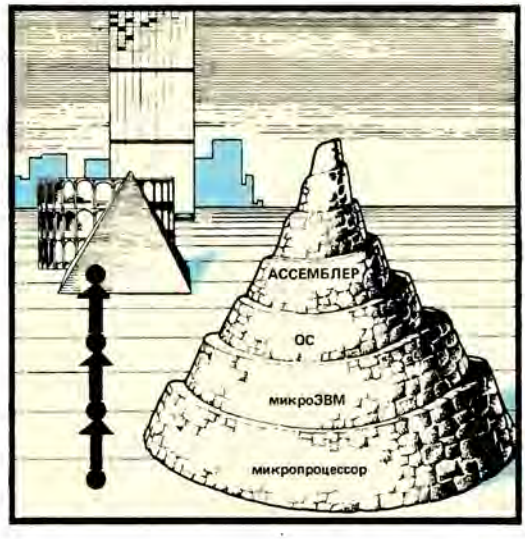


ОТ МИКРОПРОЦЕССОРА К ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

Процессор, в том числе МП, и остальное вычислительное оборудование (ЗУ, ВУ — все то, что на языке программистов именуется *hardware* — «жесткая компонента» или попросту «железо») сами по себе беспомощны. «Вдохнуть жизнь» в аппаратуру позволяет *программное обеспечение* (ПО, *software* — «мягкая компонента»): будь то несложная программа в ПЗУ микропроцессорного контроллера или целый комплекс программ для высокопроизводительной ЭВМ, одновременно обслуживающей множество терминалов. В последнем случае ПО настолько преображает возможности самой ЭВМ, что программист, работающий за терминалом, имеет дело как бы с другой, «виртуальной» машиной, а точнее — с *вычислительной системой*, понимающей слова и даже отдельные предложения на языке, близком к естественному.

Реально же ни МП, ни ЭВМ не умеют выполнять ничего, кроме ограниченного набора достаточно простых команд, представленных в ОЗУ или ПЗУ комбинациями нулей и единиц. Удобная для человека форма диалога — результат осуществляемой специальными программами интерпретации команд «виртуальной» машины. Даже для одного типа ЭВМ или МП можно создать множество новых языков диалога. В зависимости от степени близости к языку машины, их делят на *языки низкого* и *высокого уровня*. В этой главе будет рассмотрен язык самого низкого, исходного уровня для МП К1801 и сходных с ним ЭВМ — язык машинных команд. Именно на этот уровень, как на фундамент, надстраиваются «этажи» ПО, обеспечивающие более комфортный для программиста характер общения с вычислительной техникой.

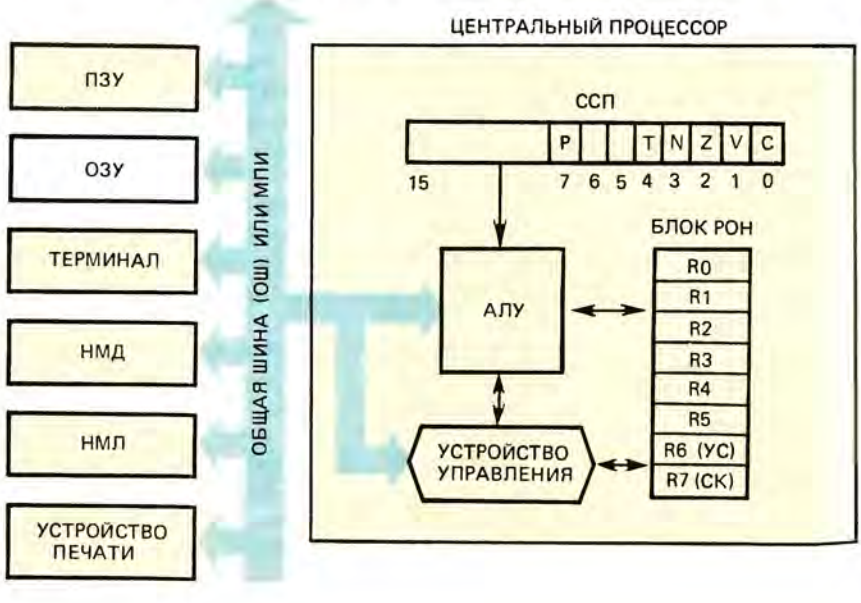
У читателя все эти знания уже есть. Напомним о них.

С точки зрения программиста ЦП представляет собой совокупность узлов, изображенных на рисунке 8.1. УУ координирует работу всех устройств. Каждая команда программы считывается с ОЗУ или ПЗУ и декодируется *(фаза выборки команды)*, затем при необходимости вычисляются адреса операндов, которые извлекаются из памяти *(фаза операндов)*, после чего выполняется заданное действие *(фаза исполнения)*. Сложность фазы операндов зависит от используемых способов адресации.

Большинство действий над операндами производится в АЛУ. Сложные операции выполняются путем многократного выполнения базовых операций АЛУ. Заметим, что АЛУ не осуществляет операций по пересылке данных, а лишь оперирует данными, поданными заблаговременно на его входы. Поэтому его нередко сравнивают со слепым жонглером, который потрясает зрителей своими головокружительными трюками с горящими факелами, но лишь после того, как их вручат ему в руки.

Для хранения слова состояния процессора служит специальный регистр, в отдельные биты которого УУ заносит информацию об особых случаях при выполнении некоторых действий. *Z-бит* и

В состав центрального процессора входят АЛУ, УУ, блок РОН и регистр ССП. Два последних узла доступны программисту: их состояния могут изменяться при выполнении команд.



N-бит устанавливаются, соответственно, при нулевом и отрицательном результатах (для представления отрицательных чисел используются дополнительные коды). *C-* и *V-биты* сигнализируют о переполнении при работе с натуральными и дополнительными кодами. Единичное значение *бита P* ССП запрещает внешним устройствам прерывания программы. Не рассматривавшийся до сих пор *бит T* (Trace — прослеживание) используется при отладке программ. Его единичное состояние вызывает внутреннее прерывание с адресом вектора 14_8 после исполнения каждой команды, что удобно, например, для выполнения программы по шагам при ее отладке.

В ЦП имеется 8 регистров общего назначения R0...R7. Регистры R6 и R7 наделены особыми функциями. Регистр R7 играет роль счетчика команд и всегда содержит адрес очередной подлежащей исполнению команды.

В процессе выборки команды УУ устанавливает на магистрали содержимое R7 и производит цикл «ввод». Слово, записанное по этому адресу в ОЗУ или ПЗУ, интерпретируется как очередная команда. Сразу после выборки команды (до ее исполнения) содержимое R7 автоматически увеличивается на 2, т. е. указывает адрес следующей команды.

R6 служит *указателем системного стека* (УС), в качестве которого может быть использована произвольная область ОЗУ (обычно — в «нижней» части памяти). УС всегда содержит адрес последней занятой ячейки, а в начале работы — верхнюю границу стека. К УС разрешено обращаться с помощью команд для работы с подпрограммами, некоторых команд, использующих R6, или как к одному из РОН. Обычно значение R6 уменьшается на 2 перед записью каждого слова в стек и увеличивается на 2 после каждого считывания. Такой механизм обеспечивает принцип «первым вошел, последним вышел», хотя сами элементы (слова) не перемещаются в ОЗУ. Изменяется лишь нижняя граница занимаемой стеком области ОЗУ. Аппаратный доступ к стеку используется в процессах обработки прерываний для хранения значений СК и ССП прерываемой программы.

Напомним, что ЦП рассматривает подключаемую к нему память как совокупность 64 Кбайт (32 Кслов) с адресами $0...177777_8$, причем адреса 0...376 обычно зарезервированы для векторов прерываний, а старшие 8 К адресов — для регистров ВУ. Это позволяет обойтись без специальных команд ввода-вывода, т. к. к регистрам ВУ ЦП может обращаться так же, как и к ОЗУ.

Если по исполняемым функциям все команды ЦП можно разделить на команды пересылки, управления и арифметико-логические (см. 2.11), то с точки зрения кодирования удобнее классификация по форматам (см. 3.7).

8.2. **Методы адресации**

Методы адресации весьма важны для понимания последующего материала.

Ограничимся рассмотрением единственной **команды очистки** содержимого регистра или ячейки памяти. Она относится к числу одноадресных (см. рис. 3.9), и ее восьмеричный код можно условно записать в виде $0050DD_8$ (либо $1050DD_8$ — для байтов), где $^0/_1050$ — код операции, DD — восьмеричные цифры, выражающие, соответственно, номера метода адресации и используемого при этом РОН. Наряду с восьмеричной записью используется и символическая запись, характерная для обозначения команд в языке *Ассемблер*. В этой форме записи КОП представляется в виде мнемонического английского обозначения, а остальные особенности станут ясными из примеров, иллюстрирующих механизм каждого из 8 основных способов адресации. В этих примерах через косую черту приводятся значения адреса и содержимого ячеек памяти (регистров), участвующих в процессе выполнения команды. Для самой команды очистки, имеющей мнемоническое обозначение **CLR** (от CLeaR — очистить), отведем постоянное место в ОЗУ по адресу 001000_8.

Метод 0. Регистровая адресация. Предполагается, что в указанном командой регистре находится сам операнд.

Пример: CLR R3

до выполнения	после выполнения
R3/123574	R3/000000
001000/005003	001000/005003

Метод 1. Косвенно-регистровая адресация. В указанном РОН содержится не сам операнд, а его адрес в ОЗУ.

Пример: CLR (R5)

до выполнения	после выполнения
R5/001216	R5/001216
001000/005015	001000/005015
001216/165123	001216/000000

Этот метод обеспечивает доступ к любой ячейке или байту ОЗУ.

Метод 2. Автоинкрементная адресация. Как и в методе 1, операнд находится по адресу, содержащемуся в указанном РОН, но после выполнения команды содержимое РОН автоматически увеличивается на 2 (или 1, если команда оперирует с байтами), указывая тем самым на следующее слово (байт) в ОЗУ.

Пример 1: CLR (R5)+

до выполнения	после выполнения
R5/020000	R5/020002
001000/005025	001000/005025
020000/145612	020000/000000

Пример 2: CLRB (R5)+

до выполнения	после выполнения
R5/020000	R5/020001
001000/105025	001000/105025
020000/145612	020000/145400

Метод 3. Косвенно-автоинкрементная адресация. Указанный в команде РОН хранит адрес ячейки, содержащей адрес операнда. После выполнения операции содержимое РОН увеличивается на 2.

Пример: CLR @ (R5)+*

до выполнения	после выполнения
R5/002066	R5/002070
001000/005035	001000/005035
002066/010100	002066/010100
010100/053530	010100/000000

Метод 4. Автодекрементная адресация. Содержимое РОН вначале уменьшается на 2 (или на 1 в случае байтовой команды), а затем используется в качестве адреса операнда. Обратите внимание на отличие в мнемонических обозначениях от автоинкрементного метода.

Пример: CLR —(R0)

до выполнения	после выполнения
R0/007000	R0/006776
001000/005040	001000/005040
006776/123456	006776/000000
007000/025340	007000/025340

Основное назначение методов 4 и 2 — обработка данных, расположенных в последовательных ячейках памяти.

Метод 5. Косвенно-автодекрементная адресация. Содержимое РОН уменьшается на 2, а затем используется в качестве адреса ячейки, содержащей адрес операнда.

Пример: CLR @—(R1)

до выполнения	после выполнения
R1/102520	R1/102516
001000/005051	001000/005051
102516/123560	102516/123560
123560/105407	123560/000000

Методы 5 и 3 позволяют организовать в ОЗУ «программные» стеки, использующие в качестве указателя любой из РОН.

* Значок @ (читается «эт») — еще одно, наряду с круглыми скобками, обозначение косвенной адресации.

Следующие 2 метода адресации на первый взгляд «запутанные», но тем не менее быстро запоминаются и помогают писать эффективные и быстро выполняющиеся программы.

Наличие в команде операнда, адресуемого методом 6 или 7, вызывает автоматическое извлечение слова, следующего непосредственно за командой, которое используется при вычислении адреса и называется индексным словом. Таким образом, команда с методом адресации 6 или 7 всегда занимает 2 последовательные ячейки памяти, т. к. записывается вместе с некоторым индексным словом, или же 3 ячейки, когда команда двухоперандная и оба операнда адресуются по методам 6 или 7. Для того чтобы индексные слова не воспринимались как команды, что приводило бы к ошибкам, процессор автоматически производит коррекцию СК.

Метод 6. Индексная адресация. Адрес операнда определяется как сумма индексного слова и содержимого указанного регистра общего назначения.

Пример: CLR 13004(R2)

до выполнения	после выполнения
R2/100512	R2/100512
001000/005062	001000/005062
001002/013004	001002/013004
113516/053720	113516/000000

Метод 7. Косвенно-индексная адресация. Сумма содержимого, указанного РОН и индексного слова, определяет адрес ячейки, содержащей адрес операнда.

Пример: CLR @ 13004(R2)

до выполнения	после выполнения
R2/100512	R2/100512
001000/005072	001000/005072
001002/013004	001002/013004
113516/053720	113516/053720
053720/044510	053720/000000

Все восемь методов адресации могут быть использованы в любых одно- и двухоперандных командах. В последнем случае каждому из операндов может соответствовать свой метод адресации.

8.3. **Использование счетчика команд для адресации операндов**

СК, т. е. РОН R7, может использоваться в командах со всеми рассмотренными методами адресации, однако в таком сочетании только четыре из них имеют практическое значение.

Помимо команды CLR, в примерах адресации с применением СК используем **команду сложения** (мнемоническое обозначение **ADD**), относящуюся к классу двухадресных (рис. 3.6). Поскольк-

ку код операции сложения — 06_8, то ее восьмеричная запись имеет вид

06SSDD,

где SS, DD — восьмеричные номера метода адресации и используемого регистра для операндов источника (S) и приемника (D). Вследствие достоинств, вытекающих из особой функции R7, эти 4 метода получили специальные названия и мнемонические обозначения:

1. Непосредственная адресация (СК + метод 2). Операнд выбирается из ячейки, следующей за командным словом.

Пример: ADD #2, R0

до выполнения	после выполнения
R0/004436	R0/004440
001000/062700	001000/062700
001002/000002	001002/000002

2. Абсолютная адресация (СК + метод 3). Из ячейки, следующей за командным словом, выбирается адрес операнда.

Пример: ADD @ #200, R0

до выполнения	после выполнения
R0/004430	R0/012070
001000/063700	001000/063700
001002/000200	001002/000200
000200/005440	000200/005440

3. Относительная адресация (СК + метод 6). Адрес операнда определяется как сумма содержимого СК и индексного слова. Заметим, что в момент суммирования с индексным словом СК скорректирован, т. е. содержит адрес очередной команды.

Пример: CLR 1120

до выполнения	после выполнения
001000/005067	001000/005067
001002/000114	001002/000114
001004/	001004/
001120/054100	001120/000000

4. Косвенно-относительная адресация (СК + + метод 7). Адрес операнда выбирается из ячейки, адрес которой равен сумме содержимого СК и индексного слова.

Пример: CLR @ 1120

до выполнения	после выполнения
001000/005077	001000/005077
001002/000114	001002/000114
001004/	001004/
001120/054100	001120/054100
054100/021577	054100/000000

Режимы адресации с использованием СК позволяют легко составлять программы, которые не требуют никакой переделки при необходимости изменить их положение в ОЗУ *(позиционно-независимый код)*, а также эффективно работать с данными, не организованными в массивы.

Особо отметим, что непосредственный и абсолютный методы адресации позволяют записывать прямо в тексте программы фиксированные адреса и константы. Фактическая длина команды в этом случае может составлять 2 или 3 машинных слова.

Безусловный переход. Подпрограммы и стеки

Любую сложную программу удобно разбить на отдельные части (фрагменты), которые можно независимо друг от друга разрабатывать и отлаживать. В этом случае собственно программа может состоять в основном из команд перехода к своим фрагментам, которые после выполнения должны передавать управление назад в основную программу. Однако при таком подходе затруднено многократное использование одних и тех же частей, например, при различных значениях аргументов *(параметров)*.

Мощный метод повышения эффективности — использование *механизма подпрограмм*, позволяющего избежать дублирования одинаковых фрагментов, а также легко включать в программы заранее написанные модули из библиотеки готовых программ. Как и для одноадресной (см. рис. 3.9) **команды JMP** (JuMP — прыгать) безусловной передачи управления, имеющей код 0001DD, обращение к подпрограмме приводит к выполнению группы команд, начиная с указанного адреса. Отличие состоит в том, что с целью обеспечить возврат к нужному месту основной программы, процессор обязан запоминать текущее содержимое СК в момент каждого обращения к подпрограмме *(адрес возврата)*.

Если же подпрограмма использует те же РОН, что и основная программа, сохранению подлежит и содержимое этих регистров. В конце подпрограммы нужно обеспечить восстановление прежнего содержимого как СК, так и этих РОН.

Последовательность возврата в основную программу усложняется, если подпрограмма, в свою очередь, может обращаться к другой подпрограмме и т. д. *(вложенные подпрограммы)*.

Эффективный порядок взаимодействия основной программы и подпрограмм обеспечивает помещение адресов возврата (а если необходимо, и содержимого РОН) в стек, расположенный в свободном месте ОЗУ. Такой стек может быть организован на основе любого РОН. Пример использования стека на базе регистра R5 для сохранения и восстановления содержимого РОН R0...R2 представлен на рисунке 8.2. В этом примере фигурирует **команда пересылки,** относящаяся к классу двухадресных (см. рис. 3.6). Ее мнемоническое обозначение — **MOV** (MOVe), а восьмеричный код $^{0}/_{1}$1SSDD.

Для работы с подпрограммами предусмотрены специальные

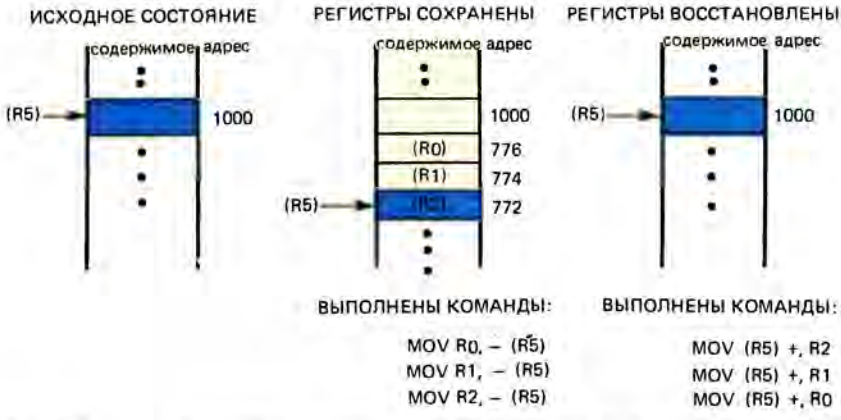


Рис. 8.2. Для работы со стеком, организованным на базе одного из РОН, например R5, можно использовать команду пересылки MOV, автоинкрементный и автодекрементный методы адресации.

команды. **Команда JSR** (Jump to SubRoutine — перейти к подпрограмме) с восьмеричным кодом 004RDD и форматом регистровой команды (рис. 3.7) вызывает занесение содержимого счетчика команд в указанный командой РОН *(регистр связи)*. Предыдущее содержимое этого регистра автоматически помещается в системный стек (на базе R6). Новое содержимое СК (адрес входа в подпрограмму) устанавливается в соответствии с адресной частью команды (DD).

Для организации возврата в основную программу последней командой подпрограммы должна быть **RTS** (ReTurn from Subroutine — вернуться из подпрограммы) с кодом 00020R, вызывающая обратное действие. При этом содержимое регистра связи (R), хранящее адрес возврата, помещается в СК, а «старое» содержимое РОН, используемого в качестве регистра связи, извлекается из системного стека.

Описанный метод гарантирует правильную последовательность действий как в случае вложенных подпрограмм, так и при вызове подпрограммой самой себя *(рекурсия)*.

При вызове подпрограммы в качестве регистра связи можно указать сам счетчик команд R7. В этом случае в системный стек помещается только содержимое СК, т. е. адрес возврата, который считывается при выполнении команды возврата. Соответствующая команда перехода к подпрограмме имеет мнемонический вид:

JSR PC, *subr*,

где PC — стандартное для Ассемблера обозначение *счетчика команд* (Program Counter), *subr* — произвольное символическое обозначение *(метка)* первой команды подпрограммы.

Для передачи значений параметров (аргументов, при которых

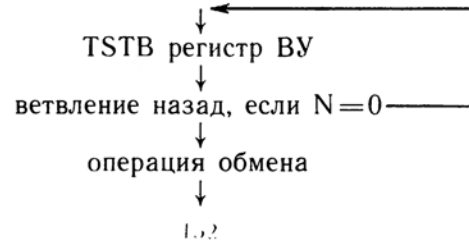
должна выполняться подпрограмма) и возврата результатов в основную программу применяют несколько способов. Самый простой из них состоит в резервировании нужного числа ячеек памяти (или РОН) для совместного использования основной программой и подпрограммой. Иногда удобнее располагать аргументы в ячейках ОЗУ, непосредственно следующих за командой JSR. Так как ее адрес сохраняется в регистре связи, то доступ к параметрам из подпрограммы осуществляют путем автоинкрементной адресации через этот регистр. Если таким же методом извлекать и самый последний параметр, то к моменту исполнения возврата регистр связи будет содержать адрес следующей за аргументами команды программы. Аналогично, вслед за JSR можно помещать адреса аргументов, а доступ к ним иметь посредством косвенной автоинкрементной адресации через регистр связи. В этом случае аргументы могут быть размещены в ОЗУ произвольно. Можно поступить и так: список параметров (адреса или собственно значения) поместить в любом месте ОЗУ, а в подпрограмму передавать только адрес первого элемента списка через любой из РОН.

Очень удобно передавать параметры, помещая их перед входом в подпрограмму в системный стек вместе с информацией о точке возврата.

8.5. Краткий обзор системы команд МП К1801ВМ1

Весь набор команд МП включает 64 команды. Они приведены в Приложении. Многие из них, такие,как INC (INCrement — увеличить на 1), DEC (DECrement — уменьшить на 1), CLR, команды операций с разрядами ССП и другие, не требуют пояснений.

Среди одноопе рандных команд отметим **команду TST** (от TeST — проверить, код $^{0}/_{1}057DD$). Ее действие заключается в установке битов N и Z ССП,если операнд (DD) соответственно отрицательный или равный нулю. Так как числа представляются в дополнительном коде, признаком отрицательного числа является установленный старший разряд, и поэтому можно считать, что при исполнении команды TST в разряд N ССП посылается старший бит проверяемого числа, т. е. бит 15 (команда TST) или 07 (команда TSTB). Поскольку в качестве флага готовности ВУ обычно используется бит 07, то команда TSTB весьма удобна для проверки флага. В этом случае за ней должна стоять команда условного перехода назад, если флаг еще не установлен.

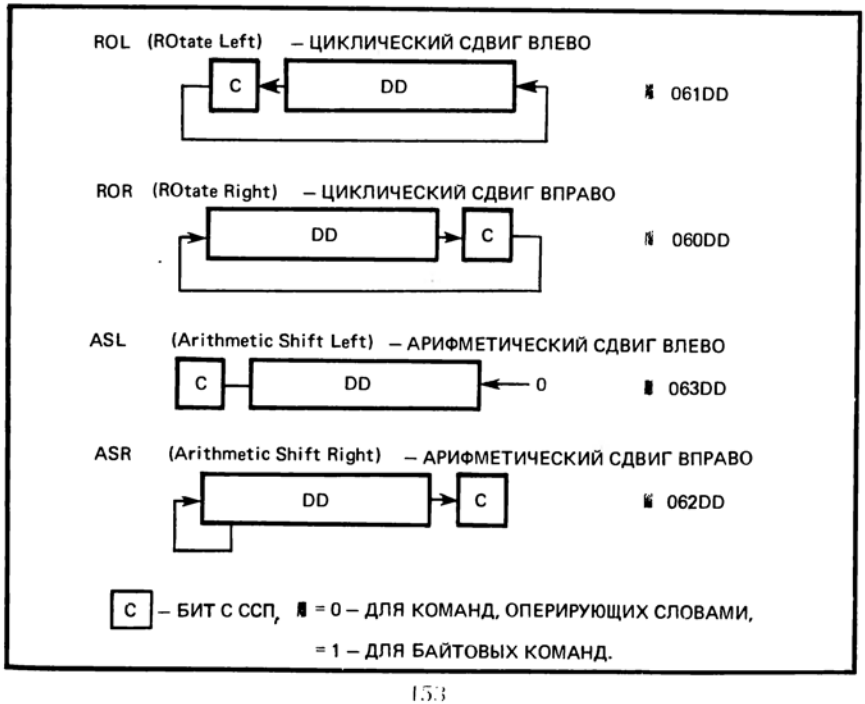


Проверку произвольных разрядов можно произвести двухоперандной **командой BIT** (BIt Test — побитная проверка), которая производит поразрядное логическое умножение двух операндов и заносит информацию о результате в разряды N и Z ССП аналогично команде TST. Сами операнды при этом не изменяются.

Для *сдвигов* операндов существуют четыре специальные команды, действие которых схематично показано на рисунке 8.3. Они широко применяются при умножении и преобразовании двоичных слов данных.

Команды ADC (B), SBC (B) служат для работы с операндами, занимающими более одного слова. Команда ADC (ADd Carry — прибавить перенос) складывает содержимое бита C с указанным операндом, а команда SBC (SuBtract Carry — вычесть перенос) производит его вычитание. В группе двухоперандных команд отметим **команду CMP(B)** (CoMPare — сравнить, код $^{0}/_{1}$2SSDD), сравнивающую значения своих операндов. Фактически она производит вычитание содержимого 2-го операнда (приемника) из 1-го (источника) с выработкой соответствующих результату значений битов ССП, однако после ее выполнения значения обоих операндов сохраняются.

Рис. 8.3. Команды ROL и ROR производят циклический сдвиг операнда на одну позицию через бит C ССП. Команды ASL и ASR — особый вариант сдвиговых операций. Их действие эквивалентно целочисленному умножению (делению) на 2.



Команда SOB (Subtract One and Branch if not equal to 0 — вычесть 1 и перейти, если не равно 0, код 077RNN) исключительно удобна при организации циклов в программе. В начале выполнения команды происходит вычитание единицы из регистра R, затем если результат оказался ненулевой, происходит ветвление назад со смещением NN. Для этого из счетчика команд вычитается удвоенное шестибитное смещение NN.

Группа команд условного перехода *(ветвления,* см. рис. 3.8) включает семнадцать команд, допускающих передачу управления на величину *смещения* (со знаком) в зависимости от значения отдельных битов ССП и их комбинаций. При невыполнении выбранного условия исполняется следующая команда. Поскольку слова имеют четные адреса, а во время исполнения команды ветвления СК уже содержит адрес следующей команды, то для перехода с адреса A на адрес B требуемое смещение можно вычислить по формуле $X = (B - A - 2) : 2$ с записью результата в дополнительном коде. Очевидно, $-128_{10} \leqslant X \leqslant 127_{10} (-200_8 \leqslant X \leqslant \leqslant 177_8)$.

Команды ветвления можно разбить на три подгруппы. В первой (команды BNE, BEQ, BPL, BMI, BVC, BVS, BCC, BCS) анализируются отдельные коды условий ССП. Команды второй подгруппы (BGE, BLT, BGT и BLE) проверяют комбинации кодов условий и позволяют сравнивать операнды с учетом знаков (в дополнительных кодах). Команды третьей подгруппы (BHI, BLOS, BHIS и BLO) применяют для сравнения беззнаковых чисел в натуральном коде. Команды ветвления обладают свойством позиционной независимости.

Безадресные команды (рис. 3.10), помимо изменения битов ССП, используются для непосредственного управления процессором. **Команда HALT** (останов, код 000000) прекращает выполнение команд. По **команде WAIT** (ожидание, код 000001) выполнение программы прекращается временно, до поступления прерывания от ВУ. **Команда RESET** (сброс, код 000005) инициирует сигнал INIT на МПИ, используемый для приведения ВУ в исходное состояние. **Команда NOP** (No OPeration, код 000240) не выполняет никакого действия и применяется при отладке программ или организации временных задержек.

8.6. Основные ВУ микро-ЭВМ

Большинство вычислительных систем на основе микро-ЭВМ имеет минимальный набор внешних устройств: алфавитно-цифровой терминал (дисплей), печатающее устройство (принтер) и ВЗУ на гибких магнитных дисках.

Дисплей, снабженный алфавитно-цифровой клавиатурой, имеет три основных состояния:

1) основное или «комплекс» (on-line) — работа с ЭВМ, при котором клавиатура и экран работают независимо друг от друга, как устройства, соответственно, ввода и вывода;

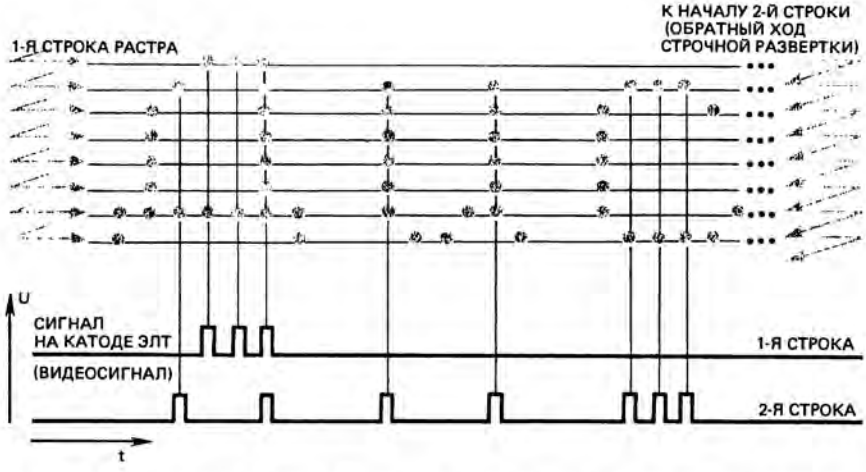
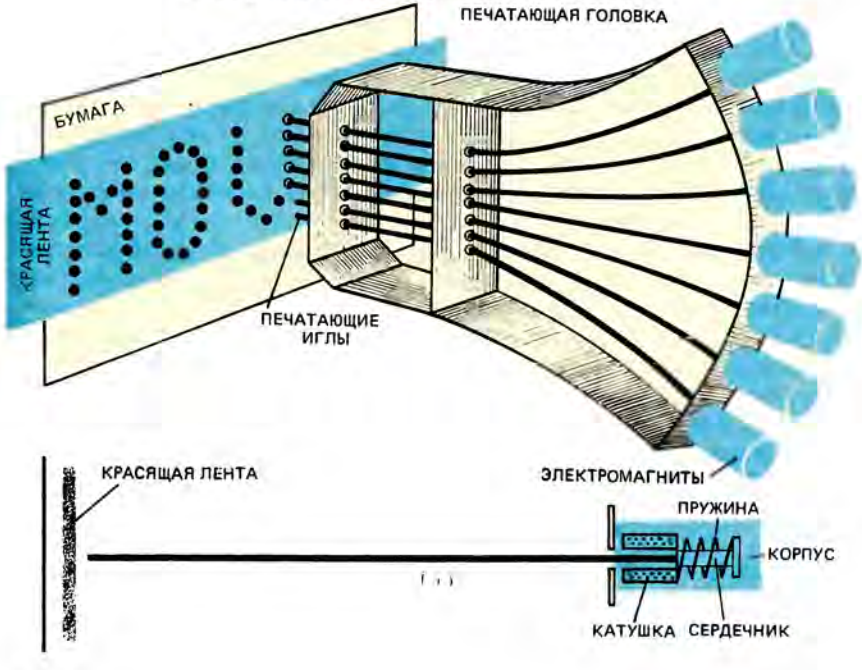


Рис. 8.4

Вывод одной строки текста на экран алфавитно-цифрового дисплея осуществляется за несколько ходов строчной развертки. Электронным лучом управляет специальная схема знакогенератора, основу которой составляет ПЗУ, содержащее (в двоичном виде) очертания используемых символов.

Рис. 8.5

Печатающая головка принтера содержит вертикальный ряд тонких упругих стержней (игл). В процессе вывода данных из ЭВМ наряду с перемещением головки вдоль печатаемой строки срабатывают выбранные электромагниты, «выталкивающие» иглы к красящей ленте, которая расположена непосредственно перед бумагой. Печатаемые символы обычно формируются в матрице 7×5 точек. При простоте устройства такие принтеры обладают быстродействием до 200 знаков в секунду и в принципе способны к воспроизведению графических изображений.



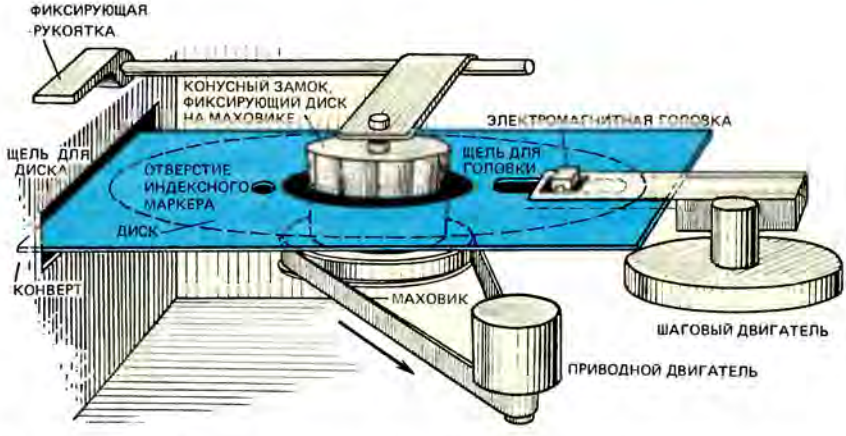


Рис. 8.6. Диск, установленный в НГМД, вращается внутри защитного конверта с частотой 6 оборотов в секунду. Электромагнитная головка может перемещаться вдоль радиуса диска и устанавливаться над одной из дорожек, после чего способна к считыванию/записи информации. Отверстие в диске (индексный маркер) позволяет синхронизировать эти операции с вращением диска посредством простого фотодатчика. Некоторые типы НГМД содержат две головки для работы с верхней и нижней поверхностями диска.

2) «автоном» (off-line) — автономная работа, когда терминал логически отсоединен от ЭВМ и вся информация с клавиатуры попадает непосредственно на экран;

3) «установка» — установка режимов работы терминала.

В режиме «комплекс» при нажатии какой-либо алфавитно-цифровой клавиши формируется соответствующий код, поступающий в регистр данных клавиатуры интерфейса терминала, что сопровождается установкой флага готовности в соответствующем регистре состояния. На экран же информация поступает побайтно из регистра данных экрана, причем коды в интервале 40... 177_8 приводят к появлению в очередной позиции экрана соответствующего символа, а коды $0...37_8$ являются *управляющими* и не отображаются. Реакция на прием управляющих кодов различна у разных типов терминалов. Для формирования изображений в дисплеях обычно используют растровый метод, аналогичный применяемому в телевизорах (рис. 8.4).

Для получения результатов вычислений, текстов программ и других документов в печатном виде микро-ЭВМ комплектуется *печатающими устройствами* (принтерами). Наиболее распространены точечно-матричные принтеры (рис. 8.5).

Накопитель на гибких магнитных дисках — наиболее распространенное ВЗУ для микро-ЭВМ (рис. 8.6). Гибкие магнитные диски изготовляются из майларовой пленки с ферромагнитным покрытием и имеют диаметр 203 мм, 133 мм и меньше. Информа-

ция записывается на концентрические *дорожки*, каждая из которых делится на *секторы*, содержащие обычно по 128 байт данных. Дорожки располагаются на поверхности диска с плотностью до 60 дорожек/мм, причем каждая из них имеет 8...26 секторов в зависимости от формата записи. Полная емкость одного диска достигает нескольких Мбайт.

8.7. **Пультовой режим работы**

Большинство микро-ЭВМ не имеет специального пульта управления, при помощи которого оператор может записать или просмотреть содержимое ячеек ОЗУ, запустить или остановить программу, контролировать состояние ЭВМ. Чаще эту роль поручают специальной программе, называемой *эмулятором пульта управления*. Она хранится в специальном системном ПЗУ. «Диалог» с пользователем ведется с помощью алфавитно-цифрового терминала. Обычно микро-ЭВМ входит в режим связи с пультовым эмулятором при включении питания, исполнении команды HALT (останов), появлении активного сигнала на линии аппаратного останова В HALT L и некоторых ошибочных ситуациях.

В этих случаях на экране терминала появляются текущее содержимое СК и символ «@» в новой строке, означающий, что микро-ЭВМ находится в режиме пультового терминала и готова принимать команды оператора. Приведем некоторые из них.

Для просмотра или изменения содержимого ячеек ОЗУ, РОН или ССП следует ввести восьмеричный адрес ячейки, либо наименование РОН (ССП), а затем ввести символ «/» *(«открыть ячейку»)*. Содержимое указанной ячейки, РОН или ССП, выводится в восьмеричном виде вслед за знаком «/». После этого можно ввести новое содержимое открытой ячейки, завершив его одним из следующих символов: <ВК>, <ПС>, «↑», «@» или «_» (подчеркивание). Если новое содержимое ячейки не введено до начала очередной команды, оно останется неизменным.

В любом случае эмулятор закрывает открытую ячейку, а затем, в зависимости от поданной команды, выполняет следующее:

<ПС> — открывает ячейку со следующим адресом;

«↑» — открывает ячейку с предыдущим адресом;

«@» — открывает ячейку, адресом которой является содержимое ранее открытой ячейки или РОН;

«_» — открывает ячейку, адресом которой является содержимое ранее открытой ячейки+ее адрес и+2;

<ВК> — закрывает открытую ячейку.

Команда «G» вызывает запуск программы. Перед ее подачей необходимо набрать стартовый адрес программы, в противном случае выполнение начнется с адреса 0.

Команда «P» позволяет продолжить выполнение программы,

остановленной по каким-либо причинам. Если при этом на линии B HALT L МПИ постоянно активный уровень, то по каждой пультовой команде «Р» будет выполняться лишь одна команда программы — это используется для пошагового выполнения программ при отладке.

Пример:

```
160000                                  Включение питания. В ячейках ОЗУ — случайная информация.
@
@  1000/177777 240 <ПС>*
001002/177777  123456<ПС>               Запись в последовательные ячейки ОЗУ чисел 240₈, 123456₈, 55₈ с адреса 1000.
001004/177777  55<ПС>
001006/000377  <ВК>

@  1000/000240  <ПС>
001002/123456  <ПС>                     Проверка записанной информации.
001004/000055  <ПС>
001006/000377  <ВК>

@  RS /000240  <ВК>                     Просмотр содержимого ССП.

@  R0/120137  <ПС>
R1/000012  <ПС>                         Просмотр содержимого РОН.
R2/000377  <ВК>
@
```

Для начальной загрузки ЭВМ используются команды: «177550L» — загрузка с перфоленты; X*n* — загрузка с накопителя на гибких мини-дисках, $n=0...3$ — номер дисковода; D*n* — загрузка с гибких дисков, $n=0,1$ — номер дисковода.

Набор команд пультового режима позволяет вводить в микро-ЭВМ небольшие программы в машинных кодах, отлаживать и выполнять их без применения внешних ЗУ.

Помимо пультового терминала, управление микро-ЭВМ осуществляется выключателем «Питание» и клавишей «Программа/Пульт», изменяющей состояние линии B HALT L магистрали МПИ.

Что такое операционная система?

Составлять программы на понятном для ЭВМ языке ее машинных команд — дело трудоемкое. Еще труднее их модифицировать и отлаживать. Нередко единственная ошибка может потребовать переделки всей программы. Использованное в предыдущих параграфах символическое обозначение машинных команд, бесспорно, упрощает их запоминание и восприятие. С его помощью легче составлять и модифицировать программы, так как подстановка вместо символов их двоичных или восьмеричных эквивалентов может производиться после всех изменений и исправлений. Очевидно, что при строгом соблюдении правил символической записи программы процесс ее перевода в машинные коды носит

---

* Символы, вводимые оператором, выделены шрифтом.

механический характер, а значит, может быть поручен самой ЭВМ. Необходимая для такого перевода *(трансляции)* программа носит название *ассемблер*. Ассемблером называется и язык, на котором составляются символические программы.

Процесс трансляции возможен при наличии хотя бы простейшего ВЗУ для хранения программы-транслятора и терминала для ввода текста программы, воспринимаемой транслятором как исходные данные.

Результат перевода в большинстве случаев целесообразно записать в то же ВЗУ для многократного использования. Роль такого ВЗУ обычно играют НМД (НГМД), реже — НМЛ, а в простейших случаях — *перфоленточные станции*, содержащие фотосчитыватель и перфоратор.

В процессе создания программы неизбежны ошибки, зачастую носящие синтаксический характер. Многие из них может обнаружить хорошо спроектированный транслятор, сообщая об этом посредством терминала. В любом случае избежать повторного ручного ввода исправленной программы позволяет специальная программа — *редактор*, предоставляющая возможность вносить изменения в программы, хранящиеся во ВЗУ. В функции редактора входит также прием исходного текста с терминала и размещение его во ВЗУ.

Процесс разработки прикладных программ упрощают и другие специальные программы: *отладчики*, позволяющие тщательно, шаг за шагом, проконтролировать выполнение прикладной программы; *библиотекари*, накапливающие уже опробованные фрагменты или целые программы универсального назначения, и другие. Все эти программы целесообразно хранить во ВЗУ, извлекая по мере надобности. Процедуры записи и считывания, в общем случае — обмена данными с ВУ, можно поручить специальным программам — *драйверам*. Но остаются заботы типа: где и в какой форме разместить программы во ВЗУ, чтобы избежать путаницы; определить, какие области ВЗУ еще свободны и нет ли в них дефектов; скопировать программу с одного носителя информации на другой; составить список имеющихся программ и т. д. В случае перфоленточных ЗУ эти операции неизбежно требуют достаточно однообразного, рутинного труда. При наличии ВЗУ типа НМД доля этого непроизводительного труда может быть сведена к минимуму. С этой целью используется главная управляющая программа — *монитор*, координирующая работу ВУ, самой ЭВМ и всех специальных программ путем интерпретации команд, подаваемых с клавиатуры терминала.

Монитор, драйверы, трансляторы, редакторы, отладчики — все это компоненты создаваемого высококвалифицированными программистами комплекса программ, называемого *операционной системой* (ОС). Одна из основных функций ОС — обеспечение удобного взаимодействия человека с аппаратурой в процессе разработки программ. «Услуги» операционной системы упрощают

и ускоряют этот процесс, и в результате ОС выступает как своеобразное средство автоматизации программирования.

Монитор или, по крайней мере, основная его часть должны постоянно *(резидентно)* находиться в ОЗУ ЭВМ, вызывая по необходимости остальные компоненты ОС или другие программы. Сам же монитор загружается в ОЗУ с помощью очень короткой *программы начальной загрузки* (bootstrap), хранящейся обычно в ПЗУ ЭВМ.

ОС — не только средство удобного взаимодействия человека с ЭВМ, но и мощный инструмент повышения эффективности использования вычислительной техники, а также среда, поддерживающая создание и выполнение прикладных программ.

8.9. Эволюция операционных систем. Мультипрограммирование

Эффективность использования всех ресурсов ЭВМ высокой производительности с дорогостоящим периферийным оборудованием может быть существенно повышена за счет одновременного коллективного доступа к ним нескольких программ или пользователей.

Коэффициент использования ЭВМ 50-х и даже 60-х годов в среднем не превышал 10 % прежде всего из-за низкой скорости используемых в ту пору ВЗУ на базе перфолент, перфокарт, магнитных лент. Для многих ЭВМ типична занятость множеством различных, непериодически выполняемых программ, в том числе трансляторов, загружаемых с медленного ВЗУ, поэтому процессор значительную часть времени занимается не вычислениями, а вспомогательными операциями ввода-вывода. При этом неизбежны и простои из-за нерасторопности программистов, вынужденных заниматься поиском нужных носителей, установкой их во ВЗУ и т. п. Парадоксально, но достоверно, что с переходом на более быстродействующие процессоры эффективность работы некоторых ЭВМ уменьшалась.

Такая ситуация формировала поиски более быстродействующих ВЗУ. Так появились ЗУ на магнитных барабанах, а затем и дисках. Рутинные же действия операторов ЭВМ постепенно брали на себя все усложняющиеся мониторы — прообразы будущих ОС. Совершенствовались и способы взаимодействия ЭВМ с ВУ: медленно работающие устройства стали снабжаться буферной памятью, а в системе ввода-вывода появился аппарат прерываний. ЭВМ вооружились, например, возможностью «сбросить» в такой буфер целый массив данных, и, не дожидаясь их обработки ВУ, продолжать выполнение программы, которая может быть прервана по мере очередной готовности ВУ к обмену.

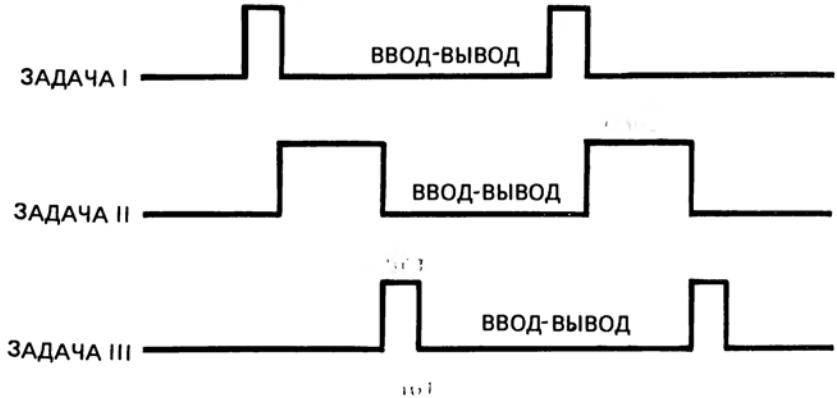
Но далеко не все программы могут продолжать работу, пока не окончился сеанс обмена с ВУ. В результате из совокупности процессора и множества ВУ работающим чаще всего оказывается лишь одно устройство.

Выход был найден в концепции *мультипрограммирования*.

Если в ОЗУ разместить несколько готовых к выполнению программ (задач), то паузой в работе одной программы могут воспользоваться другие. Функцию «переключения» задач естественно поручить монитору, а простои процессора могут быть сведены к минимуму за счет «сеанса одновременной игры» сразу со всеми задачами (рис. 8.7). Еще одна компонента монитора, называемая диспетчером, в состоянии подбирать для ОЗУ такую «смесь» задач из «пакета», установленного на ВЗУ, чтобы таким же образом загрузить почти постоянной работой не только процессор, но и большинство имеющихся ВУ.

В случае параллельного выполнения нескольких программ нельзя полностью «доверить» процессор какой-либо одной из них, поскольку все они должны конкурировать между собой за право доступа к основным ресурсам ЭВМ: ЦП, ВУ и ОЗУ. Чтобы не потерять власть над событиями, монитор должен «перехватывать» в программах команды, претендующие на монопольное использование процессора (например, изменяющие его состояние) или ВУ. Привилегия использования таких команд должна быть уделом только монитора. Проще всего это достигается, если процессор имеет два режима работы: т. н. *привилегированный* (Kernel mode), в котором возможно выполнение всех машинных команд, и *пользователя* (User mode) с ограниченными возможностями для обычных программ. Интересы программистов при этом практически не ущемляются. Они могут использовать в программах многие «запрещенные» команды. Дело в том, что попытка их выполнения процессором приводит к внутреннему прерыванию, вызывающему «на помощь» монитор, который, работая в привилегированном режиме, принимает нужное решение с учетом интересов всех находящихся в ОЗУ задач. Если программа имеет право на требуемое действие, то монитор может произвести его,

Рис. 8.7. Принцип мультипрограммирования позволяет более эффективно использовать вычислительные ресурсы за счет совмещенного выполнения вычислений и операций ввода-вывода для нескольких задач.



а затем вновь переключить процессор на выполнение программы в режиме пользователя.

Описанный механизм настолько эффективен, что целесообразно обращение пользовательских программ к монитору не только в «принудительном» порядке, но и с обычными запросами, например за услугами по обмену с ВЗУ, к стандартным и всевозможным сервисным программам. Делается это с помощью специальных *директив,* которые существенно расширяют возможности обычных языков программирования, например Ассемблера.

Последовательность выполнения различных задач в мультипрограммных ОС определяется диспетчером. Для рассмотренного нами механизма работы ОС стратегия диспетчера состоит в обеспечении максимальной загрузки всего оборудования задачами, образующими заранее подготовленный пакет. Такой режим работы ОС получил название *пакетного.* Основной недостаток его состоит в том, что ЭВМ «закрыта» для пользователей, которые не имеют возможности вмешаться в выполнение своих задач, даже если оно становится бессмысленным в результате какой-либо ошибки.

Возможность общения программистов с ЭВМ достигается с помощью иных режимов работы ОС, прежде всего — диспетчера. В режиме *разделения временных ресурсов* (РВР), или просто — разделения времени диспетчер поочередно, на равновеликие промежутки *(кванты)* времени предоставляет процессор готовым к исполнению задачам. Кванты имеют величину порядка десятой доли секунды и программист, работающий за терминалом, обычно не замечает, что ЭВМ «отвлекается» на решение других задач. Для ЭВМ эти кванты достаточно велики, и за каждый из них она в состоянии выполнить многие тысячи операций. В результате у каждого программиста создается полная иллюзия работы тет-а-тет с ЭВМ, а стратегию работы диспетчера можно выразить девизом: «всем — поровну».

Такая «уравниловка» неприемлема в целом ряде случаев. Так, в системах автоматизации эксперимента равные права на ресурсы ЭВМ для программ, обеспечивающих накопление и обработку (или вывод) данных, могут привести к потерям измеряемой информации, что недопустимо. Избежать таких ситуаций можно, если присвоить каждой задаче, в зависимости от ее степени важности, определенный *приоритет,* а работу диспетчера подчинить стратегии наискорейшего выполнения задач с наивысшими приоритетами. Такой механизм работы ОС носит название *режима реального времени* (РВ). Недостаток режима РВ — «недемократичность», проявляющаяся в простоях, а иногда — в полном подавлении низкоприоритетных задач при повышенной активности задач с высокими приоритетами.

Современные ОС обычно способны сочетать свойства сразу нескольких режимов работы, благодаря чему недостатки, свойственные каждому из них в отдельности, становятся малосущественными.

Говоря об архитектуре здания, часто имеют в виду пропорции, гармонию — эстетические соотношения между частями единого целого. Специалиста-архитектора интересуют и более конкретные, технические соотношения между подсистемами здания (водо- и электроснабжение, отопление, несущие конструкции и др.). Подобно этому под *архитектурой* сложных систем, в том числе и вычислительных (ВС), понимают способ распределения реализуемых ими функций по определенным уровням организации с точным определением функций, выполняемых на каждом уровне.

С любым уровнем связан определенный язык общения с ВС, т. е. совокупностью аппаратуры и ПО. Заманчиво наделить ЭВМ способностью непосредственно воспринимать предложения языка очень высокого уровня, но проектировать такую машину было бы очень трудно и вряд ли целесообразно. Значительно проще разработать ЭВМ с несложной, но полной системой команд, а на ее основе — ПО, транслирующее в машинные коды команды языка некоторого промежуточного уровня. В свою очередь, на этом языке проще создавать ПО для трансляции предложений языка еще более высокого уровня и т. д. Существуют два механизма трансляции: интерпретация и компиляция.

В процессе работы *интерпретатора* специальная программа распознает очередную команду и осуществляет переход к подпрограмме, выполняющей требуемую функцию. Такая организация служит предпосылкой диалогового режима работы: составляющие интерпретатор программы в состоянии незамедлительно информировать об ошибках в подаче команд, результатах их выполнения, подсказывать правильный порядок действий. Интерпретаторы удобны для обработки команд в ОС, автоматизированных системах управления. Широко распространены интерпретаторы для трансляции и выполнения программ, написанных на языках высокого уровня Бейсик, Фокал.

Трансляторы этого типа имеют и определенные недостатки. Громоздкий интерпретатор (или его часть) должен находиться в ОЗУ вместе с интерпретируемой программой, ограничивая ее размер. Сама же программа выполняется относительно медленно, особенно если она содержит цикл. В этом случае при каждом новом проходе цикла механически повторяется трансляция предложений, образующих тело цикла.

Трансляция методом *компиляции* устраняет эти недостатки. Компилятор транслирует программу целиком так, что ее выполнение возможно только после трансляции всего исходного текста.

Интерпретаторы обычно содержат в себе все компоненты, необходимые для создания, модификации и отладки программ. Так, существуют версии интерпретатора языка Бейсик, «зашитые» в ПЗУ и способные работать даже на ЭВМ минимальной конфигурации. Компиляторы, которые обычно используют для трансляции с языков типа Ассемблер, Фортран, Паскаль, нужда-

ются в определенном аппаратном и программном обеспечении (см. 8.8).

Традиционно структура архитектурных уровней ВС выглядит следующим образом. Возможности машинного языка развиваются и дополняются на уровне ОС. На базе ОС строятся уровни *языковых процессоров — трансляторов* (интерпретаторов и компиляторов) с различных языков программирования. На этих уровнях ВС обычно сохраняет свойство универсальности в задачах обработки данных. Вышележащие уровни, в числе которых т. н. *пакеты прикладных программ,* создаются на основе подходящего языка программирования и ориентированы на решение уже достаточно узкого круга типовых задач в конкретной (проблемной) области.

Перечисленные уровни образуют иерархическую систему: каждый из них опирается на один из предыдущих, но «обволакивает» его так, что все «винтики» нижележащих уровней обычно оказываются скрытыми от пользователей ВС. Это упрощает не только разработку, но и практическое использование ЭВМ: программисту, особенно начинающему, для работы на Фортране можно ничего не знать о деталях машинного языка и иметь лишь поверхностное представление об уровне ОС.

В ряде же случаев программист обязан знать особенности не только машинного, но и еще более низкого уровня. Дело в том, что для некоторых ЭВМ УУ, входящее в состав процессора, само может быть организовано как некий процессор. Этот процессор, воспринимая код команды «традиционного» машинного уровня, включает для ее выполнения «зашитую» в ПЗУ программу *(микропрограмму),* состоящую из еще более элементарных, чем команды действий, *микрокоманд* типа занесения в регистр, сдвига, стробирования и т. п. УУ этого типа носят название *микропрограммных.*

Основное преимущество микропрограммных УУ — простота их модификации, что особенно удобно на стадии проектирования. Так, для одного и того же процессора можно разработать несколько систем команд и разместить микропрограммы их выполнения в отдельных ПЗУ. Тогда переход от одной системы команд к другой, т. е. возможность работы с совершенно другой машиной достигается простой заменой ПЗУ в УУ. Такая возможность реализована в некоторых современных ЭВМ для имитации работы других машин. При этом вместо ПЗУ часто используют БИС ЗУПВ, загружаемые в начале работы нужными микропрограммами с внешнего накопителя на гибких магнитных дисках или кассетной ленте. ЭВМ этого типа называются *микропрограммируемыми.*

Аналогичным свойством обладают многие микропроцессоры, пользователям которых предоставляется возможность самим разработать систему команд, наиболее эффективных для решаемой задачи или круга задач. Микропрограммы этих команд и программы, составленные на их основе, после отладки помещаются в отдельные БИС ПЗУ.

Формулировка и уточнение границ архитектурных уровней –

важная часть работы по проектированию ВС. Программисту знание архитектуры ВС необходимо для выбора наиболее эффективных путей и средств решения поставленной задачи.

8.11. Совместимость. Семейства ЭВМ

Быстрый прогресс микроэлектроники приводит к тому, что доля ПО в общем объеме работ по созданию ВС неуклонно возрастает и уже сейчас значительно превосходит долю аппаратного обеспечения. Один из путей снижения трудоемкости создания ПО — достижение *совместимости* различных ЭВМ и ВС, созданных на их основе. Под совместимостью принято понимать возможность выполнения программ, созданных для одной ВС, на других ВС. Совместимость может быть реализована на отдельном архитектурном уровне, например, для всех программ, написанных на языке Фортран. Это достигается стандартизацией требований к разрабатываемым для различных ЭВМ трансляторам. Ведутся работы по стандартизации уровня ОС. Так, распространенная ОС UNIX путем минимальной модификации может быть адаптирована практически к любой ЭВМ. ОС этого типа называют *мобильными.*

Достигнутая на одном из уровней совместимость позволяет заимствовать для новой ВС часть ПО, накопленного в ходе эксплуатации других ВС. При идентичности системы команд двух машин возможен перенос всего ПО одной ЭВМ на другую. В результате можно говорить о совместимости самих машин и о тождественности архитектуры ВС, созданных на их основе.

А правомерно ли в этом случае утверждать о тождественности самих ЭВМ? ЭВМ как сложная техническая система обладает своей архитектурой, включающей микропрограммный, схемотехнический и другие уровни, каждый из которых выполняет свои определенные функции. Для совместимости машин, очевидно, играют роль лишь характеристики самого внешнего уровня, т. е. системы команд. Поэтому под *архитектурой ЭВМ* принято понимать лишь те функции, которые доступны программисту, работающему на уровне машинных команд. Из деталей внутренней организации машины в это понятие входит только то, что каким-либо образом отражается на системе команд: структура памяти, механизмы адресации, наличие и способы использования РОН, особенности системы ввода-вывода и т. п. Таким образом, совместимые ЭВМ обязаны обладать тождественной архитектурой, но могут различаться своим устройством, конструктивным исполнением, потребляемой мощностью, объемом адресуемой памяти, быстродействием и т. д.

Уже по этой причине целесообразен выпуск *семейств ЭВМ,* состоящих из различных модификаций *(моделей)* машин с однотипной архитектурой. Еще большее разнообразие моделей допускает *принцип совместимости «снизу-вверх».* В этом случае система команд всякой новой (более «старшей») модели обязана

содержать как подмножество базовую систему команд, характерную для самой «младшей» модели, но может расширяться за счет введения новых команд и способов адресации, дополнительных РОН, режимов работы процессора и т. д. В результате на «старшие» модели без каких-либо изменений переносятся все программы, созданные для более «младших» моделей, а дополнительные возможности могут быть использованы для дальнейшего развития ПО.

Отечественной промышленностью выпускается несколько семейств ЭВМ. Семейство ЭВМ *Единой Системы* (ЕС) охватывает диапазон от «больших» высокопроизводительных машин до персональных компьютеров (ЕС 1840, 1841). В *Системе Малых* (СМ) ЭВМ развиваются три архитектурных семейства. В одно из них входит рассмотренный нами МП К1801ВМ1. Основные характеристики этого семейства ЭВМ представлены в таблице 8.1.

Сравнить указанные в таблице модели ЭВМ по производительности довольно сложно, поскольку реальная производительность зависит от целого ряда параметров. Косвенно о производительности можно судить по числу одновременно обслуживаемых пользователей, объему ОЗУ, наличию расширенных наборов команд.

Все, что ранее говорилось о МП К1801, в полной мере справедливо для «младших» моделей ЭВМ в таблице 8.1.

Наиболее серьезное архитектурное отличие «старших» моделей — это увеличение максимально возможного объема ОЗУ до 124 Кслов и до 1920 Кслов с помощью *диспетчера памяти* MMU (от англ. Memory Management Unit — блок управления памятью), который, кроме того, позволяет разделить адресное пространство на секции, ограничивая их влияние друг на друга и обеспечивая тем самым многопользовательский и мультипрограммный режим работы. Что касается программного обеспечения, то наибольшей популярностью на ЭВМ с объемом ОЗУ до 28 Кслов пользуются различные варианты однопользовательской ОС **RT-11** (об этом — следующая глава), а на более производительных моделях применяется, в основном, многопользовательская ОС **RSX-11M** (ОС РВ). В обеих ОС есть возможность выполнять программы, составленные на большинстве широкораспространенных языков программирования: BASIC, PASCAL, FORTRAN-IV, MODULA-2, а для RSX-11M, кроме того, разработаны компиляторы языков FORTRAN-77, FORTRAN-IV PLUS, BASIC-RLUS2, ALGOL, COBOL, C, FORT, RATFOR и другие.

ХАРАКТЕРИСТИКА ПРОГРАММНО-СОВМЕСТИМЫХ МАЛЫХ ЭВМ

Таблица 8.1

ЭВМ	Тип магистрали *	Макс. об'ем ОЗУ, Кслов	Расширенные наборы команд **			Число уровней прерываний	Режим процессора	Стандартное системное ПО	Число пользователей	Тип используемого МП
			EIS	FIS	FPP					
Электроника 100/16	U	28/30	—	—	—	1	KERNEL	РАФОС	1	К581
Электроника 60	Q							ФОБОС (RT-11)		К1801ВМ1
Электроника МС1201, МС1201.01 ***	Q							ОС ДВК		
СМ 3	U									
СМ 1300	U							МОС РВ (RSX-11S)		К1802
Электроника 60М	Q		+	+	—			IDRIS		К581
Электроника 1201.02	Q									К1801ВМ2
Электроника 100/25	U	124				4	KERNEL USER	ОС РВ (RSX—11M)	8—16	
Электроника МС1211	Q							ДОС РВР (RSTS/E)		КН1811
СМ 1300.01	U							ИНМОС (UNIX)		1802/1804
СМ 4	U									
СМ 1600	U			—						
СМ 1420	U,M		+	—	+			RSX—11M/PLUS	до 64	К1804
Электроника МС1212	Q	1920		+				IAS		КН1811
Электроника 79	U,M			+			KERNEL USER SUPERVISOR	UNIX		

Примечания: * Q — магистраль «МПИ» (совмещенные линии адреса/данных;
U — магистраль «Общая шина» (раздельные линии адреса (18 разр.) и данных);
M — магистраль «MASSBUS» (22-разрядная адресная подшина;

** О командах EIS, FIS и FPP см. Приложение

*** Одноплатные микро-ЭВМ «Электроника МС 1201» входят в состав вычислительных комплексов ДВК-1 и ДВК-2.