, представляемым в памяти как совокупность одномерных массивов.

Относительная адресация ЦП BM86 реализуется только по отношению к указателю команд IP, так что сегментное смещение вычисляется как сумма смещения disp, находящегося в команде, и текущего значения IP. При этом значение IP равно адресу байта, следующего за рассматриваемой командой, которая в это время выполняется микропроцессором. В BM86 относительная адресация не используется в командах, оперирующих данными, а применяется только в командах условных и безусловных переходов, вызовов подпрограмм и управления циклами. Смещение по отношению к содержимому IP не зависит от размещения программ в адресном пространстве памяти, что обеспечивает позиционную независимость команд. При автоматизированном ассемблировании программы указывается метка команды, которой передается управление, а необходимое смещение вычисляется программой-ассемблером.

Неявная адресация. Объект, содержимым которого манипулирует команда, указывается с помощью первого байта команды вместе с кодом операции без выделения специального поля для этой цели. Чаще всего этот специфический способ адресации встречается в однобайтовых командах, где адресуемым объектом являются аккумулятор, регистр флагов или отдельные флаги. В частности, в командах обработки цепочек неявно используются индексные регистры, причем регистр SI адресует первый элемент цепочки-источника, а регистр DI - первый элемент цепочки-получателя.

Контрольные вопросы

1. Охарактеризуйте программную модель МП BM86.
2. Дайте общую характеристику команд МП BM86.
3. Охарактеризуйте формат двухоперандной команды (рис.2.2, а).
4. Какой операнд в постбайте считается первым, а какой – вторым?
5. Как кодируются регистры в полях reg и r/m постбайта?
6. Как используется поле reg в однооперандных командах?
7. Как используется поле md в постбайте?
8. Каково исключение из правил заданных в табл. 2.2?
9. Объясните два случая употребления термина “смещение”.

10. Используя табл.2.1 и 2.3 составьте двоичный код постбайта команды формата рис.2.2, а использующей регистр **CX** и эффективный адрес **BX+DI+D16.**
11. Поясните алгоритм получения физического адреса, представленный на рис. 2.3.
12. Поясните формат двухоперандной команды с непосредственным операндом приведенный на рис. 2.2,б.
13. Для каких операций используются специальные форматы команд?
14. Какие девять различных форматов команд можно выделить?
15. Охарактеризуйте регистровую адресацию.
16. Охарактеризуйте непосредственную адресацию.
17. Охарактеризуйте прямую адресацию.
18. Охарактеризуйте косвенную адресацию.
19. Охарактеризуйте базовую адресацию.
20. Охарактеризуйте индексную адресацию.
21. Охарактеризуйте базово-индексную адресацию.
22. Охарактеризуйте относительную адресацию.
23. Охарактеризуйте неявную адресацию.

2.3. Описание системы команд

Система команд ВМ86 содержит 91 мнемокод и позволяет совершать операции над байтами, двухбайтовыми словами, отдельными битами, а также цепочками байтов и слов. Имеется широкий набор арифметических команд, включающий умножение и деление, который ориентирован на обработку как беззнаковых, так и знаковых чисел. Весьма разнообразны команды пересылки данных, логических операций, переходов в программе и вызовов подпрограмм, а также управления микропроцессором. Число вариантов команд, т.е. число различных машинных кодов, превышает 3800 благодаря использованию восьми способов адресации в их различных модификациях. Тем самым обеспечивается гибкость применения большого числа команд, что позволяет выбрать наиболее рациональные способы адресации в конкретных случаях. Это особенно важно при обработке сложных структур данных.

По функциональному признаку система команд ВМ86 разбивается на шесть групп: **пересылка данных**, **арифметические операции**, **логические операции и сдвиги**, **передача управления**, **обработка цепочек** и **управление микропроцессором**.

Полный набор команд, упорядоченный по мнемокоду, машинные коды команд и описание выполняемых операций приведены в табл. 2.5. Для удобства пользования в таблице представлены все варианты мнемонических обозначений команд условных переходов и команд управления циклами.

Для каждой цепочечной команды приведено только общее мнемоническое обозначение, которое требует дополнительного указания разрядности элементов цепочки, например MOVSB — пересылка байтов, MOVSW — пересылка слов.

Данные табл. 2.5 позволяют осуществить переход от мнемокодов к машинным кодам команд. При необходимости выполнить обратный переход целесообразно воспользоваться системой команд, упорядоченной по машинному коду операции. Естественно, что наиболее удобным является автоматическое преобразование кодов с помощью соответствующих программ ассемблера и реассемблера (дисассемблера).

В табл. 2.5 введены следующие обозначения: r - общий регистр; sr - сегментный регистр; m - адрес ячейки памяти, который указывается в мнемокоде в соответствии с используемым способом адресации; d – непосредственный операнд; ac - аккумулятор AX

или AL; p - адрес 8-разрядного порта ввода - вывода; disp - смещение при адресации относительно IP; addr - указатель адреса при межсегментных переходах и вызовах; type - тип (вектор) прерывания; cond - условия в команде условных переходов; disp 8/16 □ смещение в формате команды, состоящее из одного или двух байтов; d 8/16 □ одно- или двухбайтовая константа; sbr □ имя подпрограммы; diff □ разница между адресом перехода и содержимым указателя команд IP (при адресации относительно IP); label — метка, к которой осуществляется переход.

Влияние команд на флаги иллюстрируют табл. 2.6, 2.7, в которые включены только те команды, которые это влияние оказывают. Знак вопроса соответствует случаям, когда состояние флага после выполнения команды произвольно (например, зависит от конкретных значений операндов).

Таблица 2.5. Полный набор команд, упорядоченный по мнемокоду

№ п/п	Команда	Байты кода команды			Символическое описание и/или содержание операции
		B1	B2	B3-B6	
1	AAA	00110111			ASCII - коррекция для сложения
2	AAD	11010101	00001010		ASCII - коррекция для деления
3	AAM	11010100	00001010		ASCII - коррекция для умножения
4	AAS	00111111			ASCII - коррекция для вычитания
5	ADC r,r/m	0001001W	md reg r/m	(disp8/16)	r←r+r/m+CF; сложение с переносом
	r/m,r	0001000W	md reg r/m	(disp8/16)	r/m←r+r/m+CF
	r/m,d	1000000W	md 010 r/m	(disp8/16)d8/16	r/m←r/m+d+CF
	r16/m16,d8	10000011	md 010 r/m	data L	r/m←r/m+d+CF
	ac,d	0001010W	data L	(data H)	ac←ac+d+CF
6	ADD r,r/m	0000001W	md reg r/m	(disp8/16)	r←r+r/m; сложение
	r/m,r	0000000W	md reg r/m	(disp8/16)	r/m←r+r/m
	r/m,d	1000000W	md 000 r/m	(disp8/16)d8/16	r/m←r/m+d
	r16/m16,d8	10000011	md 000 r/m	data L	r/m←r/m+d
	ac,d	0000010W	data L	(data H)	ac←ac+d
7	AND r,r/m	0010001W	md reg r/m	(disp8/16)	r←r∧r/m; конъюнкция, И
	r/m,r	0010000W	md reg r/m	(disp8/16)	r/m←r∧r/m
	r/m,d	1000000W	md 100 r/m	data L data H	r/m←r/m∧d
	ac,d	0010010W	data L	(data H)	ac←ac∧d
8	CALL				
	NEAR sbr	11101000	diff L	diff H	IP←IP+diff; вызов прямой
	NEAR r16/m16	11111111	md 010 r/m	(disp8/16)	IP←IP+r/m; вызов косвенный
	FAR sbr	10011010	IP-L	IP-H CS-L CS-H	IP←IP-L,IP-H; CS←CS-L,CS-H; вызов прямой
	FAR m32	11111111	md 011 r/m	(disp8/16)	IP←m16;CS←m16; вызов косвенный
9	CBW	10011000			Преобразование байта в слово
10	CLC	11111000			CF←0; сброс переноса
11	CLD	11111100			DF←0; сброс триггера направления
12	CLI	11111010			IF←0; запрет прерывания
13	CMC	11110101			CF←CF; инверсия переноса
14	CMP r,r/m	0011101W	md reg r/m	(disp8/16)	r-r/m; сравнение
	r/m,r	0011100W	md reg r/m	(disp8/16)	r/m-r
	r/m,d	1000000W	md 111 r/m	(disp8/16)d8/16	r/m-d
	r16/m16,d8	10000011	md 111 r/m	data L	r/m-d
	ac,d	0011110W	data L	(data H)	ac-d

Продолжение Таблицы 2.5

№ п/п	Команда	Байты кода команды			Символическое описание и/или содержание операции
		B1	B2	B3-B6	
15	CMPS	1010011W			Сравнение элементов цепочек
16	CWD	10011001			Преобразование слова в двойное слово
17	DAA	00100111			Десятичная коррекция для сложения
18	DAS	00101111			Десятичная коррекция для вычитания
19	DEC r/m	1111111W	md 001 r/m	(disp8/16)	r/m←r/m-1; декремент
	r	01001reg			
20	DIV r/m	1111011W	md 110 r/m	(disp8/16)	Деление чисел без знака
21	ESC OPCODE,r/m	11011XXX	md YYY r/m	(disp8/16)	Переключение на сопроцессор
22	HLT	11110100			Останов
23	IDIV r/m	1111011W	md 111 r/m	(disp8/16)	Деление чисел со знаком
24	IMUL r/m	1111011W	md 101 r/m	(disp8/16)	Умножение чисел со знаком
25	IN ac,port	1110010W	port8		Ввод из фиксированного порта
	ac,DX	1110110W			Ввод из порта с косвенной адресацией
26	INC r/m	1111111W	md 000 r/m	(disp8/16)	r/m←r/m+1; инкремент
	r	01000reg			r←r+1;
27	INT type	11001101			Прерывание заданного типа
	INTO	11001110			Прерывание по переполнению
	INT3	11001100			Прерывание типа 3
28	IRET	11001111			Возврат из прерывания
29	JA label	01110111	diff L		IP←IP+diff L; перейти, если выше
30	JAE label	01110011	diff L		IP←IP+diff L; если выше или равно
31	JB label	01110010	diff L		IP←IP+diff L; если ниже
32	JBE label	01110110	diff L		IP←IP+diff L; если ниже или равно
33	JC label	01110010	diff L		IP←IP+diff L; если есть перенос
34	JCXZ label	11100011	diff L		IP←IP+diff L; если CX равен нулю
35	JE label	01110100	diff L		IP←IP+diff L; если равно
36	JG label	01111111	diff L		IP←IP+diff L; если больше
37	JGE label	01111101	diff L		IP←IP+diff L; если больше или равно
38	JL label	01111100	diff L		IP←IP+diff L; если меньше
39	JLE label	01111110	diff L		IP←IP+diff L; если меньше или равно
40	JMP				
	SHORT label	11101011	diff L		IP←IP+diff L; прямой короткий переход
	NEAR label	11101001	diff L	diff H	IP←IP+diff L; прямой переход
	NEAR r16/m16	11111111	md 100 r/m	(disp8/16)	IP←IP+r/m; косвенный переход
	FAR label	11101010	IP-L	IP-H CS-L CS-H	IP←IP-L,IP-H; CS←CS-L,CS-H; прямой переход
	FAR m32	11111111	md 101 r/m	(disp8/16)	IP←m16;CS←m16; косвенный переход
41	JNA label	01110110	diff L		IP←IP+diff L; если не выше
42	JNAE label	01110011	diff L		IP←IP+diff L; если не выше или равно
43	JNB label	01110111	diff L		IP←IP+diff L; если не ниже

Продолжение Таблицы 2.5

№ п/п	Команда	Байты кода команды			Символическое описание и/или содержание операции
		B1	B2	B3-B6	
44	JNC label	01110011	diff L		IP←IP+diff L; если нет переноса
45	JNE label	01110101	diff L		IP←IP+diff L; если не равно
46	JNG label	01111110	diff L		IP←IP+diff L; если не больше
47	JNGE label	01111100	diff L		IP←IP+diff L; если не больше или равно
48	JNL label	01111101	diff L		IP←IP+diff L; если не меньше
49	JNLE label	01111111	diff L		IP←IP+diff L; если не меньше или равно
50	JNO label	01110001	diff L		IP←IP+diff L; если нет переполнения
51	JNP label	01111011	diff L		IP←IP+diff L; если нет паритета
52	JNS label	01111001	diff L		IP←IP+diff L; если результат положительный
53	JNZ label	01110101	diff L		IP←IP+diff L; если не нуль
54	JO label	01110000	diff L		IP←IP+diff L; если есть переполнение
55	JP label	01111010	diff L		IP←IP+diff L; если есть паритет
56	JPE label	01111010	diff L		IP←IP+diff L; если паритет четный
57	JPO label	01111011	diff L		IP←IP+diff L; если паритет нечетный
58	JS label	01111000	diff L		IP←IP+diff L; если результат отрицательный
59	JZ label	01110100	diff L		IP←IP+diff L; если нуль
60	LAHF	10011111			AH←FL; пересылка младшего байта F в AH
61	LDS r16,m32	11000101	md reg r/m	(disp8/16)	r,DS←m32; загрузка указателя адреса
62	LEA r16,m	10001101	md reg r/m	(disp8/16)	r←EA; загрузка эффективного адреса
63	LES r16,m32	11000100	md reg r/m	(disp8/16)	r,ES←m32; загрузка указателя адреса
64	LODS	1010110W			Загрузка элемента цепочки
65	LOOP label	11100010	diff L		IP←IP+diff L; зациклить
66	LOOPE/LOOPZ label	11100001	diff L		IP←IP+diff L; зациклить, если равно
67	LOOPNE/LOOPNZ label	11100000	diff L		IP←IP+diff L; зациклить, если не равно
68	MOV r,r/m	1000101W	md reg r/m	(disp8/16)	r←r/m; пересылка
	r/m,r	1000100W	md reg r/m	(disp8/16)	r/m←r;
	r/m,d	1100011W	md 000 r/m	(disp8/16)d8/16	r/m←d;
	r,d	1011Wreg	data L	(data H)	r←d
	ac,m	1010000W	disp L	disp H	ac←m; при прямой адресации
	m,ac	1010001W	disp L	disp H	m←ac; при прямой адресации
	sr,r/m	10001110	md 0sr r/m	(disp8/16)	sr←r/m; запись в сегментный регистр
	r/m,sr	10001100	md 0sr r/m	(disp8/16)	r/m←sr; чтение из сегментного регистра
69	MOVS	1010010W			Пересылка элемента цепочки
70	MUL r/m	1111011W	md 100 r/m	(disp8/16)	Умножение чисел без знака
71	NEG r/m	1111011W	md 011 r/m	(disp8/16)	Смена знака числа
72	NOP	10010000			Пустая операция
73	OR r,r/m	0000101W	md reg r/m	(disp8/16)	r←rvr/m; дизъюнкция, ИЛИ

Продолжение Таблицы 2.5

№ п/п	Команда	Байты кода команды			Символическое описание и/или содержание операции
		B1	B2	B3-B6	
	r/m,r	0000100W	md reg r/m	(disp8/16)	r/m←r/mvr
	r/m,d	1000000W	md 001 r/m	(disp8/16)d8/16	r/m←r/mvd
	ac,d	0000110W	data L	(data H)	ac←acvd
74	OUT ac,port	1110011W	port8		Вывод в фиксированный порт
	ac,DX	1110111W			Вывод в порт при косвенной адресации
75	POP r/m	10001111	md 000 r/m	(disp8/16)	Чтение из стека
	r	01011reg			
	sr	000sr111			
76	POPF	10011101			Загрузка регистра флагов из стека
77	PUSH r/m	11111111	md 110 r/m	(disp8/16)	Запись в стек
	r	01010reg			
	sr	000sr110			
78	PUSHF	10011100			Запись регистра флагов в стек
79	RCL r/m,1	1101000W	md 010 r/m	(disp8/16)	Циклический сдвиг
	r/m,CL	1101001W	md 010 r/m	(disp8/16)	влево через CF
80	RCR r/m,1	1101000W	md 011 r/m	(disp8/16)	Циклический сдвиг
	r/m,CL	1101001W	md 011 r/m	(disp8/16)	вправо через CF
81	RET (NEAR)	11000011			Возврат из подпрограммы
	d(NEAR)	11000010	data L	data H	Возврат с прибавлением константы к SP
	(FAR)	11001011			Возврат из подпрограммы
	d(FAR)	11001010	data L	data H	Возврат с прибавлением константы к SP
82	ROL r/m,1	1101000W	md 000 r/m	(disp8/16)	Циклический сдвиг
	r/m,CL	1101001W	md 000 r/m	(disp8/16)	влево
83	ROR r/m,1	1101000W	md 001 r/m	(disp8/16)	Циклический сдвиг
	r/m,CL	1101001W	md 001 r/m	(disp8/16)	вправо
84	SAHF	10011110			Пересылка AH в регистр F
85	SAL r/m,1	1101000W	md 100 r/m	(disp8/16)	Арифметический сдвиг
	r/m,CL	1101001W	md 100 r/m	(disp8/16)	влево
86	SAR r/m,1	1101000W	md 111 r/m	(disp8/16)	Арифметический сдвиг
	r/m,CL	1101001W	md 111 r/m	(disp8/16)	вправо
87	SBB r,r/m	0001101W	md reg r/m	(disp8/16)	r←r-r/m-CF; вычитание с заемом
	r/m,r	0001100W	md reg r/m	(disp8/16)	r/m←r-r/m-CF
	r/m,d	1000000W	md 011 r/m	(disp8/16)d8/16	r/m←r/m-d-CF
	r16/m16,d8	10000011	md 011 r/m	data L	r/m←r/m-d-CF
	ac,d	0001110W	data L	(data H)	ac←ac-d-CF
88	SCAS	1010111W			Сканировать элемент цепочки
89	SHL r/m,1	1101000W	md 100 r/m	(disp8/16)	Логический сдвиг
	r/m,CL	1101001W	md 100 r/m	(disp8/16)	влево
90	SHR r/m,1	1101000W	md 101 r/m	(disp8/16)	Логический сдвиг
	r/m,CL	1101001W	md 101 r/m	(disp8/16)	вправо
91	STC	11111001			CF←1; установка флага переноса
92	STD	11111101			DF←1; установка триггера направления
93	STI	11111011			IF←1; разрешение прерывания
94	STOS	1010101W			[DI]←ac; запоминание ac в цепочке
95	SUB r,r/m	0010101W	md reg r/m	(disp8/16)	r←r-r/m; вычитание
	r/m,r	0010100W	md reg r/m	(disp8/16)	r/m←r-r/m
	r/m,d	1000000W	md 101 r/m	(disp8/16)d8/16	r/m←r/m-d
	r16/m16,d8	10000011	md 101 r/m	data L	r/m←r/m-d
	ac,d	0010110W	data L	(data H)	ac←ac-d;
96	TEST r,r/m	1000010W	md reg r/m	(disp8/16)	r∧r/m; проверка, неразрушающее И
	m,r	1000010W	md reg r/m	(disp8/16)	r∧r/m
	r/m,d	1111011W	md 000 r/m	(disp8/16)d8/16	r/m d
	ac,d	1010100W	data L	(data H)	ac∧d
97	WAIT	00111011			Ожидание

Продолжение Таблицы 2.5

№ п/п	Команда	Байты кода команды			Символическое описание и/или содержание операции
		B1	B2	B3-B6	
98	XCHG AX,r	10010reg			AX↔r; обмен
	r,AX	10010reg			r↔AX
	r,r/m	1000011W	md reg r/m	(disp8/16)	r↔r/m
	m,r	1000011W	md reg r/m	(disp8/16)	m↔r
99	XLAT	11010111			Преобразование байта из AL
100	XOR r,r/m	0011001W	md reg r/m	(disp8/16)	r←r⊕r/m; суммирование по модулю 2 (исключающее ИЛИ)
	r/m,r	0011000W	md reg r/m	(disp8/16)	r/m←r⊕r/m;
	r/m,d	1000000W	md 110 r/m	(disp8/16)d8/16	r/m←r/m⊕d
	ac,d	0011010W	data L	(data H)	ac←ac⊕d
	Префиксы:				
	LOCK	11110000			Префикс блокировки шины
	REP/REPE/REPZ	11110011			Повторять цепочечную операцию
	REPNE/REPNZ	11110010			То же
	SEG	001sr110			Префикс замены сегмента

Таблица 2.6. Флаги состояния (статусные)

Операции	Команды	Флаги состояний					
		OF	CF	AF	SF	ZF	PF
Сложение, вычитание	ADD ADC SUB SBC	+	+	+	+	+	+
	CMP NEG CMPS SCAS	+	+	+	+	+	+
	INC DEC	+	-	+	+	+	+
Умножение	MUL IMUL	+	+	?	?	?	?
Деление	DIV IDIV	?	?	?	?	?	?
Десятичная коррекция	DAA DAS	?	+	+	+	+	+
	AAA AAS	?	+	+	?	?	?
	AAM AAD	?	?	?	+	+	+
Логические	AND OR XOR TEST	0	0	?	+	+	+
Сдвиги	SHL SHR[1]	+	+	?	+	+	+
	SHL SHR[2]	?	+	?	+	+	+
	SAR	0	+	?	+	+	+
	ROL,ROR,RCL,RCR[1]	+	+	-	-	-	-
	ROL,ROR,RCL,RCR[2]	?	+	-	-	-	-
Восстановление флагов	POPF IRET	+	+	+	+	+	+
	SAHF	-	+	+	+	+	+
Управление флагом переноса	STC	-	1	-	-	-	-
	CLC	-	0	-	-	-	-
	CMC	-	Г	-	-	-	-

Примечание. "+" - результат операции влияет на флаг; "-" - не влияет; 1 - устанавливает в "1"; 0 - устанавливает в "0"; Г - инвертирует; ? - не определен.
[1] Одиночный сдвиг.
[2] Многоразрядный сдвиг.

Таблица 2.7. Флаги управления

Операции	Команды	Флаги управления		
		DF	IF	TF
Восстановление флагов	POPF IRET	+	+	+
Прерывания	INT INTO	-	0	0
Управление флагами	STD	1	-	-
	CLD	0	-	-
	STI	-	1	-
	CLI	-	0	-

Примечание. "+" - результат операции влияет на флаг; "-" - не влияет; 1 - устанавливает в "1"; 0 - устанавливает в "0".

Контрольные вопросы

1. Дайте общую характеристику системе команд МП ВМ86.
2. Перечислите функциональные группы команд ВМ86.
3. Охарактеризуйте комментарии к таблице полного набора команд.
4. Какие обозначения введены в табл.2.5?
5. Охарактеризуйте влияние команд на флаги (табл. 2.6, 2.7).

2.4. Особенности выполнения отдельных команд

Приводится описание лишь некоторых команд, выбранных по принципу частоты их использования или трудности освоения. Описание остальных команд можно найти в [2-5].

2.4.1. Команды пересылки данных

Команды пересылки данных составляют четыре подгруппы: *общие*, *обращения к стеку*, *ввода/вывода* и *пересылки цепочек* (последние рассмотрены вместе с другими цепочечными командами). Эти команды, за исключением POPF и SAHF, не влияют на флаги.

Команда **MOV** (рис. 2.5) осуществляет пересылку содержимого источника src в получатель dst и имеет обобщенное представление:

MOV dst, src; dst←src

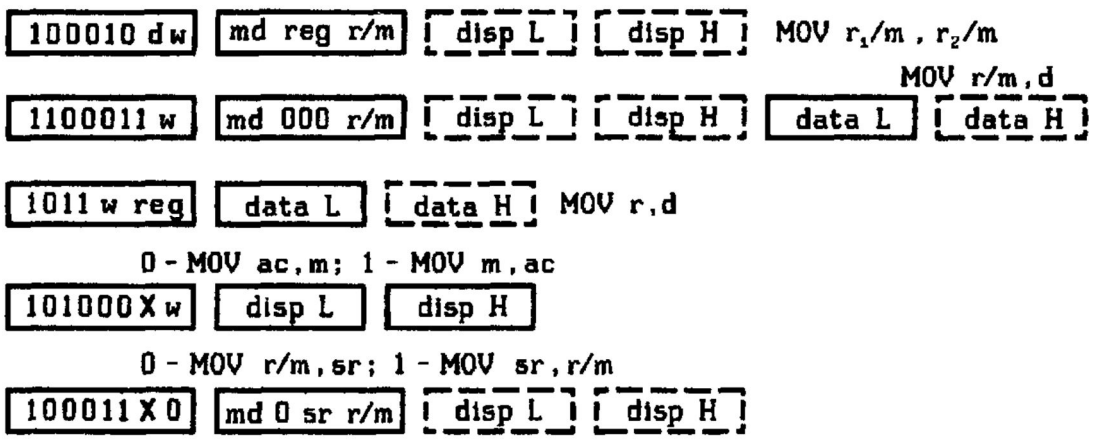

Рис. 2.5. Форматы команды MOV

Команда **MOV r_1/m, r_2/m** содержит постбайт и обеспечивает пересылки регистр – регистр – память и память – регистр при использовании любого общего регистра и любого способа адресации памяти. Бит w определяет передачу байта или слова, а бит d – направление передачи.

Команда **MOV r/m, d** позволяет передать непосредственные данные в общий регистр или ячейку памяти. Команда **MOV r, d** представляет более короткий вариант (специальный формат) предыдущей команды и осуществляет загрузку констант в общие регистры.

Команды **MOV ac, m** и **MOV m, ac** предназначены для загрузки и запоминания содержимого аккумуляторов AL и AX при использовании прямой адресации. Обращение производится к текущему сегменту данных, и адрес, указанный в команде, представляет смещение в этом сегменте. Если пересылаются два байта, то младший располагается по указанному адресу, а старший – по следующему адресу.

Команды **MOV sr, r/m** и **MOV r/m, sr** осуществляют пересылки между сегментным регистром и регистром или памятью. При этом передаются только слова, а ячейка памяти может быть определена с помощью любого допустимого способа адресации. Следует учитывать, что в команде MOV sr, r/m нельзя указывать сегментный регистр кода CS, так как при этом результат операции не определён (выполнение этой команды было бы равносильно специальному безусловному переходу в программе).

Так как не существует команды непосредственной загрузки сегментных регистров, то команда MOV sr, r/m используется для инициализации регистров SS, DS и ES, т. е. для определения соответствующих сегментов памяти. Если, например, в регистр DS необходимо загрузить число 8000, то потребуется две команды:

MOV AX, 8000H; Инициализация регистра DS на 8000

MOV DS, AX;

При этом в качестве промежуточного регистра обычно используется AX, так как команда MOV ac, d короче более общей команды MOV m/r, d.

Команда **XCHG** (рис. 2.6) осуществляет обмен данными между источником и получателем:

XCHG dst, src; dst↔src

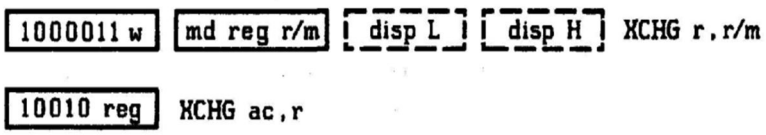

Рис. 2.6. Команда XCHG

Она имеет два формата. Общий формат позволяет произвести обмен содержимым любой пары общих регистров, а также обмен между общим регистром и ячейкой памяти при любом допустимом способе адресации. Укороченный формат осуществляет обмен любого общего регистра и аккумулятора AX. Команда XCHG AX, AX, 16-ричный код которой равен 90, используется как команда пустой операции NOP, обеспечивающая задержку 3T.

XLAT - однобайтная команда с кодом операции D7 предназначена для быстрого преобразования кодов и заменяет содержимое AL на байт из 256-байтовой таблицы, начальный (базовый) адрес которой содержится в регистре BX (рис. 2.7).

Другими словами, содержимое AL используется как индекс таблицы, находящейся в сегменте данных и адресуемой регистром BX. При выполнении этой команды к содержимому BX прибавляется содержимое AL, а полученный результат используется как смещение относительно DS. Адресуемый таким образом байт из памяти пересылается в AL.

Команды **LEA, LDS, LES** (рис. 2.8) отличаются от других команд пересылки тем, что

при их выполнении в адресуемый регистр (регистры) передаются не собственно данные из памяти, а их адреса. Основное назначение этих команд инициализация регистров перед выполнением цепочечных команд или перед вызовом подпрограммы.

Команда **LEA r, m** обеспечивает вычисление эффективного адреса EA ячейки памяти в соответствии с указанным способом адресации и загрузку EA (а не содержимого адресуемой ячейки памяти!) в указанный общий регистр. Такая операция может потребоваться, например, для загрузки начального адреса таблицы в регистр BX перед выполнением команды XLAT.

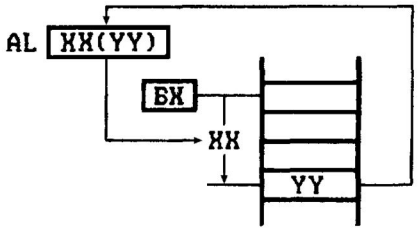

Рис. 2.7. Действие команды XLAT

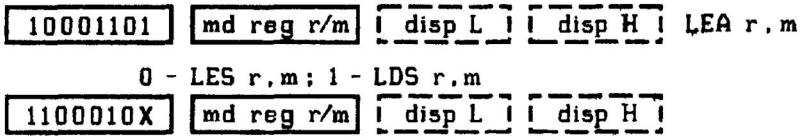

Рис. 2.8. Команды LEA, LDS, LE5

Команды **LDS r/m** и **LES r, m** загружают адресную информацию из памяти в адресуемый общий регистр и в соответствующий сегментный регистр. Сначала вычисляется адрес EA памяти, который, как обычно, суммируется с содержимым регистра DS, затем слово из памяти по вычисленному адресу загружается в общий регистр, а следующее слово в регистр DS (команда LDS) или ES (команда LES). При этом кодирование поля md = 11 (регистровая адресация) не должно использоваться, так как действия команд в этом случае не определены.

Команды LDS и LES упрощают коммутацию сегментов данных, поскольку одновременно загружают базовый или индексный регистр и сегментный регистр. Если эти команды подготавливают обращение к цепочке, то в команде LDS указывается регистр SI (цепочка-источник расположена в сегменте данных и адресуется регистром SI), а в команде LES - регистр DI (цепочка-получатель обязательно находится в дополнительном сегменте и адресуется регистром DI).

2.4.2. Арифметические команды

Арифметические команды (рис. 2.9) выполняются над целыми числами четырех типов: *беззнаковыми* и *знаковыми двоичными*, *упакованными* и *неупакованными десятичными*.

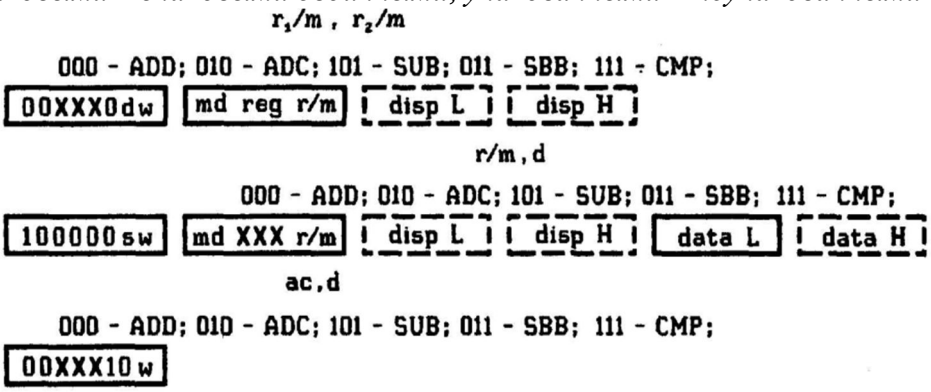

Рис. 2.9. Команды сложения и вычитания

Беззнаковые двоичные числа могут иметь длину 8 или 16 бит, каждый из которых является значащим, т. е. учитывается при определении значения числа. Это обеспечивает диапазон представления чисел 0 255 и 0 65 535 соответственно. Для таких чисел имеются команды сложения, вычитания, умножения и деления. Знаковые двоичные числа также могут содержать 8 или 16 бит, но значащими являются все биты, кроме старшего, который определяет знак числа: 0 положительное число, 1 отрицательное число.

Соответственно диапазоны значений чисел: от □128 до +128 и от □32 768 до +32 767. Число нуль содержит нули во всех разрядах и считается положительным и четным. Число, имеющее нули во всех разрядах, кроме знакового, равно $\square 2^7$ для 8-битовых чисел и $\square 2^{15}$ для 16-битовых.

Числа представляются в стандартном дополнительном коде, благодаря применению которого сложение и вычитание как знаковых, так и беззнаковых чисел выполняется с помощью одних и тех же команд (рис. 2.9). Для умножения и деления знаковых чисел предусмотрены специальные команды.

Команда **NEG** изменяет знак числа, находящегося в общем регистре или ячейке памяти, т. е. формирует его дополнительный код:

NEG dst; dst ← 0 – dst

Например, если однобайтовый операнд равен □1 (11111111), то команда NEG изменит его на +1 (00000001). Если операнд равен нулю, его значение не изменяется. Попытка изменить знак числа □ 128 (10000000) или $\square 2^{16}$ не модифицирует операнд, но устанавливает флаг переполнения OF. Отметим, что команда NEG отсутствует в ВМ80, поскольку в нем не предусмотрено специальных средств по обработке знаковых чисел.

Команды умножения и деления (рис. 2.10). В МП ВМ86 имеется по две команды умножения **(MUL и IMUL)** и деления **(DIV и IDIV),** выполняющие операции с беззнаковыми и знаковыми числами (в дополнительном коде) соответственно. Работа с десятичными числами требует использования специальных команд коррекции **AAM** и **AAD**.

Команды умножения

MUL (IMUL) src; ext:ac ← ac × src

выполняют умножение адресуемого операнда (общего регистра или ячейку памяти) и содержимого аккумулятора ac. При работе с байтами функции аккумулятора ac выполняет регистр AL, а функции его расширения (ext) — регистр AH, так что 16-битовое произведение формируется в регистре AX. Если перемножаются слова, то множимое располагается в регистре AX, функции расширения которого выполняет регистр DX, так что 32-битовый результат образуется в регистрах DX и AX.

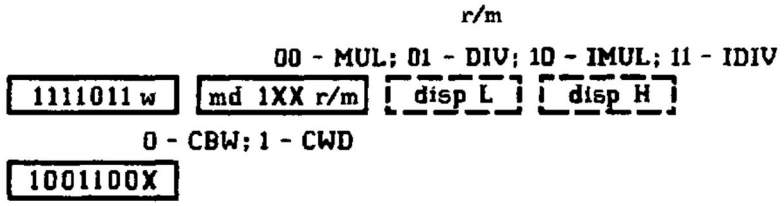

Рис. 2.10. Команды умножения и деления

Команды умножения беззнаковых и знаковых чисел осуществляют практически одни и те же действия и несколько различаются только влиянием на флаги OF и CF (значения остальных арифметических флагов после выполнения этих команд не определены, т. е. эти флаги могут принимать произвольные состояния). При выполнении команды MUL флаги OF и CF устанавливаются в единицу, если старшая половина произведения, находящаяся в регистрах AH или DX, отличается от нулевой, т. е. разрядность результата действительно превышает разрядность операндов. В противном случае эти флаги принимают нулевые значения. При выполнении команды IMUL флаги OF и CF устанавливаются в единицу, если старшая половина разрядов произведения не является расширением знака младшей половины, т. е. не содержит 00 (0000) или FF (FFFF) при умножении байтов (слов). Это означает, что в старшей половине находятся значащие цифры результата. В противном случае OF = CF = 0 и старшую половину разрядов произведения можно не сохранять.

Отсюда следует, что флаги OF и CF несут однотипную информацию о результатах выполнения команд MUL и IMUL.

Команды деления

DIV (IDIV) src; ac←quot (ext:ac/src)
ext←rem (ext:ac/src)

Команды деления производят деление содержимого аккумулятора и его расширения (AH:AL для 8-битового и DX:AX для 16-битового делителя) на содержимое регистра или ячейки памяти src. Частное quot формируется в регистре AL или AX, а остаток rem — в регистре AH или DX. Дробное частное округляется до целого путем отбрасывания дробной части результата. Состояния всех флагов не определены.

Если частное выходит за диапазоны представления чисел в байте (слове) или делитель является нулем, то автоматически генерируется прерывание по ошибке деления (тип 0), а частное и остаток не определены. В результате этого МП переходит к подпрограмме обработки прерывания, полный адрес которой CS:IP берется из ячеек ОЗУ 0000 и 0002.

В ряде случаев, например для предотвращения прерывания, может появиться необходимость до выполнения операции деления проверить возможность возникновения прерывания типа 0. Такая проверка осуществляется с помощью команд:

```
CMP ext, src;        сравнение и переход
JNB OVERFLOW;        по переполнению
```

осуществляющих сравнение старшей половины разрядов делимого с делителем и передающих управление по метке OVERFLOW, если левый операнд в команде CMP больше или равен правому (что является условием возникновения переполнения при делении).

Частное и остаток после выполнения команды IDIV всегда имеют одинаковые знаки. Например, при делении числа -47 на +3 из двух возможных результатов: -15 с остатком -2 и -16 с остатком +1 будет сформирован первый результат.

Однобайтовые команды преобразования разрядности операнда **CBW** и **CWD** примыкают к командам деления и осуществляют расширение со знаком операнда, который будет использоваться в качестве делимого. Обе команды не влияют на флаги и не изменяют значения операнда. Команда CBW (код операции 98) реализует преобразование байта в слово путем расширения (копирования) знака содержимого регистра AL в регистр AH. Команда CWD (код операции 99) осуществляет аналогичное преобразование слова в двойное слово путем передачи знака содержимого AX во все биты регистра DX.

Команды CBW и CWD удобно использовать для превращения делимого одинарной длины в делимое двойной длины, что может потребоваться для корректного выполнения команды IDIV. Выполнение команды CBW перед командой IDIV позволяет осуществлять деление 8-битовых чисел, а выполнение команды CWD — деление 16-битовых чисел.

Алгоритмы умножения и деления в МП BM86 реализованы не аппаратно, а в виде микропрограмм. Поэтому длительность команд MUL, IMUL, DIV, IDIV включает большое число тактов, причем длительность каждой команды зависит не только от разрядности операндов и расположения операнда src (в регистре или в памяти), но и от конкретных значений операндов в пределах, указанных в табл. 2.10 диапазонов длительности команд. В первом столбце в скобках указана длина сомножителей команд умножения и длина делителя (частного) для команд деления.

Таблица 2.10. Длительность выполнения команд умножения и деления в тактах

Тип операции (длина операнда)		MUL	IMUL	DIV	IDIV
r-r	(байт)	70-77	80-98	80-90	101-112
r-r	(слово)	118-133	128-154	144-162	165-184
r-m	(байт)	(76-83)+EA	(84-104)+EA	(86-96)+EA	(107-118)+EA
r-m	(слово)	(124-139)+EA	(134-160)+EA	(150-168)+EA	(171-190)+EA

2.4.3. Логические команды

К логическим командам (рис. 2. 11) относятся следующие: **AND** (конъюнкция, И), **OR** (дизъюнкция, ИЛИ), **XOR** (исключающие ИЛИ, сумма по модулю два), **TEST** (неразрушающая проверка, которая выполняет конъюнкцию операндов без изменения их значений, но с влиянием на флаги) и **NOT** (инверсия, НЕ).

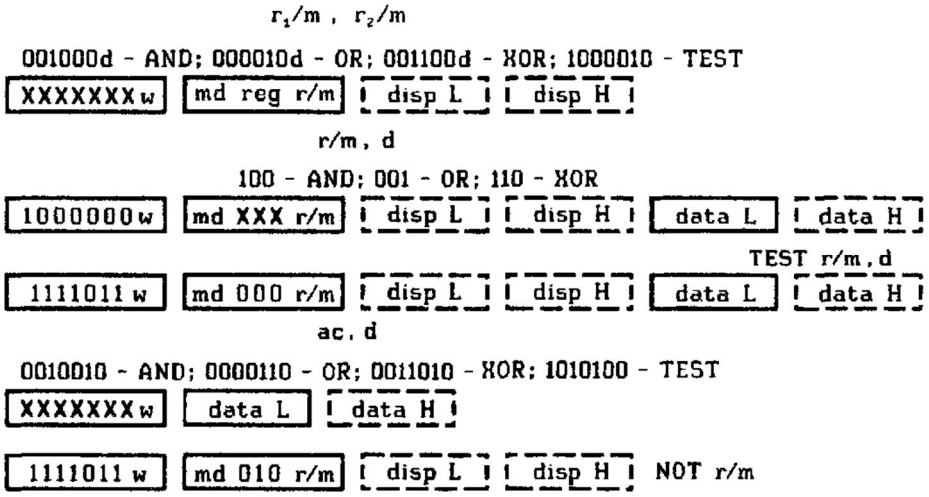

Рис. 5.11. Команды логических операций

Все логические операции выполняются поразрядно, т. е. каждый бит операндов обрабатывается независимо от других. Для этого в АЛУ микропроцессора включены наборы не связанных друг с другом логических элементов, содержащие по 16 двухвходовых элементов, И, ИЛИ и сумматоров по модулю два, а также набор из 16 инверторов.

2.4.4. Команды передачи управления

Команды передачи управления используются в разветвляющихся и циклических программах, а также при вызове подпрограмм и возврате из них.

Сегментная организация программной памяти определяет два основных типа команд передачи управления: внутрисегментные **NEAR** (близкие) и межсегментные **FAR** (далекие). При выполнении команд типа NEAR модифицируется только регистр IP, и адрес переходов представляется одним словом или даже байтом, если используется короткий вариант перехода с ограниченным диапазоном адресов. При выполнении команд типа FAR изменяется содержимое регистров IP и CS и адрес перехода представляется двумя словами (сегмент:смещение), что позволяет перейти в любую точку адресного пространства памяти.

В рассматриваемую группу команд входят *команды безусловных переходов*, *вызовов*, *возвратов*, *условных переходов*, *управления циклами* и *прерываний*.

Команды безусловных переходов JMP (рис. 2.12) производят модификацию регистра IP или регистров IP и CS без сохранения прежних, значений этих регистров. Имеется три

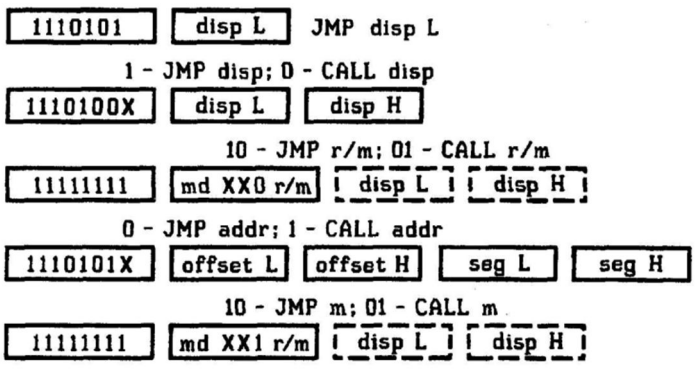

Рис. 2.12. Форматы команд JMP и CALL

формата (рис. 2.13) команды JMP типа NEAR, осуществляющих переход в пределах текущего кодового сегмента (внутрисегментный, близкий переход), и два формата команды JMP типа FAR, осуществляющих переход в любую точку адресного пространства (межсегментный, дальний переход).

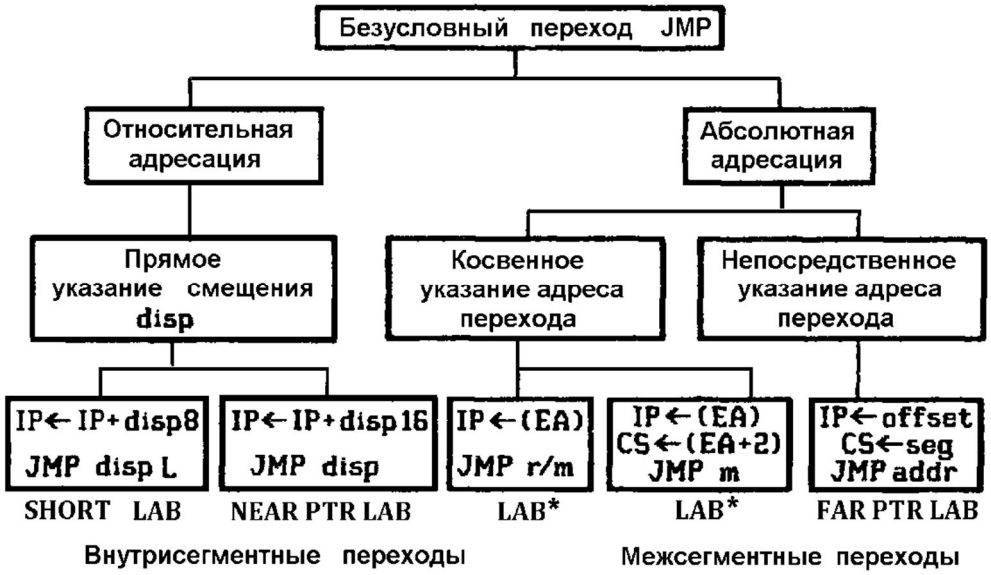

Рис. 2.13. Варианты формирования адресов переходов командой JMP.

Двухбайтовая команда **JMP disp L** во втором байте содержит смещение, которое интерпретируется как знаковое целое. Это смещение добавляется (с предварительным расширением знака до 16 бит) к содержимому IP, которое соответствует адресу команды, находящейся после данной команды JMP. Диапазон значений disp L составляет от □128 до +127, причем при положительном смещении осуществляется переход вперед, а при отрицательном — переход назад. Данная команда реализует так называемый короткий переход, который в мнемоническом обозначении отмечается указателем SHORT (например, JMP SHORT COUNT — короткий переход к метке COUNT). Она находит широкое применение, поскольку большинство переходов в прикладных программах осуществляется на небольшие расстояния.

Трехбайтовая команда **JMP disp** производит такое же действие, как и предыдущая команда, но содержит 16-битовое смещение disp (ассемблерный указатель NEAR PTR). Это смещение также интерпретируется как знаковое целое, принимающее значения от □32 768 до +32 767, что обеспечивает переход в любую точку текущего кодового сегмента. Напомним, что благодаря игнорированию переноса из бита 16, возникающего при сложении IP и disp, текущий сегмент рассматривается как кольцо. Поэтому диапазон переходов составляет от □IP до 2^{16} □ IP, а не от □2^{15} до 2^{15} □ 1. Если, например, IP = 8000 и смещение равно 7FFF или меньше, оно вызовет переход вперед, а смещение, равное 8000 или больше, вызовет переход назад.

Первые два рассмотренных формата команды JMP реализуют относительный способ адресации, который обеспечивает позиционную независимость программ. Следующие три формата определяют абсолютные адреса переходов и загружают регистр IP (а при межсегментных переходах и регистр CS) новыми значениями, не зависящими от прежних. Команда **JMP r/m** осуществляет косвенный внутрисегментный переход (ассемблерный указатель WORD PTR), при котором в регистр IP загружается содержимое регистра или ячейки памяти в соответствии с постбайтовым режимом адресации. Необязательные байты disp L, disp H в команде стандартно относятся к режиму адресации памяти.

Команда **JMP m** реализует косвенный межсегментный переход (ассемблерный указатель DWORD PTR) через содержимое двух ячеек памяти: слово из адресуемой с

помощью постбайта ячейки загружается в IP, а слово из следующей ячейки — в регистр CS. Если в постбайте указывается регистр (md = 11), то операция не определена.

Команда **JMP addr** реализует прямой межсегментный переход (ассемблерный указатель FAR PTR) и содержит 4 байта адреса перехода: два байта сегментного смещения offset загружаются в регистр IP, а два байта sr — в регистр CS.

Команды вызова подпрограммы **CALL** (см. рис. 2.12) имеют такие же форматы, как и команды JMP (кроме команды короткого перехода) и выполняются аналогичным образом, за исключением того, что автоматически запоминается адрес возврата (т. е. адрес команды, следующей за командой CALL). С этой целью при внутрисегментных вызовах в стеке запоминается содержимое IP, а при межсегментных вызовах — сначала содержимое IP, а затем CS. Напомним, что включение в стек каждого слова сопровождается уменьшением содержимого SP на два и что SP адресует последнюю заполненную 16-битовую ячейку стека. Использование стека для временного хранения адресов возврата позволяет легко реализовать правильный порядок возвратов из вложенных подпрограмм.

Перечислим действия, выполняемые 5-байтовой командой прямого межсегментного вызова CALL addr:

- содержимое регистра SP уменьшается на два;
- в адресуемую регистрами SP и SS ячейку памяти пересылается содержимое CS;
- содержимое SP уменьшается на два;
- в адресуемую регистрами SP и SS ячейку памяти засылается содержимое регистра IP;
- в IP загружается смещение offset;
- в CS загружается сегментный адрес seg.

В микропроцессоре BM86 отсутствуют команды условных вызовов подпрограмм (в отличие от BM80) и поэтому при необходимости условный вызов реализуется двумя командами. Например, если требуется вызвать подпрограмму SUBR при ненулевом результате операции, т. е. реализовать отсутствующую команду CNZ (команда МП BM80), то выполняется команда условного перехода, проверяющая противоположное условие, и осуществляется “перескок” через команду вызова CALL:

```
        JZ    SKIP
        CALL SUBR
SKIP: ...
```

В зависимости от того, как определен тип подпрограммы SUBR, ассемблер сформирует необходимую команду внутрисегментного вызова.

Команды возвратов (из подпрограмм) RET (рис. 2.14). Каждая подпрограмма должна

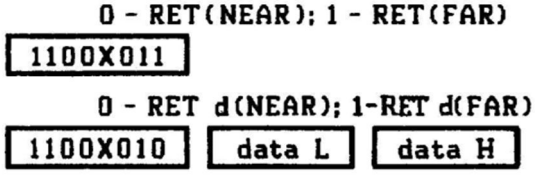

Рис. 2.14. Команды возврата из подпрограмм

содержать хотя бы одну команду возврата RET, которая возвращает управление программе, осуществившей вызов. Такая передача управления осуществляется путем извлечения из стека адреса возврата, включенного в него командой вызова подпрограммы. Поэтому команды возврата не содержат никакой адресной информации и неявно адресуют вершину стека. Тип команды возврата (внутрисегментный NEAR или межсегментный FAR) выбирается в соответствии с типом команды CALL, осуществившей вызов данной подпрограммы.

Однобайтовая команда внутрисегментного возврата RET (код операции C3) выполняет

следующие действия: слово из вершины стека загружается в IP, а содержимое SP увеличивается на два.

Однобайтовая команда межсегментного возврата RET (код операции CB) осуществляет следующие действия: слово из вершины стека передается в IP; производится инкремент SP на два; слово из следующей ячейки стека передается в CS; производится инкремент SP на два. В результате в регистрах IP и CS оказывается полный адрес возврата, а SP адресует новую вершину стека.

Две трехбайтовые команды RET (внутрисегментный возврат с кодом операции C2 и межсегментный возврат с кодом CA) содержат два байта данных, интерпретируемых как беззнаковое целое. Они производят такие же действия, как и соответствующие однобайтовые команды возврата, но дополнительно прибавляют содержащиеся в них данные к указателю стека SP (после извлечения из стека адреса возврата).

Эти команды упрощают возврат из тех подпрограмм, параметры которых передаются в стеке. Прибавление к SP числа, равного числу байтов в блоке параметров, эквивалентно удалению этого блока из стека.

Команды условных переходов (рис. 2.15). Имеется 18 команд условных переходов, которые представлены единым двухбайтовым форматом, позволяющим осуществлять короткие (в пределах от □128 до +127) переходы относительно указателя команд IP. При выполнении этих команд анализируется некоторое условие, соответствующее текущим состояниям отдельных флагов (кроме флага AF) или некоторым комбинациям флагов (табл. 2.11). Если условие выполнено, то осуществляется переход и однобайтовое смещение, расширенное со знаком до 16 бит, добавляется к содержимому IP. Если условие не выполнено, выполняется следующая по порядку команда. Время выполнения команд составляет восемь тактов в первом случае и четыре такта во втором.

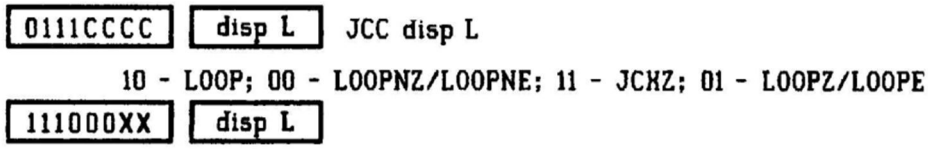

Рис. 2.15. Форматы команд условных переходов и управления циклами

Команды условных переходов обеспечивают ограниченный диапазон переходов, для расширения которого необходимо сочетать их с командами безусловных переходов. Большинство команд условных переходов имеет по два мнемонических обозначения, которые подчеркивают содержательный смысл проверяемого условия и введены для удобства программирования. Обобщенное мнемоническое обозначение команд имеет вид **Jcond label,** где cond соответствует проверяемому условию (первая графа табл. 2.11), двоичный код которого CCCC (вторая графа табл. 2.11) помещается в соответствующее поле первого байта формата команды. Операнд label обозначает команду, к которой осуществляется переход. В качестве label может использоваться метка, а также метка плюс/минус константа или выражение, вычисление которого дает константу.

Команды условных переходов позволяют проверить все отношения между знаковыми и беззнаковыми числами. В мнемокодах этих команд при сравнении знаковых чисел для обозначения условия “больше” используется буква G (Greater), а для обозначения “меньше” — буква L (Less). Для аналогичных условий при сравнении беззнаковых чисел используются соответственно буквы A (Above) и B (Below). Условие равенства обозначается буквой E (Equal), а невыполнение некоторого условия — буквой N (Not).

Первые восемь команд в табл. 2.11 предназначены в основном для использования после команд сравнения CMP, в которых второй операнд вычитается из первого.

В результате сравнения двух операндов и последующего перехода в зависимости от

состояния флагов реализуются решения, соответствующие операторам отношений <, >, ≤, ≥, ≠, =. Последние восемь команд можно использовать в любой ситуации, когда в зависимости от значения указанного в табл. 2.11 флага следует предпринять одно из двух действий.

Таблица 2.11

Мнемокод условия	Код CCCC	Условие перехода: Логика	Условие перехода: Признак
		Для чисел со знаком	
G/NLE	1111	Больше/не меньше и не равно	(SF ⊕ OF) ∨ ZF = 0
NG/LE	1110	Не больше/меньше или равно	(SF ⊕ OF) ∨ ZF = 1
L/NGE	1100	Меньше/не больше и не равно	SF ⊕ OF = 1
NL/GE	1101	Не меньше/больше или равно	SF ⊕ OF = 0
		Для чисел без знака	
A/NBE	0111	Больше/не меньше и не равно	CF ∨ ZF = 0
NA/BE	0110	Не больше/меньше или равно	CF ∨ ZF = 1
B/NAE/C	0010	Меньше/не больше и не равно	CF = 1
NB/AE/NC	0011	Не меньше/больше или равно	CF = 0
		Для прочих данных	
E/Z	0100	Равно/по нулю	ZF = 1
NE/NZ	0101	Не равно/по не нулю	ZF = 0
S	1000	Отрицательный результат	SF = 1
NS	1001	Положительный результат	SF = 0
O	0000	Есть переполнение	OF = 1
NO	0001	Нет переполнения	OF = 0
P/PE	1010	Четно	PF = 1
NP/PO	1011	Нечетно	PF = 0

Команды условных переходов реализуют относительную адресацию, как и первые два формата команды безусловного перехода JMP. При этом встает задача определения конкретного значения смещения dispL, которое должно быть добавлено к регистру IP для осуществления перехода к указанной метке. Решение этой задачи рассмотрим на примере следующего фрагмента программы:

```
0050 AGAIN:    INC CX
0052           ADD AX, [BX]
0054           JNS AGAIN
0056           MOV RESULT, CX
```

В левом столбце приведены эффективные адреса команд, т. е. содержимое IP при выборке каждой команды. Вычисление 0050 – 0056 = □6 показывает, что при ассемблировании команды JNS следует установить смещение dispL = FA, соответствующее дополнительному коду числа □6.

Команды управления циклами (см. рис. 2.15) используются для удобства реализации вычислительных циклов (итераций). Они осуществляют условный переход в зависимости от состояния регистра CX, который выполняет функции счетчика циклов. Команды **LOOP, LOOPE** и **LOOPNE**, кроме того, декрементируют регистр CX перед выполнением условного перехода, что исключает использование соответствующей команды декремента.

Как и в командах условных переходов, в данном случае при выполнении условия управление передается команде, расположенной в диапазоне адресов от □128 до +127, задаваемых смещением в байте 2 команды.

Команда LOOP, которая обычно ставится в конце цикла, осуществляет декремент CX □ 1 и если CX≠0, то добавляет смещение dispL к регистру IP; в противном случае (CX = 0 и цикл окончен) выполняется следующая по порядку команда. Таким образом, смысл этой команды состоит в повторении цикла CX раз, т. е. в выполнении переходов до тех пор, пока CX ≠ 0.

Команда **JCXZ** имеет противоположный смысл и осуществляет переход только при CX = 0. Ее удобно использовать в начале цикла, особенно в ситуации, когда цикл может не выполняться ни разу. Если команда LOOP выполняет постпроверку содержимого CX, то команда JCXZ выполняет его предпроверку.

Фактически команда LOOP label эквивалентна двум командам

DEC CX

JNZ label

и ее использование кроме удобства программирования позволяет экономить один байт объектного кода и один такт Т. При большом числе повторений цикла экономия времени может оказаться значительной.

Команда LOOPE/LOOPZ по сравнению с командой LOOP вводит дополнительное условие повторения цикла: ZF = 1. Это позволяет выходить из вычислительного цикла, как по окончании заданного числа циклов, так и при получении ненулевого результата (или неравенстве сравниваемых операндов). Команда LOOPNE/LOOPNZ вводит дополнительное условие противоположного смысла: ZF = 0, которое позволяет прекращать цикл при получении нулевого результата (или по равенству сравниваемых операндов).

2.4.5. Команды обработки цепочек.

Имеется пять однобайтовых команд (рис. 2.16), предназначенных для обработки

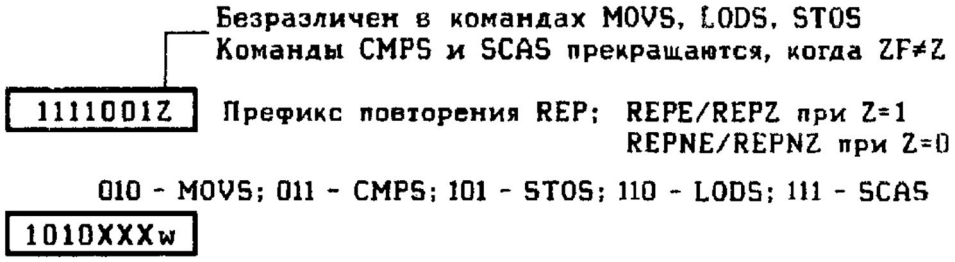

Рис. 2.16. Команды обработки цепочек

одного элемента цепочки. Напомним, что под цепочкой (строкой) понимается последовательность любых контекстно связанных байтов или слов, находящихся в смежных ячейках памяти. Цепочечной команде может предшествовать однобайтовый префикс повторения REP, который вызывает повторение действия команды над следующим элементом. Это позволяет обрабатывать цепочки значительно быстрее, чем при организации программного цикла. Повторение рассчитано на максимальную длину цепочки 64 Кбайт и может заканчиваться по одному или двум условиям. Требуемое число повторений указывается в регистре CX, содержимое которого автоматически декрементируется при каждом повторении операции. Когда содержимое CX становится равным нулю, МП выполняет следующую команду. При выполнении повторяющейся цепочечной команды МП воспринимает внешние прерывания.

В качестве операндов команды могут иметь цепочку-источник, цепочку-получатель или то и другое одновременно. Цепочка-источник по умолчанию размещается в текущем сегменте данных, и все элементы адресуются регистром SI. Имеется возможность использовать другой сегмент, указав его в префиксе смены сегмента. Цепочка-получатель всегда размещается в дополнительном сегменте, и ее элементы адресуются регистром DI.

Для начальной загрузки адресов в регистры можно использовать команды LEA, LDS и LES.

При выполнении цепочечной команды содержимое SI и DI автоматически изменяется на ±1 (при обработке байта) или на ±2 (при обработке слова), чтобы адресовать следующие элементы цепочек. Флаг DF определяет направление изменения адресов: автоинкремент (DF = 0) или автодекремент (DF = 1).

Каждая цепочечная команда способна оперировать как с байтами, так и со словами, причем тип элементов цепочек ассемблер может определить по атрибутам операндов. Однако многие версии ассемблера допускают указание типа элементов путем добавления букв B (байт) или W (слово) к мнемоническим обозначениям цепочечных команд: MOVSB или MOVSW, CMPSB или SMPSW и т. д. При этом операнды в команде становятся необязательными (ассемблер их игнорирует, так как адресация цепочек известна заранее) и могут указываться только для удобства программирования.

Префикс повторения (см. рис. 2.16) обеспечивает повторяющееся (циклическое) выполнение цепочечной команды. При отсутствии префикса команда оперирует только одним элементом цепочки. Префикс повторения имеет пять мнемокодов: REP/REPE/REPZ и REPNE/REPNZ, которые определяют только два объектных кода, различающихся однобитным полем z, и введены для лучшего отражения содержательного смысла команды. Префикс повторения не влияет на флаги. Он обеспечивает автоматическое декрементирование регистра CX на каждом повторении команды и проверку содержимого CX на нуль. Префикс REP используется с командами пересылки MOVS и STOS и инициирует действие “повторять, пока не будет достигнут конец цепочки”, т. е. по ка CX≠0. Префиксы REPE и REPZ действуют аналогично и имеют такой же машинный код, как и префикс REP. Они используются с командами сравнения CMPS и SCAS и оперируют с флагом ZF, состояние которого определяется результатом исполнения этих команд, причем появление ZF = 1 останавливает процесс повторения команд. Префиксы REPNE и REPNZ также применяются командами CMPS и SCAS, но используют противоположное условие ZE = 0. Наличие дополнительных проверяемых условий позволяет окончить выполнение этих команд по результату вычислений до завершения перебора всех элементов цепочки. Цепочечные команды пересылки MOVS, LODS и STOS на флаги не влияют, а команды неразрушающего сравнения COMS и SCAS устанавливают значения флагов в соответствии с разностью операндов, сохраняя значения самих операндов. Время выполнения цепочечных команд (в тактах синхронизации) указано в табл. 2.12, где n - число повторений, которое предварительно заносится в регистр CX.

Таблица 2.12. Время выполнения цепочечных команд

Команда	Время выполнения команды	
	без префикса повторения	с префиксом повторения
MOVS	18	9 + 17n
CMPS	18	9 + 17n
SCAS	15	9 + 15n
LODS	12	9 + 13n
STOS	11	9 + 10n

Команды с префиксом повторения при обработке длинных цепочек могут выполняться в течение значительного времени. Поэтому для сокращения времени реакции на внешние прерывания МП воспринимает внешние сигналы прерывания после обработки каждого элемента цепочки. (Сигнал INTR = 1 должен установиться не более чем за один такт до окончания цикла обработки).

После возврата из прерывания операция возобновляется, как обычно, с точки прерывания. Однако во время обработки прерывания МП помнит действие только одного префикса, непосредственно предшествующего команде. Поэтому, если с цепочечной

командой используется несколько префиксов (префикс замены сегмента, префикс блокировки), необходимо запрещать прерывания на время ее выполнения.

Обобщенные обозначения цепочечных команд и выполняемые ими действия при различных префиксах приведены в табл. 2.13.

Таблица 2.13. Обобщенные обозначения цепочечных команд

Команда, её действие	Префикс повторения	Описание действия
MOVS dst,src; dst ← src	-	Пересылка из цепочки src, адресуемой регистром SI (сегмент данных) в цепочку dst, адресуемую регистром DI (дополнительный сегмент)
	REP	Блоковая пересылка память-память
CMPS dst,src; src - dst	-	Вычитание элемента цепочки dst из элемента цепочки src (неразрушающее сравнение)
	REPE/REPZ	Сравнение элементов до обнаружения одинаковых элементов или до окончания цепочек
	REPNE/REPNZ	Сравнение элементов до обнаружения различающихся элементов или до окончания цепочек
SCAS dst; ac - dst	-	Вычитание элемента цепочки dst, адресуемой регистром DI, из содержимого AL или AX (неразрушающее сравнение)
	REPE/REPZ	Поиск элемента цепочки со значением, отличающимся от ac
	REPNE/REPNZ	Поиск элемента цепочки со значением, совпадающим с ac
LODS src; ac ← src	-	Загрузка аккумулятора элементом цепочки, адресуемым регистром SI (префикс повторения обычно не используется)
STOS dst; dst ← ac	-	Запоминание содержимого аккумулятора AL или AX в элементе цепочки, адресуемом регистром DI
	REP	Инициализация цепочки на фиксированное значение, содержащееся в аккумуляторе

Команду CMPS удобно применять для нахождения одинаковых (REPE CMPS) или различающихся (REPNE CMPS) элементов цепочек. Достижение конца цепочки при выполнении этих команд соответственно означает, что цепочки полностью различны (не содержат одинаковых элементов) или полностью совпадают (не содержат различающихся элементов). Команда SCAS производит сравнение элемента цепочки с некоторым эталоном, содержащимся в аккумуляторе. Вариант REPE SCAS осуществляет просмотр цепочки до тех пор, пока значение элемента не отличается от эталона или не достигается конец цепочки, а вариант REPNE SCAS — просмотр цепочки до обнаружения элемента, равного эталону, или до достижения конца цепочки.

Команду LODS удобно использовать в программных циклах вместо двух команд: MOV, ac, src и INC SI (или DEC SI в зависимости от направления продвижения по цепочке). Отметим, что команда LODS оперирует элементами цепочки, расположенной в сегменте данных, причем имеется возможность замены сегмента, а команда STOS оперирует элементами цепочки, которая всегда находится в дополнительном сегменте. В последнем случае префикс замены сегмента не указывается, а при его наличии — игнорируется.

Контрольные вопросы

1. Какие группы команд относятся к командам пересылки данных?
2. Охарактеризуйте варианты команды MOV.
3. Каковы особенности обмена с сегментными и стековыми регистрами?
4. Охарактеризуйте команду XCHG.
5. Охарактеризуйте команду XLAT.
6. Каковы особенности команд LEA, LDS, LES?
7. Охарактеризуйте команду LEA.
8. Охарактеризуйте команду LDS.
9. Охарактеризуйте команду LES.
10. Где используются команды LDS и LES?
11. Какие команды относятся к арифметическим?
12. Охарактеризуйте числа, с которыми работают арифметические команды.
13. Охарактеризуйте команду SBB.
14. Охарактеризуйте команды умножения.
15. Охарактеризуйте команды деления.
16. Охарактеризуйте команды CBW и CWD.
17. Как в BM86 реализованы алгоритмы умножения и деления?
18. Какие команды относятся к логическим и как они реализуются в АЛУ?
19. Дайте общую характеристику командам передачи управления.
20. Дайте общую характеристику командам безусловных переходов JMP.
21. Охарактеризуйте команды JMP disp L и JMP disp.
22. Охарактеризуйте команды JMP r/m и JMP m.
23. Охарактеризуйте команды вызова подпрограммы CALL.
24. Дайте общую характеристику командам возврата из подпрограмм RET.
25. Охарактеризуйте однобайтные команды внутрисегментного возврата RET.
26. Охарактеризуйте трехбайтовые команды RET.
27. Какие команды относятся к командам условных переходов?
28. Дайте общую характеристику командам условных переходов.
29. Каковы мнемокоды условия команд Jcc для чисел со знаком?
30. Каковы мнемокоды условия команд Jcc для чисел без знака?
31. Каковы мнемокоды условия команд Jcc для прочих данных?
32. Какая команда часто предшествует командам Jcc с мнемокодом условия для чисел со знаком и без знака?
33. Охарактеризуйте команды управления циклами.
34. Какие команды относятся к командам обработки цепочек?
35. Дайте общую характеристику командам обработки цепочек.
36. Охарактеризуйте операнды командам обработки цепочек.
37. Охарактеризуйте префиксы повторения командам обработки цепочек.
38. Охарактеризуйте контент таблицы 2.13.

3. Арифметический сопроцессор К1810ВМ87 (i8087)

3.1. Общие сведения и технические характеристики

Микросхема К1810ВМ87 представляет собой однокристальный 80-битовый арифметический сопроцессор (АСП), выполненный по высококачественной n-МОП технологии. Кристалл микросхемы с геометрическими размерами 5,5×5,5 мм содержит около 65 000 транзисторов и потребляет мощность не более 3 Вт от источника питания напряжением +5 В. Микросхема выпускается в 40-выводном корпусе. Синхронизируется однофазными импульсами с частотой повторения 2 - 5 МГц от внешнего тактового генератора.

Сопроцессор К1810ВМ87 (обозначаемый далее для краткости ВМ87) может быть использован только совместно с центральным процессором ВМ86/ВМ88, так как в нем (ВМ87) отсутствует механизм выборки команд. Сопроцессор предназначен для повышения производительности центрального процессора в среднем в 100 раз при выполнении операций с многоразрядными целыми и вещественными числами. Совмещение ЦП ВМ86 с сопроцессором ВМ87 сводится к простым соединениям соответствующих выводов без использования дополнительных интегральных схем (ИС).

Сопроцессор содержит четыре 16-битовых, один 64-битовый и восемь 80-битовых регистров. Он имеет 16-битовую шину данных для связи с ОЗУ и портами ввода/вывода, отображенными на память. Шина адреса имеет 20 линий, что позволяет при передаче данных непосредственно адресоваться к памяти емкостью до 1 Мбайт. Для экономии числа выводов младшие 16 адресных линий мультиплексированы во времени с линиями данных и составляют единую шину адреса/данных. Четыре старшие адресных линии аналогично мультиплексированы с линиями состояния АСП. Чтобы сигналы этих линий можно было использовать в системе, их обязательно разделяют с помощью тех же внешних ИС, которые используются ЦП.

Система команд АСП ВМ87 содержит 68 мнемокодов и позволяет совершать операции над целыми двоичными, двоично-десятичными и вещественными числами в широком диапазоне значений. Имеется набор арифметических команд, включающих сложение, вычитание, умножение и деление, который ориентирован на обработку чисел с высокой точностью. Весьма разнообразны команды пересылки данных, специальные вычислительные команды, такие, как вычисление квадратного корня, масштабирование, нахождение тангенса, логарифма и др.

Использование АСП ВМ87 совместно с ЦП ВМ86 расширяет систему команд последнего до 159 мнемокодов.

3.2. Структурная схема АСП

Структурная схема (рис. 3.1) содержит две относительно независимые части: *операционное устройство*, которое выполняет операции, заданные командой, и *устройство шинного интерфейса*, которое получает команды и помещает их в очередь команд, осуществляет считывание операндов из памяти и преобразование их в формат ВВ (см. рис. 3.6), а также запись результатов в память с обратным преобразованием в требуемый формат. Оба устройства могут работать параллельно, что обеспечивает совмещение во времени процессов передачи и обработки данных.

3.2.1. Операционное устройство АСП

Операционное устройство АСП содержит *группу арифметических регистров*, *модули обработки порядка и мантиссы*, *ПЗУ констант*, *регистр этикеток (TAG)* и *блок управления*.

Группа арифметических регистров (АР) *включает* восемь 80-битовых регистров, организованных в стек. Со стеком связан 3-битовый указатель стека ST, содержимое которого определяет номер одного из восьми арифметических регистров, являющегося вершиной стека. Нумерация регистров начинается всегда от вершины (рис. 3.2), и регистры

обозначаются через ST(0), ST(1),..., ST(7). Группа арифметических регистров служит для хранения обрабатываемых данных, представленных в формате ВВ. При загрузке данных в арифметический регистр содержимое указателя стека ST предварительно уменьшается ST ← ST □ 1 и указывает номер регистра, в который производится загрузка. При извлечении данных, после запоминания извлеченного значения в памяти, содержимое ST автоматически увеличивается на единицу ST ← ST + 1. С группой арифметических регистров связан регистр этикеток TAG, в котором каждому АР ставится в соответствие 2-битовое поле. В этом поле автоматически формируется код, характеризующий загруженное в АР значение, в соответствии с табл. 3.1.

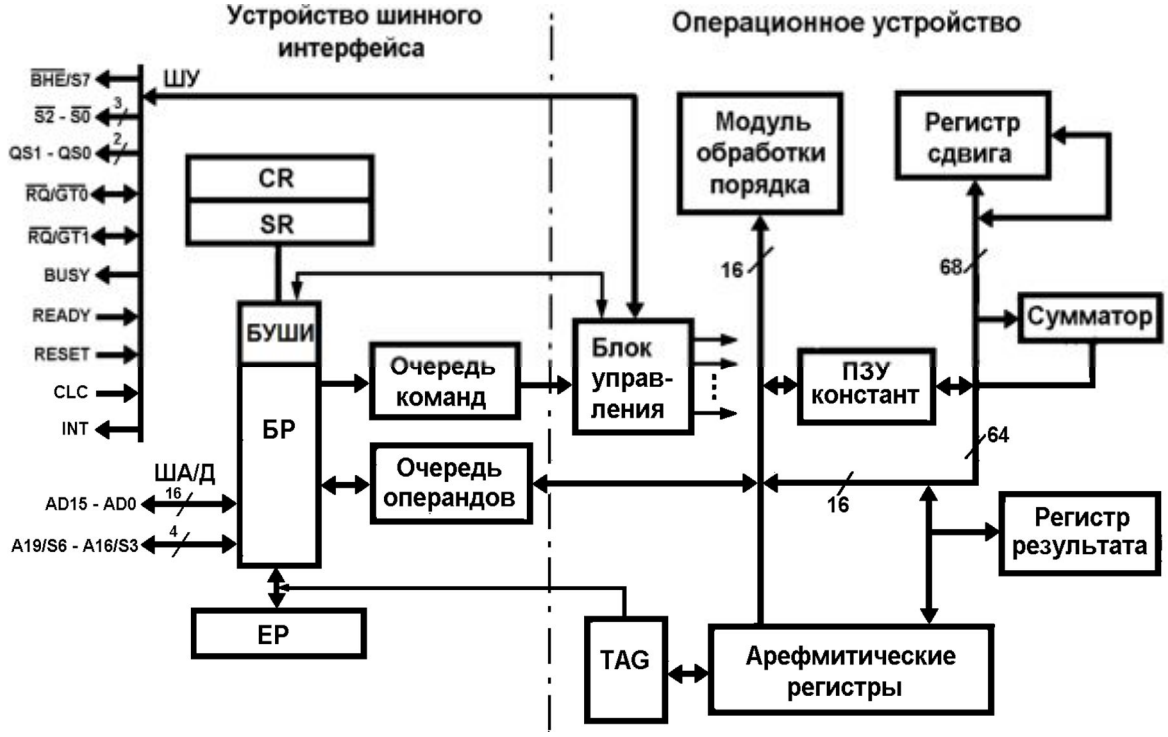

Рис. 3.1. Укрупненная структурная схема АСП ВМ87 (i8087)

Модуль обработки мантиссы состоит из 68-битового сумматора, регистра сдвига и регистра результата. Он выполняет заданные операции над мантиссами операндов и формирует признаки различных исключительных ситуаций, возникающих при обработке данных.

Модуль обработки порядка осуществляет действие над значениями поля порядка исходных операндов в соответствии с командой.

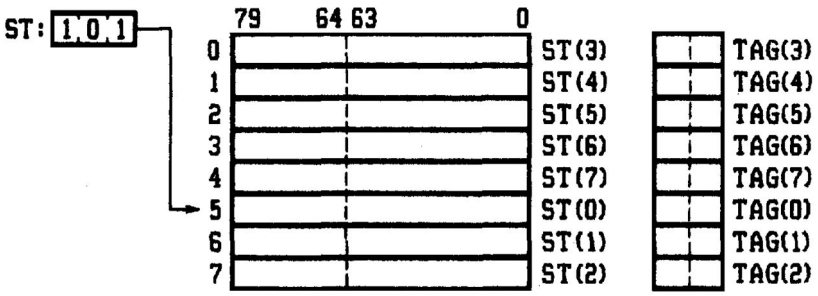

Рис. 3.2. Нумерация арифметических регистров (при ST = 5) и соответствующих полей регистра

ПЗУ констант хранит семь различных числовых констант, которые часто используются в вычислительных программах: +0.0; +1,0; π; lg2; ln2; $\log_2 10$ и $\log_2 e$. Все константы

представлены в формате ВВ.

Блок управления связан с выходом очереди из байтов команд и построен по микропрограммному принципу. Его назначение состоит в дешифрации очередной команды, поступающей на исполнение из устройства шинного интерфейса, и формировании управляющих сигналов, определяющих режим работы модулей обработки и других устройств АСП.

Таблица 3.1. Значения полей регистра этикеток TAG

Код поля TAG(i)	Содержимое регистра ST(i)
00	Конечное число, не равное нулю
01	Нуль
10	NAN/±∞
11	Регистр пуст (обозначается e)

3.2.2. Устройство шинного интерфейса

Устройство шинного интерфейса содержит *средства подключения к локальной шине* (интерфейсу) микропроцессорной системы выполненной на базе МП86, *буферный регистр* (БР), *регистры очереди команд и операндов*, *блок управления шинным интерфейсом* (БУШИ) и *группу вспомогательных регистров*.

Сигналы, использующиеся при подключении к системному интерфейсу:

AD15 — AD0 — входы/выходы для формирования адресов и передачи данных. В течение первой части цикла шины (T1) они содержат адрес, а в остальной части цикла (T2, T3, TW, T4) по ним вводятся или выводятся данные. Когда шиной управляет ЦП, выводы AD15 — AD0 являются входами (описание циклов шины см. в [1]).

A19/S6 — A16/S3 — выходы для формирования четырех старших разрядов адреса в течение первой части цикла шины (T1), в остальной части цикла (T2, T3, TW, T4) имеют постоянные значения S3 = S4 = S6 = 1, S5 = 0. Когда шиной управляет ЦП, эти выводы являются входами (рис. 3.1).

/S7 выходной сигнал разрешения старшего байта шины данных. Значение = 0 устанавливается в такте T1 при чтении или записи данных с использованием старших разрядов шины данных D15 — D8. Если разряды D15 — D8 при передаче данных не используются, то = 1. В остальной части цикла шины (T2, T3, TW, T4) действует выходной сигнал S7 = 0. Когда сопроцессор не управляет шиной, данный вывод используется как вход.

— — выходные сигналы кода состояния сопроцессора; = 101 — чтение из памяти; = 110 — запись в память; = 111—пассивное состояние сопроцессора; остальные комбинации значений не используются.

/ — вход/выход используется для запроса/предоставления доступа к локальной шине ЦП, когда сопроцессору требуется переслать операнд.

/ — вход/выход, сигнал запроса/предоставления шины для связи сопроцессора с другим процессором, использующим локальную шину.

QS1, QS0 — входные сигналы кода состояния очереди команд. Они позволяют сопроцессору следить за состоянием очереди команд ЦП с тем, чтобы синхронизировать начало выполнения очередной команды.

INT — выходной сигнал запроса прерывания.

BUSY — выходной сигнал занятости; сигнал BUSY = 1 указывает на то, что сопроцессор выполняет команду. Этот вывод соединяется с выводом TEST ЦП, обеспечивая тем самым синхронизацию его работы и сопроцессора.

READY — входной сигнал готовности от внешних устройств.

RESET — входной сигнал сброса (начальной установки), устанавливающий сопроцессор в начальное состояние. Длительность сигнала RESET = 1 должна составлять не менее четырех периодов CLK.

CLK — входной сигнал тактовой частоты от генератора тактовых импульсов К1810ГФ84, осуществляющий синхронизацию работы сопроцессора. Допустимый диапазон частот 2 - 5 МГц.

(Более подробное описание назначения сигналов можно найти и в [1] и др.)

Таким образом, все выводы сопроцессора ВМ87 (за исключением BUSY и INT) имеют обозначение, идентичное выводам ВМ86. При подключении АСП к ВМ86 одноименные выводы соединяются между собой, выход BUSY подключается к входу TEST ЦП, а выход INT — к одному из входов программируемого контроллера прерываний К1810ВН59А, используемого в системе.

Буферный регистр служит для мультиплексации и демультиплексации адресной информации, данных и состояния при операциях ввода/вывода выполняемых сопроцессором.

Регистры очереди команд и операндов служат для промежуточного хранения их значений.

Блок управления шинным интерфейсом обеспечивает параллельную и асинхронную работу устройства шинного интерфейса и операционного устройства.

Группа вспомогательных регистров включает 16-битовый *регистр состояния* SR**,** 16-битовый *регистр управления* CR и 64-битовый регистр — *указатель исключительных ситуаций* EP.

Регистр состояния SR (рис. 3.3) содержит 13 полей, разряды которых при функционировании АСП автоматически устанавливаются в соответствии с выполняемыми действиями или результатами действий. Поле **B** регистра **SR** (разряд 15) устанавливается в единицу, когда сопроцессор выполняет команду. Значение этого поля выдается на выход BUSY и осуществляет синхронизацию действий ЦП и АСП, «удерживая» ЦП в режиме ожидания. Разряды поля **ST** устанавливаются в соответствии с номером арифметического регистра, который является вершиной стека. Уменьшение или увеличение значения ST на единицу осуществляется автоматически при выполнении команд загрузки и запоминания соответственно. Разряды полей **C3 — C0** содержат значения флагов, характеризующих результаты выполнения различных команд (табл. 3.2). Разряды полей **PE, UE, OE, ZE, DE, IE** являются флагами исключительных ситуаций, возникающих в результате выполнения команд.

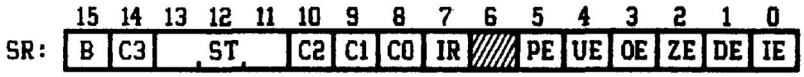

Рис. 3.3. Регистр состояния АСП ВМ87

Флаг **IE** — *недействительная операция* — устанавливается в единицу при переполнении стека, извлечения из пустого стека, выполнения действий, приводящих к неопределенным значениям типа ∞/∞, $\infty\times0$, $\sqrt{}$ или с операндом NAN (Not a Number) (см. раздел 3.3). Если флаг IE замаскирован, результат недействительной операции формируется следующим образом. Когда один из операндов равен NAN, то результатом является NAN; когда оба операнда NAN, результатом является NAN с б льшим абсолютным значением. Если ни один из операндов не является NAN, результатом является код неопределенности.

Флаг **DE** — *денормализованный операнд* (см. раздел 3.3) — устанавливается в единицу, когда хотя бы один из операндов денормализован. Если флаг DE замаскирован и операнд находится в памяти, то сопроцессор оперирует с денормализованным операндом как с нормализованным. Когда денормализованный операнд находится в регистре, сопроцессор предварительно преобразует его в ненормализованный путем сдвига мантиссы и изменения порядка.

Флаг **ZE** — *деление на нуль* — устанавливается в единицу, когда делимое — конечное число, не равное нулю, а делитель равен нулю. Если флаг ZE замаскирован, то в качестве результата выдается код ∞ со знаком, равным сумме по модулю два знаков операндов.

Флаг **OE** — *переполнение* — устанавливается в единицу, когда результат операции не может быть представлен конечным числом, т. е. его значение превышает $1{,}2\times10^{4932}$. Если флаг OE замаскирован, то в качестве результата выдается код ∞.

Флаг **UE** — *антипереполнение* — устанавливается в единицу, когда результат операции не может быть представлен нормализованным конечным числом. Если флаг UE замаскирован, то в качестве результата выдается денормализованное число.

Флаг **PE** — *неточный результат* — устанавливается в единицу, когда значение результата не может быть представлено точно в том формате, который определен командой. Результат в этом случае округляется в соответствии с заданным режимом округления. Если флаг PE замаскирован, сопроцессор округляет результат, не вызывая прерывания.

Разряд SR(7) = **IR** содержит флаг запроса прерывания, который устанавливается в «1» при возникновении какой-либо незамаскированной (из перечисленных выше) исключительной ситуации. Значение флага IR формируется на выходе INT сопроцессора. Флаг IR может быть замаскирован программным путем.

После инициализации BM87 все биты регистра состояния, за исключением C0, C1, C2, C3, устанавливаются в нуль.

Таблица 3.2. Значения разрядов C0 □ C3

Команды	C3C2C1C0	Результат
Сравнение чисел	0 0 x 1	ST(0) > X
	0 0 x 1	ST(0) < X
	1 0 x 0	ST(0) = X
	1 1 x 1	ST(0) несравнимо с X
Анализ числа, находящегося в вершине стека	0 0 0 0	+ ненормализованное
	0 0 0 1	+ NAN
	0 0 1 0	- ненормализованное
	0 0 1 1	- NAN
	0 1 0 0	+ нормализованное
	0 1 0 1	+∞
	0 1 1 0	- нормализованное
	0 1 0 1	-∞
	1 0 0 0	+0
	1 0 0 1	e
	1 0 1 0	-0
	1 0 1 1	e
	1 1 0 0	+ денормализованное
	1 1 0 1	e
	1 1 1 0	- денормализованное
	1 1 1 1	e
Получение частичного остатка	$r_0$0 r_1r_2	команда завершена
	x 1 x x	команда не завершена

Примечания: 1. ST(0) - число в вершине стека.
2. X - число в памяти или ST(i).
3. e - пустая вершина.
4. r_0 - r_2 - три младших бита частного.

Регистр управления CR (рис. 3.4) содержит десять полей, которые служат для задания требуемого режима работы АСП программным путем. Различают две основные группы режимов — вычислений и обработки исключительных ситуаций. Режимами вычислений управляют поля старшего байта регистра CR, а режимами обработки исключительных ситуаций — поля младшего байта.

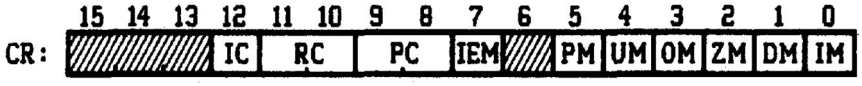

Рис. 3.4. Регистр управления АСП BM87

Разряд CR (12) = **IC** — *управление бесконечностью.* IC = 0 определяет, что + ∞ и □ ∞ считаются одной беззнаковой бесконечностью; IC = 1 — определяет обычную ситуацию, когда + ∞ и □ ∞ считаются двумя знаковыми бесконечностями. Первый режим (IC = 0) носит название «проективная арифметика», а второй (IC = 1) — «аффинная арифметика».

Поле CR (11, 10) = **RC** — *управление округлением* — определяет способ округления в соответствии с табл. 3.3. Первый режим RC = 00 является обычным способом округления. Режимы RC = 01 и 10 используются для определения граничных левых и правых значений соответственно. В режиме RC = 11 осуществляется усечение (отбрасывание) лишних битов результата.

Таблица 3.3. Управление округлением

Код поля RC	Способ округления результата
00	К ближайшему значению
01	По направлению к -∞
10	По направлению к +∞
11	По направлению к нулю

Поле CR (9, 8) = **PC** — управление точностью позволяет определить формат представления результатов вычислений в соответствии с табл. 3.4 и задать один из трех режимов работы. Режимы, соответствующие PC = 00 и 10, позволяют на основе BM87 эмулировать работу процессоров с разрядной сеткой, отведенной под мантиссу, 24 и 53 бит соответственно. В этих режимах результаты операций с плавающей запятой перед размещением в арифметический регистр округляются до указанного размера.

Таблица 3.4. Управление точностью (описание форматов см. в разделе 3.3)

Код поля PC	Точность представления результата
00	В формате КВ (24 бит)
01	Зарезервирован
10	В формате ДВ (53 бит)
11	В формате ВВ (64 бит)

Поля **CR(5) — CR(0)** позволяют установить маски исключительных ситуаций, описанных ранее, при этом: **IM** = 1 маскирует флаг **IE**; **DM** = 1 □ флаг **DE**; **ZM** = 1 □ флаг **ZE**; **OM** = 1 □ флаг **OE**; **UM** = 1 □ флаг **UE**; **PM** = 1 □ флаг **PE**.

Разряд CR(7) = **IEM** □ маскирует флаг запроса прерывания **IR**, т. е. независимо от значений перечисленных шести масок сигнал IEM = 1 заставляет BM87 обрабатывать все исключительные ситуации стандартным образом (см. раздел 3.5).

При инициализации BM87 сигналом RESET поля регистра управления устанавливаются следующим образом: IC = 0, RC = 00, PC = 11, IEM = 0, все маски устанавливаются в единицу.

Регистр-указатель исключительной ситуации EP (рис. 3.5) состоит из двух 32-битовых регистров. Когда сопроцессор выполняет очередную команду, устройство

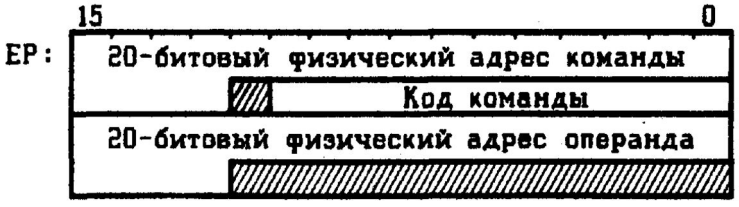

Рис. 3.5. Регистр-указатель, исключительной ситуации

шинного интерфейса засылает в первый регистр 20-битовый адрес этой команды и

11-битовый код команды, а во второй — 20-битовый адрес операнда, если он взят из памяти.

Эта информация может потребоваться центральному процессору при нестандартной обработке незамаскированной исключительной ситуации по запросу прерывания от ВМ87. Содержимое регистра ЕР может быть считано в память с помощью соответствующей команды ВМ87 для анализа в процессе обработки запроса. Неиспользуемые биты регистра-указателя ЕР заполняются нулями.

Контрольные вопросы

1. Кратко охарактеризуйте технические характеристики микросхемы ВМ87.
2. Кратко охарактеризуйте функциональные характеристики микросхемы ВМ87.
3. Кратко опишите структурные компоненты АСП ВМ87.
4. Кратко охарактеризуйте систему команд АСП ВМ87.
5. Дайте общую характеристику структурной схемы АСП ВМ87.
6. Что входит в состав операционного устройства АСП?
7. Охарактеризуйте модули операционного устройства.
8. Что входит в состав устройства шинного интерфейса?
9. Охарактеризуйте сигналы шинного интерфейса.
10. Дайте краткое описание других компонентов шинного интерфейса.
11. Охарактеризуйте флаги регистра состояния.
12. Охарактеризуйте разряды полей ST и С3 - С0 регистра состояния.
13. Охарактеризуйте контент табл. 3.2.
14. Дайте общую характеристику регистр управления CR
15. Охарактеризуйте разряды и поля старшего байта регистра CR.
16. Охарактеризуйте разряды младшего байта регистра CR.
17. Охарактеризуйте регистр-указатель исключительной ситуации ЕР.

3.3. Форматы данных.

АСП ВМ87 может оперировать данными семи различных форматов: *целых двоичных чисел* (три формата), *целых двоично-десятичных чисел* (один формат) и *вещественных чисел с плавающей запятой* (три формата). Структура форматов приведена на рис. 3.6, а наименование и примерный диапазон значений в табл. 3.5.

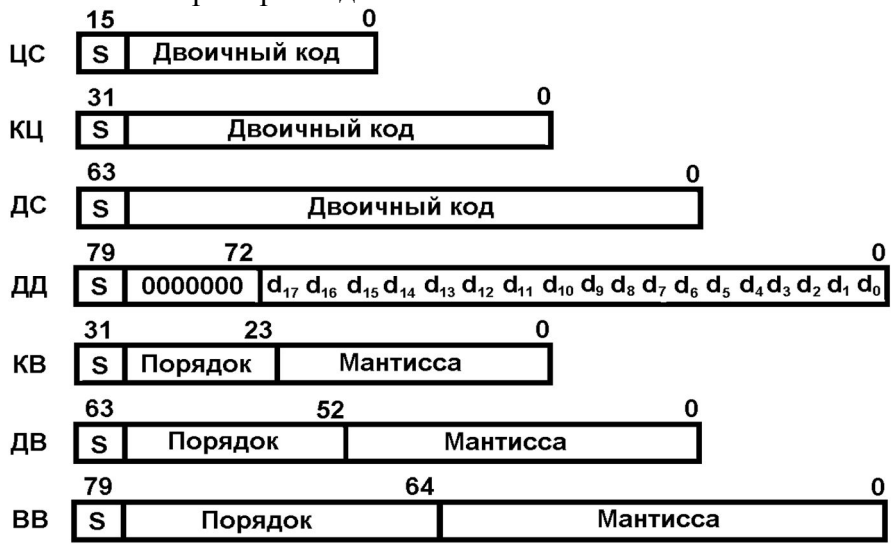

Рис. 3.6. Структура форматов данных АСП ВМ87

3.3.1. Целые числа

Целые двоичные числа записываются в памяти, начиная с младших байтов. Во всех трех форматах ЦС, КЦ и ДЦ отрицательные числа представляются в дополнительном коде

(табл. 3.6), тем самым старший бит определяет знак числа (0 □ положительное число, 1 □ отрицательное число).

Полный диапазон целых чисел представлен в табл. 3.6. Один из кодов, в котором знаковый разряд s = 1, а в значащих разрядах все нули, выделен для обозначения неопределенности. Код неопределенности генерируется в исключительных ситуациях, описываемых далее в разделе 3.5.

Таблица 3.5. Наименование и примерный диапазон значений чисел

Формат данных	Число байтов	Примерный диапазон значений
Целое слово (ЦС)	2	$\pm 10^{4}$
Короткое целое (КЦ)	4	$\pm 10^{9}$
Длинное целое (ДС)	8	$\pm 10^{18}$
Двоично-десятичное (ДД)	10	$\pm 10^{18}$
Короткое вещественное (КВ)	4	$\pm 10^{\pm 38}$
Длинное вещественное (ДВ)	8	$\pm 10^{\pm 308}$
Временное вещественное (ВВ)	10	$\pm 10^{\pm 4932}$

Целые двоично-десятичные числа записываются в плотно упакованном двоично-десятичном (ДД) коде, в котором под каждую цифру d_0 □ d_{17} (рис. 3.6) отводится четыре двоичных разряда. В памяти ДД-числа также располагаются начиная с младших разрядов и занимают 10 байт. Старший байт используется только под знак числа, который расположен в разряде 79, а остальные разряды старшего байта 78 □ 72 содержат нули.

Полный диапазон двоично-десятичных чисел представлен в табл. 3.7. В него входят коды +0 и □0; а также специальный код **неопределенности**. Код неопределенности является результатом в исключительных ситуациях, описываемых далее в разделе 3.5.

Таблица 3.6. Форматы целых чисел.

Наименование числа	Знак s	Значащие разряды ЦС(15 бит), КЦ(32 бит), ДЦ(64 бит)
Целое положительное	0	1 1 . . . 1 1
		. . . .
	0	0 0 . . . 0 1
Нуль	0	0 0 . . . 0 0
Целое отрицательное	1	1 1 . . . 1 1
		. . . .
	1	0 0 . . . 0 1
Неопределенность	1	0 0 . . . 0 0

Таблица 3.7. Диапазон двоично-десятичных чисел

Наименование числа	Знак s	Биты 78-72	Значащие цифры $d_{17}d_{16}...d_0$
Положительное двоично-десятичное	0	0000000	10011001...1001
		.....	.....
		0000000	00000000...0000
Нуль	0	0000000	00000000...0000
	1	0000000	00000000...0000
Отрицательное двоично-десятичное	1	0000000	00000000...0000
		.....	.....
		0000000	10011001...1001
Неопределенность	1	1111111	111111110..0000

3.3.2. Вещественные числа

Вещественные числа в каждом из трех форматов КВ, ДВ и ВВ выключают три поля: поле знака мантиссы, поле порядка и поле мантиссы (рис. 3.6). Мантисса числа всегда

(кроме исключительных случаев) записывается в нормализованном виде 1, m_1m_2..., где m_1m_2...— дробная часть числа.

Целая часть, всегда равная единице, прямо не представляется в форматах КВ и ДВ, а учитывается неявно. В формате ВВ старший бит мантиссы представляется явно. Порядок вещественных чисел записывается в поле порядка в смещенном виде, поэтому истинный порядок равен числу в поле порядка минус значение смещения. Применение *смещенных порядков* позволяет упростить операцию сравнения вещественных чисел, достаточно сравнить их порядки как обычные целые числа без знака. Поскольку операция сравнения является доминирующей в вычислительных программах, такое упрощение существенно повышает эффективность программ.

Учитывая разрядность полей в форматах вещественных чисел и особенности представления мантисс и порядков, можно указать формулы для вычисления истинных значений чисел:

$(-1)^S\times(1, m^1m^2\ldots m^{23})\times2^{E-127}$ для КВ

$(-1)^S\times(1, m^1m^2\ldots m^{52})\times2^{E-1023}$ для ДВ

$(-1)^S\times(1, m^1m^2\ldots m^{64})\times2^{E-16383}$ для ВВ

где s — знак мантиссы; m_i — значение i-го двоичного разряда мантиссы, считая от запятой; E — значение в поле порядка.

Примеры представления чисел в формате КВ.

Число □0,419375 представляется в виде

s	E	$m_1m_2m_3m_4m_5\ldots m_{23}$
1	01111101	1 0 1 1 0 ...0

и вычисляется по формуле

$$(-1)^1\times(1{,}1011)_2\times2^{125-127}=-(1{,}6775)_{10}\times2^{-4}=-0{,}419375.$$

Число +1,0 представляется в виде

s	E	$m_1m_2\ldots m_{23}$
0	01111111	0 0 ...0

и вычисляется по формуле

$$(-1)^0\times(1{,}0)_2\times2^{127-127}=(1{,}0)_{10}\times2^0=1{,}0.$$

Число □3,0 представляется в виде

s	E	$m_1m_2\ldots m_{23}$
1	10000000	1 0 ...0

и вычисляется по формуле

$$(-1)^1\times(1{,}1)_2\times2^{128-127}=-(1{,}5)_{10}\times2^1=-3{,}0.$$

Число $+1/3$ представляется в виде

s	E	$m_1m_2m_3m_4\ldots m_{22}m_{23}$
0	01111101	0 1 0 1 ...0 1

и вычисляется по формуле

$$(-1)^0\times(1{,}0101\ldots01)_2\times2^{125-127}=(1{,}31\ldots)\times2^{-2}\approx1/3.$$

Как видно из примеров, в формате вещественных чисел могут быть представлены абсолютно точно и целые числа, лежащие в пределах диапазона представимых значений.

Для представления вещественных чисел используется часть значений в поле порядка от 00 ... 00 до 11 ... 10. В форматах КВ и ДВ значение 11 ... 11 используется для кодирования ±∞, когда в поле мантиссы все нули, **и для обозначения «не чисел» (NAN — Not a Number — не числа), когда в поле мантиссы не нулевое значение.** Полный диапазон значений вещественных чисел в форматах КВ и ДВ представлен в табл. 3.8.

Как и в предыдущем случае, для обозначения **неопределенного значения** введен специальный код ¤. Кроме того, отметим, что диапазоны конечных чисел от $\pm8{,}43.10^{-37}$ до

$\pm 3{,}37 \cdot 10^{38}$ в формате КВ и от $\pm 4{,}19 \cdot 10^{-307}$ до $\pm 1{,}67\ 10^{308}$ в формате ДВ разбиты на два класса — **нормализованные** и **денормализованные** числа. Последние характеризуются тем, что имеют все нули в поле порядка и отличную от нуля мантиссу. Во всех случаях получения денормализованных операндов сопроцессор сигнализирует об этом и реагирует соответствующим образом (см. раздел 3.5).

Диапазон значений вещественных чисел в формате ВВ представлен в табл. 3.9. Все множество конечных чисел от $\pm 3{,}4\ 10^{-4932}$ до $\pm 1{,}2\ 10^{4932}$ разбито на три класса: нормализованные, ненормализованные и денормализованные. В числах последних двух классов после запятой всегда формируется нуль (старший бит мантиссы), при этом денормализованные числа содержат все нули в поле порядка. Появление ненормализованных и денормализованных чисел приводит к исключительным ситуациям, о которых сопроцессор сигнализирует особым образом. Подробно вопросы обработки исключительных ситуаций будут описаны далее в разделе 3.5. Формат ВВ обеспечивает максимальный рабочий диапазон значений вещественных чисел и является основным, в нем выполняются все операции сопроцессора ВМ87. Преобразование исходных данных, представленных в памяти в любом другом формате (ЦС, КЦ, ДЦ, ДД, КВ, ДВ), в формат ВВ осуществляется сопроцессором автоматически при загрузке данных. Такое преобразование никогда не приводит к потере точности. Заметим, что при необходимости можно «заставить» сопроцессор ВМ87 использовать другой формат, а именно КВ или ДВ, что осуществляется программно и используется для эмуляции на основе ВМ87 какого-либо другого вычислителя с разрядностью не превышающей 32 или 64 бит соответственно.

Таблица 3.8. Значения вещественных чисел в форматах КВ и ДВ

Наименование числа	Знак s	Порядок КВ(8 бит), ДВ(11 бит)	Мантисса КВ(23 бит), ДВ(52 бит)	Значение КВ/ДВ
Положительное "не число"	0	11 ... 11	11 ... 11	NAN
	...	.. ..	.. ..	...
	0	11 ... 11	00 ... 01	NAN
Бесконечность	0	11 ... 11	00 ... 00	$+\infty$
Положительное нормализованное число	0	11 ... 10	11 ... 11	$+3{,}37 \cdot 10^{38}/1{,}67 \cdot 10^{308}$
	...	.. ..	.. ..	...
	0	00 ... 01	00 ... 00	...
Положительное денормализованное число	0	00 ... 00	11 ... 11	...
	...	.. ..	.. ..	...
	0	00 ... 00	00 ... 01	$+8{,}43 \cdot 10^{-37}/+4{,}19 \cdot 10^{-307}$
Нуль	0	00 ... 00	00 ... 00	+0
	1	00 ... 00	00 ... 00	-0
Отрицательное денормализованное число	1	00 ... 00	00 ... 01	$-8{,}43 \cdot 10^{-37}/-4{,}19 \cdot 10^{-307}$
	...	.. ..	.. ..	...
	1	00 ... 00	11 ... 11	...
Отрицательное нормализованное число	1	00 ... 01	00 ... 00	...
	...	.. ..	.. ..	...
	1	00 ... 00	11 ... 11	$-3{,}37 \cdot 10^{38}/-1{,}67 \cdot 10^{308}$
Бесконечность	1	11 ... 11	00 ... 00	$-\infty$
Отрицательное "не число"	1	11 ... 11	00 ... 01	NAN
	...	.. ..	.. ..	...
Неопределенность	1	11 ... 11	10 ... 00	И[1]
Отрицательное "не число"	...	.. ..	.. ..	...
	1	11 ... 11	11 ... 11	NAN

[1] Знак И - здесь и далее обозначает неопределенное значение.

Результаты вычислений могут быть переданы в память в любом (из семи) желаемом формате. Таким образом, получение результата требует (в шести случаях из семи) обратного преобразования из формата ВВ в желаемый. В ряде случаев это может привести к потере точности из-за необходимости округления, о чем сигнализирует ВМ87, формируя запрос прерывания на выходе INT.

Таблица 3.9. Диапазон значений вещественных чисел в формате BB

Наименование числа	Знак	Порядок (15 бит)	Мантисса (64 бит)	Значение
Положительное "не число"	0	11 ... 11	111 ... 11	NAN
	...	.. ..	... .	...
	0	11 ... 11	100 ... 01	NAN
Бесконечность	0	11 ... 11	100 ... 00	$+\infty$
Положительное нормализованное число	0	11 ... 10	111 ... 11	$+1,2\cdot10^{4932}$
	...	.. ..	... .	...
	...	.. ..	100 ... 00	...
Положительное ненормализованное число	...	.. ..	011 ... 11	...
	...	.. ..	... .	...
	0	00 ... 01	000 ... 00	...
Положительное денормализованное число	0	00 ... 00	011 ... 11	...
	...	.. ..	... .	...
	0	00 ... 00	000 ... 01	$+3,4\cdot10^{-4932}$
Нуль	0	00 ... 00	000 ... 00	+0
	1	00 ... 00	000 ... 00	-0
Отрицательное денормализованное число	1	00 ... 00	000 ... 01	$-3,4\cdot10^{-4932}$
	...	.. ..	... .	...
	1	00 ... 00	011 ... 11	...
Отрицательное ненормализованное число	1	00 ... 01	000 ... 00	...
	...	.. ..	... .	...
	...	.. ..	011 ... 11	...
Отрицательное нормализованное число	...	.. ..	1000... 00	...
	...	.. ..	... .	...
	1	11 ... 10	111 ... 11	$-1,2\cdot10^{4932}$
Бесконечность	1	11 ... 11	100 ... 00	$-\infty$
Отрицательное "не число"	1	11 ... 11	100 ... 00	NAN
	...	.. ..	... .	...
Неопределенность	1	11 ... 11	110 ... 00	¤
Отрицательное "не число"	...	.. ..	... .	...
	1	11 ... 11	111 ... 11	NAN

Контрольные вопросы

1. Какими данными может оперировать АСП BM87?
2. Охарактеризуйте числа целого типа.
3. Как кодируется «неопределенность» в форматах целых и двоично-десятичных чисел?
4. Дайте характеристику числам вещественного типа.
5. Какое вещественное число считается значением $\pm\infty$ и «не число» (NAN — Not a Number — не число)?
6. Как кодируется «неопределенность» в форматах КВ, ДВ и ВВ?
7. Какие числа называются нормализованными и денормализованными?
8. Охарактеризуйте вещественные числа в формате ВВ.
9. В каком формате выполняются все операции в АСП BM87?
10. Как преобразует форматы чисел АСП BM87?
11. Как пользователь может преобразовать числа из одного формата в другой с помощью АСП BM87?

3.4. Система команд арифметического сопроцессора

Полная система команд АСП BM87 включает 68 мнемокодов и построена на основе команды ESC процессора BM86. Используются два формата команды ESC (рис. 3.7).

Формат команды с постбайтом применяется в тех случаях, когда операнд-источник находится в памяти или регистре ST(i) либо результат необходимо переслать в память или регистр ST(i). Кодирование полей постбайта приведено в табл. 3.10, где DISP8 означает однобайтовое смещение, а DISP16 — двухбайтовое смещение (подобная кодировка

применяется при кодировании постбайта в командах процессора ВМ86 (см. табл. 2.3), отличается только столбец с полет MOD (md) =11).

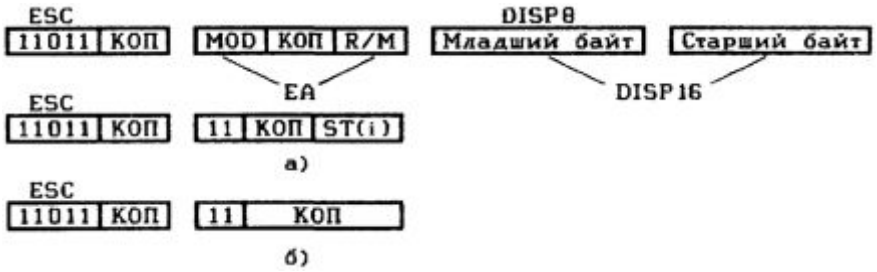

Рис. 3.7. Форматы команд АСП с постбайтом (а) и без него (б)

Таблица 3.10. Кодирование полей постбайта.

Поле	Поле MOD			
R/M	00	01	10	11
000	(BX)+(SI)	(BX)+(SI)+DISP8	(BX)+(SI)+DISP16	ST(0)
001	(BX)+(DI)	(BX)+(DI)+DISP8	(BX)+(DI)+DISP16	ST(1)
010	(BP)+(SI)	(BP)+(SI)+DISP8	(BP)+(SI)+DISP16	ST(2)
011	(BP)+(DI)	(BP)+(DI)+DISP8	(BP)+(DI)+DISP16	ST(3)
100	(SI)	(SI)+DISP8	(SI)+DISP16	ST(4)
101	(DI)	(DI)+DISP8	(DI)+DISP16	ST(5)
110	DISP16	(BP)+DISP8	(BP)+DISP16	ST(6)
111	(BX)	(BX)+DISP8	(BX)+DISP16	ST(7)

Из табл. 3.10 видно, что смещение DISP может составлять один байт (MOD = 01), два байта (MOD = 10) или отсутствовать (MOD = 00). Исключение составляет случай MOD = 00, R/M = 110, когда смещение составляет два байта и полностью определяет исполнительный адрес. Код поля MOD =11 означает, что операнд размещается в одном из арифметических регистров ST(i).

Формат команды без постбайта (рис. 3.15, *б)* применяется в тех случаях, когда операнд расположен в регистре ST(0) либо для команд управления.

Полный перечень команд сопроцессора ВМ87 приведен в табл. 3.11, где указаны среднее время Тср выполнения команды, а также диапазон значений Tmin Tmax. Среднее время соответствует наиболее часто встречающемуся (типовому) случаю выполнения команды, когда «мешающие» факторы отсутствуют. Диапазон Tmin Tmax охватывает наилучший и наихудший случаи выполнения команды, когда учитываются дополнительные задержки, связанные с арбитражем локальной шины (при извлечении операндов) и анализом входа (при выполнении команды WAIT, предшествующей ESC). Кроме того, при обращении к операндам, находящимся в памяти, следует добавлять время T_{EA}, необходимое для формирования исполнительного адреса операнда.

Мнемокод каждой команды начинается с буквы F (float—плавающий), что позволяет в программах на ассемблере отличать команды процессоров ВМ87 и ВМ86. В мнемокоде команд, оперирующих вещественными числами, используется мнемоника, определяющая основное содержание команды: LD (load) — загрузка; ST (story) — запоминание; COM (compare) — сравнение и т. д. Для команд, оперирующих целыми двоичными числами, за буквой F идет буква I (integer — целое число), а двоично-десятичными — буква B. Таким образом, различаются мнемокоды команд, оперирующих различными типами данных. В табл. 3.11 для каждого типа данных, расположенных в памяти, в скобках указан формат, например запись mem (ЦС) означает, что операндом является целое слово.

Для одного и того же типа, независимо от формата, используется один мнемокод. Например, для вещественного числа, размещенного в памяти по адресу Addr1, команда загрузки имеет вид FLD Addr1 в любом случае, когда по адресу Addr1 записано число в формате КВ, ДВ или ВВ. Для правильного формирования кода команды в **языке ассемблера** ASM-86 предусмотрены директивы объявления данных, ориентированные на форматы АСП ВМ87: DW — для формата ЦС; DD — для форматов КЦ, КВ; DQ — для форматов ДЦ, ДВ; DT — для форматов ВВ и ДД.

Система команд сопроцессора ВМ87 по функциональному признаку разбивается на четыре группы: команды передачи данных (мнемокоды 1 — 16 в табл. 3.11), арифметические команды (мнемокоды 17 — 41), специальные вычислительные команды (мнемокоды 42 — 53) и команды управления (мнемокоды 54 — 68).

Рассмотрим особенности использования команд различных групп, которые не отражены в табл. 3.11.

3.4.1. Команды передачи данных

Команды передачи данных включают четыре типа: *загрузки, запоминания, обмена и загрузки констант.*

Команды загрузки **FLD** src, **FILD** src, **FBLD** src служат для передачи данных из источника src в вершину стека ST(0). В качестве источника данных служит адрес памяти, а в команде FLD src источником может служить также арифметический регистр. Перед выполнением загрузки автоматически подготавливается новое значение указателя стека ST ← ST1 и анализируется значение TAG(0). Если TAG(0) ≠ 11, устанавливается флаг IE = 1, причем, когда он не замаскирован (IM = 0), загрузка не производится и генерируется прерывание, свидетельствующее о недействительной операции — попытке загрузки занятого регистра. В процессе загрузки осуществляется автоматический перевод в формат ВВ данных из форматов КВ, ДВ (по команде FLD), из форматов ЦС, КЦ, ДД (по команде FILD)

Таблица 3.11 Полный перечень команд сопроцессора ВМ87 (в табл. 3.11 для обозначения поля MOD используется мнемоника MD).

№ п/п	Мнемокод	Байт 1	Байт 2	nв	Tср	Tmin-Tmax	Операция dst	Операция src
1	FLD src	ESC001	MD000 R/M	2-4	43+E	38-56+E	ST(0)	mem(КВ)↓
		101	000	2-4	46+E	40-60+E	ST(0)	mem(ДВ)↓
		011	101	2-4	57+E	53-65+E	ST(0)	mem(ВВ)↓
		001	11000ST(i)	2	20	17-22	ST(0)	ST(i)↓
2	FILD src	ESC111	MD000 R/M	2-4	50+E	46-54+E	ST(0)	mem(ЦС)↓
		011	000	2-4	56+E	52-60+E	ST(0)	mem(КЦ)↓
		111	101	2-4	64+E	60-68+E	ST(0)	mem(ДЦ)↓
3	FBLD src	ESC111	MD100 R/M	2-4	300+E	290-310+E	ST(0)	mem(ДД)↓
4	FST dst	ESC001	010	2-4	87+E	84-90+E	mem(КВ)	ST(0)
		101	010	2-4	100+E	96-104+E	mem(ДВ)	ST(0)
		101	11010ST(i)	2	18	15-22	ST(i)	ST(0)
5	FIST dst	ESC111	MD010 R/M	2-4	86+E	80-90+E	mem(ЦС)	ST(0)
		011	010	2-4	88+E	82-92+E	mem(КЦ)	ST(0)
6	FSTP dst	ESC001	011	2-4	89+E	86-92+E	mem(КВ)	ST(0)↑
		101	011	2-4	102+E	98-106+E	mem(ДВ)	ST(0)↑
		011	111	2-4	55+E	52-58+E	mem(ВВ)	ST(0)↑
		101	11011ST(i)	2	20	17-24	ST(i)	ST(0)↑
7	FISTP dst	ESC111	MD011 R/M	2-4	88+E	82-92+E	mem(ЦС)	ST(0)↑
		011	011	2-4	90+E	84-94+E	mem(КЦ)	ST(0)↑
		111	111	2-4	100+E	94-105+E	mem(ДЦ)	ST(0)↑
8	FBSTP dst	ESC111	110	2-4	530+E	520-540+E	mem(ДД)	ST(0)↑
9	FXCH ST(i)	ESC001	11001ST(i)	2	12	10-15	ST(0)	ST(i)
10	FLDZ	ESC001	11101 110	2	14	11-17	ST(0)	+0,0↓
11	FLD1	ESC001	11101 000	2	18	15-21	ST(0)	+1,0↓
12	FLDPI	ESC001	11101 011	2	19	16-22	ST(0)	π↓
13	FLDL2T	ESC001	11101 001	2	19	16-22	ST(0)	$\log_2 10$↓
14	FLDL2E	ESC001	11101 010	2	18	15-21	ST(0)	$\log_2 e$↓
15	FLDLG2	ESC001	11101 100	2	21	18-24	ST(0)	lg2↓
16	FLDLN2	ESC001	11101 101	2	20	17-23	ST(0)	ln2↓
17	FCOM src	ESC000	MD010 R/M	2-4	65+E	60-70+E	ST(0)	-mem(КВ)
		100	010	2-4	70+E	65-75+E	ST(0)	-mem(ДВ)
		000	11010ST(i)	2	45	40-50	ST(0)	-ST(i)
18	FICOM src	ESC110	MD010 R/M	2-4	80+E	72-86+E	ST(0)	-mem(ЦС)
		010	010	2-4	85+E	78-91+E	ST(0)	-mem(КЦ)
19	FCOMP src	ESC000	MD011 R/M	2-4	68+E	63-73+E	ST(0)	-mem(КВ)↑
		100	011	2-4	72+E	67-77+E	ST(0)	-mem(ДВ)↑
		000	11011ST(i)	2	47	45-52	ST(0)	-ST(i)↑

Продолжение табл. 3.11

№ п/п	Мнемокод	Байт 1	Байт 2	nв	Tср	Tmin-Tmax	Операция dst	Операция src
20	FICOMP src	ESC110	MD011 R/M	2-4	82+E	74-88+E	ST(0)	mem(ЦС)↑
		010	011	2-4	87+E	80-93+E	ST(0)	mem(КЦ)↑
21	FCOMPP	ESC110	11011 001	2	50	45-55	ST(0)	ST(1)↑↑
22	FTST	ESC001	100 100	2	42	38-48	ST(0)	0,0
23	FXAM	ESC001	100 101	2	17	12-23	анализ ST(0)	
24	FADD dst,src	ESC000	MD000 R/M	2-4	105+E	90-120+E	ST(0)←ST(0)+	mem(КВ)
		100	000	2-4	110+E	95-125+E	ST(0)←ST(0)+	mem(ДВ)
		000	11000ST(i)	2	85	70-100	ST(0)←ST(0)+	ST(i)
		100	000	2	85	70-100	ST(i)←ST(0)+	ST(i)
25	FIADD src	ESC110	MD000 R/M	2-4	120+E	102-137+E	ST(0)←ST(0)+	mem(ЦС)
		010	000	2-4	125+E	108-143+E	ST(0)←ST(0)+	mem(КЦ)
26	FADDP dst,src	ESC110	11000ST(i)	2	90	75-105	ST(i)←ST(0)+	ST(i)↑
27	FSUB dst,src	ESC000	MD100 R/M	2-4	105+E	90-120+E	ST(0)←ST(0)-	mem(КВ)
		100	100	2-4	110+E	95-125+E	ST(0)←ST(0)-	mem(ДВ)
		000	11100ST(i)	2	85	70-100	ST(0)←ST(0)-	ST(i)
		100	101	2	85	70-100	ST(i)←ST(i)-	ST(0)
28	FSUBR dst,src	ESC000	MD101 R/M	2-4	105+E	90-120+E	ST(0)←mem(КВ)-	ST(0)
		100	101	2-4	110+E	95-125+E	ST(0)←mem(ДВ)-	ST(0)
		000	11101ST(i)	2	87	70-100	ST(0)←ST(i)-	ST(0)
		100	101	2	87	70-100	ST(i)←ST(0)-	ST(i)
29	FSUBP dst,src	ESC110	11100ST(i)	2	90	75-105	ST(i)←ST(0)-	ST(i)↑
30	FSUBRP dst,src	ESC110	11101ST(i)	2	90	75-105	ST(i)←ST(i)-	ST(0)↑
31	FISUB src	ESC110	MD100 R/M	2-4	120+E	102-137+E	ST(0)←ST(0)-	mem(ЦС)
		010	100	2-4	125+E	108-143+E	ST(0)←ST(0)-	mem(КЦ)
32	FISUBR src	ESC110	MD101 R/M	2-4	120+E	103-139+E	ST(0)←mem(ЦС)-	ST(0)
		010	101	2-4	125+E	109-144+E	ST(0)←mem(КЦ)-	ST(0)
33	FMUL dst,src	ESC000	MD001 R/M	2-4	118+E	110-125+E	ST(0)←ST(0)x	mem(КВ)
		100	001	2-4	140+E	112-168+E	ST(0)←ST(0)x	mem(ДВ)
		000	11001ST(i)	2	110	90-145	ST(0)←ST(0)x	ST(i)
		100	001	2	110	90-145	ST(i)←ST(0)x	ST(i)
34	FIMUL src	ESC110	MD001 R/M	2-4	130+E	124-138+E	ST(0)←ST(0)x	mem(ЦС)
		010	001	2-4	136+E	130-144+E	ST(0)←ST(0)x	mem(КЦ)
35	FMULP dst,src	ESC110	11001ST(i)	2	115	95-150	ST(i)←ST(0)x	ST(i)↑
36	FDIV dst,src	ESC000	MD110 R/M	2-4	220+E	215-225+E	ST(0)←ST(0)/	mem(КВ)
		100	110	2-4	225+E	220-230+E	ST(0)←ST(0)/	mem(ДВ)
		000	11110ST(i)	2	198	195-203	ST(0)←ST(0)/	ST(i)
		100	110	2	198	195-203	ST(i)←ST(i)/	ST(0)
37	FDIVR dst,src	ESC000	MD111 R/M	2-4	221+E	215-225+E	ST(0)←mem(КВ)/	ST(0)
		100	111	2-4	226+E	220-230+E	ST(0)←mem(ДВ)/	ST(0)
		000	11111ST(i)	2	199	193-203	ST(0)←ST(i)/	ST(0)
		100	111	2	199	193-203	ST(i)←ST(0)/	ST(i)
38	FDIVP dst,src	ESC110	11110ST(i)	2	202	198-208	ST(i)←ST(0)/	ST(i)↑
39	FDIVRP dst,src	ESC110	11111ST(i)	2	203	198-208	ST(i)←ST(i)/	ST(0)↑
40	FIDIV src	ESC110	MD110 R/M	2-4	230+E	224-238+E	ST(0)←ST(0)/	mem(ЦС)
		010	110	2-4	236+E	230-243+E	ST(0)←ST(0)/	mem(КЦ)
41	FIDIVR src	ESC110	MD111 R/M	2-4	230+E	225-239+E	ST(0)←mem(ЦС)/	ST(0)
		010	111	2-4	237+E	231-245+E	ST(0)←mem(КЦ)/	ST(0)
42	FSQRT	ESC001	11111 010	2	183	180-186	ST(0)←$\sqrt{ST(0)}$	
43	FSCALE	001	11111 101	2	35	32-38	ST(0)←ST(0)×$2^{ST(1)}$	
44	FPREM	001	11111 000	2	125	15-190	ST(0)←ST(0)MODST(1)	
45	FRNDINT	001	11111 100	2	45	16-50	ST(0)←целое ST(0)	
46	FXTRACT	001	11110 100	2	50	27-55	ST(0)←мантисса, ST(1)←порядок	
47	FABS	001	11100 001	2	14	10-17	ST(0)←|ST(0)|	
48	FCHS	001	11100 000	2	15	10-17	ST(0)←-ST(0)	
49	FPTAN	001	11110 010	2	450	30-540	ST(0),ST(1)←tg(Z/2)	
50	FPATAN	001	11110 011	2	650	250-800	ST(0)←arctg(X/Y)	
51	F2FM1	001	11110 000	2	500	210-630	ST(0)←$2^{ST(0)}$-1	
52	FYL2X	001	11110 001	2	950	900-1100	ST(0)←Y$\log_2$X	
53	FYL2XP1	001	11111 001	2	850	700-1000	ST(0)←Y$\log_2$(X+1)	
54	FINIT	ESC011	11100 011	2	5	2-8	Инициализация	
55	FENT	011	11100 000	2	5	2-8	IEM←0	
56	FDISI	011	11100 001	2	5	2-8	IEM←1	
57	FLDCW src	ESC001	MD101 R/M	2-4	10+E	7-14+E	CR←src	
58	FSTCW dst	001	111	2-4	15+E	12-18+E	dst←CR	
59	FSTSW dst	ESC101	MD111 R/M	2-4	15+E	12-18+E	dst←SR	
60	FCLEX	ESC011	11100 010	2	5	2-8	PE,UE,OE,ZE,DE,IE←0	

Окончание табл. 3.11

№ п/п	Мнемокод	Байт 1	Байт 2	пв	Тср	Tmin-Tmax	Операция	
							dst	src
61	FSTENV dst	ESC001	MD110 R/M	2-4	45+E	40-50+E	dst←CR,SR,TAG,EP	
62	FLDENV src	001	100	2-4	40+E	35-45+E	CR,SR,TAG,EP←dst	
63	FSAVE dst	ESC101	MD110 R/M	2-4	203+E	197-207+E	dst←CR,SR,TAG,EP, ST(0)-ST(7)	
64	FRSTOR src	ESC101	MD100 R/M	2-4	203+E	197-207+E	CR,SR,TAG,EP,ST(0)-ST(7)←src	
65	FINCSTP	ESC001	11110 111	2	9	6-12	ST←ST+1	
66	FDECSTP	001	11110 110	2	9	6-12	ST←ST-1	
67	FFREE ST(i)	ESC101	11000ST(i)	2	11	6-16	TAG(i)←11	
68	FNOP	ESC001	11010 000	2	13	10-16	ST(0)←ST(0)	

и из формата ДД (по команде FBLD). Отметим, что команда FLD ST(0) часто используется для дублирования вершины стека.

Команды запоминания **FST dst, FIST dst** используются для пересылки данных из вершины стека ST(0) в память по адресу dst. В команде FST dst приемником данных может также служить регистр ST(i). При выполнении этих команд содержимое указателя стека не изменяется. Число, засылаемое в память, из формата ВВ автоматически переводится в формат КВ или ДВ (по команде FST) либо в формат ЦС или КЦ (по команде FIST).

Команды запоминания **FSTP dst, FISTP dst, FBSTP dst** с одновременным выталкиванием из стека (в мнемокоде буква Р — от POP) используются для пересылки данных из вершины стека ST(0) в память по адресу dst. В команде FSTP dst приемником данных может также служить регистр ST(i). После выполнения пересылки содержимое стека автоматически увеличивается ST: = (ST) + 1. Число, засылаемое в память, из формата ВВ автоматически переводится: 1) в формат КВ, ДВ или остается в формате ВВ (по команде FSTP); 2) в формат ЦС, КЦ или ДЦ (по команде FISTP) либо в формат ДД (по команде FBSTP).

Команда обмена **FXCH dst** используется для обмена содержимого регистра-приемника ST(i), указанного в качестве адреса dst с вершиной стека ST(0). Если регистр-приемник не указан, т. е. мнемокод EXCH записан без операнда, то подразумевается регистр ST(1).

Команды загрузки констант **FLDZ, FLD1, FLDPI, FLDL2T, FLDL2E, FLDLG2, FLDLN2** используются для включения в стек значений +0.0; +1.0; π; $\log_2 10$; $\log_2 e$; lg2; ln2, часто встречающихся в вычислительных программах. Эти значения хранятся в ПЗУ констант в формате ВВ и имеют точность представления 19 десятичных цифр.

3.4.2. Арифметические команды

Арифметические команды включают шесть типов команд: *сравнения, анализа, сложения, вычитания, умножения и деления.*

Команды сравнения.

Команды сравнения **FCOM src, FICOM src** выполняют вычитание содержимого памяти из содержимого вершины стека: ST(0) □ (src), формируя значения флагов С3, С0, как показано в табл. 3.12. Команда FCOM src в качестве операнда src может указывать регистр ST(i). Если в команде FCOM операнд не указан, то подразумевается ST(1). Значение операнда, находящегося в памяти, может быть в формате КВ, ДВ (команда FCOM) или ЦС, КЦ (команда FICOM).

Команды сравнения **FCOMP src, FICOMP src** по выполняемым действиям и форматам операндов аналогичны описанным выше. Дополнительное действие этих команд заключается в выталкивании из стека сравниваемого операнда.

Команда сравнения **FCOMPP** используется без указания операндов. Предполагается,

что предварительно они должны быть размещены в двух верхних регистрах стека. Выполненное по команде FCOMPP сравнение сопровождается выталкиванием обоих сравниваемых операндов из стека, т. е. указатель стека увеличивается на два ST:= (ST) + 2.

Команда **FTST** используется для *сравнения содержимого с нулем*. Эта операция часто выполняется в вычислительных программах. Результат сравнения кодируется в разрядах С3, С0 регистра состояния SR в соответствии с табл. 3.12, где (src) = +0.0.

Команда **FXAM** осуществляет *анализ содержимого вершины стека* и помещает результат анализа в регистр состояния SR. Флаг С1 указывает знак числа, находящегося в вершине: «+» (С1 = 0), «□» (С1 = 1). Флаг С0 указывает, какое число находится в вершине: С0 = 0 — конечное число, С0 = 1 в других случаях. Флаги С2, С3 конкретизируют содержимое вершины (табл. 3.13).

Таблица 3.12

Отношение	С3С0
ST(0) > (src)	0 0
ST(0) < (src)	0 1
ST(0) = (src)	1 0
ST(0) не сравнимо с (src)	1 1

Таблица 3.13

С3С2	Содержимое вершины стека ST(0)	
	С0 = 0	С0 = 1
0 0	Не нормализовано	NAN
0 1	Нормализовано	∞
1 0	Нулевое	е (пусто)
1 1	Денормализовано	е (пусто)

Команды сложения

Команды сложения **FADD dst,src, FIADD src, FADDP dst,src** в качестве одного из слагаемых используют число из вершины стека ST(0).

В команде FADD операнды dst и src могут быть не указаны. В этом случае подразумевается dst = ST(0), src = ST(l). Если не указан один операнд dst, то подразумевается dst = ST(0), а в качестве src может быть использован ST(i) либо адрес памяти, по которому слагаемое представлено в формате КВ или ДВ. Если указаны оба операнда, то ими могут быть dst = ST(0), src = ST(i) или dst = ST(i), src = ST(0).

В команде FIADD src указывается только адрес памяти, по которому расположено слагаемое в формате ЦС или КЦ.

В команде FADDP dst, src операндами являются регистры dst = ST(i), src = ST(0). Сложение по команде FADDP ST(i), ST(0) сопровождается выталкиванием слагаемого, находящегося в вершине стека.

Команды вычитания

Команды вычитания **FSUB dst,src, FSUBR dst,src, FSUBP dst, src, FSUBRP dst,src, FISUB src, FISUBR src** в качестве одного операнда используют содержимое вершины стека ST(0).

В командах FSUB и FSUBR операнды dst и src могут быть не указаны, в этом случае подразумевается dst = ST(0), src = ST(l). Если не указан один операнд dst, то подразумевается, что dst = ST(0), а в качестве src может быть использован ST(i) либо адрес памяти, по которому значение представлено в формате КВ или ДВ. Если указаны оба операнда, то ими могут быть dst = ST(0), src = ST(i) или dst = ST(i), src = ST(0). По команде FSUBR, в отличие от FSUB, уменьшаемое и вычитаемое меняются местами (в мнемокоде R — от англ. Revers).

В командах FSUBP dst,src и FSUBRP dst,src операндами являются регистры dst = ST(i), src = ST(0). Вычитание сопровождается выталкиванием операнда, находящегося в вершине стека. По команде FSUBRP (в отличие от команды FSUBP) уменьшаемое и вычитаемое меняются местами.

В командах FISUB src и FISUBR src указывается только адрес памяти, по которому расположено значение операнда в формате ЦС или КЦ. По команде FISUBR (в отличие от FISUB) уменьшаемое и вычитаемое меняются местами.

Команды умножения

Команды умножения **FMUL dst,src, FIMUL src, FMULP dst,src** в качестве одного из сомножителей используют содержимое вершины стека ST (0). Указание (или умолчание) операндов, а также ограничение на форматы операндов в памяти аналогичны командам сложения FADD и FIADD.

Команды деления

Команды деления **FDIV dst,src, FDIVR dst,src, FDIVP dst,src, FDIVRP dst,src, FIDIV src, FIDIVR src** в качестве одного операнда используют содержимое вершины стека ST(0). Указание (или умолчание) операндов, а также ограничения на форматы операндов в памяти аналогичны соответствующим командам вычитания.

3.4.3. Специальные вычислительные команды

Группа специальных вычислительных команд **включает 12 мнемокодов, определяющих** действие, часто встречающееся в вычислительных программах.

Команды вычисления квадратного корня **FSQRT,** *нахождения абсолютного значения* **FABS** и *изменения знака* **FCHS** не требуют дополнительных пояснений. Операндом в этих командах является содержимое вершины стека ST(0).

Команда масштабирования **FSCALE** используется для изменения порядка числа, размещенного в ST(0), путем сложения масштабного коэффициента, размещенного в ST(1), со значением поля порядка ST(0). Масштабный коэффициент рассматривается как целое число со знаком. Эта операция эквивалентна умножению значения ST(0) на степень числа 2^n, где n = ST(l), причем $-2^{15} \le n \le +2^{15}$.

Команда получения частичного остатка от деления **FPREM** используется для точного деления чисел. Делимое размещается в регистре ST(0), делитель — в ST(1). Делитель можно рассматривать как некоторый модуль, и тогда команда FPREM позволяет получить остаток от деления по модулю. Действие команды заключается в сдвигах и вычитаниях, как при «ручном» делении. (Подробности см. в [1] стр. 98).

Команда округления до целого значения **FRNDINT** производит округление значение ST (0) в соответствии с установленным режимом в поле RC регистра управления CR. Например, значение 125,75 преобразуется командой FRNDINT в 125, если RC = 01 или 11, и в 126, если RC = 00 или 10 (см. табл. 3.8).

Команда выделения порядка и мантиссы **FXTRACT** "разделяет" число, размещенное в ST (0), на два компонента: мантиссу и порядок. В результате выполнения команды в регистре ST (0) формируется значение мантиссы, представленной как вещественное число с тем же знаком и порядком, равным нулю, а в регистре ST(1) — значение порядка (без константы смещения), представленного как вещественное число со знаком. Например, исходным операндом является число $+1{,}5\times2^{-7}$ размещенное в ST (0) в формате BB (s = 0, поле порядка 3FF8H, поле мантиссы C0...0H). В результате выполнения команды FXTRACT в регистре ST (0) получим число 1,5 в формате BB (s = 0, поле порядка 3FFFH, поле мантиссы C0...0H), а в ST(1) — число −7,0 в формате BB (s = 1, поле порядка 4001 H, поле мантиссы E0...0H). Использование команды FXTRACT совместно с командой FBSTP удобно для преобразования результатов вычислений с дальнейшим их отображением (на дисплее, печатающем устройстве) в десятичной системе счисления. Команда может быть также полезна при поиске ошибок, так как позволяет отдельно проанализировать порядок и мантиссу числа.

Команда **FPTAN** вычисляет частичный тангенс числа z, размещенного в ST (0), где z должно находиться в интервале $0 \le z \le \pi/4$. Результатом выполнения команды являются два числа: x и y [ST(0) = x, ST(l) = y], из которых с помощью команды деления можно получить tg(z/2) = y/x. Используя значение x, y, можно легко вычислить все другие прямые тригонометрические функции, например

$$\operatorname{ctg}(z/2) = x/y,\ \sin z = \frac{2y/x}{1+(y/x)^2},\ \cos z = \frac{1-(y/x)^2}{1+(y/x)^2}$$

по известным из тригонометрии формулам. Ограничения на значения аргумента z ($0 \le z \le \pi/4$) можно избежать, используя предварительно команду FPREM [с модулем ST(l) = $\pi/4$], которая помимо понижения значения аргумента укажет один из восьми секторов единичного круга, где вычисляется функция. Напомним, что код сектора формируется в C3 C1 C0 после полного завершения команды FPREM.

Команда **FRATAN** вычисляет частичный арктангенс z = arctg (x/y). Исходные операнды x и y размещаются в регистрах ST (0) и ST (1) соответственно, причем $0 < y < x < \infty$, а результат получается в ST (0). Перед записью результата осуществляется выталкивание из стека; и, таким образом, оба исходных операнда теряются. Используя значение z, можно легко вычислить другие обратные тригонометрические функции по известным формулам.

Команда **F2XM1** вычисляет значение функции $y = 2^x - l$, где исходный операнд x размещается в регистре ST (0) и должен лежать в интервале $0 \le x \le 0{,}5$. Результат y размещается на месте исходного операнда. Команда позволяет получать точные значения даже тогда, когда x очень близок к нулю. Она используется для вычисления различных показательных функций, например функция $10^x = 2^x \times \log_2 10 = 1 + (2^x \times \log_2 10 - 1)$ может быть вычислена с помощью последовательности команд FLDL2T, FMUL x, F2XM1, FADD «1».

Команда **FYL2X** вычисляет значение функции $z = y \times \log_2 x$, где исходные операнды x и y размещены в ST (0) и ST (1) соответственно, причем $0 < x < \infty$, $-\infty < y < \infty$. Команда осуществляет выталкивание из стека и записывает результат в новую вершину, т. е. оба исходных операнда теряются. Команда FYL2X позволяет изменять основания логарифмов согласно известной формуле $\log_n x = \log_n 2 \times \log_2 x$ путем использования команды умножения.

Команда **FYL2XP1** вычисляет значение функции $z = y \times \log_2(x + 1)$, где исходные операнды x и y размещаются в ST (0) и ST (1) соответственно, причем $0 < |x| < 1 - 1/\sqrt{2}$, $-\infty < y < \infty$. Команда осуществляет выталкивание из стека и записывает результат в новую вершину, т. е. оба исходных, операнда теряются.

3.4.4. Команды управления

Команды управления включают 15 мнемокодов, которые используются не для вычислений, а для выполнения системных функций. Большинство команд этой группы имеют альтернативные мнемокоды, которые образуются путем добавления к мнемокоду буквы N, указывающей на то, что при ассемблировании данного мнемокода перед ним не надо помещать команду WAIT. Альтернативный мнемокод используется на участках программ, критичных по времени выполнения, где лишние такты, затрачиваемые на выполнение команды WAIT, нежелательны. Конечно, когда разрешены и возможны запросы прерывания на ЦП ВМ86, должен быть использован обычный, а не альтернативный мнемокод. Все команды (исключая FNSTENV и FNSAVE), имеющие альтернативный мнемокод, являются самосинхронизируемыми, т. е. не требуют для синхронизации команды WAIT.

Команда инициализации сопроцессора **FINIT/FNINIT** эквивалентна внешнему сигналу RESET, за исключением того, что она не влияет на синхронизацию выборки команд для процессоров ВМ86 и ВМ87. Отметим также, что если команда FNINIT начнет выполняться в то время, когда ВМ87 выполняет предыдущую команду, связанную с обращением к памяти, обращение будет прекращено.

Команда разрешения прерывания **FENI/FNENI** устанавливает в нуль маску IEM = 0 в регистре SR, разрешая АСП ВМ87 генерировать запросы на выходе INT.

Команда запрещения прерывания **FDISI/FNDISI** устанавливает маску IEM = 1, запрещая генерировать запросы прерывания.

Команда загрузки управляющего слова **FLDCW src** осуществляет загрузку регистра CR данными из памяти по адресу src. Эта команда обычно используется для задания или изменения режимов работы ВМ87 (см. рис. 3.4). Следует отметить, что если какой-либо

флаг исключительной ситуации в регистре SR установлен, то при загрузке нового управляющего слова, в котором этот флаг не замаскирован, и IEM = 0, немедленно генерируется запрос прерывания, до выполнения следующей команды. С учетом этой особенности рекомендуется при изменении режима сначала установить в нуль флаги исключительных ситуаций, а затем загружать новое управляющее слово.

Команда запоминания управляющего слова **FSTCW/FNSTCW dst** записывает текущее содержимое регистра CR и в памяти по адресу dst.

Команда запоминания слова-состояния **FSTSW/FNSTSW dst** записывает текущее содержимое регистра SR в память по указанному адресу dst. Эта команда обычно используется при организации условных переходов после выполнения команд сравнения (FCOM, FCOMP, FCOMPP, FICOM), для чего результат сравнения, полученный во флагах C0, C3 регистра SR, записывается в память и затем, после его загрузки в регистр F процессора ВМ86, используется команда условного перехода из системы команд ВМ86; для обработки исключительных ситуаций, путем анализа младшего байта регистра SR без использования прерываний (при IEM = 1).

Команда установки в нуль флагов исключительных ситуаций **FCLEX/FNCLEX** сбрасывает флаги IE, DE, ZE, OE, UE, PE, а также поле B и флаг IR в регистре состояния, SR.

Команда записи содержимого всех вспомогательных регистров сопроцессора в память **FSTENV/FNSTENV dst** по адресу dst. На рис. 3.8 показано размещение регистров в памяти при dst = addr. Обычно адрес addr выбирается в стековом сегменте ВМ86. Эта команда часто используется при обработке незамаскированных исключительных ситуаций, так как обеспечивает доступ ЦП к содержимому регистра EP, определяющему команду, при выполнении которой обнаружилась исключительная ситуация, а также адрес ее операнда.

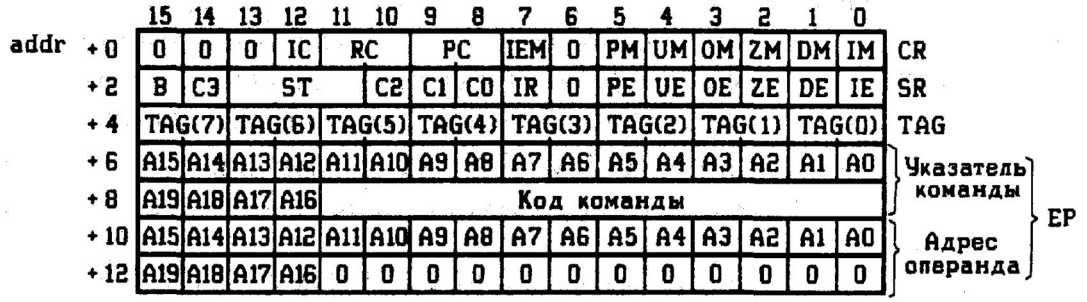

Рис. 3.8. Размещение содержимого регистров в памяти по команде FSTENV

После записи содержимого вспомогательных регистров в память команда FSTENV /FNSTENV устанавливает маски всех исключительных ситуаций в единицу, но не влияет на значение IEM. Если команда FNSTENV декодирована до завершения предыдущей команды, то она не осуществит запись, пока эта команда не будет выполнена полностью. Команда FNSTENV должна быть «защищена» от возможных прерываний в ЦП в момент выполнения. Ее можно использовать в тех случаях, когда прерывания в ВМ86 запрещены либо перед ней использована команда WAIT при разрешенных прерываниях (т. е. мнемокода FSTENV).

Команда **FLDENV src** осуществляет загрузку вспомогательных регистров данными из памяти по адресу src. Вслед за командой FLDENV может следовать любая команда для ЦП ВМ86, но никакая команда для АСП ВМ87 не может следовать непосредственно за FLDENV без промежуточной команды WAIT. Если загруженный с помощью команды FLDENV регистр SR содержит незамаскированный взведенный флаг исключительной ситуации (при IEM = 0), то он немедленно выдает запрос прерывания.

Команда **FSAVE/FNSAVE dst** осуществляет запись содержимого всех вспомогательных, а также арифметических регистров в память по адресу dst. На рис. 3.9 показано размещение регистров в памяти при dst = addr. Область для хранения этой информации (94 байт) обычно выбирается внутри стека ЦП ВМ86. Если команда FNSAVE декодирована до

завершения предыдущей команды, то ВМ86 задержит ее выполнение до полного завершения этой предыдущей команды, тем самым сохраняемая информация всегда отражает состояние ВМ87, соответствующее моменту полного завершения любой предшествующей команды. После записи информации команда FSAVE/FNSAVE осуществляет инициализацию ВМ87, т. е. выполняет дополнительно все действия, определяемые командой FINIT/FNIIT. Команда FSAVE/FNSAVE используется в программе, когда необходимо сохранить текущее состояние сопроцессора и инициализировать его на выполнение другой программы, например, когда ВМ86 при обработке прерывания нуждается в использовании ВМ87. Альтернативный мнемокод FNSAVE нельзя использовать, когда разрешены прерывания в процессоре ВМ86. В этом случае используется мнемокод FSAVE.

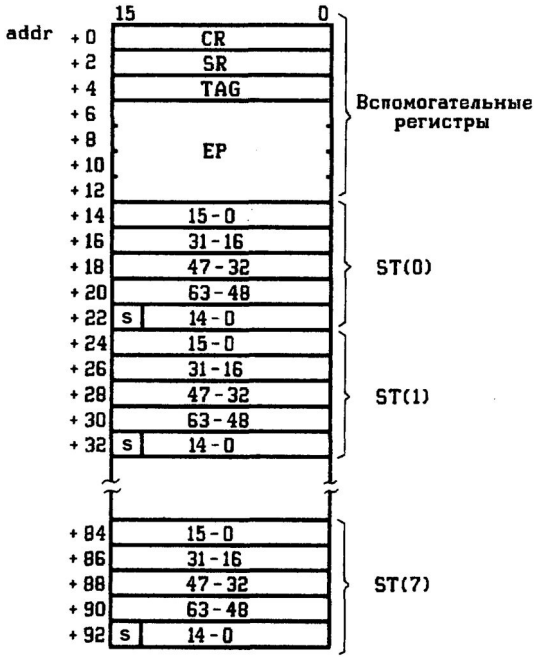

Рис. 3.9. Размещение содержимого регистров в памяти по команде FSAVE

Команда **FRSTOR src** осуществляет загрузку регистров сопроцессора данными из 94-байтовой области памяти, определяемой адресом src. Вслед за командой FRSTOR может следовать любая команда для ЦП ВМ86, но никакая команда для АСП ВМ87 не может следовать непосредственно за FRSTOR без промежуточной команды WAIT. После перезагрузки сопроцессор немедленно реагирует на новое состояние, например он сразу генерирует прерывание, если новое состояние укажет на наличие незамаскированной исключительной ситуации (конечно, только в случае, когда IEM = 0).

Команда **FINCSTP** увеличивает содержимое указателя стека ST на единицу. Она не изменяет содержимого арифметического регистра и соответствующих ему разрядов регистра TAG, а также не выполняет каких-либо пересылок. Команда FINCSTP не эквивалентна команде выталкивания из стека, так как не устанавливает разряды TAG, соответствующие предыдущей вершине, в код 11. Увеличение содержимого указателя стека, когда ST = 7, дает ST = 0.

Команда **FDECSTP** уменьшает содержимое указателя стека ST на единицу. Эта команда не влияет на содержимое арифметического регистра и регистра TAG, не осуществляет никаких пересылок данных. Выполнение команды FDECSTP, когда ST = 0. дает ST = 7.

Команда **FFREE ST(i)** устанавливает в соответствующие разряды регистра TAG (i) код 11, т. е. освобождает указанный арифметический регистр ST (i), не изменяя его содержимого.

Команда **FNOP** не выполняет никаких действий.

Команды управления не оказывают влияния на флаги исключительных ситуаций, за исключением команды FCLEX/FNCLEX, которая их сбрасывает. Влияние команд на флаги исключительных ситуаций показано в табл. 3.14.

В заключение отметим, что в программах на языке ассемблера ASM-86 можно использовать мнемокод **FWAIT,** являющийся альтернативным мнемокоду WAIT, когда программист хочет быть уверенным в том, что следующая команда не начнет выполняться раньше, чем полностью завершится предыдущая. Обычно команда FWAIT применяется для того, чтобы ЦП не воспользовался каким-либо операндом из памяти до тех пор, пока АСП ВМ87 не закончит выполнять предыдущую команду, связанную с обращением в память. Например, следующий фрагмент показывает возможное применение команды:

Таблица 3.14. Влияние команд на флаги исключительных ситуаций

№ п/п	Команда	Флаги исключительных ситуаций	№ п/п	Команда	Флаги исключительных ситуаций
1	FABS	IE,	27	FLDLN2	IE
2	FADD	IE,DE,OE,UE,PE	28	FLDL2E	IE
3	FADDP	IE,DE,OE,UE,PE	29	FLDL2T	IE
4	FBLD	IE	30	FLDPI	IE
5	FBSTP	IE	31	FLDZ	IE
6	FCHS	IE	32	FLD1	IE
7	FCOM	IE,DE	33	FMUL	IE,DE,OE,UE,PE
8	FCOMP	IE,DE	34	FMULP	IE,DE,OE,UE,PE
9	FCOMPP	IE,DE	35	FPATAN	IE,PE (операнды не проверяются)
10	FDIV	IE,DE,ZE,OE,UE,PE	36	FPREM	IE,DE,OE
11	FDIVP	IE,DE,ZE,OE,UE,PE	37	FPTAN	IE,PE (операнды не проверяются)
12	FDIVR	IE,DE,ZE,OE,UE,PE	38	FRNDINT	IE,PE
13	FDIVRP	IE,DE,ZE,OE,UE,PE	39	FSCALE	IE,OE,UE
14	FIADD	IE,DE,OE,PE	40	FSQRT	IE,DE,PE
15	FICOM	IE,DE	41	FST	IE,OE,UE,PE
16	FICOMP	IE,DE	42	FSTP	IE,OE,UE,PE
17	FIDIV	IE,DE,ZE,OE,UE,PE	43	FSUB	IE,DE,OE,UE,PE
18	FIDIVR	IE,DE,ZE,OE,UE,PE	44	FSUBP	IE,DE,OE,UE,PE
19	FILD	IE	45	FSUBR	IE,DE,OE,UE,PE
20	FIMUL	IE,DE,OE,PE	46	FSUBPR	IE,DE,OE,UE,PE
21	FIST	IE,PE	47	FTST	IE,DE
22	FISTP	IE,PE	48	FXCH	IE
23	FISUB	IE,DE,OE,PE	49	FXTRACT	IE
24	FISUBR	IE,DE,OE,PE	50	FYL2X	PE (операнды не проверяются)
25	FLD	IE,DE	51	FYL2XP1	PE (операнды не проверяются)
26	FLDLG2	IE	52	F2XM1	UE,PE (операнды не проверяются)

Примечания: 1. Команды перечислены в алфавитном порядке.
2. 16 команд, не включенных в таблицу, влияния на флаги исключительных ситуаций не оказывают.

```
FNSAVE ADDR1
FWAIT       ;  ожидание завершения FNSAVE
MOV AX, ADDR2
```

где ADDR2 — адрес в 94-байтовой области, расположенной с начального адреса ADDR1. Ассемблирующая программа заменяет мнемокод FWAIT на объектный код команды WAIT.

Контрольные вопросы

1. Какие форматы команд поддерживаются в АСП ВМ87?
2. Когда используется формат команды с постбайтом?
3. Охарактеризуйте кодирование полей постбайта.
4. Когда используется формат команды без постбайта?
5. Какая информация приводится в табл.3.11 для каждой команды и как она интерпретируется?
6. Охарактеризуйте мнемонику команд АСП ВМ87.
7. На какие группы разбита система команд АСП ВМ87?
8. Охарактеризуйте группу команд передачи данных.
9. Какие команды относятся к арифметическим?

10. Охарактеризуйте команды сравнения.
11. Охарактеризуйте команду анализа
12. Охарактеризуйте команды сложения.
13. Охарактеризуйте команды вычитания.
14. Охарактеризуйте команды умножения и деления
15. Какие команды относятся к специальным вычислительным?
16. Охарактеризуйте команду масштабирования **FSCALE** и команду получения частичного остатка от деления **FPREM.**
17. Охарактеризуйте команду округления до целого значения **FRNDINT.**
18. Охарактеризуйте команду выделения порядка и мантиссы **FXTRACT.**
19. Охарактеризуйте команды **FPTAN** и **FRATAN**.
20. Охарактеризуйте команды **F2XM1, FYL2X и FYL2XP1.**
21. Дайте общую характеристику командам управления.
22. Охарактеризуйте команды **FINIT/FNINIT, FENI/FNENI** и **FDISI/FNDISI.**
23. Охарактеризуйте команды **FLDCW src** и **FSTCW/ FNSTCW dst**.
24. Охарактеризуйте команды **FSTSW/FNSTSW dst и FCLEX/FNCLEX.**
25. Охарактеризуйте команды **FSTENV/FNSTENV dst** и **FLDENV src.**
26. Охарактеризуйте команды **FSAVE/FNSAVE dst и FRSTOR src.**
27. Охарактеризуйте команды **FINCSTP, FDECSTP, FFREE ST(i) и FNOP.**
28. Охарактеризуйте контент таблицы 3.14.
29. Когда можно пользоваться командой **FWAIT**?

3.5. Специальные случаи использования арифметического сопроцессора

Специальные случаи достаточно редко встречающиеся на практике, но представляют определенный интерес для пользователей. К ним относятся действия сопроцессора над **денормализованными** и **ненормализованными числами**, **кодами нулей**, **±∞, NAN и кодами неопределённости.** В табл. 3.15 кратко перечислены условия, приводящие к исключительным ситуациям, и дается описание реакции на них сопроцессора.

Денормализованные числа возникают в результате вычислений, приводящих к исключительной ситуации антипереполнения (UE = 1), когда она замаскирована (UM = 1). Антипереполнение происходит, если порядок результата настолько мал, что не может быть представлен в нужном формате. Например, значение порядка □130 вызывает антипереполнение, когда результат должен быть представлен в формате KB, поскольку наименьший представимый порядок равен □126. Конечно, это значение порядка не вызывает антипереполнения при формате результата DB или BB, так как их наименьшие представимые порядки равны □1 023 и □16 383 соответственно.

Незамаскированная реакция АСП BM87 на антипереполнение выражается в прекращении дальнейших вычислений и выдаче запроса прерывания, если результат должен быть записан в память. Если приемником результата является регистр, сопроцессор прибавляет к истинному значению порядка константу 24 576, записывает результат и выдает запрос прерывания. Эта константа возвращает значение экспоненты в представимый формат BB и при дальнейшей обработке прерывания ее следует вычесть для получения истинного значения порядка.

Маскирование антипереполнения позволяет продолжать вычисления, не обращая внимания на денормализованные числа. Часто в процессе дальнейших вычислений получается нормализованный результат. Чтобы убедиться в этом, достаточно после завершения вычислений проанализировать флаг IE. Если IE = 0, то конечный результат правильный.

Таблица 3.15. Реакция сопроцессора на исключительные ситуации

Условия	Реакция ВМ87
Недействительная операция (IE)	
Регистр-источник пуст (TAG=11)	Возвращает код неопределенности (в формате ВВ)
Регистр-приемник заполнен (TAG 11)	Записывает в него код неопределенности (в формате ВВ)
Один или оба операнда NAN	Возвращает код NAN с большим абсолютным значением
Один или оба операнда NAN, один операнд проективная (в командах сравнения и FTST)	Устанавливает код "Несравнимы" (C3C0=11)
Операнды - коды бесконечности с противоположными знаками (аффинная арифметика) или оба операнда (проективная арифметика, знак не играет роли) - команды сложения и вычитания	Возвращает код неопределенности (в формате ВВ)
Умножение ∞ x0 или 0x∞	То же
Деление ∞/∞, 0/0, 0/псевдонуль, делитель не нормализован или денормализован	>>
Команда FPREM - делитель не нормализован или денормализован	Возвращает код неопределенности (в формате ВВ), устанавливает код "Команда завершения" (C2=0)
Команда FSQRT - операнд не нормализован или денормализован, отрицательное число, не равное нулю, - ∞ (аффинная арифметика), ∞ (проективная арифметика)	Возвращает код неопределенности (в формате ВВ)
Команды FIST, FISTP - регистр-источник пуст, денормализован или не нормализован, NAN, ∞, непредставим в указанном формате	Записывает код неопределенности (в формате ВВ)
Команда FBSTP - регистр-источник пуст, денормализован или не нормализован, NAN, ∞, превышает 18 десятичных цифр	Записывает код неопределенности (в формате ДД)
Команды FST, FSTP - регистр-источник содержит ненормализованные значения, а регистр-приемник определен форматом КВ или ДВ	Запоминает код неопределенности (в формате КВ или ДВ)
Команда FXCH - один или оба регистра пусты	Заполняет пустой/пустые регистры кодом неопределенности и совершает обмен
Денормализованный операнд (DE)	
Команда FLD - операнд-источник денормализован	Загружает как обычный (никаких специальных действий)

Окончание табл. 3.15

Условия	Реакция ВМ87
Арифметические команды - один или оба операнда денормализованы	Преобразует операнд (операнды) в эквивалентное ненормализованное значение и выполняет команду
Деление на нуль (ZE)	
Деление - делитель равен ±0	Возвращает код ∞ со знаком, равным сумме по модулю два операндов
Переполнение (OE)	
Арифметические команды - округление к ближайшему значению или по направлению к нулю, а порядок результата больше 16383	Возвращает код ∞ с соответствующим знаком и устанавливает PE=1
Команды FST, FSTP - округление к ближайшему значению или по направлению к нулю, а порядок результата больше 127 (приемник КВ) или 1023 (приемник ДВ)	То же
Антипереполнение (UE)	
Арифметические команды - порядок результата меньше 16382	Денормализует вплоть до порядка -16382 (в смещенном виде 00...00), если 64-битовая мантисса при этом содержит хотя бы одну единицу, возвращает денормализованное значение, иначе возвращает код 0
Команды FST, FSTP - приемник в формате КВ/ДВ, а порядок результата меньше -126/-1022	Денормализует вплоть до порядка -126/-1022 (в смещенном виде 00...00), если 24-битовая/53-битовая мантисса содержит хотя бы одну единицу, возвращает денормализованное значение, иначе возвращает код 0
Неточный результат (PE)	
Округление неточное	Никаких специальных действий
Замаскированный флаг OE (OM=1) в ряде команд	То же

Ненормализованные числа образуются из денормализованных чисел и также возникают при замаскированном антипереполнении. Ненормализованные числа существуют только в формате ВВ (см. табл. 3.9), они могут иметь порядок, схожий с порядком нормализованных чисел, но отличаются от них тем, что первая цифра их мантиссы всегда равна «0». С ненормализованными числами сопроцессор оперирует как с нормализованными, причем в процессе вычислений результат может нормализоваться.

Рядом особенностей обладают действия сопроцессора над числами, представленными **кодами нулей**. Как показано в табл. 3.6, форматы целых чисел содержат код +0, в то время как форматы вещественных и двоично-десятичных чисел включают коды +0 и –0 (табл. 3.8 и 3.7).

Только один формат ВВ включает специальный класс значений, которые можно назвать **псевдонулями**. Это такие ненормализованные числа, мантисса которых содержит все нули, а порядок не нулевой (обычные нули имеют нулевой порядок). Никакой псевдонуль не может иметь порядок, состоящий из всех единиц, так как он зарезервирован для NAN. Псевдонулевой результат может быть получен в тех случаях, когда два ненорма-

лизованных операнда, имеющие в сумме более чем 64 старших нулевых бита, перемножаются.

Псевдонулевые операнды обрабатываются как обычные ненормализованные операнды, за исключением случаев выполнения команд сравнения и команды FTST; команды FRNDINT; команды деления, когда делимое нуль или псевдонуль (делитель псевдонуль). В перечисленных ситуациях псевдонули дают те же результаты, что и обычные нули.

Когда в качестве операндов выступают **коды** ±∞, результаты действий сопроцессора в ряде случаев зависят от режима работы сопроцессора (проективный или аффинный) (см. § 3.3, стр. 85 [1]).

Действия сопроцессора над **кодами NAN**, обозначающими «не числа», приводят к установке флага недействительной операции IE = 1 и запросу прерывания. В тех случаях, когда эта исключительная ситуация замаскирована, то любая операция с операндом NAN дает в качестве результата тот же код NAN. Если в операции оба операнда являются NAN, то в качестве результата выбирается код того из них, который имеет большее абсолютное значение.

(Более подробную информацию по контенту § 3.5 можно найти в [1] стр.104 и в [2] стр.114).

3.6. Мнемоника команд сопроцессора ВМ87 в среде отладчиков Debug и AFD

В отличие от команд МП ВМ86, в которых используется представление данных в виде байтов и 16-разрядных слов, в командах ВМ87 дополнительно используются данные в форматах: короткого целого – 4 байта (КЦ), длинного целого – 8 байт, двоично-десятичное – 10 байт (ДД), короткое вещественное – 4 байта (КВ), длинное вещественное – 8 байт и временное вещественное – 10 байт (ВВ) (см. рис. 3.6 и табл. 3.5).

В мнемонике Ассемблера, которую поддерживает отладчик Debug, в командах использующих данные применяются префиксные описания их размерности: WORD PTR – 2 байта, DWORD PTR – 4 байта, QWORD PTR – 8 байт и TBYTE PTR – 10 байт.

В Ассемблере и в Debug используется также общепринятая мнемоника команд сопроцессора ВМ87 начинающаяся с символа F (float) (см. табл. 3.11).

В отладчике AFD не поддерживаются команды начинающиеся с символа F. Вместо использования общепринятой мнемоники в ATD предусмотрено использование команды микропроцесора ВМ86 с префиксом ESC (11011). Команда с таким префиксом передается для выполнения в сопроцессор ВМ87. Форматом этой команды (см. команду с номером 21 в табл.2.9) предусмотрено кодирование команд сопроцессора разрядами 2 – 0 первого байта и разрядами 5 – 3 второго байта (постбайта) команды. В постбайте также предусмотрены поля md (разряды 7 и 6) и r/m (разряды 2 – 0) для задания способа адресации оперативной памяти. Значеня этих полей определяются по таблице 3.10. **Следует отметить что в среде AFD в последнем столбце таблицы 3.10 для адресации регистров ST(i) необходимо использовать предпоследний столбец таблицы 2.3. Например: вместо мнемоники ST(0) необходимо использовать мнемонику AL, ST(1) – CL, ST(2) – DL, ST(3) – BL, ST(4) – AH, ST(5) – CH, ST(6) – DH, ST(7) – BH.**

Для примера в таблице 3.16 приводится список команд сопроцессора ВМ87 соответственно в мнемонике отладчиков Debug и AFD. Список приведен для частных случаев адресации оперативной памяти и арифметических регистров сопроцессора. В командах обмена с оперативной памятью применяется косвенная регистровая адресация с использованием регистра BX для указания адреса ячейки памяти. В командах с явной адресацией арифметических регистров используется регистр ST(1) в отладчике Debug и соответствующая ему мнемоника CL в отладчике AFD.

В таблице также приведены шестнадцатеричные значения первого и второго байта соответствующего двоичного кода команды.

Таблица 3.16. Мнемоника команд сопроцессора ВМ87 в среде отладчиков Debug и AFD

№ п/п	Мнемокод	Кодировка DEBUG		Кодир. AFD	Операция dst src	Байты кода команды B1	B2
1.	FLD src	FLD	DWORD PTR [BX]	ESC 08,[BX]	ST(0) ← mem(КВ)↓	D9	07
		FLD	QWORD PTR [BX]	ESC 28,[BX]	ST(0) ← mem(ДВ)↓	DD	07
		FLD	TBYTE PTR [BX]	ESC 1D,[BX]	ST(0) ← mem(ВВ)↓	DB	2F
		FLD	ST(1)	ESC 08,CL	ST(0) ← ST(1)↓	D9	C1
2.	FILD src	FILD	WORD PTR [BX]	ESC 38,[BX]	ST(0) ← mem(ЦС)↓	DF	07
		FILD	DWORD PTR [BX]	ESC 18,[BX]	ST(0) ← mem(КЦ)↓	DB	07
		FILD	QWORD PTR [BX]	ESC 3D,[BX]	ST(0) ← mem(ДЦ)↓	DF	2F
3	FBLD src	FBLD	TBYTE PTR [BX]	ESC 3C,[BX]	ST(0) ← mem(ДД)↓	DF	27
4.	FST dst	FST	DWORD PTR [BX]	ESC 0A,[BX]	mem(КВ) ← ST(0)	D9	17
		FST	QWORD PTR [BX]	ESC 2A,[BX]	mem(ДВ) ← ST(0)	DD	17
		FST	ST(1)	ESC 2A,CL	ST(1) ← ST(0)	DD	D1
5.	FIST dst	FIST	WORD PTR [BX]	ESC 3A,[BX]	mem(ЦС) ← ST(0)	DF	17
		FIST	DWORD PTR [BX]	ESC 1A,[BX]	mem(КЦ) ← ST(0)	DB	17
6.	FSTP dst	FSTP	DWORD PTR [BX]	ESC 0B,[BX]	mem(КВ) ← ST(0)↑	D9	1F
		FSTP	QWORD PTR [BX]	ESC 2B,[BX]	mem(ДВ) ← ST(0)↑	DD	1F
		FSTP	TBYTE PTR [BX]	ESC 1F,[BX]	mem(ВВ) ← ST(0)↑	DB	3F
		FSTP	ST(1)	ESC 2B,CL	ST(1) ← ST(0)↑	DD	D9
7.	FISTP dst	FISTP	WORD PTR [BX]	ESC 3B,[BX]	mem(ЦС) ← ST(0)↑	DF	1F
		FISTP	DWORD PTR [BX]	ESC 1B,[BX]	mem(КЦ) ← ST(0)↑	DB	1F
		FISTP	QWORD PTR [BX]	ESC 3F,[BX]	mem(ДЦ) ← ST(0)↑	DF	3F
8.	FDSTP dst	FDSTP	TBYTE PTR [BX]	ESC 3E,[BX]	mem(ВВ) ← ST(0)↑	DF	37
9.	FXCH ST(1)	FXCH	ST(1)	ESC 09,CL	ST(0) ↔ ST(1)	D9	C9
10.	FLDZ	FLDZ		ESC 0D,DH	ST(0) ← +0.0↓	D9	EE
11.	FLD1	FLD1		ESC 0D,AL	ST(0) ← +1.0↓	D9	E8
12.	FLDPI	FLDPI		ESC 0D,BL	ST(0) ← π↓	D9	EB
13.	FLDL2T	FLDL2T		ESC 0D,CL	ST(0) ← $\log_2 10$↓	D9	E9
14.	FLDL2E	FLDL2E		ESC 0D,DL	ST(0) ← $\log_2 e$↓	D9	EA
15.	FLDLG2	FLDLG2		ESC 0D,AH	ST(0) ← lg2↓	D9	EC
16.	FLDLN2	FLDLN2		ESC 0D,CH	ST(0) ← ln2↓	D9	ED
17.	FCOM src	FCOM	DWORD PTR [BX]	ESC 02,[BX]	ST(0) – mem(КВ)	D8	17
		FCOM	QWORD PTR [BX]	ESC 22,[BX]	ST(0) – mem(ДВ)	DC	17
		FCOM	ST(1)	ESC 02,CL	ST(0) – ST(1)	D8	D1
18.	FICOM src	FICOM	WORD PTR [BX]	ESC 32,[BX]	ST(0) – mem(ЦС)	DE	17
		FICOM	DWORD PTR [BX]	ESC 12,[BX]	ST(0) – mem(КЦ)	DA	17
19.	FCOMP src	FCOMP DWORD PTR [BX]		ESC 03,[BX]	ST(0) – mem(КВ)↑	D8	1F
		FCOMP QWORD PTR [BX]		ESC 23,[BX]	ST(0) – mem(ДВ)↑	DC	1F
		FCOMP ST(1)		ESC 03,CL	ST(0) – ST(1)↑	D8	D9
20.	FICOMP src	FICOMP WORD PTR [BX]		ESC 33,[DX]	ST(0) – mem(ЦС)↑	DE	1F
		FICOMP DWORD PTR [BX]		ESC 13,[DX]	ST(0) – mem(КЦ)↑	DE	17
21.	FCOMPP	FCOMPP ST(1)		ESC 33,CL	ST(0) – ST(1)↑↑	DE	D9
22.	FTST	FTST		ESC 0C,AH	ST(0) – 0,0	D9	E4
23.	FXAM	FXAM		ESC 0C,CH	анализ ST(0)	D9	E5
24.	FADD dst,src	FADD DWORD PTR [BX]		ESC 0D,DL	ST(0) ← ST(0) + mem(КВ)	D8	07
		FADD QWORD PTR [BX]		ESC 20[BX]	ST(0) ← ST(0) + mem(ДВ)	DC	07
		FADD ST, ST(1)		ESC 00,CL	ST(0) ← ST(0) + ST(1)	D8	C1
		FADD ST(1), ST		ESC 20,CL	ST(1) ← ST(0) + ST(1)	DC	C1
25.	FIADD src	FIADD WORD PTR [BX]		ESC 30,[BX]	ST(0) ← ST(0) + mem(ЦС)	DE	07
		FIADD DWORD PTR [BX]		ESC 10,[BX]	ST(0) ← ST(0) + mem(КЦ)	DA	07
26.	FADDP dst,src	FADDP ST(1), ST		ESC 30,CL	ST(1) ← ST(0) + ST(1)↑	DE	C1
27.	FSUB dst,src	FSUB DWORD PTR [BX]		ESC 04,[BX]	ST(0) ← ST(0) – mem(КВ)	D8	27
		FSUB QWORD PTR [BX]		ESC 24,[BX]	ST(0) ← ST(0) – mem(ДВ)	DC	27
		FSUB ST,ST(1)		ESC 04,CL	ST(0) ← ST(0) – ST(1)	D8	E1
		FSUB ST(1),ST		ESC 25,CL	ST(1) ← ST(1) – ST(0)	DC	E9
28.	FSUBR dst,src	FSUBR DWORD PTR [BX]		ESC 05,[BX]	ST(0) ← mem(КВ) – ST(0)	D8	2F
		FSUBR QWORD PTR [BX]		ESC 25,[BX]	ST(0) ← mem(КВ) – ST(0)	DC	2F
		FSUBR ST,ST(1)		ESC 05,CL	ST(0) ← ST(1) – ST(0)	D8	E9
		FSUBR ST(1),ST		ESC 24,CL	ST(1) ← ST(0) – ST(1)	DC	E1
29.	FSUBP dst,src	FSUBP ST(1),ST		ESC 35,CL	ST(1) ← ST(1) – ST(0)↑	DE	E9
30.	FSUBRP dst,src	FSUBRP ST(1),ST		ESC 34,CL	ST(1) ← ST(0) – ST(1)↑	DE	E1

Таблица 3.16 (продолжение)

№ п/п	Мнемокод	Кодировка DEBUG	Кодир. AFD	Операция dst	Операция src	Байты кода команды B1	Байты кода команды B2
31.	FISUB src	FISUB WORD PTR [BX]	ESC 34,[BX]	ST(0) ← ST(0) – mem(ЦС)		DE	27
		FISUB DWORD PTR [BX]	ESC 14,[BX]	ST(0) ← ST(0) – mem(КЦ)		DA	27
32.	FISUBR src	FISUBR WORD PTR [BX]	ESC 35,[BX]	ST(0) ← mem(ЦС) – ST(0)		DE	2F
		FISUBR DWORD PTR [BX]	ESC 15,[BX]	ST(0) ← mem(КЦ) – ST(0)		DA	2F
33.	FMUL dst,src	FMUL DWORD PTR [BX]	ESC 01,[BX]	ST(0) ← ST(0) × mem(КВ)		D8	0F
		FMUL QWORD PTR [BX]	ESC 21,[BX]	ST(0) ← ST(0) × mem(ДВ)		DC	0F
		FMUL ST,ST(1)	ESC 01,CL	ST(0) ← ST(0) × ST(1)		D8	C9
		FMUL ST(1),ST	ESC 21,CL	ST(1) ← ST(0) × ST(1)		DC	C9
34.	FIMUL src	FIMUL WORD PTR [BX]	ESC 31,[BX]	ST(0) ← ST(0) × mem(ЦС)		DE	0F
		FIMUL DWORD PTR [BX]	ESC 11,[BX]	ST(0) ← ST(0) × mem(КЦ)		DA	0F
35.	FMULP dst,src	FMULP ST(1),ST	ESC 31,CL	ST(1) ← ST(0) × ST(1)↑		DE	C9
36.	FDIV dst,src	FDIV DWORD PTR [BX]	ESC 06,[BX]	ST(0) ← ST(0) / mem(КВ)		D8	37
		FDIV QWORD PTR [BX]	ESC 26,[BX]	ST(0) ← ST(0) / mem(ДВ)		DC	37
		FDIV ST,ST(1)	ESC 06,CL	ST(0) ← ST(0) / ST(1)		D8	F1
		FDIV ST(1),ST	ESC 27,CL	ST(1) ← ST(1) / ST(0)		DC	F9
37.	FDIVR dst,src	FDIVR DWORD PTR [BX]	ESC 07,[BX]	ST(0) ← mem(КВ) / ST(0)		D8	3F
		FDIVR QWORD PTR [BX]	ESC 27,[BX]	ST(0) ← mem(ДВ) / ST(0)		DC	3F
		FDIVR ST,ST(1)	ESC 07,CL	ST(0) ← ST(1) / ST(0)		D8	F9
		FDIVR ST(1),ST	ESC 26,CL	ST(1) ← ST(0) / ST(1)		DC	F1
38.	FDIVP dst,src	FDIVP ST(1),ST	ESC 37,CL	ST(1) ← ST(1) / ST(0)↑		DE	F9
39.	FDIVRP dst,src	FDIVRP ST(1),ST	ESC 36,CL	ST(1) ← ST(0) / ST(1)↑		DE	F1
40.	FIDIV src	FIDIV WORD PTR [BX]	ESC 36,[BX]	ST(0) ← ST(0) / mem(ЦС)		DE	37
		FIDIV DWORD PTR [BX]	ESC 16,[BX]	ST(0) ← ST(0) / mem(КЦ)		DA	37
41.	FIDIVR src	FIDIVR WORD PTR [BX]	ESC 37,[BX]	ST(0) ← mem(ЦС) / ST(0)		DE	3F
		FIDIVR DWORD PTR [BX]	ESC 17,[BX]	ST(0) ← mem(КЦ) / ST(0)		DA	3F
42.	FSQRT	FSQRT	ESC 0F,DL	ST(0) ← $\sqrt{ST(0)}$		D9	FA
43.	FSCALE	FSCALE	ESC 0F,CH	ST(0) ← ST(0) × $2^{ST(1)}$		D9	FD
44.	FPREM	FPREM	ESC 0F,AL	ST(0) ← ST(0) × MODST(1)		D9	F8
45.	FRNDINT	FRNDINT	ESC 0F,AH	ST(0) ← целое ST(0)		D9	FC
46.	FXTRACT	FXTRACT	ESC 0E,AH	ST(0) ← мантисса ST(1) ← порядок		D9	F4
47.	FABS	FABS	ESC 0C,CL	ST(0) ← |ST(0)|		D9	E1
48.	FCHS	FCHS	ESC 0C,AL	ST(0) ← –ST(0)		D9	E0
49.	FPTAN	FPTAN	ESC 0E,DL	ST(0), ST(1) ← tg(z/2)		D9	F2
50.	FPATAN	FPATAN	ESC 0E,BL	ST(0) ← arctg(X/Y)		D9	F3
51.	F2FM1	F2XM1	ESC 0E,AL	ST(0) ← $2^{ST(0)}$–1		D9	F0
52.	FYL2X	FYL2X	ESC 0E,CL	ST(0) ← $Y\log_2 X$		D9	F1
53.	FYL2XP1	FYL2XP1	ESC 0F,CL	ST(0) ← $Y\log_2(X+1)$		D9	F9
54.	FINIT	FINIT	ESC 1C,BL	Инициализация		DB	E3
55.	FENT	FENI	ESC 1C,AL	IEM ← 0		DB	E0
56.	FDISI	FDISI	ESC 1C,CL	IEM ← 1		DB	E1
57.	FLDCW src	FLDCW [BX]	ESC 0D,[BX]	CR ← src		D9	2F
58.	FSTCW dst	FSTCW [BX]	ESC 0F,[BX]	dst ← CR		D9	3F
59.	FSTSW dst	FSTSW [BX]	ESC 2F,[BX]	dst ← SR		DD	3F
60.	FCLEX	FCLEX	ESC 1C,DL	PE,UE,OE,ZE,DE,IE ← 0		DB	E2
61.	FSTENV dst	FSTENV [BX]	ESC 0E,[BX]	dst←CR,SR,TAG,EP		D9	37
62.	FLDENV src	FLDENV [BX]	ESC 0C,[BX]	CR,SR,TAG,EP←src		D9	27
63.	FSAVE dst	FSAVE [BX]	ESC 2E,[BX]	dst←CR,SR,TAG,EP, ST(0) – ST(7)		DD	37
64.	FRSTOR src	FRSTOR [BX]	ESC 2C,[BX]	CR,SR,TAG,EP, ST(0) –ST(7) ← src		DD	27
65.	FINCSTP	FINCSTP	ESC 0E,BH	ST ← ST + 1		D9	F7
66.	FDECSTP	FDECSTP	ESC 0E,DH	ST ← ST – 1		D9	F6
67	FFREE ST(1)	FFREE ST(1)	ESC 28,CL	TAG(1) ← 11		DD	C1
68	FNOP	FNOP	ESC 0A,AL	ST(0) ← ST(0)		D9	D0

Достаточно подробное описание отладчиков Debug и AFD можно найти в [3].

Контрольные вопросы

1. Охарактеризуйте специальные случаи, возникающие в АСП ВМ87.
2. Что такое денормализованные числа и антипереполнение, когда оно возникает и как реагирует на него АСП ВИ87?
3. Как возникают ненормализованные числа, и как реагирует на них АСП ВИ87?
4. Какие бывают нули и псевдонули в АСП ВИ87?
5. Как реагирует АСП ВИ87 на коды ±∞**?**
6. Как реагирует АСП ВИ87 на коды NAN?
7. Какие префиксные описания размерности данных команд сопроцессора ВМ87 используются в ассемблере и отладчике Debug?
8. Как описываются команды сопроцессора ВМ87 а среде отладчика AFD?
9. Как обозначаются имена арифметических регистров сопроцессора МВ87 в мнемонике отладчиков AFD и Debug?
10. Как можно расшифровать шестнадцатеричные коды команд сопроцессора приведенные в таблице 3.16?
11. Охарактеризуйте форму описания операций сопроцессора используемую в таблице 3.16.

Список литературы

1. Микропроцессорный комплект К1810: Структура, программирование: Справочная книга/ Ю.М. Казаринов и др.; Под ред. Ю.М. Казаринова. – М.: Высш. шк., 1990. – 269 с.: ил.
2. Григорьев В.Л. Архитектура и программирование арифметического сопроцессора. – М.: Энергоатомиздат, 1991. – 208 с.: ил.
3. В.А. Павлов. Система ввода-вывода. Параллельный порт. Учебно-методическое пособие и практикум. СарФТИ, Саров, 2015. -204 с.: ил. (http://sarfti.ru/wp-content/uploads/2014/05/pavlov-v.a.-lpt-port.-posobie-i-praktikum.pdf)