Глава 1

МИКРОПРОЦЕССОР К1810ВМ86 КАК ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР

Микропроцессор К1810ВМ86 относится к классу однокристальных центральных процессоров с фиксированными системой команд и длиной слова. Он является базовым для нескольких моделей профессиональных персональных компьютеров, обладающих очень широкими возможностями. Безусловно, этот микропроцессор будет применяться и во многих специализированных системах, где требуется высокий уровень «машинного интеллекта». С появлением арифметического сопроцессора будут реализованы мощные микросистемы, предназначенные для сложной обработки данных. Однокристальный процессор ввода-вывода расширит область их применения на системы с интенсивными операциями ввода-вывода.

Естественным следствием увеличения возможностей микропроцессоров является все более громоздкое и сложное описание их функционирования, что приводит и к усложнению языка ассемблера. В то же время для программирования систем с арифметическим сопроцессором К1810ВМ87 необходимо знать именно язык ассемблера, так как языки высокого уровня с поддержкой арифметического сопроцессора пока не получили широкого распространения. Поэтому в настоящей главе очень кратко рассмотрена архитектура микропроцессора К1810ВМ86 [1—3].

1.1. ОБЩАЯ ХАРАКТЕРИСТИКА МИКРОПРОЦЕССОРА

Микросхема К1810ВМ86 выполнена по высококачественной NMOП — технологии с плотностью упаковки около 30 000 транзисторов. Адресное пространство памяти составляет 1М байт, и адрес любого байта состоит из 20 бит. Два смежных байта образуют слово; адресом слова считается меньший адрес, по которому хранится младший байт слова. Принцип «младшее — по меньшему адресу» распространяется на хранение всех производных от байта единиц данных, в том числе и на данные арифметического сопроцессора. Микропроцессор (МП) оперирует только байтами и словами, поэтому все обрабатываемые им адресные объекты должны иметь длину 16 бит. Для образования 20-битных физических адресов памяти применяется специальный механизм сегментации памяти.

С точки зрения программиста наиболее важными элементами МП являются внутренние регистры, к которым приходится обращаться при разработке ассемблерных программ. Микропроцессор имеет 14 16-битных регистров. Четыре регистра общего назначения (РОН) могут содержать любые данные и находятся в полном распоряжении программиста, хотя в некоторых командах они выполняют специальные функции. Два указательных и два индексных регистра предназначены для адресации данных в памяти, т. е. привлекаются для формирования эффективного адреса в соответствии с режимами адресации. Но они же могут участвовать в арифметических и логических операциях, что повышает гибкость адресации памяти. Четыре сегментных регистра, идентифицирующих базовые (начальные) адреса сегментов, привлекаются для образования физических адресов памяти. При каждом обращении к памяти один из сегментных регистров участвует в формировании физического адреса. Регистр указателя команды IP адресует программную память, т.е. выполняет функции программного счетчика PC (или счетчика команд). Регистр флажков разделен на две половины. Его младший байт содержит пять арифметических флажков, фиксирующих

особенности (признаки) результатов операций. В старшем байте находятся флажок переполнения и три флажка управления МП.

Микропроцессор имеет развитую систему команд, многие из которых оперируют байтами и словами. Предусмотрены команды всех арифметических операций, разнообразных сдвигов, команды манипуляций цепочками, множество команд передачи управления, команды прерываний и управления МП. Общая организация «регистр — память» предполагает, что в двухоперандных командах нельзя адресовать две ячейки памяти. Режимы адресации МП ориентированы на эффективную реализацию языков высокого уровня и обработку сложных структур данных.

Пространство ввода-вывода образуют 64К портов ввода-вывода, причем их адресация осуществляется без привлечения механизма сегментации. Имеются прямая и косвенная адресация портов. Через порты разрешается вводить/выводитъ байты и слова.

Взаимодействие МП с другими компонентами системы производится с помощью мультиплексной шины адреса/данных/состояния и управляющих сигналов. Особенностью МП является возможность работы в двух конфигурациях, определяемых уровнем сигнала на одном из входов. В зависимости от конфигурации изменяются функции восьми управляющих сигналов. В максимальной конфигурации, предназначенной для сложных мультипроцессорных систем, в частности систем с сопроцессором, управляющие сигналы формирует контроллер шины (микросхема К1810ВГ88).

Во внутренней организации МП выделяются два сравнительно автономных устройства: шинный интерфейс и операционное устройство, первое из которых осуществляет выборку команд и считывание/запись данных, а второе собственно выполняет команды. Эти устройства работают параллельно, т.е. шинный интерфейс самостоятельно инициирует выборку следующей команды из памяти, в то время как операционное устройство занято выполнением ранее выбранной команды. Считанная шинным интерфейсом команда попадает во внутреннюю очередь (буфер) команд, и, когда операционное устройство заканчивает выполнение текущей команды, оно обычно находит следующую команду в очереди. Благодаря такой опережающей выборке команд повышается производительность МП. Когда операционному устройству требуется обращение к памяти данных, оно запрашивает его у шинного интерфейса. Принцип опережающей выборки команд показан на рис. 1.1.

1.2. СИГНАЛЬНЫЕ ЛИНИИ

Взаимодействие МП с внешней средой заключается в выполнении одного из двух действий: МП либо выводит (записывает) данные, либо вводит (считывает) данные или команды. В каждом из этих действий он должен адресовать ячейку памяти или порт ввода-вывода.

Для передачи данных или выборки команды МП инициирует цикл шины: строго определенную последовательность событий, в результате которой слово (байт) передается из источника в получатель. В течение цикла шины МП выдает адрес ячейки памяти или порта ввода-вывода, а также показывает сигналами состояния тип цикла шины. Эти сигналы контроллер шины преобразует в необходимые управляющие импульсы, определяющие конкретные действия в цикле шины. Адресованное устройство либо воспринимает данные с шины (МП выдает их после адреса), либо помещает на шину свои данные.

На рис. 1.2 показано группирование сигнальных линий МП по функциональному

назначению. Уровень напряжения на входе MN/MX задает конфигурацию МП — минимальную или максимальную. Рассмотрим *назначение сигнальных линий в максимальной конфигурации.*

Системную шину адреса/данных/состояния образуют 20 линий.

AD15-AD0 — мультиплексная двунаправленная шина адреса/данных. По ее 16 тристабильным линиям с разделением во времени передаются младшие 16 бит адреса памяти (или полные адреса ввода-вывода) и данные.

A19/ST6-A16/ST3 — мультиплексная выходная шина адреса/состояния. В начале цикла шины на эти линии выдаются старшие 4 бита адреса памяти, а затем сигналы состояния ST, которые обычно не используются.

Шесть выходных сигналов определяют действия МП в текущем цикле шины.

R (или RD) — считывание. Сигнал R идентифицирует считывание из памяти или ввода-вывода.

ST2—ST0 — сигналы состояния. Эти сигналы в закодированной форме показывают тип текущего цикла шины. Они подаются в контроллер шины, который формирует управляющие импульсы для системной шины.

LOCK — блокировка шины. Активный сигнал на этом выходе информирует о том, что доступ к шине должен быть заблокирован (запрещен). Он генерируется по однобайтному префиксу блокировки шины и поддерживается активным до окончания следующей за префиксом команды.

BHE/ST7 — разрешение старшего байта/состояние. Низкий уровень этого сигнала в начале цикла шины означает, что байт данных передается по старшей половине AD15-AD8 шины данных.

Два сигнала предназначены только для сопроцессоров.

QSl-QS0 — состояние очереди команд. Значения эти. выходных сигналов идентифицируют операцию очереди команд. Сопроцессор также поддерживает очередь команд, но его сигналы QS1-QS0 являются входными Соединение линии микропроцессора и сопроцессора обеспечивает синхронизацию действий обоих устройств.

Две двунаправленные линии предназначены для разделения шины с другими компонентами системы, которые могут управлять шиной.

RQ/E1, RQ/E0 — запрос/разрешение (подтверждение) шины. Последовательность запроса/разрешения представляет собой трехфазный цикл. Сначала запрашивающий шину процессор генерирует отрицательный импульс на линии RQ/E. Микропроцессор, действующий как центральный процессор, отвечает выходным импульсом на той же линии, сигнализируя о приостановке своих действий, и освобождает шину. В течение последующего временного интервала шинный интерфейс логически отключен от шины. Когда, другой процессор закончил операции на шине, он генерирует импульс на линии RQ/E, и МП начинает снова управлять шиной. Оба взаимодействующих процессора должны синхронизироваться от одного источника синхроимпульсов. Запрос по линии RQ/EO имеет больший приоритет, чем запрос по линии RQ/E1. Если в системе имеются арифметический сопроцессор и процессор ввода-вывода, то запрос от арифметического сопроцессора подается по линии RQ/E1.

Две сигнальные линии относятся к прерываниям.

NMI — немаскируемое прерывание. Этот входной импульсный сигнал МП распознает по завершении текущей команды независимо от того, разрешены прерывания или нет. По входу NMI сообщается о таких критических ситуациях, как аварийное отключение сети или ошибка в памяти.

INTR— запрос прерывания. Входной потенциальный сигнал показывает, что внешнее устройство требует обслуживания со стороны МП. Если прерывания разрешены, МП завершает текущую команду и выполняет два цикла шины подтверждения прерывания. В результате их МП вводит по линиям AD7—AD0 байт типа прерывания, а затем осуществляет косвенный переход к соответствующей процедуре прерывания. Когда прерывания запрещены, МП игнорирует сигнал INT.

Пять входных сигналов предназначены для управления работой МП.

RDY — готовность. Сигнал готовности позволяет приостановить действия МП (RDY=0) на произвольное число тактов синхронизации. Обычно он используется в интерфейсе памяти, быстродействия которой недостаточно для работы синхронно с МП.

TEST — проверка. Этот входной сигнал применяется только вместе с командой ожидания WAIT. Выполняя ее, МП проверяет уровень сигнала TEST и либо переходит к следующей по порядку команде (TEST=0), либо зацикливается и периодически проверяет уровень сигнала на входе TEST (TEST==1). Таким образом, вход и команда WAIT позволяют синхронизировать действия МП с внешним событием. Этот механизм применяется для синхронизации работы МП и арифметического сопроцессора.

CLR — сброс или начальная установка. При активном уровне сигнала CLR действия МП прекращаются и он переходит в известное начальное состояние.

CLC — синхронизация или тактирование. На этот вход подаются периодические прямоугольные сигналы с коэффициентом заполнения 1/3, частотой 2—5 МГц, предназначенные для синхронизации всех действий МП.

MN/MX — минимальный/максимальный. Уровень этого сигнала определяет конфигурацию МП.

При *минимальном режиме* работы:

1) линии **QS0, QS1** выполняют функции строба адреса **EA** и подтверждения прерывания **INTA**;

2) линии **ST0-ST2** используются в качестве разрешения обмена данными **ED**, ввода/вывода данных **ID/OD**, работы с памятью/внешними устройствами **IO/M** соответственно;

3) вместо сигнала **LOCK** выполняется сигнал записи **WR**;

4) вместо сигналов **RQ/E0, RQ/E1** используются сигналы запроса прямого доступа (захвата шины) **HOLD** и разрешения прямого доступа **HLDA**.

1.3. ПРОГРАММНАЯ МОДЕЛЬ МИКРОПРОЦЕССОРА

В программной (регистровой) модели МП, показанной на рис. 1.3, фигурируют регистры, которые доступны программисту на уровне языка ассемблера. Модель показывает те ресурсы МП, которыми программист может распоряжаться при разработке программ.

В группу 16-битных регистров общего назначения (РОН), называемых также регистрами данных, входят регистры AX, BX, CX и DX. Их особенность заключается в том, что допускается индивидуально указывать их старшие (H) и младшие (L) байты. Двойственный характер РОН обеспечивает простые манипуляции байтами и словами. Остальные регистры МП можно использовать только как 16-битные.

Регистры AX-DX находятся .в полном распоряжении программиста и единообразно участвуют в арифметических и логических операциях. Однако имеются команды, которые неявно специализируют их на определенные функции, что отражено в названиях регистров.

Группа указательных и индексных регистров представлена регистрами SP, BP, SI и DI. Они предназначены в основном для хранения относительных (внутрисегментных) адресов, обеспечивают косвенную адресацию памяти и участвуют в вычислениях эффективного адреса.

Гибкость таких вычислений достигается тем, что эти регистры могут участвовать в арифметических и логических операциях так же, как и РОН. Поэтому регистры этой группы и регистры AX-DX часто называют общими регистрами. Указатель стека SP адресует текущую вершину аппаратного стека в памяти. С помощью указателя базы BP организуется простой доступ к любым данным, находящимся в стеке. Наиболее часто регистр BP привлекается для адресации параметров, передаваемых подпрограммам. Индексные регистры SI и DI применяются для адресации цепочек.

Наличие в программной модели четырех специальных сегментных регистров объясняется способом адресации памяти. Хотя МП выдает 20-битный физический адрес памяти, он оперирует двумя 16-битными значениями, называемыми логическим адресом: базовый (начальный) адрес сегмента и внутрисегментный адрес или смешение. Эти компоненты логического адреса называются также сегментным адресом и относительным адресом, а весь логический адрес сегмент: смещение называется еще сегментированным адресом. Схема преобразования адресов в составе шинного интерфейса превращает пару сегмент: смешение в 20-битный физический адрес.

Пространство памяти 1М байт разделяется на логические сегменты с максимальной емкостью 64К байт. Реальная емкость сегмента может варьироваться от 16 до максимальной. При выполнении программы МП может обращаться к четырем сегментам, базовые адреса которых находятся в сегментных регистрах CS (кода), DS (данных), SS (стека) и ES (дополнительных данных).

Регистр CS указывает на начало текущего сегмента кода, т.е. программы, откуда выбираются команды. Регистр SS адресует начало текущего сегмента стека, в котором реализуются все стековые операции. Регистр DS определяет текущий сегмент данных, содержащий программные переменные. Наконец, регистр ES указывает дополнительный (Extra — «лишний») сегмент, который обычно применяется для хранения данных.

В большинстве команд фигурирует только внутрисегментное смещение, определяемое в соответствии с указанным режимом адресации. Физический адрес формируется путем суммирования смещения с умноженным на 16 содержимым одного из сегментных регистров (подробнее в параграфе 1.4).

Указатель команды IP осуществляет адресацию внутри текущего сегмента кода, т.е. функционирует аналогично программному счетчику PC (или счетчику команд) большинства других МП. Шинный интерфейс модифицирует его так, что он указывает на ту команду (и даже. Часть команды), которую шинный интерфейс будет выбирать при очередном обращении к памяти. Из-за опережающей выборки команд содержимое IP не совпадает со смещением той следующей команды, которую будет выполнять операционное устройство. Поэтому при сохранении содержимого IP в стеке при вызовах подпрограмм и прерываний оно автоматически корректируется так, чтобы адресовать правильную команду, которая будет выполняться при восстановлении IP. Эта механика “прозрачна“ для программиста, и он может считать IP обычным программным счетчиком.

Формат регистра флажков приведен на рис. 1.4. Шесть его арифметических флажков

фиксируют определенные свойства или признаки результата арифметической или логической операции. Команды МП воздействуют на эти флажки по-разному, но в общем они отражают следующие особенности результата.

1. *Флажок переноса* **CF** фиксирует значение бита переноса (заема), возникающего при сложении (вычитании) байт или слов, а также значение выдвигаемого бита во всех операциях сдвигов. Кроме того, он отражает особенность результата операций сравнения и умножения.

2. *Флажок паритета* **PF** (или четности) показывает наличие четного (0) или нечетного (1) числа единиц в младшем байте результата.

3. *Флажок вспомогательного переноса* **AF** аналогичен флажку CF, но фиксирует перенос (заем) из младшей тетрады. Этот флажок необходим в операциях десятичной арифметики.

4. *Флажок нуля* **ZF** сигнализирует о получении нулевого результата операции.

5. *Флажок знака* **SF** повторяет значение старшего бита результата, который в дополнительном коде соответствует знаку числа.

6. *Флажок переполнения* **OF** отмечает потерю старшего бита результата операции сложения или вычитания над знаковыми числами. Он же показывает изменение старшего (знакового) бита в арифметических сдвигах влево.

Три оставшихся флажка предназначены для управления некоторыми действиями МП. Программист может одной или несколькими командами задать состояние любого из этих флажков.

1. *Флажок направления* **DF** определяет сканирование цепочек от меньших адресов к большим (DF = 0) или наоборот (DF==1). Такие действия называются еще прохождением цепочек «слева направо» и «справа налево».

2. *Флажок прерывания* **IF** задает реакцию МП на запрос внешнего прерывания по входу INT. Если IF=0, запрос прерывания игнорируется (прерывания запрещены или замаскированы), а если IF==1, МП распознает и соответственно реагирует на запрос прерывания.

3. *Флажок прослеживания (трассировки)* **TF** при установке в 1 переводит МП в одношаговый или командный режим работы, который применяется для отладки программ. В этом режиме МП автоматически генерирует внутреннее прерывание после выполнения каждой команды.

1.4. ОРГАНИЗАЦИЯ И РЕЖИМЫ АДРЕСАЦИИ ПАМЯТИ

Для программы пространство памяти 1М байт выглядит как группа сегментов (блоков или областей), определяемых самой программой. Сегмент, как уже отмечалось, есть логическая единица памяти размером 64 К байт. Каждый сегмент идентифицируется начальным адресом (базой), являющимся адресом его первого байта. Единственное требование к размещению сегментов заключается в том, что они должны начинаться на 16-границах памяти. Следовательно, младшие 4 бита базового адреса сегмента должны содержать нули. Других ограничений на размещение сегментов в памяти нет.

В сегментных регистрах CS, DS, SS и ES находятся базовые адреса четырех текущих сегментов, причем младшие нулевые биты адресов не хранятся, а подразумеваются. Если, например, регистр DS содержит 2345, то базовый адрес сегмента данных равен 23 450 (здесь и далее адреса приводятся исключительно в 16-ричной системе счисления). На рис. 1.5,а показана максимальная память, доступная при фиксированном содержимом сегментных регистров (всего адресуется 256 К байт), а рис. 1.5,б более «умеренный» случай, когда размер

сегмента кода составляет 16К байт, сегмента данных 4К байт и сегмента стека 256 байт.

Ранее было сказано о том, что внутри МП любая ячейка памяти идентифицируется логическим адресом, состоящим из 16-битных базового адреса сегмента и внутрисегментного смещения, показывающего расстояние в байтах от начала сегмента до конкретной ячейки.

Когда шинный интерфейс обращается к памяти, он образует из логического адреса сегмент: смещение физический адрес.

Рис. 1.5. Адресуемая память при различном содержимом сегментных регистров

Для этого база сегмента сдвигается влево на 4 бита (т. е. умножается на 16) и суммируется со смещением (рис. 1.6). Шинный интерфейс получает компоненты логического адреса из различных источников в зависимости от типа текущего обращения к памяти, что показано в табл. 1.1.

Таблица 1.1. Источники компонентов логического адреса

Тип обращения к памяти	умолча-	Смещение	Вариант нвю)
Выборка команды	CS	IP	Нет
Стековая операция	SS	SP	»
Обращение к переменной	DS	EA	CS, SS, ES
Цепочка — источник	DS	SI	CS, SS, ES
Цепочка — получатель.	ES	DI	Нет
BP как базовый регистр	SS	EA	CS, DS, ES

Команды всегда выбираются из текущего сегмента кода — логический адрес находится в регистрах CS : IP. Стековые операции всегда обращаются к текущему сегменту стека — привлекаются регистры SS : SP. Обычно переменные находятся в текущем сегменте данных, определяемом регистром DS, но программист может заставить МП обратиться к переменной в сегменте, базовый адрес которого находится не в регистрах DS, а в каком-то другом сегментном регистре. Такая возможность показана в столбце «Вариант» табл. 1.1. Явная спецификация сегмента достигается с помощью так называемого префикса замены сегмента. Байт префикса предшествует команде и сообщает шинному интерфейсу, какой сегментный регистр использовать в следующей за ним команде для формирования физического адреса памяти. В ассемблерных программах необходимость замены сегмента указывается оператором замены сегмента. Например, команда MOV AX, [DI] будет обращаться к текущему сегменту данных, т. е. через регистр DS, а команда MOV AX, ES:[DI] —к текущему дополнительному сегменту.

Смещение переменной вычисляет операционное устройство в соответствии с режимом адресации в команде. Результат вычисления называется эффективным адресом EA операнда. Режим адресации должен идентифицировать операнд(ы) команды, в частности указывать способ формирования EA. Эффективный адрес является либо адресом данных (в командах манипуляций данными), либо адресом перехода (в командах передачи управления).

Команды МП адресуют максимум два операнда. Первым операндом (в порядке записи команды на языке ассемблера) является содержимое регистра или ячейки памяти, а вторым — содержимое регистра или непосредственный операнд. Впрочем, нумерация «первый» и «второй» довольно условна.

Общий формат двухоперандной команды приведен на рис. 1.7,а, причем штриховые

линии показывают необязательные байты. Первый байт содержит код операции (КОП) и два однобитных поля: *d* и *w.* Поле d определяет направление передачи: если *d=1,* то направление «в», а если d=0 — направление «из». Само направление относится ко второму операнду: регистру, указанному в поле *reg* второго байта. Поле *w* показывает тип операнда: байт *(w=0)* или слово *(w=1).*

Второй байт команды называется постбайтом режима адресации, как видно, он сострит из трех полей. Поле *reg* (регистр) показывает один из регистров МП (табл. 1.2). Поле *mod* (режим) определяет интерпретацию поля r*/m* (регистр/память) при нахождении первого операнда. Если *mod=11,* операнд находится в регистре и поле *r/m* показывает конкретный регистр, а в остальных случаях адресуется, память, при этом поле *mod* показывает, чему равно смещение *disp,* содержащееся в команде как константа.

Примечание. *D8=dispL* — один байт смещения; *Dl6=dispL, dispH — д*ва байта смещения.

В случае адресации памяти (mod=\11) поле *r/m о*пределяет способ формирования ЕА в соответствии с табл. 1.2.

В командах с непосредственным операндом (рис. 1.7,6) второй операнд адресовать не нужно, поэтому поле *reg* отводится под код операции. Кроме того, здесь не нужен бит *d,* так как результат можно поместить только на место первого операнда. Однако необходимо определить тип непосредственного операнда, для чего служат поля s и *w.* .

Наконец, на рис. 1.7, в показан формат однооперандной команды, который не имеет ничего нового.

Рассмотрим стандартные режимы адресации МП с учетом приведенных способов формирования ЕА.

Регистровая адресация. Операнд или операнды находятся во внутренних регистрах МП. Команды с таким режимом адресации оказываются наиболее короткими и быстрыми.

Непосредственная адресация. Микропроцессор может оперировать непосредственными операндами (константами) длиной в байт или слово и содержимым регистров или ячеек памяти. Однако команды непосредственной загрузки сегментных регистров и включения .константы в стек отсутствуют.

Абсолютная адресация. В абсолютной адресации ЕА берется из поля *disp* в команде. Этот режим применяется для обращения к простым переменным (скалярам).

Косвенная регистровая адресация. В этом режиме ЕА находится в одном из регистров BP, BX, SI и DI, что указывается путем заключения имени регистра в квадратные скобки.

Базовая адресация. При указании этого режима ЕА равен сумме значения *disp,* находящегося в команде, и содержимого регистра BX или BP. Напомним, что при задании регистра BP шинный интерфейс обращается к операнду в текущем сегменте стека, что упрощает доступ к параметрам подпрограммы, передаваемым в стеке.

Основное применение базовой адресации связано с обработкой структур данных, когда номер элемента структуры известен при ассемблировании программы, а начальный адрес структуры должен определяться при выполнении программы.

Обычно ассемблеры допускают задание базовой адресации в нескольких формах. Например, следующие три команды производят одно и то же действие:

MOV AX,[BP+4]
MOV AX,4[BP]
MOV AX[BP]+4

Индексная адресация. В режиме индексной адресации ЕА равен сумме значения *disp* и содержимого регистра SI или DI, Обычно *disp* определяет известный при ассемблировании

начальный адрес одномерного массива, а индексный регистр показывает нужный элемент. Изменяя содержимое индексного регистра, можно обращаться к различным элементам массива.

Базовая индексная адресация. В этом режиме EA равен сумме содержимого базового регистра BP или BX, индексного регистра SI или DI и необязательного значения *disp.* Гибкость такой адресации объясняется тем, что два компонента адреса можно определять и варьировать при выполнении программы. С помощью базовой индексной адресации обеспечивается удобный доступ к элементам массива, находящегося в стеке, а также обращение к двумерному массиву.

Относительная адресация. В режиме относительной адресации EA равен сумме значения *disp* и текущего содержимого указателя команды IP. При этом полагается, что в IP находится адрес байта, следующего за командой с таким режимом адресации. В МП К1810ВМ86 относительная адресация применяется только в командах передачи управления. Смещение *disp* имеет длину 8/16 бит, представлено в дополнительном коде и имеет диапазон от —128 до +127 и от —32768 до +32 767 соответственно. В программах указывается не значение смещения, а метка той команды, которой передается управление. Значение смещения вычисляет программа-ассемблер.

Адресация цепочек. При обработке цепочек предполагается, что регистр SI адресует в сегменте данных первый или последний байт (слово) цепочки-источника, а регистр DI — в дополнительном сегменте адресует первый или последний байт (слово) цепочки-получателя. В повторяющихся операциях МП автоматически корректирует SI и DI по мере продвижения к другим элементам цепочек. Специальные цепочечные команды рассчитаны на такую неявную адресацию цепочек.

Адресация портов ввода-вывода. Для обращений к портам в пространстве ввода-вывода применяются два режима адресации. При прямой адресации фиксированный адрес порта находится непосредственно в команде, как ее второй байт, что обеспечивает. доступ к портам 0—255. При косвенной адресации номер порта находится в регистре DX и имеет диапазон от 0 до 65 535.

1.5. СИСТЕМА КОМАНД

В систему команд МП К1810ВМ86 входят 113 базовых команд, многие из которых допускают разнообразные режимы адресации. При рассмотрении системы команд принято группировать команды с примерно одинаковым функциональным назначением.

1.5.1. КОМАНДЫ ПЕРЕДАЧ ДАННЫХ

Микропроцессор имеет обширную группу команд, предназначенных для пересылок данных между регистрами и между регистрами и памятью. Команды этой группы, за исключением команд POPF и SAHF (см. далее), не воздействуют на флажки.

Команда **MOV.** Наиболее гибкая из всех команд передач данных MOV имеет следующее содержательное представление:

MOV dst, src dst <-(src)

Содержимое источника *src* копируется в получатель *dst.*

Чтобы показать разнообразие машинных команд микропроцессора, на рис. 1.8 представлены все форматы команд MOV, а далее форматы команд не приводятся. Приняты следующие сокращения: *reg* — регистр, *mem* — память, *data* — данные, *disp* — смещение (в

команде), *ac* — аккумулятор (AX или AL), *sreg* — сегментный регистр. Как видно из рисунка, с помощью команды MOV можно осуществить следующие передачи байта или слова: из регистра в регистр, из регистра в память и наоборот, непосредственного операнда в регистр или память. Приведенные также более короткие команды, в которых фигурируют аккумулятор AX или AL.

Дадим несколько примеров ассемблерных команд MOV с различными источниками и получателями:

```
MOV  DX,CX          ;Слово, из регистра в регистр
MOV  BX,ES:[SI]     ;Слово, из памяти в регистр
MOV TEMP[DI],AL  ;Байт, из регистра в память
MOV  [BX]SI30H     ; Непосредственный байт в память
```

Конечно, при всей своей универсальности команда MOV имеет и определенные ограничения. Во-первых, как уже отмечалось, невозможно одной командой передать содержимое какой-либо ячейки памяти в другую ячейку. При необходимости такую передачу следует производить через внутренний регистр МП. Рекомендуется максимально привлекать для этого аккумулятор AX или AL, так как команда получается короче на один байт. Во-вторых, команда MOV не может загрузить непосредственное значение в сегментный регистр. Поэтому для инициализации сегментных регистров приходится применять две команды и промежуточный общий регистр, которым чаще всего выступает AX, например:

```
MOV  AX, DATA_S   ;Инициализировать   регистр DS
MOV DS,AX              ;через  аккумулятор AX
```

Наконец, командой MOV невозможно передать содержимое одного сегментного регистра в другой. Такая передача осуществляется через промежуточный общий регистр:

```
MOV AX, DS    ;Передача из регистра DS
MOV  ES,AX    ;в регистр ES
```

Команда **XCHG.** Команда обмена

XCHG dst, src (dst)^(src)

позволяет обменять содержимое двух общих регистров, а также любого общего регистра и ячейки памяти. В обмене могут участвовать байты и слова. Однако в этой команде нельзя указывать сегментные регистры. Отметим, что команда XCHG AX, AX используется как холостая команда NOP.

Примеры ассемблерной записи команды XCHG:

```
XCHG AL,DH    ;Обмен байт в регистрах
XCHG AX,[DI]  ;Обмен слов регистр—память
```

Команда XLAT. Однобайтная команда преобразования XLAT с пустым полем операнда

```
XLAT AL          <-((BX) + (AL))
```

заменяет содержимое AL на байт из 256-байтной таблицы, начальный адрес которой находится в регистре BX, при этом содержимое AL действует как индекс таблицы (рис. 1.9). Команда XLAT обычно .применяется для быстрого преобразования символов из одного символьного кода в другой, так как она выполняется всего за 11 тактов синхронизации.

Отметим, что иногда для лучшего понимания программы в поле операнда команды XLAT указывается “бесполезное” для нее символическое имя начального адресы таблицы преобразования. Кроме того, некоторые ассемблеры допускают мнемонику XLATB, подчеркивающую тот факт, что команда XLAT преобразует байты.

Команды LEA, LDS и LES. При выполнении этих команд в указанный(е) регистр(ы) передаются не данные, а адреса, поэтому их основное применение связано с инициализацией адресных регистров.

При выполнении команды загрузки эффективного адреса LEA *reg, mem* образуется эффективный адрес EA памяти и именно его значение, а не адресуемое им слово памяти, загружается в общий регистр *reg*. Передача адреса требуется, например, для инициализации регистра BX до его использования в команде XLAT.

Команды загрузки полных указателей LDS *reg, mem* и LES *reg, mem* выполняют более сложные действия: вычисляется EA, и производится обращение к памяти, а затем считываемое слово загружается в общий регистр *reg* и следующее слово из памяти передается в регистр DS (команда LDS) или в регистр ES (команда LES). Обычно в команде LDS как *reg* указывается индексный регистр SI, а в команде LES — индексный регистр DI, что согласуется с использованием регистров в цепочечных командах. Команды LDS и LES должны адресовать в памяти двойное слово. Примеры команд:

```
LEA BX,TABLE ;Адрес таблицы в BX
LDS S1,[DI]  ;Инициализировать реги-
LES DI,[DI+4] ;стры для пересылки цепочки
```

Команды SAHF и LAHF. Однобайтные безоперандные команды LAHF и SAHF введены только для упрощения программной совместимости микропроцессоров К1810ВМ86 и КР580ИК80. В 8-битном МП КР580ИК80 регистр флажков, полностью соответствующий младшему байту регистра флажков МП К1810ВМ86, вместе с аккумулятором А образует слово состояния процессора PSW. Команда LAHF передает младший байт регистра флажков в регистр AH, образуя в аккумуляторе AX аналог PSW (сами флажки при передаче, конечно, не изменяются). Команда SAHF осуществляет обратную передачу — содержимое регистра AH передается в младший байт регистра флажков.

Стековые команды. Вершина TOS *(Top of Stack)* аппаратного стека в сегменте стека оперативной памяти адресуется регистрами SS и SP. Все стековые команды оперируют только словами и сопровождаются автоматической модификацией SP: при включении *(push)* в стек производится декремент, а при извлечении *(pop)* из стека — инкремент SP.

Команды PUSH включения в стек

PUSH *src* SP <- (SP) — 2, TOS <- *(src)*

осуществляет запись в стек содержимого общего регистра, сегментного регистра или ячейки памяти, указываемой в любом режиме адресации. Команда POP извлечения из стека

POP *dst* *dst<-(TOS),* SP<-(SP)+2

выполняет противоположные действия.

До использования стековых команд необходимо правильно инициализировать регистры SS и SP (помня о том, что стек в памяти «растет вверх», в область меньших адресов), а также тщательно следить за тем, чтобы в программе каждой команде PUSH соответствовала команда POP. В МП К1810ВМ86 нет аппаратных средств контроля переполнения и антипереполнения («опустошения») стека, поэтому ответственность за правильное использование стека возлагается на программиста. При этом следует учитывать, что некоторые команды (например CALL, RET и INT) автоматически используют стек.

Однобайтные безоперандные команды PUSHF и POPF манипулируют регистром флажков.

Примеры стековых команд на языке ассемблера:

```
PUSH SI    ;Включение в стек общего регистра
PUSH ES    ;Включение в стек сегментного регистра
PUSH [BX][SI] ;Включение в стек слова из памяти
PUSHF      ;Включение в стек регистра флажков
POPF       ;Извлечение из стека в регистр флажков
```

```
POP ALPHA[SI] ;Извлечение из стека в память
POP DS     ;Извлечение из стека в сегментный регистр
POP AX     ;Извлечение из общий регистр
```

С помощью команд PUSH и POP можно передать содержимое одного сегментного регистра, в другой без привлечения промежуточного общего регистра:

```
PUSH DS    ;Эти команды передают
POP ES     ;содержимое DS в ES
```

Конечно, из-за обращений к памяти эти команды выполняются дольше эквивалентных команд:

```
MOV AX,DS  ;Эти команды также передают
MOV ES.AX  ;содержимое DS в ES
```

Команды ввода-вывода. Команды ввода IN и вывода OUT могут передавать байты и слова. Получателем при вводе и источником при выводе могут быть только аккумуляторы AX или AL. Команды

IN *ac, port*
OUT *port, ac*

содержат байт прямого адреса входного или выходного порта (так называемый статический ввод-вывод), а в командах

IN *ac*, DX
OUT *DX, ac*

16-битный адрес порта берется из регистра DX (так называемый динамический ввод-вывод). Косвенная адресация через регистр DX удобна в тех случаях, когда адрес нужного порта определяется при выполнении программы.

Примеры команд ввода-вывода:

```
IN AX,80H   ;Ввод слова (статический)
IN AL,DX    ;Ввод байта (динамический)
OUT 04H,AL  ;Вывод байта (статический)
OUT DX,AX   ;Вывод слова (динамический)
```

1.5.2. **АРИФМЕТИЧЕСКИЕ КОМАНДЫ**

Микропроцессор К1810ВМ86 имеет достаточно обширный набор арифметических команд, что позволяет применять его для сложной обработки данных. Арифметические операции выполняются над целыми числами в четырех форматах: беззнаковые двоичные, знаковые двоичные, упакованные десятичные и неупакованные десятичные (рис. 1.10). В данной книге операции с десятичными

Рис. 1.10. Форматы численных данных

числами не рассматриваются. Поэтому отметим только, что они реализуются в два этапа: сначала исполняется соответствующая операция с двоичными числами, а полученный ею результат корректируется специальной командой (как исключение, при делении коррекция делимого осуществляется первой). Беззнаковые двоичные числа представляются байтами и словами, имеющими диапазоны от 0 до 255 и от 0 до 65535 соответственно. Знаковые двоичные числа в форматах байта и слова изображаются в обычном дополнительном коде,

имея диапазоны от —128 до +127 для байта и от —32 767 до +32 767 для слова. Сложение и вычитание знаковых и беззнаковых чисел выполняют одни и те же команды (в этом заключается одно из достоинств дополнительного кода), а для умножения и деления предусмотрены отдельные команды.

В арифметических операциях особенности получающихся результатов фиксируются в шести арифметических флажках, состояния которых (за исключением флажка AF вспомогательного переноса) можно проверить командами условных переходов.

Команды сложения. Микропроцессор имеет команду ADD собственно сложения и команду ADC сложения с переносом:

ADD *dst, src* *dst<-(dst)+(src)*
ADC *dst, src* *dst<-(dst}+(src)*+(CF)

В качестве *dst* и src можно указывать общие регистры и ячейки памяти с любым режимом адресации, а в качестве src — еще и непосредственные операнды.

К командам сложения обычно относят однооперандную команду инкремента (увеличения на l):

INC *dst* *dst<-(dst)*+1

В этой команде операнд *dst,* которым может быть общий регистр или ячейка памяти, считается беззнаковым двоичным числом, и при ее выполнении состояние флажка CF не изменяется. Команда INC часто применяется для инкремента счетчика цикла и продвижения указателей (т.е. регистров адреса) при обработке регулярных структур данных.

Команды вычитания. Команды вычитания

SUB *dst, src* *dst<-(dst)—(src)*
SBB *dst, src* *dst<- (dst) — (src)* — (CF)
DEC *dst* *dst<-(dst)—1*

отличаются от соответствующих команд сложения только выполняемой операцией. Следует подчеркнуть, что флажки CF и AF в операциях вычитания становятся флажками заема и устанавливаются в 1, когда вычитаемое больше уменьшаемого.

Команды ADC и SBB (вычитание с заемом) предназначены для обработки операндов, длина которых составляет несколько байт или слов (так называемые числа многократной точности),

К командам вычитания традиционно относят команду NEG изменения знака, так как ее действие эквивалентно вычитанию операнда из нуля:

NEG dst dst <-*0*— (dst)

Если операнд равен нулю, его значение не изменяется. Попытка изменить знак максимального (по модулю) отрицательного числа, т.е. числа вида 100...00, не вызывает изменения операнда, но при этом устанавливается флажок переполнения OF. При выполнении команды NEG флажок CF устанавливается в 1, кроме случая, когда операнд равен нулю, — тогда CF=0.

Необходимо отчетливо различать смысл флажков переполнения OF и переноса (заема) CF. Состояние CF==1 показывает, что при сложении (вычитании) возник перенос (заем) из старшего бита. При обработке чисел длиной в несколько байт или слов перенос (заем) следует учитывать в операции со следующим байтом или словом. Однако состояние CF=1 можно интерпретировать и как переполнение в операциях сложения и вычитания беззнаковых чисел. Оно показывает, что при сложении или вычитании произошло переполнение, т.е. результат выходит за диапазон представимых чисел и, следовательно, неверен. Покажем различия между флажками CF и OF на нескольких примерах.

Операция	Беззнаковые числа	Знаковые числа	Флажки
0 1 1 1 1 0 0 1	121	(+121)	CF=1
1 1 0 1 0 1 1 0	214	(-42)	OF=0
0 1 0 0 1 1 1 1	79(?)	+79	
0 1 1 1 1 0 0 1	121	(+121)	CF=0
0 1 0 1 0 1 1 0	86	(+86)	OF=1
1 1 0 0 1 1 1 1	207	-49(?)	
1 0 0 0 0 0 0 0	128	(-128)	CF=1
1 0 0 0 0 0 0 0	128	(-128)	OF=1
0 1 0 0 1 1 1 1	0(?)	0(?)	

Примеры ассемблерных команд сложения и вычитания:

```
ADD   AX, CX        ;Регистр — регистр, слово
ADD   ALPHA, DI     ;Память — регистр, слово
ADD   CL,3          ;Константа к регистру, байт
INC   WORD PTR      ;Инкремент слова в памяти
GAMMA[DI]
SUB   DX, [BP+2]    ;Регистр — память, слово
DEC   BYTE PTR [BX]      '.Декремент байта в памяти
NEG   WORD PTR [SI]      -.Изменение знака слова в памяти
```

Команда сравнения. Команда CMP сравнения

CMP *dst, src (dst)—(src)*

осуществляет вычитание источника из получателя, но разность нигде не запоминается, т. е. команда реализует так называемое неразрушающее сравнение операндов. О результате сравнения сигнализируют арифметические флажки (табл. 1.3).

Команды умножения. Микропроцессор К1810ВМ86 имеет две команды умножения:

```
MUL   src
IMUL  src          ;ext:ac<-(ac) x (src)
```

предназначенные для умножения беззнаковых и знаковых операндов соответственно. Обе они выполняют умножение адресуемого операнда *src* (им может быть общий регистр или ячейка памяти и не может быть непосредственный операнд) и аккумулятора *ac.* В операции над байтами аккумулятором *ac* служит регистр AL, а произведение двойной длины образуется в регистрах AH и AL, причем регистр AH называется расширением *ext* аккумулятора AL. В операции над словами аккумулятором *ac* является регистр AX, а произведение получается в регистрах DX и AX. Здесь регистр DX выступает расширением аккумулятора AX.

Таблица 1.3. Результаты операции сравнения

Флажки-			
Условие	*i*	*1*	*i*
OF | SF | ZP (CP			

Беззнаковые операнды:
(src)<(dsf) X X O O
(src)==(dst) X X . 1 0
(src)>(dsl) *X* X 0 1
Знаковые операнды:
(src)<(dst) • 0/1 O O X
(src)^(dst) 0 0 1 X
(src)>(dst) 0/1 1 0 X

Команды умножения своеобразно воздействуют на флажки, позволяя оценить произведение. Если после выполнения команды MUL старшая половина произведения не является нулевой, а после команды IMUL она не является расширением знака младшей половины, флажки CF и OF устанавливаются в 1, а в противном случае CF, OF==0. Состояния всех остальных флажков после выполнения команды MUL и IMUL не определены.

Примеры записи команд умножения на языке ассемблера:

```
MUL   BL              ;Байты, без знака
MUL   WORD PTR [DI]   ;Слова, без знака
IMUL   CX             ;Слова, со знаком
IMUL   BYTE PTR [BX] [SI] ;Байты, со знаком
MOV   DX, 100         ;Эти две команды
MUL   DX              ;умножают AX на 100
```

Команды деления. В двух командах деления операндами являются беззнаковые (DIV) и знаковые (IDIV) двоичные числа:

DIV *src } ac<- quot((ext:ac)/(src))*
IDIV *src } ext<-rem((ext:ac)/(src))*

Здесь делимым двойной длины служит содержимое аккумулятора *ac* (AX или AL) и его расширения *ext* (DX или AH), делителем — адресуемый операнд *src* (общий регистр или ячейка памяти, но не константа), частное *quot* образуется в аккумуляторе *ac*, а остаток *rem* — в его расширении. Дробное частное усекается до
целого, а состояния всех арифметических флажков не определены. Если длина частного превышает длину аккумулятора (например, при делении беззнаковых байт частное больше 255). МП генерирует внутреннее прерывание типа 0 и автоматически переходит к соответствующей процедуре обработки прерывания. При этом содержимое регистров *ext* и *ac* непредсказуемо.

Примеры ассемблерных команд деления:

```
DIV  BH              ;Байты, без знака
DIV   WORD PTR [BP]   ;Слова, без знака
DIV   BX              ;Слова, со знаком
IDIV  BYTE PTR [DI]    ;Байты, со знаком
MOV   CX, 1000         ;Эти две команды делят
DIV   CX               ;DX:AX на 1000
```

К командам деления относятся также две команды преобразования, позволяющие удвоить длину знакового операнда с сохранением его численного значения. Команда CBW преобразования байта в слово копирует в регистр AH знак числа, находящегося в регистре AL, а команда CWD преобразования слова в двойное слово копирует в регистр DX знак числа, находящегося в регистре AX. Обе команды не влияют на состояние флажков. Наличие этих команд упрощает операции над данными смешанных типов, например:

```
CWD             ;Деление слова из регистра AX
IDIV  BX        ;на слово из регистра BX
CBW             ;Суммирование байта из AL
ADD   AX, BX    ;и слова из BX
```

1.5.3. ЛОГИЧЕСКИЕ КОМАНДЫ И КОМАНДЫ СДВИГОВ

Логические команды. Поразрядные логические операции МП К1810ВМ86 представлены булевыми операторами NOT (инверсия), AND (конъюнкция), OR (дизъюнкция), XOR (сложение по *mod* 2) и командой TEST проверки, которая выполняет конъюнкцию операндов, но не изменяет ни один из них:

```
AND  dst,src     dst<-(dst)/\(src)
OR   dst,src     dst<-(dst)\/(src)
XOR  dst,src     dst<-(dst)^(src)
TEST dst,src     (dst)/\(src)
NOT  dst         dst<-(dst)
```

Бинарные операции AND, OR, XOR и TEST воздействуют на арифметические флажки следующим образом:

флажки CF и OF переводятся в нулевое состояние;

состояния флажков SF, ZF и PF зависят от результата;

состояние флажка AF не определено. Команда NOT не изменяет состояний флажков. В качестве *dst* и *src* можно указывать общие регистры и ячейки памяти с любым режимом адресации, а в качестве src — еще и непосредственные данные (за очевидным исключением команды NOT). Операнды могут быть байтами и словами.

Команды AND и OR применяются в основном для установки в 0 и 1 тех бит операнда *dst,* которые определяются другим операндом src, называемым маской. С помощью команды XOR можно инвертировать выбираемые маской биты операнда *dst,* сравнивать операнды на абсолютное равенство (при равенстве результат равен нулю и ZF==1) и переводить регистр в нулевое состояние.

Примеры ассемблерной записи логических команд:

```
AND AX, BX          ; Конъюнкция двух регистров
AND [SI], 1         ;Проверка младшего бита
OR   AX.8000H       ;Установка в 1 старшего бита
XOR  BX, BX         ;Сброс регистра BX
XOR  DI, 1          ;Инвертирование младшего бита
NOT  BYTE PTR [SI];Инвертирование байта в памяти
```

Команды сдвигов. Действия команд сдвигов представлены на рис. 1.11. Поле операнда всех команд имеет вид *mem/reg, count.* Здесь *mem/reg* обычным образом адресует общий регистр или ячейку памяти, а счетчик *count* определяет число сдвигов. Он может быть указан как константа 1 (статический сдвиг на один бит) или как регистр CL (динамический сдвиг, в котором число сдвигов задается содержимым регистра **CL),**

С помощью команд сдвигов осуществляются циклические и обычные сдвиги. В циклических сдвигах выдвигаемый бит помещается на место освобождающегося бита (см. первые четыре команды на рис. 1.11). Команды RCL и RCR называются циклическими-сдвигами через перенос, так как в кольцо сдвига включается флажок CF. «Обычные» сдвиги

подразделяются на логические .(команды SHL и SHR), в которых операнды считываются беззнаковыми числами, и арифметические {команды SAL и SAR), оперирующие знаковыми числами. Для логического сдвига характерно, что в освобождающийся бит помещается 0, а выдвигаемый бит теряется. В арифметическом сдвиге вправо (команда SAR) знаковый бит сохраняется путем его дублирования, а в арифметическом сдвиге влево (команда SAL) он не сохраняется, но изменение его фиксируется установкой флажка OF ==1.

При выполнении команд сдвигов флажки модифицируются следующим образом:

состояние флажка AF всегда не определено;

флажок CF всегда содержит значение последнего выдвинутого бита;

в статических сдвигах флажок OF==1, если знаковый бит изменился;

в динамических сдвигах состояние OF не определено;

циклические сдвиги воздействуют только на флажки OF и CF;

в «обычных» сдвигах флажки SF, ZF и PF фиксируют соответствующие признаки результата.

Приведем примеры ассемблерных команд сдвигов:

```
SHL   DH, 1                 ;Логический  сдвиг байта в регистре
SAR   BYTE PTR [SI], CL     ;Арифметический сдвиг байта в памяти
ROR  WORD PTR [BXj [DI], 1  ;Циклический сдвиг слова в памяти
```

Одно из применений команд сдвигов заключается в быстром (по сравнению с соответствующими явными командами) умножении и делении операндов на степени 2. Следующие команды показывают, как умножить и разделить содержимое регистра BX на 8 с предположении, что регистр CL содержит 3:

```
SHL   BX, CL      ;Умножение беззнакового числа
SAL   BX, CL      ;Умножение знакового числа
SHR   BX, CL      ;Деление беззнакового числа
SAR   BX, CL      ;Деление знакового числа
```

Следующий фрагмент осуществляет умножение содержимого регистра AX на 15:

```
MOV CL, 4          ;Константа сдвига
MOV DX, AX         ;Сохранить содержимое AX
SAL  AX, CL        ;Умножить содержимое AX на 16
SUB  AX, DX        ;корректировать для умножения на 15
```

1.5.4. **КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ**

В разветвляющихся и циклических программах, при организации подпрограмм и обработке прерываний требуется выполнять не следующую по порядку команду, а команду, находящуюся в другой ячейке программной памяти и определяемую адресом перехода. Специальные команды, которые каким-либо образом модифицируют указатели программной памяти (регистры PC и CS), называются командами передачи управления. Они не изменяют состояний флажков (за исключением команды IRET возврата из прерывания).

Сегментная организация памяти определяет две разновидности команд передачи управления. Передача управления в пределах текущего сегмента кода называется внутрисегментной, при этом модифицируется только PC и адрес перехода представлен одним словом. Передача управления вне текущего сегмента кода называется межсегментной; здесь необходимо модифицировать регистры PC и CS и адрес перехода должен быть представлен двумя словами сегмент:смещение. В ассемблерных программах операндом команд передачи управления служит адресное выражение, определяющее ту команду *(target},*

которой передается управление.

Команда безусловного перехода. В МП К1810ВМ86 мнемоника JMP обозначает команды безусловного перехода пяти форматов, которые имеют одно и то же ассемблерное представление: .

JMP *target*

где *target* каким-либо образом определяет адрес перехода. Формат команды ассемблер выбирает автоматически в соответствии с атрибутами операнда.

Двухбайтная команда JMP во втором байте содержит знаковое значение из диапазона — 128—+127. При выполнении команды оно прибавляется к содержимому регистра PC, которое соответствует адресу команды, находящейся после команды JMP. Эта команда удобна для организации коротких программных циклов.

Трехбайтная команда JMP производит такое же действие, как и предыдущая команда, но содержит 16-битное смещение. Оно по-прежнему считается знаковым целым, поэтому область перехода составляет от —32 768 до +32767 байт. Обе эти команды реализуют внутрисегментный переход.

Еще одна команда JMP осуществляет внутрисегментный косвенный переход. Здесь адресом перехода, загружаемым в PC, служит содержимое общего регистра или слова из памяти, определяемого в любом режиме адресации. Например, команда JMP BX загружает в PC содержимое регистра BX, а команда JMP WORD PTR. [SI] загружает в PC слово из памяти, находящееся в сегменте данных и имеющее смещение в регистре SI.

Последние две команды JMP реализуют прямой и косвенный межсегментные переходы. Первая .из них содержит два слова прямого адреса перехода, которые загружаются в регистры PC и CS. В команде косвенного межсегментного перехода допускается указывать только память. При ее выполнении слово из адресуемой ячейки загружается в регистр PC, а следующее слово — в регистр CS.

Примеры команд безусловного перехода:

```
JMP   SHORT LABEL          ;Короткий переход
JMP   NEAR PTR LABEL       ;Ввнутрисегментный переход
JMP   DWORD PTR ES: [BX][SI];Межсегментный косвенный переход
```

Команды условных переходов. При выполнении любой из 19 двухбайтных команд условных переходов проверяется. некоторое условие, представленное текущими состояниями флажков (а в команде JCXZ — содержимым регистра CX), и в зависимости от удовлетворения условия переход осуществляется или нет. Эти команды позволяют проверить оба состояния всех арифметических флажков, кроме флажка AF, а также ряд комбинаций нескольких флажков. Если условие «истинно», управление передается по адресу перехода путем прибавления к содержимому регистра PC однобайтного знакового смещения, а если условие «ложно», выполняется следующая по порядку команда. Следовательно, все условные переходы являются короткими и диапазон перехода составляет от —128 до +127 байт.

Многие команды условных переходов имеют две мнемоники, подчеркивающие содержательный смысл проверяемого условия. Например, в командах

```
SUB   AX, DX
JZ   ZERO   ;Перейти, если нуль
```

переход к метке ZERO происходит, если команда вычитания образует в регистре AX нулевой результат. Вместе с тем команды

```
CMP AX, DX
JE   ZERO   ;Перейти, если равно
```

выполняют переход к метке ZERO, если в регистрах AX и DX находятся одинаковые числа. Команды JZ и JE осуществляют переход, если ZF==1, но их мнемоники лучше передают содержательный смысл.

Обычно команды условного перехода применяются после команды сравнения и позволяют проверить все отношения между знаковыми и беззнаковыми числами в соответствии с табл. 1.4. ′

^Таблица 1.4. Условные переходы после команды сравнения

_ „ Беззнаковое Знаковое » Перейти, если число число

; Получатель больше источника JA JQ t Получатель равен источнику JE JE Получатель ие равен источнику JNE JNE Получатель меньше источника JB Jb ′• Получатель меньше или равен источнику JBE JLE ; Получатель больше или равен источнику JAE JGE

Термины *больше* (Greater) и *меньше* (Less) относятся к знаковым числам, а термины *выше* (Above) , и *ниже* (Below) — к беззнаковым. Например, число BE_{16} (равное 190 как беззнаковое и —66 как знаковое) оказывается *меньше* и *выше* числа 37_{16} (как беззнаковое S и знаковое оно равно 55).

Команды вызова подпрограмм. В поле операнда команды CALL вызова подпрограммы находится адресное выражение (в простейшем случае — метка), определяющее точку входа подпрограммы, и последнее действие этой команды заключается в загрузке значения адресного выражения в регистр PC (внутрисегментный вызов) или в регистры PC и CS (межсегментный вызов). Однако при переходе к подпрограмме необходимо временно запомнить точку вызова или адрес возврата — им является адрес команды, находящейся после команды CALL. Завершающая подпрограмму команда RET (urn) возврата должна передать управление по запомненному адресу возврата. Удобным «хранилищем» адресов возвратов является стек. Поэтому первое действие команды CALL заключается в том, чтобы включить в стек содержимое регистра PC (внутрисегментный вызов) или содержимое регистров PC и CS (межсегментный вызов) с соответствующей коррекцией указателя стека SP. Команда CALL допускает такие же режимы адресации, как и команда JMP, но короткий вызов отсутствует (обычно подпрограмма удалена более чем на 128 байт). Примеры команд вызова подпрограммы:

```
CALL SUBR              ;Внутрисегментный прямой вызов
CALL BX                ;Внутрисегментный косвенный вызов
CALL NEAR PTR [BX]     ;Внутрисегментный косвенный вызов
CALL FAR PTR SUBR      ;Межсегментный прямой вызов
```

Команды возврата из подпрограммы. Каждая подпрограмма должна иметь минимум одну команду RET возврата, которая передает управление вызывающей программе, привлекая для этого сохраненный в стеке адрес возврата. В соответствии с типами команды CALL имеются команды возврата двух типов: внутрисегментные и межсегментные. Выполнение команды внутрисегментного возврата заключается в том, что слово из вершины стека передается в PC и производится инкремент указателя стека, на 2. Выполнение команды межсегментного возврата несколько сложнее: слово из вершины стека передается в PC; производится инкремент SP на 2; слово из новой вершины стека передается в CS, и производится заключительный инкремент SP на 2. Длина таких команд RET составляет всего 1 байт.

В трехбайтных командах возврата вида RET *n* содержатся 2 байта данных, интерпретируемых как беззнаковое целое число. Дополнительное действие этих команд

заключается в прибавлении к SP числа *n* (после извлечения из стека адреса возврата). Коррекция SP упрощает возврат из тех подпрограмм, параметры которым передаются в стеке, так как продвижение SP вперед эквивалентно удалению параметров из стека (говорят также про «очистку» стека).

Примеры команд возврата из подпрограмм:

```
RET          ;Тип возврата определяется соответствующей
RET  16      ; командой вызова
```

Команды управления циклами. Три двухбайтные команды управления циклами упрощают организацию программных циклов. В них предполагается, что счетчиком цикла служит регистр CX. Второй байт команд представляет собой знаковое число, которое при переходе к началу цикла прибавляется к содержимому регистра PC. В ассемблерных программах поле операнда команд управления циклами занято меткой первой команды цикла.

При выполнении команды LOOP производится декремент регистра CX и, если (CX)=0, второй байт команды прибавляется к PC, т.е. осуществляется переход к началу цикла; в противном случае выполняется следующая по порядку команда, что соответствует выходу из цикла. Таким образом, команда LOOP MORE эффективно заменяет две команды: DEC CX и JNZ MORE.

Довольно часто требуется организовать цикл, условием окончания которого является не просто достижение нуля в регистре CX, а какое-то дополнительное событие. Мнемоники LOOPE и LOOPZ определяют одну и ту же машинную команду, которая производит декремент CX, а затем передает управление в начало цикла, если (CX)=0 и ZF=1. В командах LOOPNE и LOOPNZ дополнительным условием перехода к началу цикла является нулевое состояние флажка ZF.

Следующий фрагмент отыскивает в массиве байт первый ненулевой элемент. Предполагается, что длина массива находится в регистре CX, а начальный адрес — в регистре BX.

```
DEC   BX                ;Подготовить цикл
MORE: INC   BX                  ;Продвинуть указатель
CMP   BYTE PTR [BX], 0;Сравнить элемент с нулем
LOOPE MORE              ;Повторять
JNZ   NOFOUND                   ;Нулевых байт нет;Адрес нулевого байта в BX
```

Команды прерываний. Микропроцессор К1810ВМ86 имеет три команды программных прерываний. Выполнение их напоминает реакцию МП на запросы аппаратных прерываний по входам INT и NMI, но без циклов шины подтверждения прерывания: в стек последовательно включается содержимое регистров флажков, CS и PC, а затем в соответствии с типом прерывания (он встроен в команду)' осуществляется косвенный межсегментный переход через память к нужной процедуре прерывания.

Двухбайтные команды INT *n* имеют во втором байте тип прерывания, который программист задает в поле операнда как беззнаковое число из диапазона от 0 до 255. Если поле операнда пустое, ассемблер образует однобайтную команду прерывания с фиксированным типом 3 — это так называемое прерывание контрольной точки или контрольного останова. Наконец, команда INTO прерывания при переполнении генерирует прерывание с фиксированным типом 4, если только установлен флажок переполнения OF.

Возврат из процедур прерываний осуществляет специальная команда IRET, которая извлекает из стека и возвращает «по назначению» запомненное содержимое регистров PC, CS и флажков.

1.5.5. **ЦЕПОЧЕЧНЫЕ КОМАНДЫ**

Цепочкой (или строкой) называется последовательность байт или слов, находящихся в смежных ячейках памяти. Микропроцессор К1810ВМ86 имеет пять однобайтных команд, оперирующих одним элементом цепочки (эти команды иногда называются элементарными операциями или примитивами). Однако команде может предшествовать префикс повторения (модификатор) REP(eate), который вызывает повторение действия команды над следующими элементами. Благодаря префиксу повторения цепочки обрабатываются значительно быстрее, чем при организации цикла. Повторение рассчитано на максимальную длину цепочек 64К байт и заканчивается по одному или двум условиям.

Аппаратно подразумевается, что цепочка-источник по умолчанию находится в текущем сегменте данных (но допускается замена сегмента) и смещение ее текущего элемента содержится в регистре SI, — отсюда и название этого регистра «индекс источника». Цепочка-получатель всегда находится в дополнительном сегменте данных, а смещение ее текущего элемента берется из регистра DI. Следовательно, явно адресовать цепочки не нужно и ассемблер привлекает поле операнда только для определения типа элементов цепочки — байты или слова. При выполнении команд индексные регистры автоматически модифицируются так, чтобы адресовать следующие элементы цепочек. Флажок направления DF показывает их автоинкремент (DF==0) или автодекремент (DF==l). Инкремент/декремент производится на 1 при обработке цепочек байт и на 2 при обработке цепочек слов.

При указании префикса повторения в каждой элементарной операции осуществляется декремент регистра CX. Когда его содержимое достигает нуля, управление передается следующей по порядку команде.

Префикс повторения. Этот префикс имеет пять мнемонических обозначений, которые определяют всего 2 байта кода префикса. Префикс REP в командах передачи MOVS и запоминания STOS означает «повторять до достижения конца цепочки», т.е. до получения (CX)=0. Префиксы REPE и REPZ в командах сравнения CMPS и сканирования SCAS для инициирования следующего повторения дополнительно требуют ZF==l. Наконец, префиксы REPNE и REPNZ для повторения этих же команд требуют ZF==0.

Команда MOVS. Команда MOVS передачи цепочки

MOVS *dst, src dst<-(src)*

передает байт (слово) из цепочки *src* в цепочку *dst* и соответственно модифицирует регистры SI и DI. Эта команда с префиксом повторения REP осуществляет блоковую передачу память—память. Для реализации передачи необходима следующая подготовка:

установить нужное состояние флажка DF;

загрузить смещение цепочки-источника в регистр SI;

загрузить смещение цепочки-получателя в регистр DI;

загрузить в регистр CX число передаваемых элементов (байт или слов).

После этого выполняется команда MOVS с префиксом REP.

Следующий фрагмент копирует 100 слов из цепочки SOURCE (находящейся в сегменте данных) в цепочку DEST (она находится в дополнительном сегменте данных):

```
DEST  DW 100 DUP (?)     ;Зарезервировать место
...
CLD                      ;Движение вперед
LEA   SI, SOURCE         ;Адрес источника
LEA   DI, ES: DEST       ;Адрес получателя
```

```
        MOV CX, 100              ;Счетчик элементов
REP  MOVS   DEST, SOURCE   ;Пересылать слова
```

Ассемблер узнает о передаче слов по типу переменной DEST, которая определена директивой DW и формирует команду передачи именно слов. Если бы переменная DEST была определена директивой DB, ассемблер сформировал бы команду передачи байт. Таким образом, ассемблер обращается к полю операнда в общей команде MOVS только для определения типа элементов цепочек, а местоположение цепочек задают регистры DS:SI и ES:DI. Поэтому целесообразно предусмотреть в ассемблере мнемоники MOVSB и MOVSW, явно указывающие тип элементов (B—байт, W—слово) и не требующие никакой информации в поле операнда.

Команда CMPS. Команда сравнения цепочек CMP имеет следующее общее представление:

CMPS *dst, src (src)—(dst)*

Она производит вычитание текущих элементов цепочек и по результату вычитания устанавливает все арифметические флажки. Сами операнды не изменяются, а регистры SI и DI продвигаются на следующие элементы цепочек.

Префикс REPE (или REPZ) придает команде смысл «сравнивать до тех пор, пока не достигнут конец цепочек или элементы цепочек будут не равны», а префикс REPNE (или REPNZ) — «сравнивать до тех пор, пока не достигнут конец цепочек или элементы цепочек будут равны». Например, фрагмент

```
        CLD
        MOV   CX,.100
REPE    CMPS   DEST, SOURCE
```

будет сравнивать до 100 элементов цепочек DEST и SOURCE, пытаясь найти,неодинаковые элементы. Следующий фрагмент

```
        CLD
        MOV   CX, 100
REPNE  CMPS   DEST, SOURCE
```

также сравнивает цепочки DEST и SOURCE. Операция прекращается после производства 100 сравнений или обнаружения одинаковых элементов цепочек.

Для явного указания типа элементов обычно допускаются мнемоники CMPSB и CMPSW.

Команда SCAS. Команда сканирования (просмотра) цепочки

SCAS *dst (ac)—(dst)*

вычитает элемент цепочки, адресуемый ES:DI, из содержимого аккумулятора AL (байт) или AX (слово). Разность определяет состояния арифметических флажков, но сами операнды не изменяются. С префиксом REPE (или REPZ) команду SCAS можно использовать, для поиска элемента цепочки, отличающегося от заданного значения, а с префиксом REPNE (или REPNZ) — равного заданному в аккумуляторе значению. Мнемоники SCASB и SCASW явно задают тип элементов цепочки. Рассмотрим, например, следующий фрагмент:

```
CLD              ;Движение вперед
MOV   AL, 30     ;Отыскиваемое значение
MOV   CX, 100    ;Длина цепочки
REPNZ SCASB      ;Сканировать цепочку
```

Здесь команда SCASB просматривает до 100 байт цепочки, пытаясь найти элемент, который содержит 30. Если такой элемент обнаруживается, в регистре DI возвращается смещение следующего за ним элемента, а флажок ZF будет содержать 1. Условие окончания

поиска можно проверить командой JCXZ (перейти, если в CX содержится 0): переход будет выполнен, если в цепочке нет байта, содержащего 30.

Команда LODS. Команда загрузки элемента цепочки в аккумулятор

LODS *src* *ac<-(src)*

передает элемент цепочки, адресуемый DS:SI, в аккумулятор AL или AX и продвигает SI на следующий элемент. Обычно команда LODS с префиксом повторения не применяется, но ее удобно использовать в программных циклах вместо команд MOV *ac, src* и INC SI (или DEC SI). Обычно допускаются мнемоники LODSB и LODSW, явно указывающие тип элементов цепочки.

Команда STOS. Команда запоминания аккумулятора в цепочке

STOS *dst* *dst<-(ac}*

передает байт (слово) из аккумулятора AL (AX) в элемент цепочки, адресуемый ES:DS и продвигает DI на следующий элемент. С префиксом повторения этой командой можно инициализировать целую цепочку (т.е. область памяти) на некоторое фиксированное значение, например на нуль или пробел. Мнемоники STOSB и STOSW явно индентифицируют тип элементов цепочки.

1.5.6. КОМАНДЫ УПРАВЛЕНИЯ МИКРОПРОЦЕССОРОМ

Несколько команд этой группы предназначены для управления состоянием флажков. С их помощью можно установить (STC), сбросить (CLC) и инвертировать (CMC) флажок переноса CF, установить (STD) и сбросить (CLD) флажок направления DF, установить (STI) и сбросить (CLI) флажок прерывания IF.

Команда HLT останова прекращает действия МП и переводит его в состояние останова. Из этого состояния МП выводится сигналом CLR сброса, а также запросами прерываний на входах NMI и INT. Таким образом, с помощью команды HLT действия МП синхронизируются с прерываниями от внешних источников.

Команда WAIT. ожидания заставляет МП циклически проверять уровень сигнала на входе TEST. При появлении активного (низкого) уровня сигнала TEST будет выполняться команда, находящаяся за командой WAIT. При выполнении команды WAIT микропроцессор стандартным образом реагирует на внешние прерывания, возвращаясь к прерванной команде после обслуживания прерывания. Механизм WAIT—TEST позволяет синхронизировать действия МП с работой какого-либо внешнего устройства.

Команда LOCK, называемая префиксом блокировки шины, заставляет МП сформировать активный сигнал LOCK на время выполнения следующей команды. В мультипроцессорных конфигурациях сигнал LOCK подается в арбитр шины, который не разрешает доступ к шине других устройств (процессоров).

Холостая команда NOP не производит никаких действий, кроме инкремента программного счетчика PC, и обычно применяется в программных циклах задержки. С помощью команды NOP можно также удалять из программы ненужные команды без ее повторного ассемблирования.

Команда ESC переключения на сопроцессор не является в строгом смысле командой МП К1810ВМ86. За этой мнемоникой скрываются все команды арифметического сопроцессора К1810ВМ87. Встретив такую команду, МП К1810ВМ86 может только обратиться к памяти за операндом и выдать его на шину данных (подробнее см. в гл. 2).