В. А. ВИШНЯКОВ
А. А. ПЕТРОВСКИЙ

СИСТЕМНОЕ ОБЕСПЕЧЕНИЕ МИКРОЭВМ

Допущено Министерством народного образования БССР в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности 22.01 "Вычислительные машины, системы, комплексы и сети", 22.04 "Программное обеспечение вычислительной техники и автоматизированных систем"

Минск
"Вышэйшая школа"
1990

ББК 32.973.2-01я73
В55
УДК 681.3.06(075.8)

Рецензенты: кафедра вычислительной техники Ленинградского института точной механики и оптики; доктор технических наук, профессор кафедры вычислительной техники Киевского политехнического института *Г.М. Луцкий*

Вишняков В.А., Петровский А.А.
В55 Системное обеспечение микроЭВМ: Учеб. пособие для вузов. – Мн.: Выш. шк., 1990. – 304 с.: ил.
ISBN 5-339-00154-7.

Рассматриваются вопросы логической организации микроЭВМ на базе микропроцессорных комплектов К580, К588, К1810, их система команд, прерывание, ввод-вывод, элементы программирования на языках ассемблера. Разбираются основы построения трансляторов, редакторов текста, компоновщиков, загрузчиков, отладчиков, а также простейших ОС. Рассматриваются основы моделирования на базе сетей Петри, системного обеспечения мультипроцессорных микроЭВМ, ПЭВМ, вопросы интеллектуализации микроЭВМ и построения ЭВМ пятого поколения.

Для студентов специальностей "Вычислительные машины, комплексы, системы и сети" и "Программное обеспечение вычислительной техники и автоматизированных систем", а также для аспирантов, инженерных и научных работников.

В $\frac{2404090000-053}{М304(03)-90}$ 31–90

ББК 32.973.2-01я73

ISBN 5 339-00154-7

ПРЕДИСЛОВИЕ

Широкое применение микроЭВМ в народном хозяйстве предъявляет к вузам страны высокие требования по подготовке специалистов. В последние годы вышло немало книг, посвященных аппаратным и программным вопросам построения микроЭВМ. Однако системное обеспечение микроЭВМ либо оставалось неосвещенным, либо отражалось частично.

В настоящем учебном пособии рассматриваются вопросы построения системного программного обеспечения современных микроЭВМ и организации вычислительных процессов в них. В соответствии с этим особое внимание уделяется следующим темам: программным моделям микроЭВМ, способам адресации памяти, системам команд, а также языкам ассемблера; принципам построения трансляторов, включая ассемблер и простейшие компиляторы; разбору методов построения редакторов текста, компоновщиков, загрузчиков, отладчиков, а также построению простейших операционных систем микроЭВМ; основам микропрограммирования для микроЭВМ, построенных на секционных микропроцессорах; организации вычислительных процессов микроЭВМ и средств их автоматизации проектирования; системным средствам современных персональных ЭВМ; созданию микроЭВМ с элементами искусственного интеллекта, а также перспективным направлениям в архитектуре, языках и программном обеспечении.

В основу пособия положены курсы лекций, прочитанных авторами в Минском радиотехническом институте, а также на факультетах повышения квалификации. Изложение материала поясняется примерами.

В пособии главы 1, 2 авторы написали совместно, 4, 7, 9–11 – В.А. Вишняков, 3, 5, 6, 8, 12 – А.А. Петровский.

Авторы выражают благодарность рецензентам – коллективу кафедры вычислительной техники Ленинградского института точной механики и оптики и доктору технических наук, профессору кафедры вычислительной техники Киевского политехнического института Г.М.Луцкому за ряд ценных замечаний, способствовавших улучшению учебного пособия.

При написании книги использованы материалы исследований, проведенных В.А. Вишняковым совместно с А.С. Папковым (§ 9.2–9.5) и с Н.И.Кузьмицким (§ 11.3, 11.5).

Все отзывы и пожелания просим направлять по адресу: 220048, Минск, проспект Машерова, 11, издательство "Вышэйшая школа".

Авторы

СПИСОК СОКРАЩЕНИЙ

АЛУ – арифметико-логическое устройство
АТД – абстрактный тип данных
БА – буфер адреса
БД – буфер данных
БИС – большая интегральная схема
БМУ – блок микропрограммного управления
БНФ – форма Бэкуса–Наура
БСВВ – базовая система ввода-вывода
БУП – блок управления процессом
БУФ – блок управления файлом
ВС – вычислительная система
ВУ – внешнее устройство
ДВК – диалоговый вычислительный комплекс
ДП – дескриптор процесса
ДС – диалоговая система
ЗУ – запоминающее устройство
ИИ – искусственный интеллект
КВВ – контроллер ввода-вывода
КОП – код операции
КШ – контроллер шины
ЛП – лингвистический процессор
МК – микрокоманда
МП – микропроцессор
МПП – микропрограммная память
НГМД (ГМД) – накопитель на гибких магнитных дисках
НР – накапливающий регистр
ОЗУ – оперативное запоминающее устройство
ОП – оперативная память
ОПП – область памяти процесса
ОС – операционная система
ОШ – общая шина
ПДП – прямой доступ к памяти
ПЗ – плавающая запятая
ПЗУ – постоянное запоминающее устройство
ПЛМ – программируемая логическая матрица
ПМ – процессорный модуль
ПП – прикладная программа
ППМ – пакет прикладных программ моделирования
ППП – пакет прикладных программ
ПЭВМ – персональная ЭВМ
РВ – реальное время
РОН – регистры общего назначения
РСП – регистр состояния процессора
СИРЗ – система интеллектуального решения задач
СК – счетчик команд
СМО – специализированное математическое обеспечение
СОС – специализированная операционная система
СПЗ – стек плавающей запятой
ССП – слово состояния процессора
СУБД – система управления базой данных
СУБЗ – система управления базой знаний
УП – управляющая память
УС – указатель стека
УУ – устройство управления

1. ЛОГИЧЕСКАЯ ОРГАНИЗАЦИЯ И ФУНКЦИОНИРОВАНИЕ МИКРОЭВМ

1.1. ВВЕДЕНИЕ В АРХИТЕКТУРУ МИКРОЭВМ

1.1.1. Основные понятия

Под архитектурой ЭВМ понимают общую конфигурацию ее основных компонентов (структуру) , главные их особенности, возможности, способы соединения и функционирование. На рис. 1.1. показана обобщенная структурная схема микроЭВМ. МикроЭВМ – законченная ЭВМ, состоящая из следующих компонентов: центрального процессора (ЦП), оперативной памяти (ОП), внешних устройств (ВУ) с интерфейсными блоками ввода и вывода, шины, с помощью которых названные устройства связаны друг с другом.

Ядром микроЭВМ является микропроцессор (МП) – функционально законченное электронное цифровое устройство, реализованное на одной или нескольких БИС и предназначенное для выполнения команд, хранимых в оперативной памяти. Основные параметры ЦП (разрядность слова, быстродействие, набор выполняемых команд, режимы адресации памяти, число программно-доступных регистров, схема обработки прерываний и др.) определяют характеристики всей микроЭВМ. Микропроцессор включает ряд взаимосвязанных составных элементов (среди них можно выделить арифметико-логическое устройство (АЛУ) с блоками регистров, микропрограммную память (МПП) и устройство управления (УУ)), связанных друг с другом внутренней магистралью для передачи микрокоманд, адресов микрокоманд и управляющих сигналов. ЦП формирует синхронизирующие и управляющие сигналы для всех компонентов микроЭВМ, осуществляет выборку команд и данных из памяти, их декодирование, выполняет арифметические, логические и другие операции, закодированные в командах, управляет вводом-выводом информации в микроЭВМ.

Другим важным компонентом микроЭВМ является память: оперативная память, реализованная на электронных запоминающих устройствах с произвольной выборкой информации, предназначенная для хранения макрокоманд и данных; внешняя память, в качестве которой чаще применяются накопители на гибких магнитных дисках (НГМД). В микроЭВМ обычной конфигурации два НГМД. Здесь хранится операционная система, пакеты прикладных программ (ППП), организуются банки данных и рабочие файлы.

В зависимости от конструктивного исполнения принято различать одноплатные, многоплатные и однокристальные микроЭВМ. В одноплатной микроЭВМ ее ЦП, ОП и ряд других компонентов размещены на одной плате (например, в микроЭВМ "Электроника С5"). В многоплатной микроЭВМ ее компоненты, например ЦП и ОП, располагаются на отдельных платах и находятся в общем корпусе с соответствующими органами управления, индикации и блоком питания. К данному классу микроЭВМ относятся "Электроника 60",

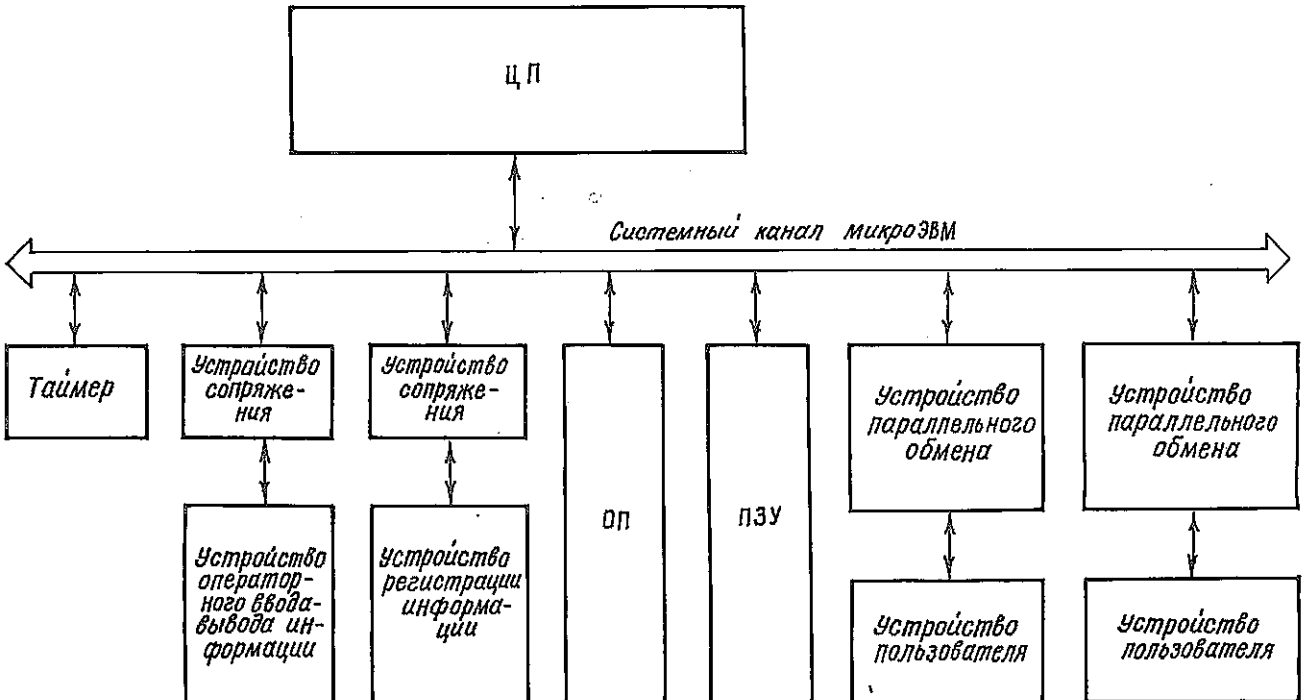

ЦП
Системный канал микроЭВМ
Таймер
Устройство сопряжения
Устройство сопряжения
ОП
ПЗУ
Устройство параллельного обмена
Устройство параллельного обмена
Устройство операторного ввода-вывода информации
Устройство регистрации информации
Устройство пользователя
Устройство пользователя

Рис. 1.1

а также программно совместимые с ней микроЭВМ серии ДВК. Наконец, однокристальная микроЭВМ представляет собой одну БИС, объединяющую ЦП, ОП ограниченного объема и контроллеры ввода-вывода, например БИС серии 1816, 1806 и др.

1.1.2. МикроЭВМ семейства "Электроника 60"

МикроЭВМ "Электроника 60" – одна из первых отечественных микроЭВМ, имеющая структуру с общей магистралью. Она предназначена для применения в системах автоматического и автоматизированного управления станками и технологическими процессами, в коммуникационных системах, в контрольно-измерительном оборудовании, для выполнения научно-технических расчетов и других целей. Важным достоинством этой микроЭВМ является ее программная совместимость с отечественными мини-ЭВМ "Электроника 100-25", СМ-3 и СМ-4, а также с зарубежными мини-ЭВМ семейства PDP-11 фирмы DEC.

Так, в микроЭВМ семейства "Электроника 60" реализованы структурные особенности мини- и микроЭВМ: организация ввода-вывода с использованием принципа обращения к памяти; коды условий перехода N , Z , V , C (знака, нуля, переполнения, переноса); отдельный указатель стека SP для вызова подпрограмм и организации прерываний; короткие относительные переходы; развитые механизмы адресации; многоуровневые системы прерываний; программные прерывания; структура памяти с побайтовой адресацией.

В микроЭВМ "Электроника 60" микропроцессор выполнен в виде набора БИС серии К581. ЭВМ имеет модульный принцип построения, т. е. все ее функциональные блоки выполнены как конструктивно законченные устройства (модули), связь между которыми осуществляется через канал обмена информацией. Пользователь сам определяет необходимую конфигурацию системы в зависимости от конкретного применения ЭВМ.

Центральный процессор включает блок резидентной ОП, которая расположена на одном модуле и имеет емкость 4 К 16-разрядных слов. Общая емкость ОП микроЭВМ определяется числом подключенных к каналу модулей памяти, обычно 4 К 16-разрядных слов или 16 К слов. Вместе с резидентной ОП общая емкость памяти машины может достигать величины 28 К слов.

Важнейшие особенности архитектуры (в том числе состав программно-доступных регистров) и система команд микроЭВМ "Электроника 60" нашли воплощение в одноплатной микроЭВМ "Электроника НЦ-8001Д", в которой в качестве элементной базы использован набор БИС серии К1801. На ее основе собираются диалоговые вычислительные комплексы (ДВК). В состав этой микроЭВМ входят однокристальный процессор, реализованный на БИС К1801ВМ1, системное постоянное запоминающее устройство (ПЗУ) емкостью 8 К байт, оперативное запоминающее устройство (ОЗУ) емкостью 56 К байт, интерфейсный блок байтового параллельного ввода-вывода данных и контроллер для НГМД ("Электроника ГМД-7012" или "Электроника ГМД-6022").

1.1.3. МикроЭВМ на базе МП К580ИК80

Микропроцессор К580ИК80 используется для построения 8-разрядных микроЭВМ (СМ-1800, К-20) и ПЭВМ (Роботрон 1715, Микроша, Ириша, Кор-

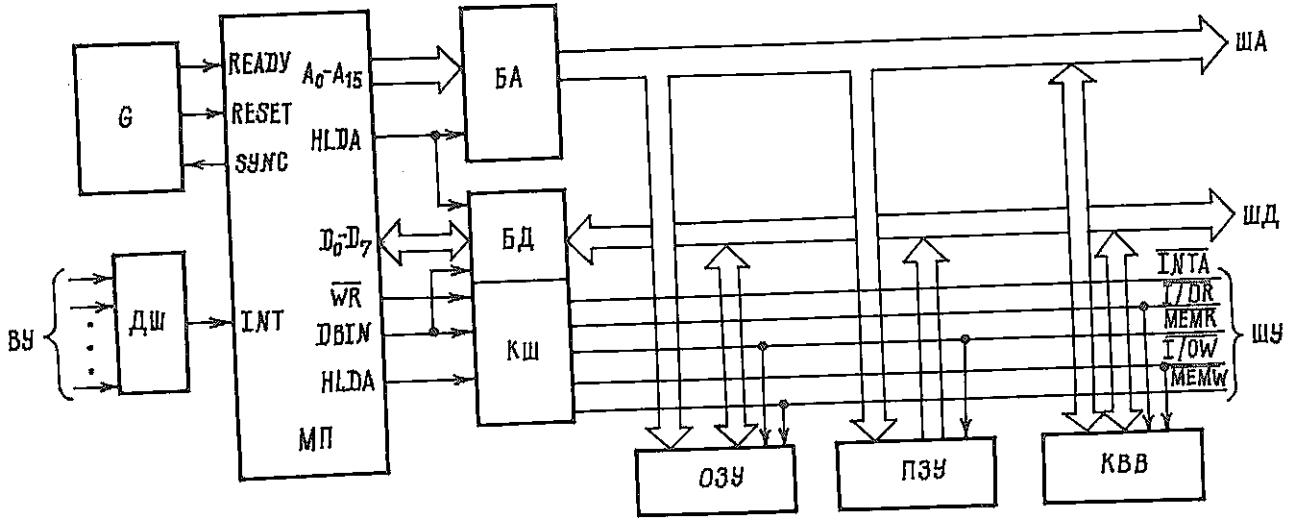

G
READY
RESET
SYNC
A0-A15
HLDA
D0-D7
WR
DBIN
HLDA
МП
БА
БД
КШ
ВУ
ДШ
INT
ША
ШД
ШУ
INTA
I/OR
MEMR
I/OW
MEMW
ОЗУ
ПЗУ
КВВ

Рис. 1.2

вет). На рис. 1.2 представлена структурная схема микроЭВМ на данном микропроцессоре. К МП подключается генератор G тактовых сигналов с сигналами готовности (READY), сброса (RESET), синхронизации (SYNC).

Для подключения оперативной и постоянной памяти (ОЗУ, ПЗУ), а также ВУ через контроллеры ввода-вывода (КВВ) организуется системная шина, состоящая из шин адреса (ША), данных (ШД) и управления (ШУ). Для формирования ША и ШД используются соответственно буфер адреса (БА) и буфер данных (БД). Для выработки управляющих сигналов применяется контроллер шины (КШ), который подключается к микропроцессору сигналами приема (DBIN), выдачи ($\overline{WR}$) и подтверждения захвата шины (HLDA). Последний сигнал используется для управления буферами адреса и данных. Поскольку буфер данных является двунаправленным, он еще управляется сигналом приема (DBIN). КШ вырабатывает следующие управляющие сигналы: подтверждения прерывания ($\overline{INTA}$), который выдается в ответ на сигналы прерывания, пришедшие на вход INT через дешифратор (ДШ) от ВУ; чтения и записи ВУ ($\overline{I/OR}$, $\overline{I/OW}$), чтения и записи памяти ($\overline{MEMR}$, $\overline{MEMW}$).

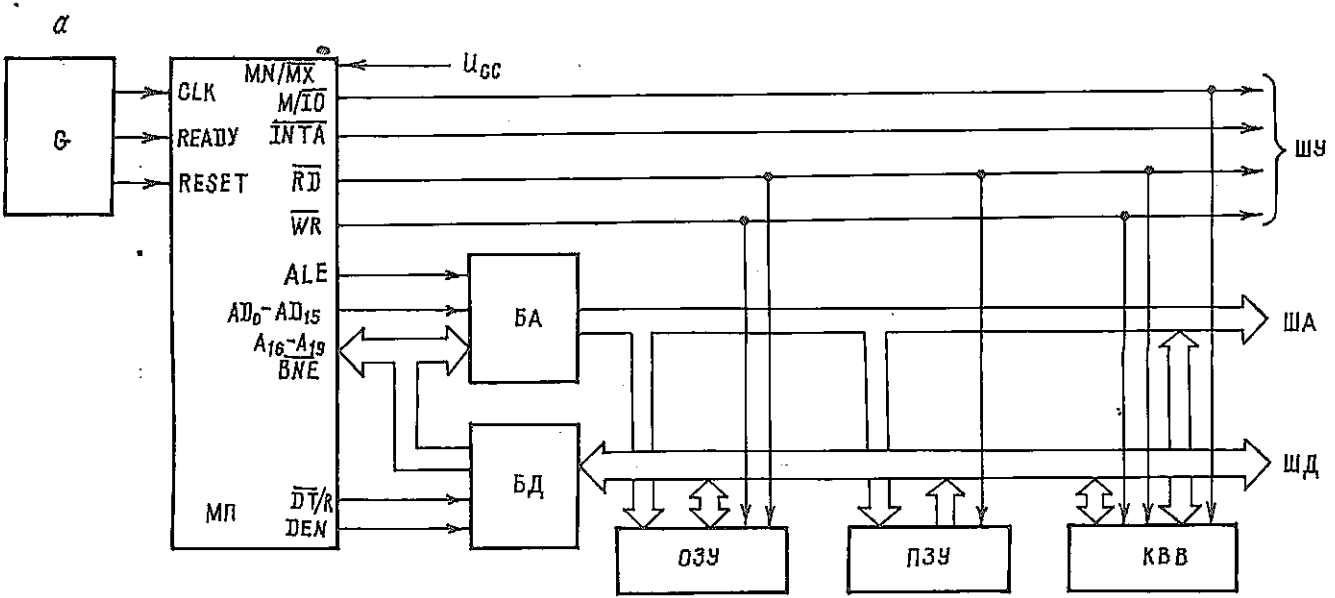
Блок ОЗУ подключается к шине адреса, данных и чтения-записи памяти; блок ПЗУ – к шинам адреса, данных (только на чтение), сигналу чтения памяти; блок КВВ – к шинам адреса, данных и сигналам чтения-записи ВУ.

МикроЭВМ работает следующим образом. Выполняемая программа может храниться как в ОЗУ, так и в ПЗУ, откуда она выбирается покомандно. Команда выполняется за 1–5 циклов. Каждый цикл включает 3–5 тактов (Т1–Т5). В Т1 на ША выдается код адреса, на ШД – код состояния МП. В такте Т2 МП анализирует признак подтверждения останова HLDA и сигнал готовности READY и соответственно переходит в состояния подтверждения останова или ожидания (при наличии указанных сигналов). В состоянии Т3 действия МП определяются типом машинного цикла. В цикле выборки команды МП интерпретирует код на ШД как код операции. В циклах чтения из памяти/ВУ, записи в память/ВУ осуществляется обмен байтом между памятью/ВУ по шине данных. Состояния Т3, Т4 используются для выполнения однобайтовой команды и являются нетипичными. Поэтому после Т3 МП может переходить к выполнению следующего цикла команды (если она больше одного байта).

1.1.4. МикроЭВМ на базе МП К1810ВМ86

Микропроцессор К1810ВМ86 используется для построения 16-разрядных микроЭВМ в однопроцессорном (минимальном MN/$\overline{MX}$=U_{CC}) и мультипроцессорном (максимальном MN/$\overline{MX}$=0) вариантах. В первом варианте МП предназначен для построения малых и средних ЭВМ (рис. 1.3, *а*). Управление шиной осуществляется сигналами самого МП: чтения ($\overline{RD}$), записи ($\overline{WR}$), обращения к памяти/периферийным устройствам (M/$\overline{IO}$), направления передач по шине данных (DT/$\overline{R}$), подтверждения прерывания ($\overline{INTA}$), стробирования адреса (ALE), данных ($\overline{BHE}$), управления шинными приемопередатчиками данных ($\overline{DEN}$).

Цикл шины МП состоит из четырех тактов Т1–Т4. Между тактами Т3 и Т4 с помощью сигнала готовности READY может быть введено произвольное


а
G
CLK
READY
RESET
МП
MN/MX
M/IO
INTA
RD
WR
ALE
AD0-AD15
A16-A19
BHE
DT/R
DEN
Ucc
БА
БД
ОЗУ
ПЗУ
КВВ
ШУ
ША
ШД

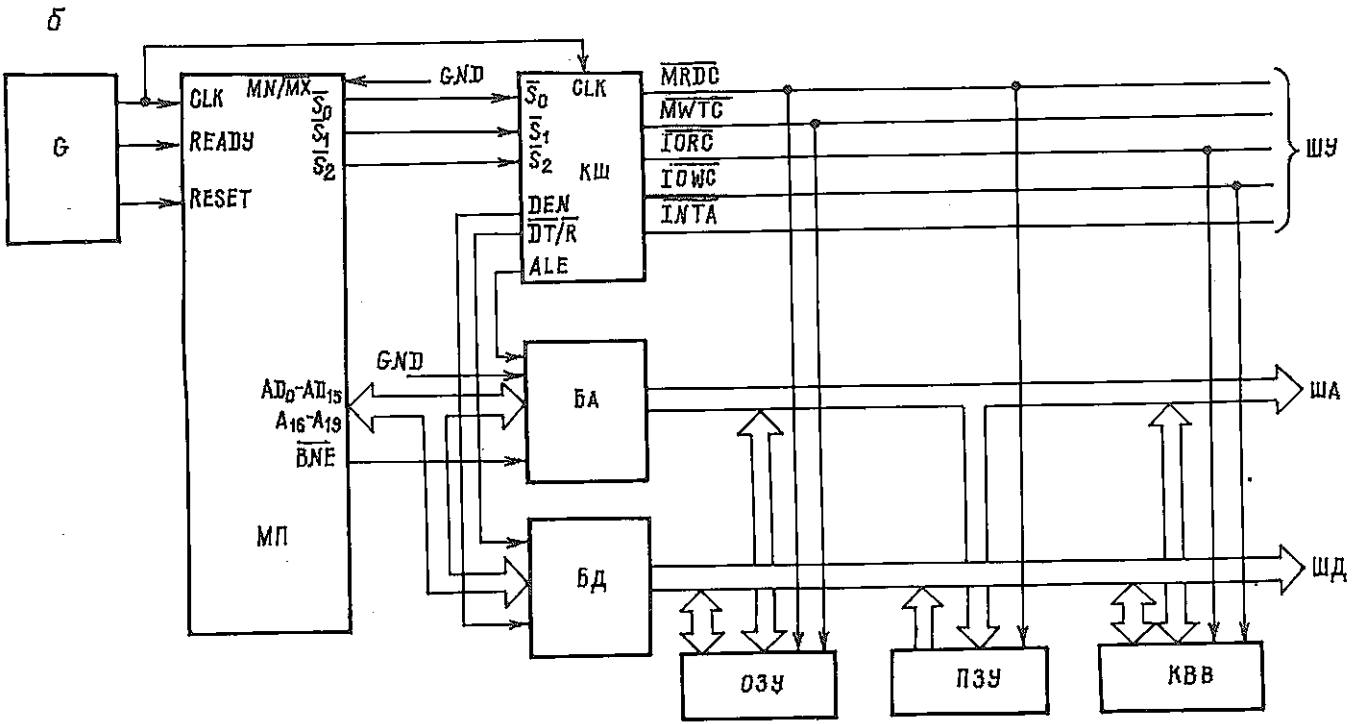

б
Г
CLK
READY
RESET
MN/MX
S0
S1
S2
GND
S0
S1
S2
CLK
КШ
DEN
DT/R
ALE
MRDC
MWTC
IORC
IOWC
INTA
ШУ
GND
AD0-AD15
A16-A19
BHE
МП
БА
БД
ША
ШД
ОЗУ
ПЗУ
КВВ

Рис. 1.3

число состояний ожидания для синхронизации МП с памятью или периферийными устройствами. В такте Т1 МП выдает на линии AD_{0-19} адрес и формирует сигнал ALE, по которому адрес выдается на адресный регистр. В тактах Т2–Т4 происходит чтение-запись данных. Максимальный режим допускает многопроцессорные конфигурации и работу с сопроцессорами. При этом управляющие сигналы вырабатываются КШ (рис. 1.3, *б*).

В такте Т2 в максимальном режиме на линии S_0–S_2 выдается код состояния, определяющий тип цикла шины. Кроме того, информация состояния в тактах Т2–Т4 выдается на линиях S_3–S_7 (A_{16}–A_{19}).

В тактах Т2–Т4 осуществляется прием или передача данных МП – память/периферийные устройства. Сигнал READY является сигналом подтверждения адресуемой памяти/порта (регистра) ввода-вывода о том, что они закончили передачу данных. Уровень сигнала READY проверяется в начале такта Т3. Если READY=0, то МП переходит в состояние ожидания, вырабатывая холостые такты TW.

В максимальном режиме тристабильные линии M/$\overline{IO}$, DT/$\overline{R}$, DEN используются для вывода в начале машинного цикла сигналов состояния S_0–S_2, которые подаются на КШ для выработки сигналов управления: чтения-записи памяти $\overline{MRDC}$, $\overline{MWTC}$ и ВУ– IORC, $\overline{IOWC}$ (см. рис. 1.3, *б*). Сигнал $\overline{WR}$ используется для блокировки шины, становясь выходным сигналом $\overline{LOCK}$. В этом режиме процессоры, входящие в систему, могут разделять ресурсы через системную шину, используя специальную БИС – арбитр шины.

1.2. ПРОГРАММНЫЕ МОДЕЛИ МИКРОЭВМ

1.2.1. Программная модель микроЭВМ "Электроника 60"

Составляя программу для микроЭВМ в машинных кодах или на языке ассемблера, программист абстрагируется от многообразия элементов МП и имеет дело лишь с системой команд и ограниченным числом его регистров, называемых программно-доступными. При этом никакие другие элементы микроЭВМ, кроме программно-доступных, не находят отражения в программах, написанных в кодах машины или на ассемблере.

Рассмотрим теперь особенности программных моделей отечественных микроЭВМ на основе МП К580ИК80, МП К1810ВМ86, а также микроЭВМ семейства "Электроника 60".

Программно-доступными в микроЭВМ семейства "Электроника 60" являются следующие функциональные узлы (рис. 1.4): АЛУ, восемь регистров общего назначения (РОН), регистр состояния процессора (РСП). Любое устройство ввода или вывода, или АЛУ может поместить информацию в магистраль, называемую общей шиной, и, наоборот, каждому из них доступна информация из магистрали. Таким образом, общая шина (ОШ) является эффективным средством передачи информации между функциональными устройствами, подсоединенными к ней.

В ЦП имеется восемь 16-разрядных регистров общего назначения, пронумерованных, начиная с нуля: R0–R7. Эти регистры могут использоваться как аккумуляторы, указатели адресов ячеек ОП, указатели стека (УС), индексные регистры. У регистров R6 и R7 есть, кроме того, специальное назначение.

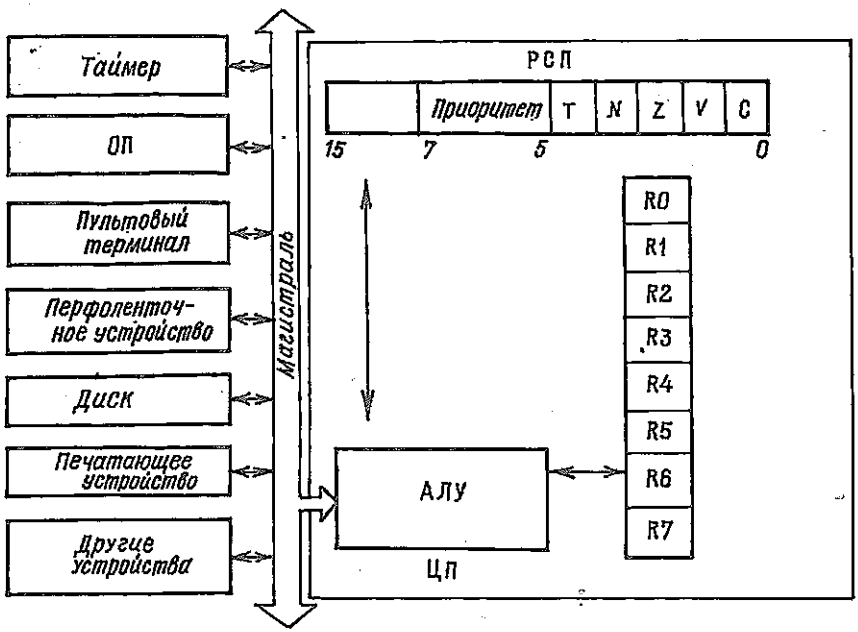


Рис. 1.4

Регистр R6 используется как аппаратно-реализованный указатель стека (SP). При записи в стек слова (стек может располагаться в любой области ОП) процессор уменьшает содержимое УС на два (длину слова в байтах) и затем записывает это слово по адресу, содержащемуся в УС. При чтении слова из стека содержимое УС увеличивается на два.

Регистр R7 выполняет функцию программного счетчика или счетчика команд (СК) – (PC). СК всегда содержит адрес очередной команды, подлежащей выполнению. После чтения из ОП очередной команды содержимое СК увеличивается на два (длину команды в байтах), определяя тем самым адрес следующей команды. Если команда состоит из двух (или трех) слов, то после чтения из ОП каждого очередного слова команды содержимое СК увеличивается на два, автоматически формируя адрес следующего слова.

Слово состояния процессора (ССП) (Processor Status Word–PSW) хранится в РСП и содержит информацию о текущем состоянии процессора. Разряд приоритета процессора P (7-й бит ССП) находится в состоянии 0 или 1. В последнем случае внешние устройства могут вызывать прерывание текущей программы. При организации ввода-вывода без прерываний программы разряд P должен быть равен 0.

T-разряд (4-й бит ССП) используется для отладки программы. Если он равен 1, то после завершения выполнения текущей команды произойдет прерывание программы с вектором 14 и из ячейки 16 ОП в регистр ССП будет занесено новое ССП, т. е. при выборке из стека нового ССП и занесении его в регистр ССП может устанавливаться или очищаться разряд T, что удобно использовать в отладочных программах.

Разряды 3–0 ССП содержат коды условия, которые отражают признаки результата выполнения очередной команды. Установка кода условий в соответствующее состояние выполняется командами обработки данных (арифметическими, логическими, сдвига и т. д.). Биты кода условий обозначаются буквами N , Z , V , C. Они характеризуют результат операции следующим образом: N (Negative) принимает значение 1, если результат отрицателен, т. е. если 15-й бит результата в случае обработки слов или 7-й бит в случае обработки байтов равен 1; Z (Zero) принимает значение 1, если результат (без учета переполнения) равен нулю, т. е. биты 15–0 результата в случае обработки слова или биты 7–0 при обработке байтов равны нулю; C (Carry) принимает значение переноса из старшего бита (т. е. из 15-го бита при обработке слов или 7-го бита при обработке байтов); V (over flow) принимает значение 1, если имеет место переполнение. Подчеркнем, что на коды условий воздействуют главным образом арифметические и логические команды. Неиспользуемые поля PSW оставлены для будущих расширений.

Минимальной адресуемой единицей ОП является байт. Для передач адресов в канале ЭВМ используется 16 линий данных – адресов. При этом возможно обращаться к 32 К 16-разрядным ячейкам ОП.

Слово ОП имеет длину 16 разрядов и состоит из двух байтов. Нумерация разрядов в слове производится от 15 до 0, в байте – от 7 до 0. Адрес слова ОП всегда четный. Слово делится на старший и младший байты. Допустимый диапазон 16-разрядных адресов (от 000000Q до 177777Q) обеспечивает доступ к 32 768 словам (32 К слова) или 65 536 байтам (64 К байт). Адреса байтов могут быть как четными, так и нечетными.

Младшие ячейки ОП с адресами 000000Q–000376Q зарезервированы под векторы прерываний и использовать их для других целей не рекомендуется. Адреса в диапазоне 160000Q–177776Q зарезервированы для регистров внешних устройств, поэтому максимальный объем реальной памяти равен 28 К 16-разрядных слов.

В микроЭВМ числа и нечисловая информация представляются совокупностью двоичных разрядов. При этом число двоичных разрядов определяет формат данных.

Данные делятся на три группы: логические коды, числа с фиксированной запятой и числа с плавающей запятой (ПЗ).

Логические коды могут размещаться в отдельных байтах и в словах. Для их представления используются все разряды: для байта – 7–0, для слова – 15–0. Логическими кодами могут быть числа без знака, которые имеют диапазон представления от 000Q до 377Q для байта и от 000000Q до 177777Q для слова.

Число с фиксированной запятой может занимать байт или слово. Знак в случае байта содержится в 7-м разряде, а в случае слова – в 15-м. Отрицательные числа представляются в дополнительном коде. Диапазон представления *чисел с фиксированной запятой*: а) для байта от –128D до +127D; б) для слова от –32768D до +32767D.

В микроЭВМ класса "Электроника 60" числа с ПЗ могут храниться в двух прилегающих друг к другу 16-битовых словах (рис. 1.5). Бит 15-й старшего слова предназначен для знака числа, следующие 8 бит обозначают порядок и отделяют старшую часть мантиссы от ее знака, а младшая часть хранится в сле-

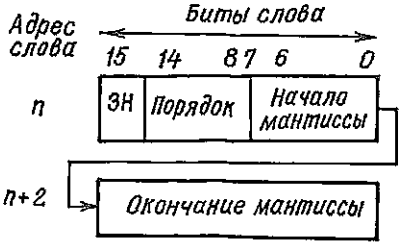


Рис. 1.5

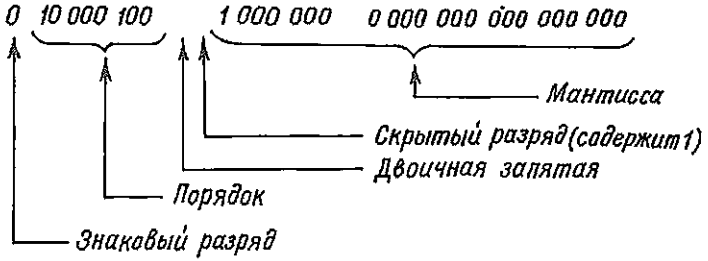


Рис. 1.6

дующем, старшем по адресу, 16-битовом слове. Мантисса всегда должна быть нормализована. Самый старший разряд (справа от двоичной запятой) является скрытым (избыточным) Поскольку этот бит должен быть равен 1, он не входит в число, что позволяет хранить мантиссу в 23 битах.

Порядок задается как смещенное 8-разрядное положительное число, т.е. если из порядка вычитается 128D–200Q , то результат представляет степень 2, на которую умножается мантисса, для получения истинного значения числа с ПЗ.

Приведем десятичный, восьмеричный и двоичный коды, соответствующие различным величинам порядка:

Десятичный код	Восьмеричный код	Двоичный код
+127	377	11 111 111
+1	201	10 000 001
0	200	10 000 000
–1	177	01 111 111
–128	000	00 000 000

Заметим, что если порядок равен 0Q (–128D), то операнд считается равным 0, 0, несмотря на то что знаковый разряд и мантисса не равны нулю. В этом случае происходит аппаратная очистка 32-разрядного слова. Например, число 12D=1100B=2^4D * 0 , 11B в форме с ПЗ в двоичном коде формата F имеет вид, представленный на рис. 1.6.

1.2.2. Программная модель микроЭВМ на базе МП К580ИК80

Модель включает регистры микропроцессора, ячейки памяти (ОЗУ, ПЗУ), а также порты ввода-вывода (рис. 1.7, *а*).

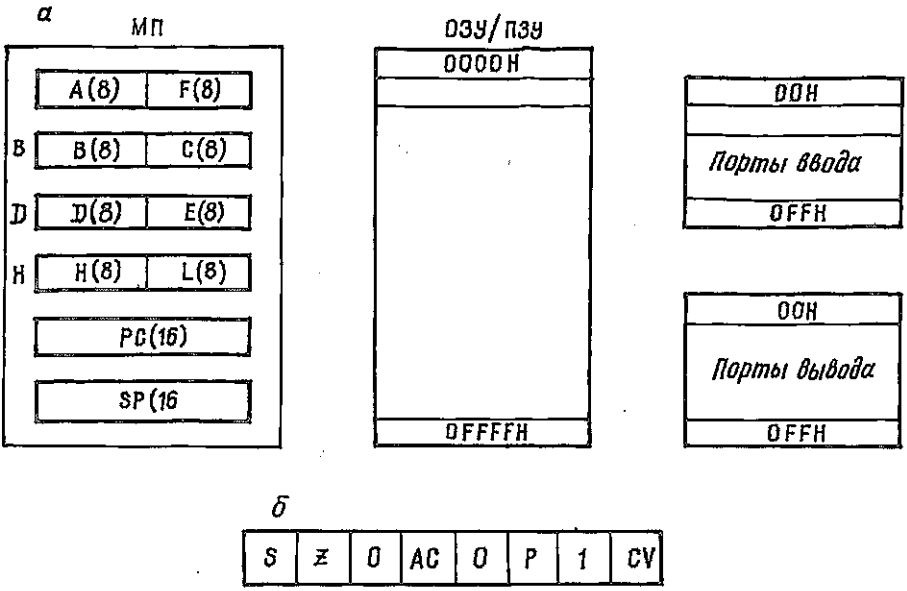


Рис. 1.7

Программная модель микропроцессора содержит десять регистров: 8-разрядный аккумулятор, шесть 8-разрядных регистров общего назначения B , C , D , E , H , L , 16-разрядный счетчик команд PC (program counter), 16-разрядный указатель стека SP (stack pointer) и регистр флажков F . Последний включает биты условий выполнения команд.

В аккумулятор всегда заносится результат операций, он хранит код первого операнда. Нумерация его разрядов, как и регистров B–L, производится слева направо D_7–D_0 . Поскольку в МП адрес имеет 16 разрядов, то для его хранения и модификации могут использоваться 16-разрядные регистры, получаемые за счет объединения B и C, D и E , H и L . Они обозначаются по имени первого регистра (B , D , H соответственно). Кроме того, аккумулятор и регистр образуют слово состояния программы PSW , которое может сохраняться в стеке при выполнении программ обработки прерываний.

Счетчик команд PC позволяет адресовать память объемом $2^{16}=65536$ байт и содержит адрес байта команды, которая будет выбираться из памяти для выполнения. После чтения очередной команды содержимое PC увеличивается на ее длину, равную одному, двум или трем байтам.

Регистр SP всегда содержит адрес верхней ячейки стека, поэтому он указывает (является указателем) на последний занятый байт. Физически стек представляет собой часть ОЗУ, запись в которую производится (как и считывание) словом (двумя байтами). При записи слово пишется в два последовательных байта в направлении младших адресов, а указатель уменьшается на два. При считывании слово выбирается по адресу, начиная с указателя, после чтения содержимое регистра SP увеличивается на два. Например, если до записи SP=1222, то запись будет производиться в ячейки с адресами 1221 и

1220, после чего SP установится на последнюю занятую ячейку с адресом 1220. Считывание будет происходить из ячеек с адресами 1220, 1221, после чего SP установится на ячейку с адресом 1222.

Пять разрядов регистра флажков F хранят биты признаков, которые устанавливаются после выполнения последней команды. К ним относятся флажки знака S , нуля Z , вспомогательного (межтетрадного) переноса AC, четности P и переноса CY (рис. 1.7, *б*).

Флажок знака S=1 , если результат последней команды отрицательный (в седьмом разряде аккумулятора 1); флажок нуля Z=1, если результат последней команды равен нулю (все биты аккумулятора нулевые); флажок AC=1, если при выполнении последней команды возник перенос из младшей тетрады в старшую (из D_3 в D_4); флажок P=1, если после выполнения команды число единичных битов аккумулятора четно, и P=0, если нечетно. Наконец, флажок CY=1, если после выполнения команды возник перенос из старшего разряда аккумулятора, в противном случае CY=0.

Биты S , Z , P , CY используются в командах передачи управления, бит AC – в команде десятичной коррекции при обработке двоично-десятичных чисел.

Бит CY , кроме того, применяется при сложении и вычитании многобайтовых чисел. Для беззнаковых чисел диапазон представления 0–255, для знаковых – –128–127.

Программная модель памяти – это пространство последовательно расположенных ячеек с адресами от 0000H до FFFFH. При этом данное адресное пространство разделяется между ОЗУ и ПЗУ. За одно обращение к памяти МП может записать или прочитать один байт.

Программная модель устройств ввода-вывода представляется набором соответствующих портов (см. рис. 1.7, *а*). При этом как для ввода, так и для вывода используется 256 портов с адресами от 00H до FFH . Заметим, что порты ввода и вывода могут иметь одинаковые адреса, выставляемые на ША, но они различаются при обращении к ВУ посредством сигналов $\overline{\text{I/OR}}$ $\overline{\text{I/OW}}$ соответственно. Для передачи данных между МП и ВУ используются команды IN и OUT . Команда IN идентифицирует адрес порта на ША, и его содержимое будет перенесено в аккумулятор МП. Команда OUT задает адрес порта на ША, и значение аккумулятора передается по соответствующему адресу. Обычно одному устройству назначается несколько портов ввода и вывода данных. Часть из них используется как управляющие, а часть как информационные.

1.2.3. Программная модель МП 1810ВМ86

Программная модель МП 1810ВМ86 включает три группы программно-доступных 16-битовых регистров по четыре в каждой, а также 16-битовый регистр условий F и 16-разрядный счетчик команд IP (рис. 1.8, *а*), который является программно-доступным лишь для команд перехода.

Регистры AX, BX, CX , DX (первая группа) являются регистрами общего назначения (РОН). Они могут участвовать в арифметических операциях без ограничений. Эти регистры имеют следующие мнемонические обозначения: AX – аккумулятор; BX – база; CX – счетчик; DX – данные. При этом старший байт регистров имеет имя H, младший L (AH, AL, BH, BL, CH, CL, DH, DL).

а

РОН				
AX	AH	*Аккумулятор*	AL
BX	BH	*База*	BL
CH	CH	*счетчик*	CL
DX	DH	*Данные*	DL

SP	*(указатель стека)*
BP	*(указатель базы)*
SI	*(индекс-регистр операнда)*
DI	*(индекс-регистр результата)*

IP	*(счетчик команд)*
F	*(биты условий)*

CS	*(сегмент программ)*
DS	*(сегмент данных)*
SS	*(сегмент стека)*
ES	*(сегмент экстракодов)*

б

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
*	*	*	*	OF	DF	IF	TF	SF	ZF	*	AF	*	PF	*	CF

в ОЗУ/ПЗУ

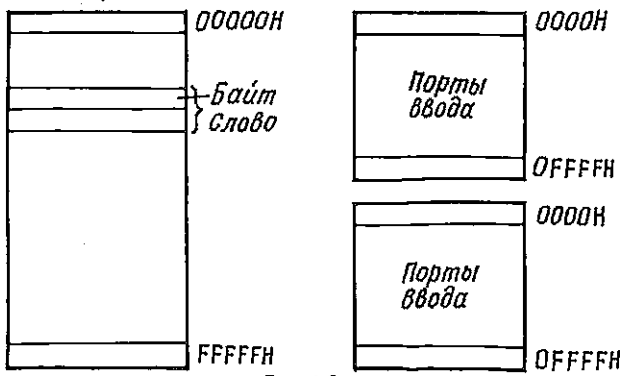


Рис. 1.8

Регистры SP, BP, SI, DI (вторая группа) называются указательно-индексными. Они содержат смещения для вычисления адресов. Эти регистры делятся на подгруппу указательных регистров P(SP, BP) и подгруппу индексных регистров I(SI, DI). Разница заключается в том, что смещение в регистрах P рассматривается относительно номера сегмента стека, а смещение в регистрах I – относительно текущего номера сегментов данных DS, ES. Обозначение этих регистров следующее: SP – указатель стека; BP – указатель базы, SI – индексный регистр операнда; DI – индексный регистр результата.

Третья группа включает регистры сегментов, содержимое которых представляет базовый адрес сегмента. Число в регистре CS определяет сегмент программы, расположенной выше базового адреса объемом до 64 К байт. Число в регистре DS представляет номер базового адреса данных, относительно которого они располагаются.

Сегмент стека – это 64 К байт памяти, задающие память стека и расположенные выше адреса, указанного в регистре SS . Основные операции со стеком – чтение и запись информации, используемые при выполнении команд CALL , операций обработки прерываний и возврата из подпрограмм.

Экстракодовый сегмент – это 64 К байт памяти, распюложенные выше адреса, указанного в регистре ES ; он создается как дополнительный сегмент данных и обычно используется в строковых операциях для хранения массива приемника (массив источника задается через DS).

Если последовательные адреса загрузить в несколько сегментных регистров (например, 1200H в ES, 1201H в DS, 1202H в SS), то к переменной по адресу 12038H можно будет обратиться с использованием регистра ES и смещения 38H, регистра DS и смещения 28H и регистра SS и смещения 18H. В этом случае сегменты будут перекрываться. Если же в качестве базовых взять адреса 1200H, 3200H, 5200H, то ни один байт памяти не будет доступен одновременно с применением всех сегментных регистров, т. е. такие сегменты в памяти не пересекаются. Имеется 16 непересекающихся сегментов памяти и 64 К байт пересекающихся.

Для отображения информации о состоянии процесса вычисления и для опроса состояний МП используются биты условий. К ним относятся: AF – вспомогательный перенос, CF – перенос, DF – направление, IF – разрешение прерываний, OF – переполнение, PF – четность, SF – знак, TF – трассировка, ZF – нуль (рис. 1.8, *б*). Биты AF, CF, PF, SF, ZF аналогичны соответствующим битам МП КР580ИК80А и отображают состояние процессора после выполнения очередной арифметической или логической операции. К новым битам условий относятся: DF – направление обработки байт при выполнении действий над строками (автоувеличение или автоуменьшение адреса); IF – бит, разрешающий или запрещающий внешние прерывания; OF – бит, отражающий наличие переполнения при завершении арифметических операций над числами со знаками; TF – бит, переводящий МП в пошаговый режим для отладки. На рис. 1.8, *б* показаны флажки условий в порядке их запоминания в стеке.

МП использует 20-разрядную ША и имеет доступ к 2^{20} байтам (1 М байт). Адресуемая единица памяти – один байт (рис. 1.8, *в*). Слово включает два байта, может начинаться с четного или нечетного адреса (во втором случае требуется два цикла обращения к памяти). Доступ к байту или слову осуще-

ставляется, если адресуемый элемент находится в пределах одного из четырех текущих сегментов по 64 К байт, задаваемых регистрами сегментов. В пределах сегментов байт адресуется с помощью 16-разрядного адреса (смещения).

Адресация памяти производится с использованием адреса сегмента (базовый адрес) и адреса смещения. МП автоматически производит сложение смещения со сдвинутым на четыре разряда влево адресом сегмента и формирует 20-разрядный физический адрес. Например, если адрес сегмента 1230Н, а смещение 3400Н, то адресуемый байт будет иметь адрес 12300Н + 3400Н= =15700Н.

В качестве модели периферийных устройств используются порты ввода-вывода. Существует 65 536 адресов портов ввода и 65 536 адресов портов вывода, поскольку порты адресуются 16-разрядными адресами (см. рис. 1.8, *в*). При этом регистры сегментов не требуются. Операции передачи данных инициируются по командам IN и OUT. По команде IN номер порта выдается на адресную шину. Как только байт (слово) поступает на ШД, он передается в аккумулятор AL(AX). По команде OUT номер порта выдается на ША, а данные из регистра AL (байт) или AX (слово) поступают на ШД.

1.3. ВИДЫ АДРЕСАЦИИ ПАМЯТИ

1.3.1. Режимы адресации микроЭВМ семейства "Электроника 60"

Часть архитектуры процессора, определяющая структуру оперативной памяти и доступ к ее элементам (ячейкам), называется *системой адресации*. От эффективности системы адресации и тесно связанных с ней средств управления ходом программы зависит и эффективность работы микроЭВМ в целом. Она достигается главным образом сокращением числа обращений к оперативной памяти и более компактным представлением адресной информации в программах.

Таблица 1.1

Формат команды	Адресация	Наименование режима	Описание	Время выборки операнда, мкс			
				источник		приемник	
				слово	байт	слово	байт
R_n	0N	Регистровый	ORN:=n	0	0	0	0
$(R_n)+$	2N	Автоинкрементный	EA:=(R_n) Inc (R_n)	1,6	1,2	2,4	2,0
$-(R_n)$	4N	Автодекрементный	Decr(R_n) EA:=R_n	2,4	2,0	3,2	2,8
$X(R_n)$	6N	Индексный	EA:=R_n+m(PC) PC:=PC+2	4,8	4,6	5,6	5,2

Примечание. Здесь и в дальнейшем приняты обозначения: ORN – операнд; R_n – n-й регистр общего назначения; EA – исполнительный адрес (адрес операнда); $m(\cdot)$ – косвенная адресация; N – используемый регистр; X – индексное слово.

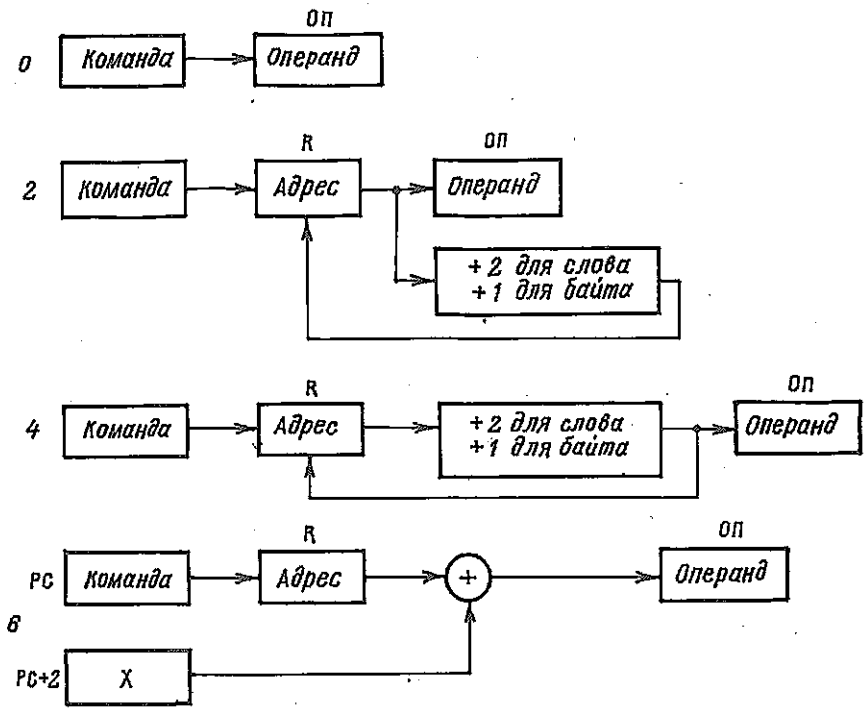


Рис. 1.9

Способы адресации микроЭВМ позволяют не только значительно расширить адресное пространство при жестко ограниченной длине команды, но и получить существенную экономию памяти, необходимой для размещения программ, а в ряде случаев и времени. Более того, способы адресации лежат в ос-

Таблица 1.2

Формат команды	Адресация	Наименование режима	Описание	Время выборки операнда, мкс			
				источник		приемник	
				слово	байт	слово	байт
$@R_n$	1N	Регистровый косвенный	$EA:=R_n$	1,6	1,2	2,4	2,0
$@(R_n)+$	3N	Автоинкрементный косвенный	$EA:=m(R_n)$ $R_n:=R_n+2$	4,0	3,6	4,8	4,8
$@-(R_n)$	5N	Автодекрементный косвенный	$R_n:R_n-2$ $EA:=m(R_n)$	4,8	4,4	5,6	5,6
$@X(R_n)$	7N	Индексный косвенный	$EA:m(R_n+m(PC))$ $PC:PC+2$	7,2	6,8	7,6	8,0

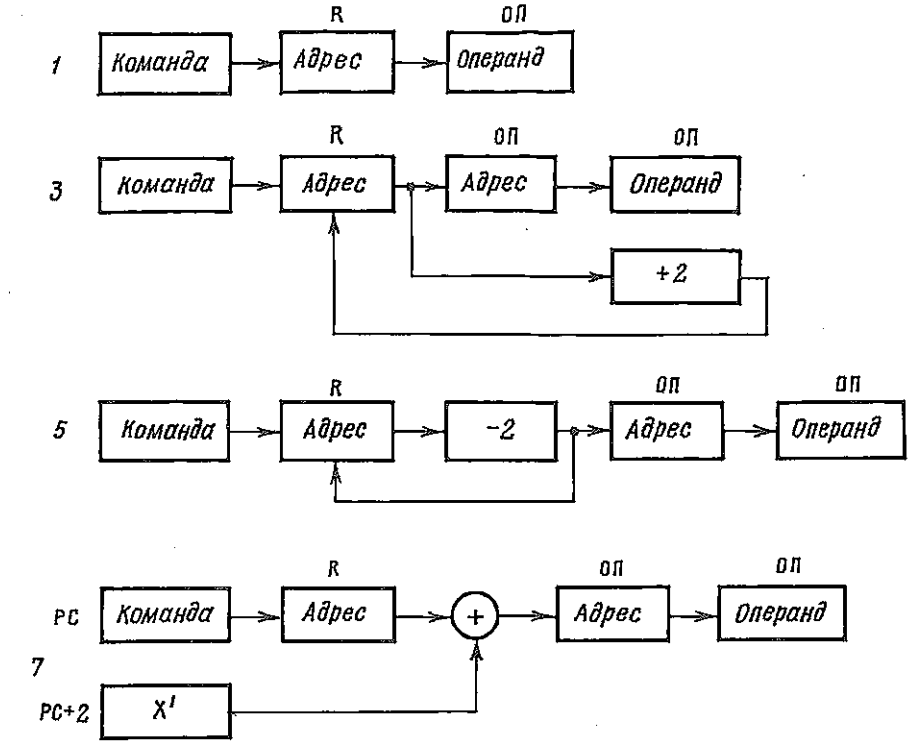


Рис. 1.10

нове современной техники программирования (перемещаемости программ, автоматического распределения и защиты памяти, мультипрограммной работы, структурирования данных).

В ЭВМ "Электроника 60" используются различные приемы и механизмы

Таблица 1.3

Формат команды	Адресация	Наименование режима	Описание	Время выборки операнда, мкс			
				источник		приемник	
				слово	байт	слово	байт
#E	27	Непосредственный	EA:=PC PC:=PC+2	1,6	1,6	2,4	2,8
@#E	37	Абсолютный	EA:=*m*(PC) PC:=PC+2	4,0	3,6	4,8	4,8
E	67	Относительный	EA:=*m*(PC) PC:PC+2; EA:=EA+PC	4,8	4,6	5,6	5,2
@E	77	Косвенно-относительный	EA:=*m*(PC) PC:= PC+2 EA=*m*(EA+PC)	7,2	6,8	7,6	8,0

адресации: прямая, косвенная и адресация посредством счетчика команд.

Основные режимы прямой адресации (режимы 0, 2, 4, 6) даны в табл. 1.1.

Действие режимов прямой адресации показано на рис. 1.9.

Если бит 3 команды установлен, то значит, что специфицирована косвенная адресация, включающая четыре основных режима (1, 3, 5, 7) (табл. 1.2).

Диаграммы, графически представляющие действие режимов косвенной адресации, приведены на рис. 1.10. В общем случае счетчик команд РС может применяться в любых стандартных режимах адресации микроЭВМ семейства "Электроника 60". Использование РС дает преимущества при работе в случае

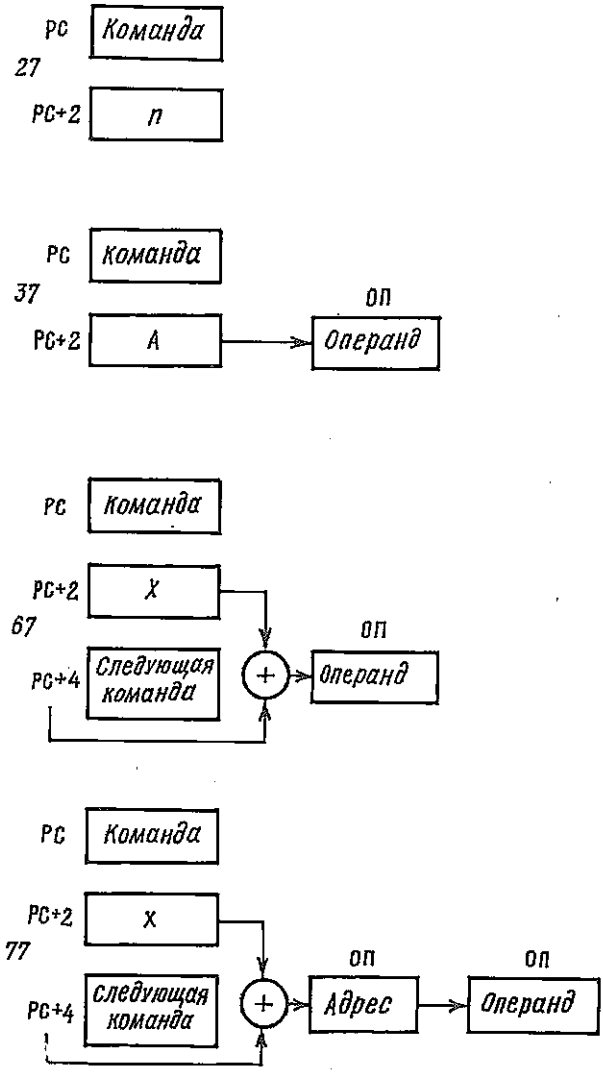


Рис. 1.11

позиционно независимого кода и неструктурированных данных. Это позволяет писать позиционно независимые программы (так называемые PIC-программы). В табл. 1.3 приведены режимы, использующие PC. Их графическая иллюстрация дана на рис. 1.11.

1.3.2. Виды адресаций МП К580

В МП К580 используются неявная, регистровая, непосредственная, прямая, регистровая косвенная и стековая адресации.

При неявной адресации адрес операнда неявно задается в коде операции. В МП первый операнд всегда находится в аккумуляторе, а второй адресуется непосредственно, прямо или косвенно. Примерами команд с неявной адресацией являются команды сдвига RAL, RAR , десятичной коррекции DAA.

Использование регистровой адресации предусматривает указание в поле операнда одного из регистров A, B, C, D, E, H, L , в котором находится один операнд (второй всегда в аккумуляторе). Примерами такой адресации являются команды арифметические ADD, логические ORA и т.д.

При непосредственной адресации операнд является частью выполняемой команды. Это может быть один байт – команда MVI B, 10H (в регистр B загружается код 10H) или два байта – команда LXI SP, 1000H (в указатель стека загружается адрес 1000H).

При прямой адресации памяти выполняемая команда включает адрес данных, участвующий в операции. Это либо адрес данных в памяти – 16 разрядов, либо порта ввода-вывода – 8 разрядов. Например, команда JMP 1000H осуществляет переход по адресу 1000H, задавая значение адреса памяти. Команда OUT 10H выводит содержимое аккумулятора в порт с номером 10H.

При косвенной регистровой адресации фактически используются два адреса. Первый адрес включен в состав команд и является адресом пары регистров, содержимое которых определяет второй адрес байта памяти. По этому адресу и расположен операнд. Например, команда LDAX B загружает аккумулятор байтом, адрес которого находится в паре регистров BC.

При стековой адресации используется указатель стека SP в командах PUSH, POP.

1.3.3. Виды адресации в МП К1810ВМ86

Кроме методов адресации, используемых в МП К580ИК80 (неявная, регистровая, непосредственная, прямая, регистровая косвенная и стековая) в МП К1810ВМ86, добавляется относительная адресация. Последняя характеризуется тем, что физический адрес операнда в памяти определяется как сумма базового адреса и смещения. В данном МП используется два вида относительной адресации: базовая и индексная, а также их комбинация.

В командах с двумя операндами правый (источник) может быть константой, тогда левый (приемник) – регистром либо ячейкой памяти. В других случаях один операнд находится в регистре, второй – в регистре или ячейке памяти.

Команды МП занимают в памяти от 1 до 6 байт. Первый байт команды содержит код операции. Обобщенный формат команды (рис. 1.12) включает по-

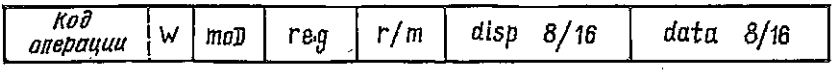


Рис. 1.12

ля адресации (второй байт), 8- или 16-разрядное смещение в команде (третий и четвертый байты), непосредственные 8- и 16-разрядные данные (пятый и шестой байты). Однобитовое поле W определяет разрядность операндов (при W = 0–8-разрядные, при W = 1–16-разрядные). Поля mod и r/m второго байта команды задают один из 24 режимов формирования относительного адреса операнда в сегменте данных или сегменте стека в соответствии с данными в табл. 1.4.

Если смещение однобайтовое, то происходит знаковое расширение до двух байт, при этом значение всех разрядов второго байта устанавливается равным значению старшего разряда исходного байта. Существуют четыре способа, с помощью которых можно обратиться к операндам, находящимся в памяти: 1) указание прямого смещения MOV REL, AL; 2) использование косвенного обращения через базовый регистр, содержимое которого суммируется с 8- или 16-разрядным смещением MOV ON[BX+4] , BL; 3) применение косвенного обращения через индексный регистр, содержимое которого суммируется со смещением MOV ON [DI+2], CL; 4) использование косвенного обращения через базовый и индексный регистры, содержимое которых суммируется со смещением MOX ON[BX + 2][DI], DL.

Таблица 1.4

Поле r/m	Поле mod							
	00		01		10		11	
	Смещение	Сегмент	Смещение	Сегмент	Смещение	Сегмент	W=0	W=1
000	BX+SI	данных	BX+SI+ +disp8	данных	BX+SI+ +disp16	данных	AL	AX
001	BX+DI	"	BX+DI+ +disp8	"	BX+DI+ +disp16	"	CL	CX
010	BP+SI	стека	BP+SI+ +disp8	стека	BP+SI+ +disp16	стека	DL	DX
011	BP+DI	"	BP+DI+ +disp8	"	BP+DI+ +disp16	"	BL	BX
100	SI	данных	SI+disp8	данных	SI+d16	данных	AH	SP
101	DI	"	DI+disp8	"	DI+d16	"	CH	BP
110	disp 16	"	BP+disp8	стека	BP+disp16	стека	DH	SI
110	BX	"	BX+disp8	данных	BX+disp16	данных	BH	DI

Рассмотрим подробнее содержимое второго байта команды. Поле метода адресации mod (2 бита) указывает, каким образом используется r/m (при mod = 11 операнд находится в регистре, иначе в памяти) (табл. 1.4). Поле регистра REG занимает 3 бита и указывает на 8- или 16-разрядный регистр, содержащий операнд. В поле r/m определяется положение операнда (в памяти или регистре) либо способ размещения операнда в памяти (совместно с полем mod). Эти поля автоматически формируются ассемблером при генерации кода. Адрес операнда вычисляется по содержимому полей mod и r/m.

Регистры BX и BP могут служить базовыми. Если регистр BX является базовым, то по умолчанию операнд находится в сегменте данных и для вычисления его физического адреса используется регистр DS. Если базовым является BP, то операнд находится в стеке и для вычисления адреса используется регистр SS. Если применяются базовый и индексный регистры, то операнд находится в сегменте, на который указывает базовый регистр. Если приводится только индексный регистр, то операнд будет находиться в сегменте данных.

Названные виды адресации позволяют при одном и том же коде операции, изменяя значение полей mod, r/m и REG, в качестве операндов использовать содержимое любого из РОН или памяти.

Форматы команд МП К1810ВМ86 зависят от типа команд, вида адресации, размера непосредственных данных и могут включать от одного до шести байт (рис. 1.13).

Однобайтовые команды задают операцию, выполняемую над содержимым регистра. Двухбайтовые команды являются группами команд регистр-регистр

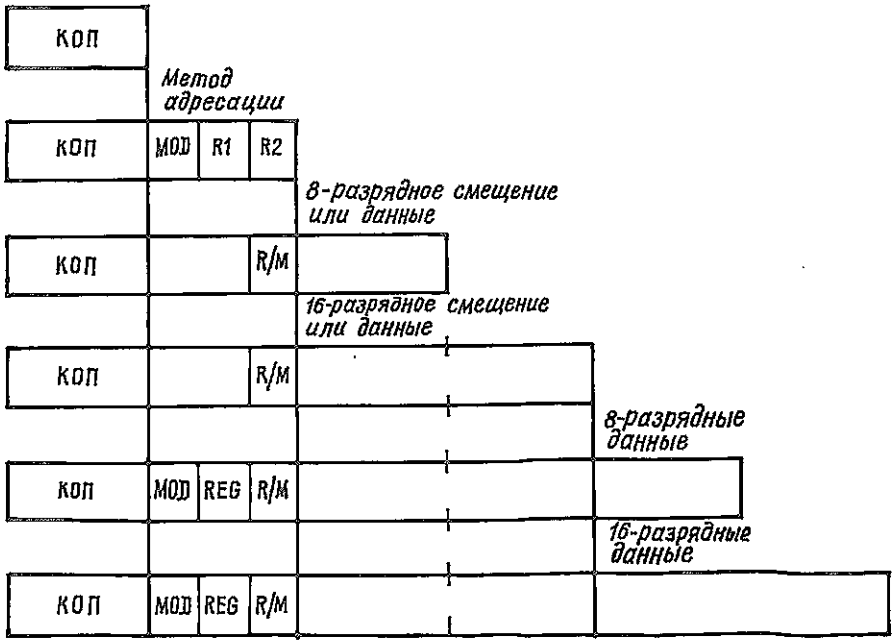


Рис. 1.13

и ввода-вывода. Второй байт определяет адреса регистра источника, регистр приемника и задает способ адресации.

Трех- и четырехбайтовые команды содержат в третьем (четвертом) байте либо 8(16)-разрядное смещение при обращении к памяти, либо 8(16)-разрядный непосредственный операнд. Пяти- и шестибайтовые команды содержат в третьем и четвертом байтах 16-разрядное смещение, а в пятом (шестом) байте – 8(16)-разрядный непосредственный операнд.

Рассмотрим на нескольких примерах использование видов адресации данного микропроцессора в его командах:

1) регистровый

MOV DX, AX; пересылка содержимого регистра AX в DX;

2) непосредственный

MOV AX, 5511; пересылка в регистр AX непосредственных данных;

3) прямой

MOV RR, AX; пересылка содержимого AX по адресу RR. Адрес берется из поля смещения команды;

4) косвенный (с использованием косвенного обращения через базовый или индексный регистр)

MOV CX, [BX]; (или MOV DX, [SI];) содержимое ячейки памяти, адресуемой косвенно через базовый BX (или индексный SI) регистр, пересылается в регистр CX (или DX);

5) базовый или индексный с использованием косвенного обращения к ячейке памяти через базовый или индексный регистр, содержимое которых суммируется со смещением

MOV DX, SN[BX]; (или MOV DX, SN[DI];) содержимое ячейки памяти, адрес которой задается базовым (или индексным) регистром плюс смещение SN, пересылается в DX;

6) базовый и индексный с использованием косвенного обращения к ячейке памяти, адресуемой суммой базового и индексного регистров

MOV DX,[BX][SI]; содержимое ячейки, адресуемой суммой BX, SI , пересылается в DX;

7) базовый и индексный со смещением с использованием косвенного обращения к ячейке памяти, адресуемой суммой базового индексного регистров и смещения

MOV DX, SN[BX][SI] содержимое ячейки, адресуемой суммой содержимого регистров BX, SI и значения SN , пересылается в DX .

Время вычисления эффективного адреса при обращении к памяти зависит от режима адресации (табл. 1.5).

Таблица 1.5

Адресация	Обозначение	Число тактов
Прямая	disp	6
Регистровая косвенная	[BX], [BP], [SI], [DI]	5
Базовая или индексная	[BX, BP, SI, DI] + disp	9
Базовая, индексная (без смещения)	[BP + SI], [BX + DI]	7
Базовая, индексная (со смещением)	[BP+DI] + disp, [BX+SI]+ disp [BX+DI]+disp, [BP+SI]+disp	11 12

1.4. СИСТЕМА КОМАНД МП КР580

В настоящее время в микроЭВМ в основном используются двух-, одно- и безадресные команды. Команды МП КР580ИК80А состоят из одного, двух или трех байт, расположенных в памяти последовательно (рис. 1.14). В однобайтовых командах задается код выполняемой операции и регистры. Двухбайтовые команды — это команды с непосредственным операндом или команды ввода-вывода, трехбайтовые команды содержат во втором и третьем байтах либо адрес памяти, либо непосредственные данные (LDA 1000H, LXI B, 2000H).

По функциональному назначению все команды можно разбить на следующие группы: пересылки, арифметические, логические, передачи управления и специальные (табл. 1.6).

Таблица 1.6

	Мнемоника	Операция	МЦ	МТ	#	Признаки
		Группа команд пересылки				
1	MOV R1,R2	(R1)←(R2)	1	5	1	Все признаки сохраняют свои значения
2	XCHG	(HL)↔(DE)	1	4	1	
3	SPHL	(SP)←(HL)	1	5	1	
4	MOV R,M	(R)←M(HL)	2	7	1	
5	MOV M,R	M(HL)←(R)	2	7	1	
6	LDAX RP'	(A)←M(RP)	2	7	1	
7	STAX RP'	M(PR)←(A)	2	7	1	
8	LDA A16	(A)←M(A16)	4	13	3	
9	STA A16	M(A16)←(A)	4	13	3	
10	LHLD A16	(HL)←M(A16)	5	16	3	
11	SHLD A16	M(A16)←(HL)	5	16	3	
12	MVI R,D8	(R)←D8	2	7	2	
13	LXI RP,D16	(RP)←D16	3	10	3	
14	MVI M,D8	M(HL)←D8	3	10	2	
15	PUSH RP*	M(SP-1)←(RPH) M(SP-2)←(RPL) (SP)←(SP)-2	3	11	1	
16	POP RP*	(RPL)←M(SP) (RPH)←M(SP+1) (SP)←(SP)+2	3	11	1	
17	XTLH	M(SP)↔(H) M(SP-1)↔(L)	5	18	1	

18	IN PORT	(A)←I(PORT)	3	10	2	
19	OUT PORT	O(PORT)←(A)	3	10	2	

Группа команд арифметических операций

20	ADD R	(A)←(A)+(R)	1	4	1	S,Z,AC,P,CY
21	ADC R	(A)←(A)+(R)+CY	1	4	1	S,Z,AC,P,CY
22	SUB R	(A)←(A)-(R)	1	4	1	S,Z,AC,P,CY
23	SBB R	(A)←(A)-(R)-CY	1	4	1	S,Z,AC,P,CY
24	INR R	(R)←(R)+1	1	5	1	S,Z,AC,P
25	DCR R	(R)←(R)-1	1	5	1	S,Z,AC,P
26	DAD RP	(HL)←(HL)+(RP)	3	10	1	CY
27	INX RP	(RP)←(RP)+1	1	5	1	- - - - - -
28	DCX RP	(RP)←(RP)-1	1	5	1	- - - - - -
29	ADD M	(A)←(A)+M(HL)	2	7	1	S,Z,AC,P,CY
30	ADC M	(A)←(A)+M(HL)+CY	2	7	1	S,Z,AC,P,CY
31	SUB M	(A)←(A)-M(HL)	2	7	1	S,Z,AC,P,CY
32	SBB M	(A)←(A)-M(HL)-CY	2	7	1	S,Z,AC,P,CY
33	INR M	M(HL)←M(HL)+1	3	10	1	S,Z,AC,P
34	DCR M	M(HL)←M(HL)-1	3	10	1	S,Z,AC,P
35	ADI D8	(A)←(A)+D8	2	7	2	S,Z,AC,P,CY
36	ACI D8	(A)←(A)+D8+cy	2	7	2	S,Z,AC,P,CY
37	SUI D8	(A)←(A)-D8	2	7	2	S,Z,AC,P,CY
38	SBI D8	(A)←(A)-D8-CY	2	7	2	S,Z,AC,P,CY
39	DAA	Десятичная коррекция(A)	1	4	1	S,Z,AC,P,CY

Группа команд логических операций

40	ANA R	(A)←(A)AND(R)	1	4	1	S,Z,P,AC*,CY=0
41	XRA R	(A)←(A)XOR(R)	1	4	1	S,Z,P,AC=CY=0
42	ORA R	(A)←(A)OR(R)	1	4	1	S,Z,P,AC=CY=0
43	CMP R	(A)-(R)	1	4	1	S,Z,AC,P,CY
44	RLC	Сдв. влево цикл.	1	4	1	CY←A(7),AC=0
45	RRC	Сдв. вправо цикл.	1	4	1	CY←A(0),AC=0
46	RAL	Сдв. влево цикл. через A(0)←CY	1	4	1	CY←A(7),AC=0
47	RAR	Сдв. вправо через A(7)←CY	1	4	1	CY←A(0),AC=0
48	CMA	(A)←INV(A)	1	4	1	- - - - - -
49	ANA M	(A)←(A)AND M(HL)	2	7	1	S,Z,P,AC,CY=0
50	XRA M	(A)←(A)XOR M(HL)	2	7	1	S,Z,P,AC=CY=0
51	ORA M	(A)←(A)OR M(HL)	2	7	1	S,Z,P,AC=CY=0
52	CMP M	(A)-M(HL)	2	7	1	S,Z,P,AC,CY
53	ANI D8	(A)←(A)AND D8	2	7	2	S,Z,P,AC*,CY=0
54	XRI D8	(A)←(A)XOR D8	2	7	2	S,Z,P,AC=CY=0
55	ORI D8	(A)←(A)OR D8	2	7	2	S,Z,P,AC=cy=0
56	CPI D8	(A)-D8	2	7	2	S,Z,AC,P,CY
57	CMC	(CY)←INV(CY)	1	4	1	CY
58	STC	(CY)←1	1	4	1	CY=1

Группа команд передачи управления

59	PSHL	(PCH)←(H) (PCL)←(L)	1	5	1	Все признаки сохраняют свои

60	JMP A16	(PC)←A16	3	10	3	значения
61	J(COND)A16	Если условие вы- полняется,то (PC)←A16,иначе (PC)←(PC)+1	5	17	3	
62	CALL A16	M(SP-1)←(PCH) M(SP-2)←(PCL) (SP)←(SP)-2 (PC)←A16	3	11	-	
63	C(COND) A16	Если условие вы- полняется,то см. ком.62,иначе (PC)←(PC)+1	5 3	17 11	3 -	
64	RST N	M(SP)←(PC) (PC)←N*8	3	11	1	
65	RET	(PCL)←M(SP) (PCH)←M(SP+1) (SP)←(SP)+2	3	10	1	
66	R(COND)	Если условие вы- полняется,то см. ком.65,иначе (PC)←(PC)+1	3 1	11 3	1	

Группа специальных команд

67	EI	Разрешить преры- вание (триггер РПР)←1	1	4	1	Все признаки сохраняют свои значения
68	DI	Запретить пре- рывания (триггер РПР)←0	1	4	1	
69	HLT	Останов	1	7	1	
70	NOP	Пустая операция	1	4	1	

В табл. 1.6 приняты следующие условные сокращения: ← – операция пересылки; <–> – операция обмена; A16 – шестнадцатиразрядный адрес; AND – конъюнкция (И); C – наличие переноса или заема (СУ-1); COND – одно из восьми условий; D8 – восьмиразрядный непосредственный операнд; D16 – шестнадцатиразрядный непосредственный операнд; INV – инверсия; I(PORT) – содержимое порта ввода с адресом PORT; M – "минус" (S=1); MT – количество машинных тактов; МЦ – количество машинных циклов; M(RP) – содержимое ячейки памяти по адресу, хранящемуся в регистровой паре RP; N – один из восьми уровней прерывания: 0, 1, 2, 3, 4, 5, 6, 7; NC – отсутствие переноса из старшего разряда или заема в старший разряд (CY=0); NZ – ненулевой результат (Z=0); O(PORT) – содержимое порта вывода с адресом PORT; OR – дизъюнкция (ИЛИ); P – "плюс" (S=0); PE – четность числа единиц в результате (P=1); PO – нечетность числа единиц в результате (P=0); PORT – восьмиразрядный адрес порта ввода-вывода; R – один из семи регистров: A, B, C, D, E, H, L; (R) – содержимое регистра; RP – одна из регистровых пар: B, D, H или SP; (RP) – содержимое регистровой пары; RP' – одна из регистровых пар: B или D; RP" – одна из регистровых пар: B, D, H или PSW; RPH – старший регистр в регистровой паре; RPL – младший регистр в регистровой паре; Z – нулевой результат (Z=1); XOR – сложение по модулю 2 (исключающее ИЛИ).

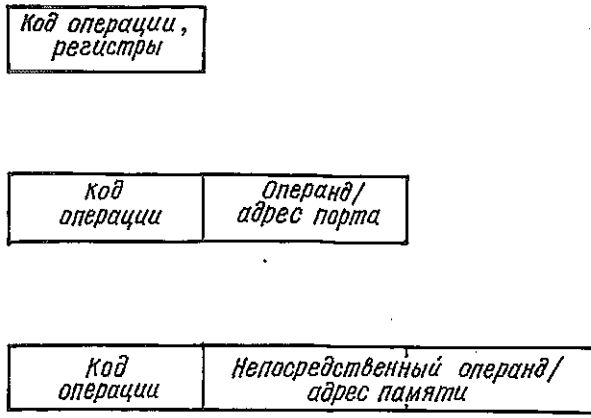


Рис. 1.14

Команды пересылки включают: MOV – пересылка между регистрами или регистром и ячейкой памяти (MOV B, C; MOV B, M); MVI – пересылка непосредственных данных в регистр или память (MVI B, 10); LXI – пересылка адреса в пару регистров (LXI D,1000); LDA, STA – загрузка и выгрузка аккумулятора по указанному адресу (LDA 1000, STA 2000). Команды LHLD, SHLD выполняют загрузку и выгрузку пары регистров HL двумя байтами по указанному адресу (LHLD 1100H; SHLD 2200H). Команды LDAX, STAX выполняют косвенную загрузку-выгрузку аккумулятора по адресу, указанному в паре регистров (LDAX B; STAX D). Команды PUSH, POP выполняют загрузку пары регистров в стек (PUSH PSW) и выгрузку вершины стека в пару регистров (POP B). Команды IN, OUT пересылают байт из порта ввода в аккумулятор и из аккумулятора в порт вывода соответственно (IN 20 H,OUT 01 H).

Команды XCHG и XTHL выполняют обмен словами между содержимым регистров HL и DE, HL и верхушкой стека (XCHG, XTHL). По команде SPHL содержимое регистровой пары HL пересылается в указатель стека SP (SPHL).

В группе арифметических команд основными операциями являются сложение и вычитание однобайтовых чисел, инкремент или декремент регистра, ячейки памяти, пары регистров, сложение пары регистров, а также десятичная коррекция аккумулятора при обработке двоично-десятичных чисел. При выполнении команд сложения-вычитания первый операнд и результат будут находиться в аккумуляторе. В отличие от команд пересылки данная группа устанавливает флажки после выполнения почти всех команд. В табл. 1.6 перенос обозначен CY, а межтетрадный перенос – AC.

К командам сложения относятся команды сложения аккумулятора с регистром ячейкой памяти без переноса (ADD B, ADD M) и с учетом переноса (ADC D, ADC M); сложение аккумулятора с константой без переноса (ADI 0BH) и с учетом переноса (ACI 12H), а также сложение пары регистров HL с указанной парой, при этом результат останется в HL (DAD B; HL ← HL + BC).

К командам вычитания относятся: вычитание из аккумулятора содержимого регистра или ячейки памяти без заема (SUB C, SUB M); вычитание из ак-

кумулятора содержимого регистра или ячейки памяти с заемом (SBB C, SBB M); вычитание константы без заема и с заемом (SUI 5, SBI 10H). Вычитание пары регистров можно произвести, используя сложение дополнительного кода по команде DAD, например для вычитания 0001H достаточно сложить код FFFFH, поместив его в пару BC (DAD B).

Команды инкрементирования, декрементирования регистра или ячейки памяти позволяют соответственно добавлять и вычитать единицу (INC B, DCR B). Для выполнения этих же операций над парами регистров используются команды (INX H, DCX B).

Команда десятичной коррекции преобразует результат сложения двоично-десятичных чисел в двоично-десятичный результат путем добавления кода 6 в тетраду, значение которой больше девяти или из нее был перенос. Например, при сложении десятичных чисел 59+59 в аккумуляторе будет код B2. В процессе выполнения команды DAA к коду первой тетрады добавится 6 (из нее был перенос), к коду второй тетрады тоже добавится 6 (ее содержимое превышает код 9). После коррекции в аккумуляторе будет код 18, в флажке C – код 1, что соответствует правильному результату 118 при сложении десятичных чисел 59+59.

Логические команды включают выполнение операций И, ИЛИ, исключающее ИЛИ. Один операнд находится в аккумуляторе, а второй может быть содержимым регистра, ячейки памяти или константой (ANA B, ORA M, XRI 10 H). При этом результат остается в аккумуляторе. Командами сравнения (по ним из содержимого аккумулятора вычитается регистрами ячейка памяти) результат не фиксируется, а только устанавливаются флажки. Заметим, что логические команды сбрасывают в нуль флажки байтового CY и межтетрадного переносов AC. Безоперандные команды RRC, RLC выполняют сдвиг аккумулятора вправо, влево циклически (младший разряд – в старший при сдвиге вправо и наоборот – при сдвиге влево). Команды RAR, RAL выполняют аналогичные действия, только через флажок CY. Команды CMA, CMC инвертируют соответственно аккумулятор и флажок CX, а STC устанавливает флажок CY в 1.

В группе команд передачи управления выполняется безусловный переход по адресу JMP 1000H либо условный по 1 или 0 флажков Z, CY, P, S (JZ 1000H, JNZ 100H, JC 50H, JNC 40H, JPE 30H, JPO 101H, JM 10H, JP 40H). Команда CALL выполняет безусловный переход к подпрограмме и отличается от команды JMP тем, что в стеке сохраняется адрес возврата, по которому командой RET осуществляется переход на команду, следующую после CALL. По команде CALL 400H передается управление по адресу 400, команда RET (последняя в подпрограмме) возвращает управление по адресу 100+3 с учетом того, что команда CALL расположена по адресу 100 (ее длина 3 байта).

Аналогично условным командам перехода имеется восемь условных команд обращения к подпрограмме и восемь команд условных возвратов из нее (CZ 1000H, RZ, CNZ, 100H, RNZ, CC 50H, RC, CNC 40H, RNC, CPE 30H, RPE, CPO 10H, RPO).

По команде PCHL содержимое пары регистров HL передается в программный счетчик PC. Эта команда может использоваться для передачи управления на подпрограмму, поскольку код в PC является адресом перехода. По команде RST n происходит следующее: вычисляется результат $8 * n$, $n = 0, ..., 7$,

который помещается в счетчик PC. Таким образом организуется переход на один из восьми векторов прерываний. По этому адресу может находиться программа обработки прерываний, но чаще всего команда JMP < адрес >, где < адрес > и есть начало этой программы.

Наконец,по командам управления EI, DI соответственно разрешается и запрещается прерывание по входу INT (триггер прерывания устанавливается в 1). Команда NOP не производит никаких действий и в основном используется для организации временных задержек. Команда HLT производит останов микропроцессора, из которого он может быть выведен сигналом прерывания.

1.5. СИСТЕМА КОМАНД МИКРОЭВМ СЕМЕЙСТВА "ЭЛЕКТРОНИКА 60"

Наиболее удачно и интересно задачи адресации и выбора формата команды решены в микроЭВМ семейства "Электроника 60". Систему команд в соответствии с их функциональным назначением принято разделять на пять групп: 1) пересылки информации; 2) арифметических операций; 3) логических операций; 4) ветвления; 5) управления.

Доступ к данным, хранящимся в памяти, а также манипуляция ими в микроЭВМ обеспечиваются командами, обычно специфицирующими: 1) функцию, которую необходимо исполнить (т. е. код операции); 2) общий регистр, используемый для определения операнда — источника и/или операнда—приемника; 3) режим адресации (для указания того, как избранный режим должен применяться).

Введение РОН как для манипуляции данными, так и для вычисления адресов приводит к форматам команд переменной длины. При этом программист может использовать РОН в качестве: 1) аккумуляторов, в которых находятся данные для манипуляции; 2) указателей данных (в противоположность функции аккумуляторов); 3) индексных регистров, чтобы определять относительное положение элементов данных в таблице или списке. Так как команды могут быть длиной в одно, два или три слова , ЭВМ "Электроника 60" иногда описывают как процессор с переменной длиной команд. Формат последних требует от программиста указания определенного режима адресации. ЭВМ "Электроника 60" имеет команды четырех основных форматов: управляющие (безадресные); одноопе рандные (одноадресные); двухоперандные (двухадресные); переходов.

Формат одноадресных команд представлен на рис. 1.15. Команда К(15:0)

Рис. 1.15

Таблица 1.7

Команда		Наименование	Признак				Алгоритм
мнемоника	код		N	Z	V	C	
CLR(B)	*050DD	Очистка	0	1	0	0	(dst)← 0
COM(B)	*051DD	Инвертирование	+	+	0	1	(dst) ← $\overline{(dst)}$
INC(B)	*052DD	Инкремент	+	+	+	–	(dst) ← (dst)+1
DEC(B)	*053DD	Декремент	+	+	+	–	(dst) ← (dst)–1
NEG(B)	*054DD	Изменение знака (дополнительный код)	+	+	+	+	(dst) ← $\overline{(dst)}$+1
ADC(B)	*055DD	Прибавление переноса	+	+	+	+	(dst) ← (dst)+(C)
SBC(B)	*056DD	Вычитание переноса	+	+	+	+	(dst) ← (dst)–(C)
TST(B)	*057DD	Проверка	+	+	0	0	(dst) ← (dst)
ROR(B)	*060DD	Циклический сдвиг вправо	+	+	+	+	(dst) ← $\overrightarrow{C, dst}$
ROL(B)	*061DD	То же влево	+	+	+	+	(dst) ← $\overleftarrow{C, dst}$
ASR(B)	*062DD	Арифметический сдвиг вправо	+	+	+	+	(dst) ← (dst)/2
ASL(B)	*063DD	То же влево	+	+	+	+	(dst) ← (dst)2
SXT	*0067DD	Расширение знака	–	+	0	–	(dst) ← 0, если N=0 (dst) ← –1, если N=1
MTPS	1064SS	Запись ССП	+	+	+	+	ССП ← (src)
MFPS	1067DD	Чтение ССП	+	+	0	–	(dst) ← ССП
SWAB	0003DD	Перестановка байтов	+	+	0	0	

Примечание. Знак * имеет значение 0 для команд с полными словами и 1 для байтовых команд; ( ) – содержимое ячейки; ← – символ присваивания; + (плюс), – (минус) – бит (признака) соответственно меняется и не меняется в зависимости от результата операции; 0 и 1 – бит признака устанавливается в 0 или 1.

Рис. 1.16

Таблица 1.8

Команда		Наименование	Признак				Алгоритм
мнемоника	код		N	Z	V	C	
MOV(B)	*1SSDD	Пересылка	+	+	0	–	(dst) ← (src)
CMP(B)	*2SSDD	Сравнение	+	+	+	+	(src) ← (dst)
BIT(B)	*3SSDD	Проверка разрядов	+	+	0	–	(src)∧(dst)
BIC(B)	*4SSDD	Очистка разрядов	+	+	0	–	(dst) ← $(\overline{src})$∧(dst)
BIS(B)	*5SSDD	Логическое сложение	+	+	0	–	(dst) ← (src)∨(dst)
ADD	06SSDD	Сложение	+	+	+	+	(dst) ← (src)+(dst)
SUB	16SSDD	Вычитание	+	+	+	+	(dst)←(dst)–(src)
XOR	074RDD	Исключающее ИЛИ	+	+	0	–	(dst)←(R_n)⊕(dst)

состоит из кода операции K(15:6) и поля адресации K(5:0) для задания приемника информации (DST) или источника (SRC). Одноадресные команды могут выполнять операции как с байтами, так и со словами (исключение составляют команды SXT, MTRS, MFPS, SWAB). Признаком байтовой операции является единица в 15-м разряде команды. В мнемоническом обозначении команды в случае операции над байтом добавляется буква B. Формальное описание команд приведено в табл. 1.7.

Формат двухадресных команд BIT, BIC, BIS, ADD, SUB, MOV, CMP приведен на рис. 1.16. Команда K(15:0) состоит из кода операции (КОП) K(15:12), поля SS адресации первого операнда src K(11:6) и поля DD адресации второго операнда dst K(5:0). Формальное описание команд приведено в табл. 1.8, в которой символами $\overline{\wedge}$, $\vee$, $\oplus$ обозначены логические операции И, ИЛИ, исключающее ИЛИ.

Команды MOV(B), CMP(B), BIT(B), BIC(B), BIS(B) могут выполнять операции как над словами, так и над байтами. Как и в одноадресных командах, признаком байтовой операции является единица в 15-м разряде команды. В мнемоническом обозначении команды в случае операции над байтом добавляется буква B.

При исполнении команды MOV признаки Z и N устанавливаются в соответствии с src. Это можно использовать для последующего ветвления по N или Z . При пересылке байта из памяти в регистр производится запись в младший байт регистра, а в старшем байте выполняется "распространение" знака операнда.

Пример. Команда "Сложение":

```
ADD @2534 (R1), R3
```

При этом R1/52; R3/4122; 2606/27342; 27342/123. Наклонная черта обозначает раскрытие содержимого регистра или ячейки памяти.

Анализ команды показывает, что для источника задан косвенный индексный режим адресации, а для приемника – регистровый. Следовательно, команда занимает два слова памяти:

```
PC      06SSDD
PC+2    индексное слово.
```

Так как косвенный индексный режим адресации имеет значение 7, а регистровый 0, машинный код заданной команды выглядит следующим образом:

PC 067103
PC+2 2534

Адрес адреса операнда источника равен 2534 + (R1) = 2534 + 52 = 2606. Адрес операнда источника из условия примера равен 27342, а сам операнд-источник – 27342/123. Таким образом, в результате выполнения команды сложения имеем 123 + 4122 = 4245. Данный результат заносится в приемник, т. е. в R3.

Таблица 1.9

Команда		Наименование	Признак				Алгоритм
мнемоника	код		N	Z	V	C	
BR	000400+xxx	Ветвление безусловное	–	–	–	–	(PC) ← (PC)+2xxx
BNE	001000+xxx	Ветвление, если ≠ 0	–	–	–	–	(PC) ← (PC)+2xxx, если Z = 0
BEQ	001400+xxx	Ветвление, если = 0	–	–	–	–	(PC) ← (PC)+2xxx, если Z = 1
BGE	002000+xxx	Ветвление, если ⩾ 0					(PC) ← (PC)+2xxx, если N⊕V = 0
BLT	002400+xxx	Ветвление, если < 0	–	–	–	–	(PC) ← (PC)+2xxx, если N⊕V =1
BGT	003000+xxx	Ветвление, если > 0	–	–	–	–	(PC) ← (PC)+2xxx, если ZV(N⊕V) = 0
BLE	003400+xxx	Ветвление, если ⩽0	–	–	–	–	(PC) ←(PC)+2xxx, если ZV(N⊕V) = 1
BPL	100000+xxx	Ветвление, если плюс	–	–	–	–	(PC) ← (PC)+2xxx, если N=0
BMI	100400+xxx	Ветвление, если минус	–	–	–	–	(PC) ← (PC)+2xxx, если N=1
BNI	101000+xxx	Ветвление, если больше	–	–	–	–	(PC) ← (PC)+2xxx, если CVZ=0
BLOS	101400+xxx	Ветвление, если меньше	–	–	–	–	(PC) ← (PC)+2xxx, если CVZ=1
BVC	102000+xxx	Ветвление, если нет переполнения	–	–	–	–	(PC) ← (PC)+2xxx, если V=0
BVS	102400+xxx	Ветвление, если переполнение	–	–	–	–	(PC) ← (PC)+2xxx, если V=1
BCC (или BHIS)	103000+xxx	Ветвление, если нет переноса	–	–	–	–	(PC) ← (PC)+2xxx, если C=0
BCS (или BLO)	103400+xxx	Ветвление, если перенос	–	–	–	–	(PC) ← (PC)+2xxx, если C=1
SOB	077R00+NN	Вычитание единицы и ветвление, если результат ≠ ≠ 0	–	–	–	–	(R_n)←(R_n)–1; если результат ≠ 0, то (PC) ← (PC)–2NN, иначе (PC) ← (PC)
JMP	0001DD	Безусловный переход	–	–	–	–	(PC) ← (dst)

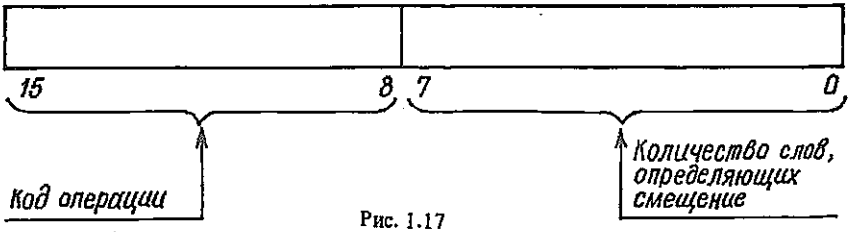


Рис. 1.17

К командам управления программой относятся команды ветвлений, перехода к подпрограмме, возврата из подпрограммы, безусловного перехода и др. Формат команд ветвления (кроме команды SOB) приведен на рис. 1.17. Эти команды состоят из 8-разрядного поля кода операции, для которого отведены биты К(15:8) команды, и 8-разрядного поля смещения К(7:0). Смещение записывается тремя восьмеричными цифрами ххх, из которых старшая может изменяться от 0Q до 3Q, а две младших – от 0Q до 7Q . Команды ветвления представляют собой команды условных переходов. В случае выполнения условия ветвления (т. е. условия перехода) осуществляется переход по адресу, являющемуся суммой удвоенного значения смещения и текущего содержимого РС, т. е. выполняется действие (РС) ← (РС) + 2 ххх.

Смещение ххх показывает, на сколько ячеек надо перейти относительно текущего содержимого СК в ту или иную сторону. Старший разряд смещения К(7) является знаковым. Если он равен 1, т. е. установлен, то смещение отрицательное и представлено в дополнительном коде; в этом случае ветвление происходит в сторону уменьшения адреса, т. е. в обратном направлении. Если знак смещения равен 0, оно положительное, и ветвление происходит в сторону увеличения адресов, т. е. в прямом направлении.

Смещение 8-разрядное может изменяться в диапазоне от –128D (т. е. –200Q) до +127D (т. е. +177Q). Поэтому возможно ветвление в обратном направлении максимально на 200Q слов от слова, на которое указывает текущее содержимое СК, и на 177Q слов – в прямом направлении.

Если при исполнении команды ветвления условие перехода не выполняется, то сохраняется естественная последовательность реализации команд, т. е. далее выполняется команда, следующая за командой ветвления.

Формальное описание команд ветвления и безусловного перехода приведено в табл. 1.9. Здесь ххх – код смещения (8 разрядов).

Если при выполнении команд ветвления удвоенное смещение складывается с содержимым РС, то РС в этот момент указывает на слово (команду), следующее за командой перехода; другими словами, если адрес команды ветвления обозначить через А, то адрес перехода вычисляется следующим образом:

(РС) ← А+2+2 ххх .

В поле К(15:9) команды SOB содержится КОП, имеющий значение 077Q, в поле К(8:6) задается регистр, а в поле К(5:0) – 6-разрядное смещение, которое условно записывается двумя 8-разрядными цифрами NN . При выполнении этой команды содержимое указанного регистра уменьшается на 1; если результат равен 0, то в СК загружается новое содержимое, определяемое вы-

читанием из текущего содержимого СК удвоенного смещения, т.е. выполняется действие (PC) ← (PC) – 2NN.

В команде SOB смещением является 6-разрядное положительное число. Эта команда может быть эффективно использована для организации различного рода счетчиков. Следует иметь в виду, что команда SOB не может быть примененена для передачи управления в прямом направлении.

Пример. Надо очистить память, начиная с ячейки MEM до ячейки WRD включительно. Это следует осуществить такой последовательностью команд:

```
        MOV     #MEM,R0
        MOV     #(WRD-MEM)/2+1,R1
LOOP:   CLR     (R0)+
        SOB     R1,LOOP
```

На какое максимальное расстояние можно передать управление в команде SOB?

Команда безусловного перехода JMP (см. табл. 1.9) обеспечивает передачу управления по адресу, определяемому в соответствии с заданным режимом адресации. Эта команда является одноадресной с форматом, соответствующим одноадресным командам (см. рис. 1.15). Использование регистровой адресации в данной команде недопустимо, так как передача управления на регистр не имеет смысла. Использование регистровой адресации вызывает прерывание программы по условию "запрещенная команда" с адресом вектора 4.

Пример.

Безусловный переход на команду, адрес которой находится в R1:

```
JMP  (R1)+
```

При этом для операндов R1/002700, R7/005000 (восьмеричный код команды 000121) после выполнения команды получаем R1/002702, R7/002700.

Безусловный переход на команду, адрес которой задан содержимым R1, уменьшенным на 2: JMP–(R1). При этом для операндов R1/002700, R7/005000 (восьмеричный код команды 000151) после выполнения команды R1/002676 получаем R7/002676.

Рассмотрим фрагмент программы, когда управление передается на оператор с меткой:

```
      JMP LAB  001000/000167
       ...     001002/000020
       ...     ...     ...
LAB:INC R1     001024/005203
```

До выполнения команды JMP R7/001000, после выполнения R7/001024, т. е. передано управление на команду с меткой LAB.

Для организации подпрограмм используются три команды: JSR, RTS и MARK. Формальное описание этих команд приведено в табл. 1.10.

Команда JSR в разрядах К(8:6) содержит R_n, а в разрядах К(5:0) указывается dst. При выполнении команды адрес возврата (т. е. текущее значение СК) запоминается в R_n, а содержимое R_n, которое находилось там до передачи в него значения PC, засылается в стек. Регистр R_n служит "указателем связи". Таким образом, обращение к вложенным подпрограммам осуществ-

ляется с помощью R_n, и вся связующая информация сохраняется в стеке. Благодаря такой организации при прерывании программы подпрограмма обслуживания прерывания может обращаться к той же самой прерванной подпрограмме.

При выполнении команды RTS реализуется процедура, обратная описанной для команды JSR, а именно: содержимое R_n заносится в PC, а из стека читается его первоначальное содержимое. Команда RTS содержит в разрядах К(15:4) КОП, а в разрядах К(3:0) – R_n. В качестве R_n должен использоваться тот же регистр, что и в команде JSR, по которой выполнялось обращение к данной подпрограмме.

К командам управления машиной относятся команды HALT, WAIT, RESET. Формальное описание этих команд приведено в табл. 1.11. При выполнении команды HALT происходит останов машины. В PC находится адрес следующей за HALT команды. ЭВМ остается в состоянии ОСТАНОВ до тех пор, пока оператор не продолжит выполнение программы при помощи команд пультового терминала. Команда WAIT прекращает выполнение всех операций ЦП. Выход из состояния ОЖИДАНИЕ может быть только по прерыванию. Команда RESET вызывает формирование в канале ЭВМ сигнала СБРОС, который устанавливает все внешние устройства в исходное состояние.

Команды изменения признаков позволяют устанавливать и сбрасывать коды условий N, Z, V, C, находящихся в ССП. Перечень команд установки кодов условий приведен в табл. 1.12.

Таблица 1.10

Команда		Наименование	Признак				Алгоритм
мнемоника	код		N	Z	V	C	
JSR	004RDD	Обращение к подпрограмме	–	–	–	–	↓(SP) ← (R_n) R_n ← (PC) PC ← (dst)
RTS	00020R	Возврат из подпрограммы	–	–	–	–	PC ← (R_n), R_n ← (SP)↑
MARK	0064NN	Восстановление указателя стека	–	–	–	–	SP ← (PC)+2NN PC (R5), R5 ← (SP)↑

Таблица 1.11

Команда		Наименование	Признак				Алгоритм
мнемоника	код		N	Z	V	C	
HALT	000000	ОСТАНОВ	–	–	–	–	Переход в состояние ОСТАНОВ
WAIT	000001	ОЖИДАНИЕ	–	–	–	–	Переход в состояние ОЖИДАНИЕ
RESET	000005	СБРОС	–	–	–	–	Сброс ВУ

Таблица 1.12

Команда		Наименование	Признак				Алгоритм
мнемоника	код		N	Z	V	C	
CLC	000241	ОЧИСТКА C	–	–	–	0	C ← 0
CLV	000242	ОЧИСТКА V	–	–	0	–	V ← 0
CLZ	000244	ОЧИСТКА Z	–	0	–	–	Z ← 0
CLN	000250	ОЧИСТКА N	0	–	–	–	N ← 0
SEC	000261	УСТАНОВКА C	–	–	–	1	C ← 1
SVV	000262	УСТАНОВКА V	–	–	1	–	V ← 1
SEZ	000264	УСТАНОВКА Z	–	1	–	–	Z ← 1
SEN	000270	УСТАНОВКА N	1	–	–	–	N ← 1
SCC	000277	УСТАНОВКА C,V,Z,N	1	1	1	1	C,V,Z,N ← 1
CCC	000257	ОЧИСТКА C,V,Z,N	0	0	0	0	C, V, Z, N ← 0
NOP	000240	Нет операции	–	–	–	–	

Таблица 1.13

Команда		Наименование	Признак				Алгоритм
мнемоника	код		N	Z	V	C	
MUL	00RSS	Умножение целых чисел	+	+	0	+	$R_n, R_n \vee 1 \leftarrow (R_n)(srs)$
DIV	071RSS	Деление целых чисел	+	+	+	+	$R_n, R_n \vee 1 \leftarrow (R_n, R_n \vee 1) : (src)$
ASH	072RSS	Арифметический сдвиг	+	+	+	+	$R_n \leftarrow$ сдвинутое (R_n)
ASCH	073RSS	Арифметический сдвиг двойного слова	+	+	+	+	$R_n, R_n \vee 1 \leftarrow$ сдвинутое $(R_n, R_n \vee 1)$

В группу команд расширенной арифметики входят команды MUL, DIV, ASH и ASCH. Формат команд расширенной арифметики представлен на рис. 1.18. Формальное описание в табл. 1.13.

Команда MUL вызывает перемножение операндов, первый из которых (операнд-источник) задается полем SS команды, второй (операнд-приемник) — полем R . Оба операнда представляются в дополнительном коде, а 32-разрядный результат умножения помещается в регистр-приемник, заданный полем R команды, и в следующий за ним по номеру регистр, если регистр-приемник имеет четный номер. Если же регистр-приемник имеет нечетный номер, то в нем сохраняется только младшая часть результата. После умножения N и Z устанавливаются в соответствии с результатом, V сбрасывается в 0, C устанавливается в 1 в случае, если результат меньше -2^{15} или больше $2^{15}-1$.

Пример. Умножить

MUL #10, R1.

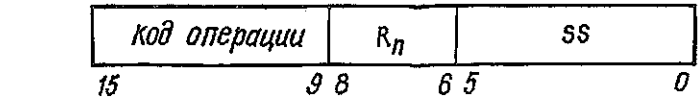


Рис. 1.18

Команда транслируется в два слова: 070127 – первое слово, 000010 – второе слово. Происходит умножение (R1) на восьмеричную константу 10; если до выполнения команды R1/000400Q, то в результате будет вычислена младшая часть результата и помещена в R1, т. е. получится R1/004000Q.

Необходимо иметь в виду, что на ассемблере порядок следования источника и приемника в командах расширенной арифметики такой же, как в обычных двухадресных командах (например, MUL S, R2, где S – источник, R2 – приемник).

Команда DIV вызывает деление 32-разрядного делимого, находящегося в регистрах R_n и R_nV1, на операнд источника. Частное помещается в R_n, а остаток со знаком делимого – в R_nV1. При делении номер R_n должен быть четным: N и Z определяются в соответствии с результатом; V устанавливается в 1, если (src) = 0; C устанавливается в 1 при попытке деления на нуль, если $|(R_n)| > |(src)|$ (в этом случае частное занимает больше 15 разрядов).

Пример. Разделить

```
DIV #2, R0
```

Команда транслируется в два слова: 071027 – первое слово, 000002 – второе слово. Осуществляется деление содержимого (R0, R1) на константу 2; если до операции R0/000000, R1/040001, то после операции R0/020000 (частное) и R1/000001 (остаток).

Для выполнения операций над числами с ПЗ используются команды FADD, FSUB, FMUL и FDIV. Формат чисел с ПЗ представлен на рис. 1.6.

Команда для операций с ПЗ имеет в разрядах К(15-3) код операции, а в разрядах К(2:0) – номер регистра R_n, содержимое которого вводится в качестве указателя стека операндов и результата.

Операнды, используемые для операции с ПЗ, задаются в специально организуемом стеке плавающей запятой (СПЗ). Содержимое R_n указывает на область размещения операндов в СПЗ, которые расположены следующим образом: (R_n) – адрес первого слова операнда B (второй операнд); $(R_n)+2$ – адрес второго слова операнда B; $(R_n)+4$ – адрес первого слова операнда A (первый операнд); $(R_n)+6$ – адрес второго слова операнда A.

После выполнения команды с ПЗ результат помещается в СПЗ на место первого операнда следующим образом: $(R_n)+4$ – адрес первого слова результата; $(R_n)+6$ – адрес второго слова результата, где (R_n) – содержимое регистра перед началом выполнения команды. После выполнения команды R_n будет указывать на первое слово результата, т. е. на $(R_n)+4$.

СПЗ может размещаться в любом месте ОП и в общем случае не является стеком в прямом смысле. Начальный адрес СПЗ перед выполнением команды с ПЗ должен быть помещен в R_n, указываемый в команде.

Если при выполнении команды с ПЗ имеет место переполнение, антипереполнение или попытка деления на 0, происходит прерывание с адресом векто-

Таблица 1.14

Команда		Наименование	Признак				Алгоритм
мнемоника	код		N	Z	V	C	
FADD	07500R	Сложение с ПЗ	+	+	0	0	A←A+B
FSUB	07501R	Вычитание с ПЗ	+	+	0	0	A←A–B
FMUL	07502R	Умножение с ПЗ	+	+	0	0	A←A×B
FDIV	07503R	Деление с ПЗ	+	+	0	0	A←A:B

ра 244. В этом случае признаки N , Z , V , C будут иметь следующие значения:

N=1, если произошло антипереполнение или деление на 0;

Z=1;

V=1, если имеет место переполнение, антипереполнение или попытка деления на 0.

C=1, если произошло деление на 0.

Список команд с ПЗ приведен в табл. 1.14.

Команды FADD, FSUB, FMUL, FDIV обработки с плавающей запятой выполняются по схеме A ← A * B, т. е. $((R_n)+4, (R_n)+6) \leftarrow ((R_n)+4, (R_n)+6) * ((R_n), (R_n)+2)$, где символ * использован для обозначения любой из четырех указанных операций с ПЗ. Если в результате выполнения операции результат по модулю получается меньше 2^{-128}, ему присваивается значение машинного нуля, т. е. $((R_n)+4, (R_n)+6) \leftarrow 0$. Признаки N и Z формируются в соответствии с результатом, а признак C очищается (при нормальном завершении операции).

1.6. СИСТЕМА КОМАНД МП 1810ВМ86

В зависимости от выполняемых функций команды МП 1810ВМ86 разделяются на следующие группы: пересылки, арифметические, порязрядной обработки данных, работы со строками, передачи управления (ветвления) и управления микропроцессором. Рассмотрим команды каждой из групп. При описании команд используем следующие обозначения в алгоритмах: E — операнд в памяти или в РОН, определенный полями mod и r/m; R — операнд в РОН; R_s — операнд в сегментном регистре; D — непосредственные данные в команде; A — при W = 0 операнд в AL , при W = 1 — в AX; St — стек; M (...) — операнд в памяти, в скобках указывается смещение; Port ( )- операнд в регистре ВУ, в скобках указывается адрес ВУ; EA — исполнительный адрес в памяти, определяемый полями mod и r/m; ext — знаковое расширение байта в слово или слова в двойное слово; & — объединение байта в слово, слова в двойное слово; B2, B3 — второй и третий байты команд; $f(...)$ — функция определения значения признака, результата операции, указанной в скобках.

Команды пересылки (общие, пересылки адресов, признаков, ввода-вывода и перекодировки) обеспечивают передачу оператора источника в операнд приемника без содержательного преобразования (табл. 1.15).

Команды пересылки данных общего назначения применяются с большин-

Таблица 1.15

Мнемоника	Алгоритм	Мнемоника	Алгоритм
	Общие		Упрощенного формата
MOV	E ← R , R ← E , E ← D; R ≠ CS	MOV	R ← D; R ≠ R_s
PUSH	St ← E16	PUSH	St ← R16
POP	E16 ← St	POP	St → R16
XCHG	E ⇔ R; R ≠ R_s	XCHG	AX ⇔ R
	Пересылки адресов		Пересылки признаков
MOV	R_s ← E; R_s ≠ CS	PUSHF, POPF	St ← F; F ← St
PUSH, POP	St ← R_s; R_s ← St	LAHF, SAHF	AH ← F; F ← AH
LEA	R_{16} ← EA		
			Ввода-вывода
LDS	DS ← M16(EA+2), R_{16} ← M16(EA)	IN	A ← Port(B2); A ← Port(DX)
LES	ES ← M16(EA+2), R_{16} ← M16(EA)	OUT	Port(B2) ← A; Port(DX) ← A
			Перекодировки
		XLAT	AL ← M(BX + AL)

стом операндов. Они (кроме команды XCHG) в качестве операнда могут использовать регистры сегмента (кроме CS).

По команде MOV производится пересылка байта или слова из источника (правый операнд) в приемник (левый операнд). Например, MOV AX, BX . Команда PUSH пересылает операнд в стек по адресу, указанному регистром SP , и уменьшает содержимое регистра SP на 2. Команда POP пересылает слово из стека по адресу, указанному регистром SP , в поле операнда и увеличивает регистр SP на 2. По команде XCHG слово или байт, адресуемые операндами, обмениваются между собой.

Команды пересылки адресов позволяют определять текущие сегменты памяти. Команда LEA (загрузка исполнительного адреса) передает в регистр (общий или индексный) величину смещения (EA). Команда LDS(LES) (загрузка указателя в регистр DS(ES)) передает 32-битовый объект (адрес сегмента и смещение), хранящийся в виде двойного слова в памяти, в пару регистров: DS(ES) и указанный регистр. По командам PUSH и POP значение адреса сегмента засылается (выбирается) в (из) стек(а): например, PUSH CS; POP DS.

По команде PUSHF содержимое регистра флажков сохраняется в стеке, а по команде POPF выбирается из него. Команда LAHF заносит правый байт регистра флажков в AH, а команда SAHF передает содержимое AH в правый байт регистра F .

Команда IN передает байт (слово) из вводного порта в регистр AL(AX). Порт указывается либо байтом данных (0–255), либо номером порта в регистре DX (до 65 536 портов). Команда OUT осуществляет передачу данных

Таблица 1.16

Мнемоника	Алгоритм	Мнемоника	Алгоритм
	Сложение		Вычитание
ADD	E ← E+R, R ← R+E, E←E+D	SUB	E ← E–R, R ← R–E, E ← E–D
ADC	E ← E+R+CF, R ← R+E+CF, E ← E+D+CF	SBB	E ← E–R–CF, R←R–E–CF, E←E–D–CF
INC	E ← E+1	DEC	E ← E–1
AAA, DAA	Коррекция результата сложения распакованных и упакованных двоично-десятичных чисел	NEG	E ← 0–E
		CMP	E ← f(R–E), F ← f(E–R) F ← f(E–D)
	Сложение, вычитание упрощенное	AAS,DAS	Коррекция результата вычитание для распакованных и упакованных десятичных чисел
ADD(ADC)	A ← A+D (A ← A+D+CF)		
SUB(SBB)	A ← A–D (A ← A–D–CF)		
INC (DEC)	R ← R+1 (R ← R–1)		
CMP	F ← f(A–D)		Преобразование форматов
	Умножение без знака	AAM	Коррекция числа в AX, полученного в результате умножения распакованных десятичных чисел
MUL	AX ← AL×E, DX&AX←AX×E		
	Умножение со знаком		
IMUL	AX ← AL×E, DX & AX←AX×E	AAD	Коррекция двоично-десятичного числа в AH перед делением
	Деление без знака		
DIV	AL ← AX/E, AH ← mod(AX/E) AX←(DX&AX)/E, DX ← mod((DX&AX)/E)	CBW	AX ← ext AL
		CWD	DX ← ext AX
	Деление со знаком		
IDIV	AX ← (DX&AX)/E DX ← mod((DX&AX)/E) AL ← AX/E, AH ← mod(AX/E)		

из аккумулятора в порт вывода и работает подобно IN. Например, IN AL, 20H; IN AX, DX; OUT 40H, AX; OUT DX, AL.

По команде XLAT осуществляется поиск в таблице и замена операнда на соответствующий байт из таблицы. При этом содержимое регистра AL используется в качестве индекса в 256-байтовой таблице, адресуемой регистром BX.

Арифметические команды (сложение, вычитание, умножение и деление без знака и со знаком, преобразование форматов) предназначены для выполнения операций над порядковыми, целыми и двоично-десятичными данными (табл. 1.16).

Существует пять команд сложения:

1) ADD производит сложение операндов и запоминает результат по адресу первого операнда. Например, ADD CX, [BX], сложение регистра с ячейкой памяти;

2) ADC выполняет сложение операндов с битом переноса CF и запоминает результат по адресу первого операнда. Например, ADC AL, 35H; сложение аккумулятора с константой;

3) INC увеличивает на 1 содержимое операнда и помещает результат по тому же адресу. Например INC SI, INC [DI]; инкремент регистра и ячейки памяти;

4) AAA выполняет коррекцию числа в регистре AL , полученного в результате сложения двух распакованных десятичных операндов;

5) DAA производит коррекцию упакованной десятичной суммы, полученной в результате сложения упакованных десятичных операндов.

Для вычитания используются семь разновидностей команд:

1) SUB вычитает из второго операнда первый, результат записывает по адресу первого операнда. Например SUB CL, [BP]; вычитание из CL ячейки памяти;

2) SBB вычитает из разности второго и первого операндов бит CF, результат записывает по первому адресу. Например, SBB BL, DL; вычитание BL ← BL - DL - CF;

3) DEC вычитает 1 из содержимого адресуемого элемента. Например, DEC SI, DEC [BX]A1;

4) NEG производит инвертирование содержимого адресуемого элемента. Например, NEG BX;

5) CMP производит вычитание второго операнда из первого, устанавливает биты условий без запоминания результата. Например, CMP AL, 0CH; сравнение аккумулятора с константой;

6) AAS выполняет десятичную коррекцию числа в регистре AL, полученного в результате вычитания двух распакованных десятичных операндов;

7) DAS производит десятичную коррекцию результата вычитания двух упакованных десятичных чисел.

Для умножения используются три команды:

1) MUL производит умножение без знака аккумулятора (AL или AX) и второго операнда, результат помещается в AH&AL для 8-разрядных операндов и в DX&AX – для 16-разрядных операндов. Например, MUL DL; умножение AL на DL, результат помещается в AX;

2) IMUL подобна команде MUL; по ней выполняется умножение с учетом знаков. Например, IMUL CX; AX умножается на CX; результат заносится в DX (старшие разряды) и AX (младшие);

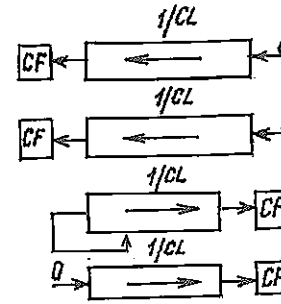
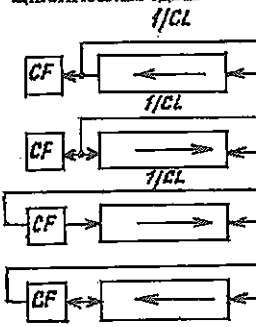
3) AAM производит коррекцию числа в регистре AX, полученного в результате умножения двух распакованных десятичных операндов для нахождения распакованного десятичного операнда. Пусть AL=03, BL=09 . После команд MUL BL; AAM получим AX=0207, т. е. 27.

Для деления используется три команды:

1) DIV производит деление содержимого аккумулятора (и его расширения для 16-разрядной операции) на содержимое, адресуемое операндом, и размещает частное в AL(AX) и остаток в AH(DX). Деление на нуль дает прерывание по вектору 0. Например, DIV CL; команда производит деление AX на CL, результат помещает в AL, остаток – в AX;

2) IDIV производит те же действия, что и команда DIV, только над операндами со знаками. Например, IDIV CX; делимое в DX, AX делится на CX

Таблица 1.17

Мнемоника	Алгоритм	Мнемоника	Алгоритм
	Логические		Линейный сдвиг
NOT	$\bar{E} \leftarrow E$	SHL	
AND	$E \leftarrow E \wedge R, R \leftarrow R \wedge E,$ $E + E \wedge D$	SAL SAR SHR	
OR	$E \leftarrow E \vee R, R \leftarrow R \vee E,$ $E \leftarrow E \vee D$		
XOR	$E \leftarrow E \oplus R, R \leftarrow R \oplus E,$ $E \leftarrow E \oplus D$		
TEST	$F \leftarrow f(E \leftarrow E \wedge R)\ f(R \leftarrow R) \wedge$ $\wedge E)\ f(E \leftarrow E \wedge D)$		
			Циклический сдвиг
	Упрощенного формата		
AND	$A \leftarrow A \wedge D$	ROL	
OR	$A \leftarrow A \vee D$	ROR	
XOR	$A \leftarrow A \oplus D$	RCL	
TEST	$F \leftarrow f(A \wedge D)$	RCR	

Результат помещается в AX, остаток – в DX;

3) AAD производит коррекцию делимого перед делением двух десятичных чисел, так что в результате получается распакованное десятичное число. Например, AX = 0604D. Выполнится AAD, получим AX = 0040H.

Для преобразования форматов используются команды:

CWW (производит распространение знака числа из регистра AL в AH). Например, AL=81. Выполнится команда CWW, получим (AX)=FF81;

CWD (производит распространение знака числа из AX в DX). Например, AX=5000. Выполнится команда CWD, получим (DX) =0000, AX=5000.

Логические команды реализуют операции NOT, И, ИЛИ, исключающее ИЛИ, а также операции линейного и циклического сдвига (табл. 1.17).

Команда NOT осуществляет инвертирование содержимого операнда, NOT AX. Логические операции над двумя операндами биты CF и OF сбрасывают, а биты SF, PF и ZF устанавливают. К этим операндам относятся: AND – поразрядное логическое умножение операндов (AND AX,[SI]);OR – поразрядное логическое сложение операндов (OR TI, DL); XOR – поразрядное сложение по модулю двух операндов (XOR AL,DL). TEST аналогична команде AND, но не устанавливает результат (TEST [SI] , CX).

Четыре команды линейного сдвига выполняются над содержимым памяти или регистров. Первые две (SHR,SHL) реализуют логический сдвиг, вторые (SAR, SAL) – арифметический. Сдвиги могут выполняться на один разряд и на несколько разрядов, указанных в регистре CL. Например, SAL AX,1; SAR [BP], CL.

Четыре команды циклического сдвига (ROL, ROR, RCL, RCR) с использованием бита CF выполняются над ячейкой памяти или регистром. При этом сдвиги возможны на один разряд или несколько (указаны в регистре CL). Например, RCL D1, 1; RCR [SI], CL.

Команды работы со строковыми данными позволяют не только вести расчеты, но и обрабатывать тексты, пересылать и сравнивать строки, производить поиск (пропуск) элемента в строке, заполнять строки конкретными символами (например, пробелом). Строковые команды можно использовать для обработки массивов данных других типов (целых, порядковых и т.д.).

В эту группу входят следующие команды: MOVS выполняет пересылку строки источника по адресу элемента строки приемника; CMPS производит сравнение строк путем вычитания строки приемника из строки источника (устанавливает признаки результата); LODS загружает элемент строки источника в регистр AX; SCAS сравнивает содержимое регистра AX и элемента строки приемника; STOS пересылает содержимое регистра AX по адресу элемента строки приемника.

В качестве элементов строк могут выступать одно- или двухбайтовые данные. Строки источника располагаются в сегменте данных DS, смещение элементов строк в сегменте задается содержимым SI . Строки приемника находятся во вспомогательном сегменте ES, смещение элементов задается содержимым регистра DI .

Строковые команды автоматически увеличивают или уменьшают содержимое регистров SI и DI на 1 или 2. Направление (увеличение или уменьшение смещения) задается битом DF (0 – увеличение, SI, DI, 1 – уменьшение SI, DI).

Для выполнения действий над строками в программе перед строковой командой необходимо поместить префикс повторения REP. Он задает многократное повторение последующей строковой команды, сопровождаемое вычитанием 1 из регистра CX. Как только CX становится равным 0, выполнение строковой команды прекращается. Для команд CMPS, SCAS используется два варианта префикса: REPE/Z (повторить, пока равно), REPNE/NZ (повторить, пока не равно). В первом случае команда SCAS повторяется до тех пор, пока содержимое аккумулятора не перестает совпадать с ячейкой памяти или CX не станет равным нулю. Команда REPNE выполняет сравнение элементов, пока они не равны, т. е. комбинация команд REPNE и SCAS позволяет быстро выполнить поиск в таблице. Приведем примеры обнуления массива и сравнения массивов байт:

```
REPNZ   CMPSB          ;ПОВТОРИТЬ СРАВНЕНИЕ ЭЛЕМЕНТОВ
        JZ EQUAL

        LES DI,A1      ;АДРЕС МАССИВА В ES:DI
        MOV CX,LENGTH  ;ДЛИНА В CX
        MOV AX,0       ;НУЛЬ В AX
REP     STOSW          ;ОБНУЛЕНИЕ МАССИВА
```

Таблица 1.18

Мнемоника	Алгоритм
JMP	Безусловная передача управления
	Межсегментная прямая
	IP ← B3&B2; CS ← B5&B4
	Межсегментная косвенная
	CS&IP ← M16(EA+2)&M16(EA)
	Внутрисегментная прямая
	IP ← IP+B3&B2
	Внутрисегментная прямая короткая
	IP ← IP+B2ext
	Внутрисегментная косвенная
	IP ← M16(EA)
CALL	Организация подпрограмм
	Межсегментная прямая
	St ← CS; St ← IP
	IP ← B3&B2;
	CS ← B5&B4
	Межсегментная косвенная
	St ← CS: St ← IP
	CS&IP ← M16(EA+2) & M2(EA)
	Внутрисегментная прямая
	St ← IP; IP ← B3&B2
	Внутрисегментная косвенная
	St ← IP; IP ← M16(EA)
RET	Возврат из подпрограмм
	Межсегментный
	IP ← St; CS ← St
	Внутрисегментный
	IP ← St
INT n	Прерывания
	St ← F; St ← CS; St ← IP
	CS&IP ← VEC n
INTO	if OF = 1 then St ← F, St←CS,
	St ← IP CS&IP ← VEC4
IRET	Возврат из прерываний
	IP ← St; CS ← St; F ← St

Команды передачи управления выполняют безусловную передачу управления, организацию работы с подпрограммами, обращение к прерыванию, условный переход и организацию циклов, которые изменяют естественный порядок выполнения программы (табл. 1.18).

Команды JMP, CALL организуют передачу управления как внутри текущего сегмента кодов, так и в другой сегмент, который становится текущим. Кроме прерывания, эти команды позволяют переходить в другой сегмент. При естественном ходе выполнения программы при достижении конца сегмента управление передается на начало сегмента. Например, JMP E8; JMP BX; JMP 0010 A000; CALL 0B00; CALL AX.

По команде RET управление передается из вызываемой подпрограммы в основную и при необходимости восстанавливается регистр IP, если в стек были помещены параметры.

К командам прерывания относятся: INT n – двухбайтовая, задающая программное прерывание, определяемое пользователем; управление передается по одному из 256 векторов прерываний; INT3, INTO – однобайтовая, определяющая программные прерывания по точкам разрыва и по переполнению. Указатель для загрузки регистров CS и IP при выполнении этих команд выбирается из одной четырехбайтовой области (вектора прерываний), их число достигает 256; IRET – для возврата из процедур обработки программных или аппаратных прерываний.

В МП реализуется ряд команд условного перехода, которые являются двухбайтовыми. Четыре команды позволяют проверить все отношения между знаковыми операндами (больше, меньше) и четыре команды – между беззнаковыми операндами (ниже, выше). Например, число BA меньше или выше

Таблица 1.19

Мнемоника	Условие	Вариант перехода	Мнемоника	Условие	Вариант перехода
JE/JZ	ZF = 1	Равно/нуль	JG/JNLE	(SF=OF) ZF=0	Не меньше или равно/больше
JL/JNGE	ZF⊕OF=1	Меньше/не больше или равно	JAE/JNB/JNC	CF=0	Не ниже/выше или равно
JLE/JNG	(SF⊕OF)∨ ∨ ZF=1	Меньше или равно/не больше	JA/JNBE	(CF∧ZF)= =0	Не ниже или равно/выше
JB/JNAE/JC	CF=1	Ниже/не выше или равно	JNE/JNZ	ZF=0	Не равно/не нуль
JBE/JNA	(CF∨ZF)= =1	Ниже или равно/не выше	JNP/JPO	PF=0	Нет четности
JP/JPE	PF=1	Четность	JNO	OF=0	Нет переполнения
JO	OF=1	Перепол-	JCXZ	CX=0	По нулю счетчика
JS	SF=1	Отрицательный знак	JNS	SF=0	Положительный знак
JGE/JNL	SF=OF	Не меньше/больше или равно			

Таблица 1.20

Мнемоника	Алгоритм
STC; CLC; CMC	CF ← 1; CF ← 0; CF ← $\overline{CF}$
STD; CLD	DF ← 1; DF ← 0
STI; CLI	IF ← 1; IF ← 0

числа 35. Девять команд проверяют условия по нулю, знаку, четности, переполнению и счетчику. При этом для выполнения условия второй байт команды знаково расширяется до слова и складывается с содержимым регистра IP. Тем самым обеспечивается переход в пределах – 128–127 байт от адреса следующей команды. Анализируемые условия и мнемоника команд представлены в табл. 1.19.

Для организации циклов используются команды LOOP, LOOPZ, LOOPNZ. Их выполнение заключается в уменьшении индекса цикла, помещаемого в регистр CX. При этом к IP прибавляется содержимое второго байта команды, знаково расширенное до слова. В команде LOOP условием перехода является CX≠0, в команде LOOPZ CX≠0&ZF=1, а в команде LOOPNZ CX≠0&ZF=0.

Команды управления микропроцессором подразделяются на команды управления признаками состояния (табл. 1.20) и синхронизации (WAIT, ESC, HLT). По командам, работающим с битами условий, устанавливаются (в 1 или 0) биты переполнения CF , направления увеличения (уменьшения) адреса DF , разрешения прерывания IF .

По команде WAIT МП переходит в состояние ожидания, которое может быть нарушено внешним прерыванием. При этом сохраненный адрес будет адресом команды WAIT , поэтому после возвращения из прерывающей программы состояние ожидания сохраняется и сбрасывается по команде TEST.

Команда ESC обеспечивает "условие", благодаря которому другие процессоры могут воспринимать команды из потока команд МП, используя его методы адресации. По этой команде осуществляется только доступ операндов к памяти. Данные условия применимы для мультимикропроцессорных систем. Команда останова HLT переводит МП в состояние останова, из которого его можно вывести сигналом прерывания или начальной установкой и запуском.

Рассмотренная система команд МП 1810ВМ86 по сложности операций, методам адресации, форматам обрабатываемых данных соответствует командам современных ЭВМ (ЕС, СМ). В то же время она отражает многие тенденции в организации вычислительных систем и программирования. Это касается сегментации памяти, обеспечения взаимодействия с другими процессорами, поддержки аппарата процедур, упрощения доступа к сложным структурам данных, реализации операций цикла и работы со строковыми данными.

1.7. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА МИКРОЭВМ

Виды ввода-вывода. Для передачи данных между микроЭВМ и внешним устройством применяются три основных способа: 1) программируемый ввод-вывод (или программный ввод-вывод); 2) ввод-вывод по прерываниям; 3) прямой доступ к памяти (ПДП).

Для программирования ввода-вывода в микроЭВМ имеется набор команд, называемых командами ввода-вывода и используемых для следующих целей: 1) вывода байта данных на ЭВМ во ВУ (например, при передаче символа в коде ASCII из аккумулятора или памяти на терминал); 2) посылки в ВУ управляющего приказа (например, посылки специального управляющего слова в интерфейс для разрешения генерирования прерывания, когда данные готовы для передачи в ЦП); 3) ввода байта данных из внешнего устройства в аккумулятор (например, ввода символа в коде ASCII с терминала); 4) проверки состояния готовности внешнего устройства к обмену, чтобы ЦП мог принять решение о передаче данных. Например, в МПК 580 есть две команды ввода-вывода: IN и OUT . Однако в микроЭВМ семейства "Электроника 60" нет отдельных пространств ввода-вывода и памяти; часть адресного пространства памяти, как отмечалось, резервируется для адресов ввода-вывода, и для передач данных используются команды обращения к памяти.

Программный ввод-вывод оказывается самым простым из всех способов и требует минимума вспомогательной логики. Он осуществляется полностью под управлением ЦП. Интерфейс содержит несколько регистров, которые участвуют в передачах ввода-вывода. Обычно используются два регистра: состояния и буферный. Регистр состояния содержит текущие состояния внешних устройств и передаваемых данных. На рис. 1.19 приведена схема типичной подпрограммы ввода-вывода.

Программный ввод-вывод. Программная модель ВУ содержит четыре регистра (рис. 1.20): входной и выходной порты, регистр управления и регистр

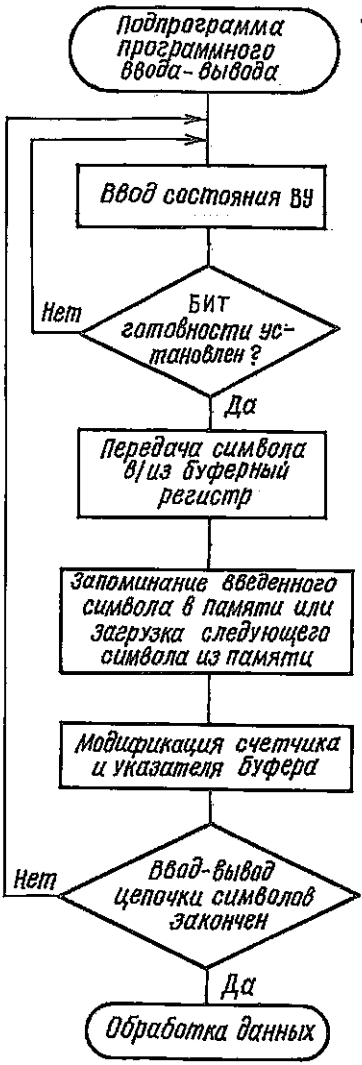


Рис. 1.19

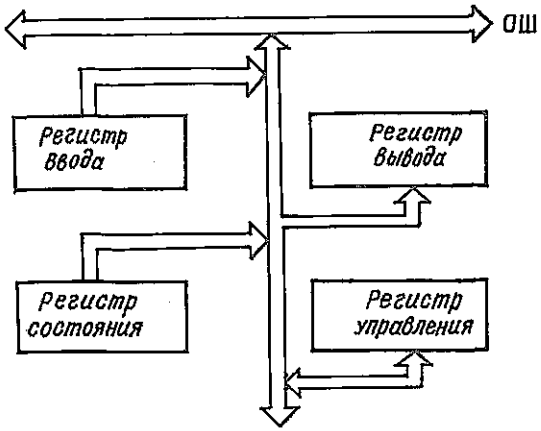


Рис. 1.20

состояния. Общее состояние готовности ВУ характеризуется флажком готовности READY(R). Для устройства ввода при R=0 входные данные отсутствуют, при R=1 – присутствуют. Для устройств вывода R=0 означает недоступность регистра для приема данных, R=1 означает, что регистр доступен. При вводе-выводе данных флажок сбрасывается.

Рассмотрим простейшую программу ввода. Пусть бит готовности R находится в седьмом разряде. Перед вводом необходимо проверить его состояние. Пусть адрес порта ввода 01H, а регистра готовности 00H. Тогда фрагмент ввода имеет вид:

```
VVOD:IN  00H ;ВВОД РЕГИСТРА СОСТОЯНИЯ В А
     RAL     ;СДВИГ 7-ГО БИТА А В С
     JNC VVOD;ЗАЦИКЛИВАНИЕ ДО УСТАНОВКИ БИТА ГОТОВНОСТИ
     IN  01H ;ВВОД СИМВОЛА ИЗ ПОРТА ВВОДА
     RET
```

Рассмотрим еще примеры программного ввода-вывода в системе команд микроЭВМ "Электроника 60". Символьные имена TPS и TPB определяют адреса регистра состояния печатающего устройства и буферного регистра ввода. Седьмой бит регистра TPS является битом готовности: он сбрасывается на время занятости буфера вывода и устанавливается в момент, когда буфер готов принять новую литеру. Итак, перед посылкой литеры ЦП в TPB программа должна ждать, пока седьмой бит регистра TPS не будет установлен. Удобно, что седьмой бит есть знаковый разряд младшего байта слова. Цикл печати следующий:

```
        TPS=177564
        TPB=177566
        ...
        MOV     #MESSGE,R1
LOOP1:  TSTB    (R1)
        BEQ     DONE
LOOP2:  TSTB    TPS
        BPL     LOOP2
        MOVB    (R1)+,TPB
        BR      LOOP1
        ...
        ...
MESSGE: .ASCII  /TESTING OUTPUT/
```

Заметим, что, поскольку мы выводим информацию, хранящуюся в соответствующих байтах массива MESS GE, применение байтовой команды MOVB с автоинкрементной адресацией корректно. Седьмой бит регистра TPS устанавливается и сбрасывается электроникой терминального устройства и только ею. Любая программная попытка изменить его значение игнорируется. Это бит только для чтения.

Ввод литер осуществляется с терминала, подключенного к микроЭВМ, так что по адресу можно получить доступ к буферу клавиатуры и регистру состояния точно так же, как и к буферу вывода. Программа чтения набираемых на терминале литер, которая заносит их в блок байтов, начиная с адреса 2000 и заканчивая свою работу по клавише ↵, выглядит следующим образом:

```
        TKS=177560
        TKB=177562
        ...
        MOV     #2000,R1
        MOV     #TKS,R2
LOOP:   TSTB    (R2)
        BPL     LOOP
        MOVB    TKB,(R1)
        CMPB    #12,(R1)+  ;ПРОВЕРКА НА СОВПА-
                           ;ДЕНИЕ С LINE FEED,Т.Е С↵
        BNE     LOOP
        ...
```

Программа дублирования перфолент на высокоскоростных или низкоскоростных перфоленточных устройствах ввода-вывода имеет вид:

```
000000 016700  C: MOV   177550,R0
000002 000024
000004 016701     MOV   177554,R1
000006 000022
000010 005210     INC   (R0)
000012 105710  A: TSTB  (R0)
000014 100376     BPL   A
000016 105711  B: TSTB  (R1)
000020 100376     BPL   B
000022 022021     CMP   (R0)+,(R1)+
000024 111011     MOVB  (R0),(R1)
000026 000764     BR    C
000030 177550
000032 177554
```

Ввод-вывод по прерываниям. Очевидно, большая часть времени в подпрограмме ввода-вывода уходит на ожидание готовности ВУ принять или передать информацию, но в процессе ожидания ЦП может обрабатывать данные, уже находящиеся в памяти. Другими словами, работу ЦП можно представить во времени двумя независимыми программами: передачи данных и их обработки. Такое совмещение вычислений и управления ВУ обеспечивается средствами прерываний.

Полное машинное состояние программы, непосредственно выполненной перед прерыванием, обычно запоминается в слове состояния программы (PSW). В микроЭВМ семейства "Электроника 60" PSW разделяется на две части, чтобы разместить информацию о состоянии процессора (PS) и содержимом счетчика команд (PC). Техника манипуляции прерыванием состоит в том, что текущее PSW заменяется на PSW программы, обрабатывающей прерывание. А текущее PSW сохраняется в некотором слове памяти. Внешнее устройство после принятия сигнала подтверждения прерывания выдает в процессор 16-битовый адрес по линиям данных/адреса, называемый *вектором прерываний*. Он однозначен для каждого ВУ и встроен в интерфейс. Затем ЦП выполняет программу в определенной последовательности (рис. 1.21): 1) ЦП вводит 16-битовый вектор прерывания, сформированный выбранным прерывающим внешним устройством; 2) в стеке запоминается PS; 3) в стек заносится PC, содержащий адрес возврата; 4) ячейка памяти, адресуемая вектором прерываний, содержит начальный адрес подпрограммы обслуживания, который загружается в PC; 5) следующее слово (вектор прерываний +2) загружается в PS и становится новым словом состояния программы; 6) процессор начинает подпрограмму обслуживания прерывания, выбирая следующую команду по адресу из PC.

Для терминальной клавиатуры отклик процессора эквивалентен такой последовательности команд:

```
MOV     PS,-(SP)
MOV     PC,-(SP)
MOV     60,PC
MOV     62,PS
```

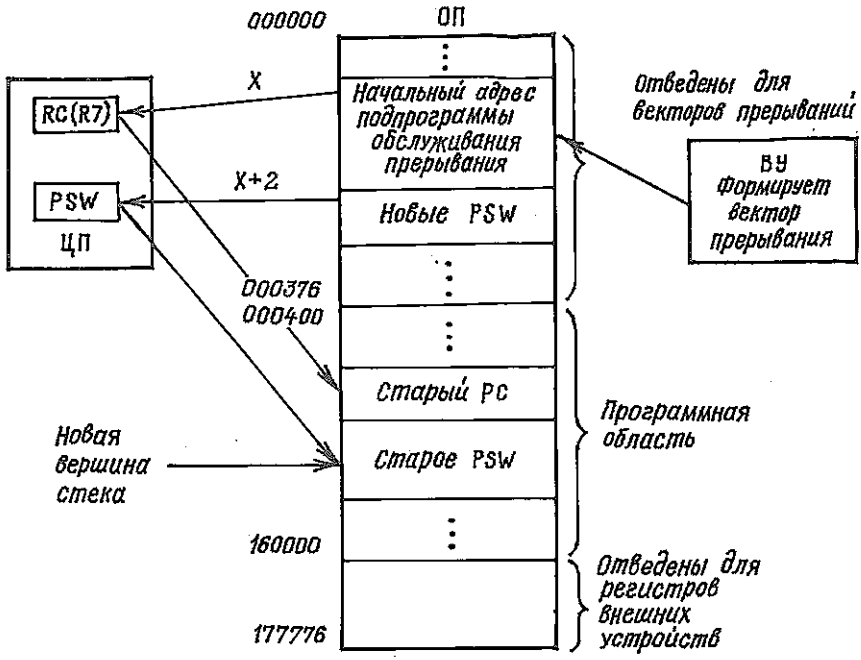


Рис. 1.21

Таблица 1.21

Команда		Наименование	Признак	Алгоритм
мнемоника	код		N, Z, V, C	
EMT	104000–104377	Командное прерывание	Из нового ССП	↓(SP)←(PCП); ↓(SP)←(PC); PC←(30); PCП ← (32)
TRAP	104400–104777	То же	"	↓(SP) ← (PCП); ↓(SP) ← (PC) (PC)← (34); PCП ← (36)
IOT	000004	Прерывание для ввода-вывода	"	↓(SP) ← (PCП); ↓(SP) ←(PC) PC← (20); PCП←(22)
BPT	000003	Прерывание для отладки	"	↓(SP) ← (PCП); ↓(SP) ← (PC) (PC) ← (14); PCП ← (16)
RTI	00002	Возврат из прерывания	Из стека	PC ←(SP)↑; PCП ← (SP)↓
RTT	000006	То же	"	PC ← (SP)↑; PCП ← (SP)↑

Здесь адрес вектора прерывания терминальной клавиатуры 60, а новое PSW загружается в регистр состояния PS из ячейки 62.

Вектор прерывания 16-битовый может адресовать любую ячейку памяти, и поэтому в системе возможно большое число ВУ. Однако наличие в дейзи-цепочке (система приоритетного прерывания; наибольший приоритет имеет ВУ, расположенное ближе всех к модулю процессора) слишком большого числа внешних устройств удлиняет цикл шины во время прерывания. В типичной системе вектор прерываний находится в диапазоне от 0 до 400. В интерфейсе каждого ВУ седьмой бит PS управляет разрешением прерывания. Если PS(7)= =1, то прерывание обрабатывается, в противном случае интерфейс не может выдать запрос на прерывание.

Иллюстрация механизма прерывания показана на примере подпрограммы обработки прерываний от печатающего устройства. Адрес вектора прерывания печатающего устройства равен 64:

```
        TITLE BPRINT
        R1=%1
        TPS=177564
        TPB=177566
START:  MOV    64,R1
        MOV    #SERV,64  ;ЗАГРУЗКА В ЯЧЕЙКУ
                         ;64 АДРЕСА ПОДПРОГРАМ-
                         ;МЫ ОБРАБОТКИ ПРЕРЫВАНИЯ
        BIS    #100,TPS  ;РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ
IN:     BR     IN
SERV:   MOVB   #102,TPB  ;ПЕЧАТЬ БУКВЫ B
        BIS    #100,TPS  ;ЗАПРЕЩЕНИЕ ПРЕРЫВАНИЯ
        MOV    R1,64
        RTI              ;ВЫХОД ИЗ ПРЕРЫВАНИЯ
        .END START
```

Для выхода из подпрограмм обслуживания прерываний используются команды RTI и RTT (табл. 1.21). Команда возврата из прерывания RTI является последней в подпрограмме обработки прерывания. При ее выполнении из стека извлекаются значения PC и PSW прерванной программы, и управление передается для ее продолжения. Если при выполнении команды RTI установится разряд PS(T), то следующая после RTI команда выполняться не будет (т.е. очередная команда прерванной программы), а произойдет новое прерывание. В случае использования команды RTT для выхода из прерывания ее действия будут идентичны действиям команды RTI, за исключением того, что при установке T-разряда прерывание возможно после того, как выполнится первая команда, следующая за RTT, т.е. очередная команда прерванной программы.

В системе МП 580 прерывающее устройство обычно формирует команду RST , которая вызывает одну из 8-байтовых подпрограмм, расположенных в первых 64 байтах памяти. Например, нулевое устройство при каждом его запросе на прерывание формирует команду RST0 (рис. 1.22, *а*). Для обработки

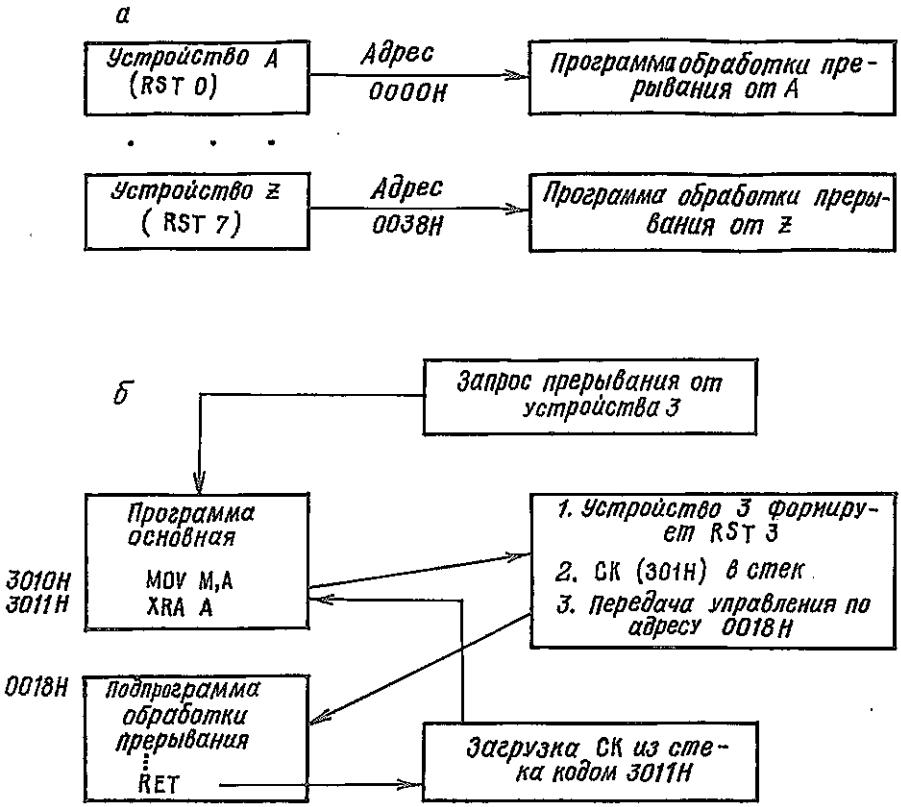


Рис. 1.22

данных, поступающих от нулевого устройства, управление передается соответствующей подпрограмме группой команд, расположенных в байтах памяти 0Н–07Н (рис. 1.22, *б*).

Рассмотрим программу обработки прерывания от принтера для вывода одиночного символа. Адрес буфера вывода расположен в ячейках с адресами 1000Н и 1001Н. В буфере вывода находится текст печати, конец которого помечен символом * . Адрес регистра данных принтера 10Н. Тогда программа имеет вид:

```
PRINT:  PUSH   PSW   ;СОХРАНЕНИЕ А И РЕГИСТРА F
        PUSH   H     ;СОХРАНЕНИЕ РЕГИСТРОВ H,L
        LHLD   1000  ;ЗАГРУЗКА В HL УКАЗАТЕЛЯ БУФЕРА
        MOV    A,M   ;ПЕРЕСЫЛКА ОЧЕРЕДНОГО СИМВОЛА В А
        CPI    '*'   ;СРАВНЕНИЕ С КОНЦОМ ТЕКСТА
        JZ     1200  ;ПЕРЕХОД ПО АДРЕСУ КОНЦА ТЕКСТА
        OUT    10H   ;ВЫВОД СИМВОЛА В ПОРТ ПРИНТЕРА
        INX    H     ;УВЕЛИЧЕНИЕ УКАЗАТЕЛЯ БУФЕРА
        SHLD   1000  ;ЗАПОМИНАНИЕ УКАЗАТЕЛЯ БУФЕРА
        POP    H     ;ВОССТАНОВЛЕНИЕ РЕГИСТРОВ H,L,
        POP    PSW   ;A И F
        RET
```

Специальные прерывания. МикроЭВМ "Электроника 60" располагает несколькими командами специальных прерываний, которые обеспечивают передачу управления на другие программные модули (программы отладки, специальные программы пользователя, программы управления вводом-выводом и др.). К ним относятся команды: EMT, TRAP, IOT, BPT (см. табл. 1.20).

Команды специальных прерываний выполняются так же, как и обычные прерывания. Адрес вектора прерывания (первого слова вектора прерывания) автоматически формируется при выполнении команды прерывания и имеет следующие значения: 30 – для команды EMT, 34 – для команды TRAP, 20 – для команды IOT, 14 – для команды BPT. Второе слово вектора (новое значение ССП) читается из следующей ячейки ОП.

Формальное описание команд прерывания приведено в табл. 1.20. Признаки N , Z , V , C при выполнении команд прерывания загружаются из вектора прерывания в соответствии с новым значением ССП. Команда BPT обычно используется для инициирования прерывания по Т-разряду в программах отладки.

Команды прерывания программы состоят только из кода операции, причем для EMT и TRAP значения разрядов К(7:0) несущественны. Для программиста число в младшем байте видно из следующего примера:

```
  1    000034            TRAPPC=34                    ;TRAP-ВЕКТОР (PC
  2    000036            TRAPPS=36                    ; И PSW)
                         ...
 14    000142  012767    START:  MOV     XFER,TRAPPC  ;УСТАНОВИТЬ PC В TRAP-
               001016'                                 ;ВЕКТОРЕ
               000034
 15    000150  005067            CLR     TRAPPS       ; И ЕГО PSW
               000036
                         ...
 49    000436  104404            TRAP    4            ;TRAP С КОДОМ 4
 50    000440                                         ;(СЛЕДУЮЩАЯ ИНСТРУКЦИЯ)
                         ...
 64    000512  104401            TRAP    1            ;TRAP С КОДОМ 1
 65    000514                                          ;(СЛЕДУЮЩАЯ ИНСТРУКЦИЯ)
                         ...
 88                      ;ЭТА ПОДПРОГРАММА ПЕРЕДАЕТ УПРАВЛЕНИЕ К УКАЗАН-
 89                      ;НОМУ МОДУЛЮ
                         ;
 90    001016  011603    XFER:   MOV     (SP),R3      ;ПОЛУЧ. АДРЕС ВОЗВРАТА
 91    001020  116303            MOVB    -2(R3),R3    ;ПОЛУЧИТЬ TRAP-КОД
               177776
 92    001024  006303            ASL     R3           ; И УДВОИТЬ ЕГО
 93    001026  000173            JMP     @DSPTCH-2(R3) ;ПЕРЕЙТИ "ЧЕРЕЗ ВЕКТОР"
 94            001110'                                 ;К УКАЗАННОЙ ПРОГРАММЕ
                         ...
                         ;
126                      ;ВЕКТОР ДЛЯ ПЕРЕДАЧИ УПРАВЛЕНИЯ К ПРОГРАММАМ
127                      ;
128    001112  001266'   DSPTCH:         .WORD   MOD1
129    001114  001344'           .WORD   MOD2
130    001116  001410'           .WORD   MOD3
131    001120  001436'           .WORD   MOD4
```

```
                        ...
153   001266          MOD1:                   ;ТОЧКА ВХОДА В МОДУЛЬ 1
                        ...
167   001342  000006        RTT               ;ВОЗВРАТ
168
169   001344          MOD2:                   ;ТОЧКА ВХОДА В МОДУЛЬ 2
                        ...
181   001406  000006        RTT               ;ВОЗВРАТ
182
183   001410          MOD3:                   ;ТОЧКА ВХОДА В МОДУЛЬ 3
                        ...
195   001434  000006        RTT               ;ВОЗВРАТ
196
197   001436          MOD4:                   ;ТОЧКА ВХОДА В МОДУЛЬ 4
                        ...
212   001470  000006        RTT               ;ВОЗВРАТ
```

Полагаем, что в данном случае программный сегмент включает четыре подпрограммы, точки входа которых содержат метки MOD1, MOD2, MOD3, MOD4. Необходимо иметь возможность войти в любой из этих модулей с помощью инструкции TRAP , но без модификации первого слова PC TRAP-вектора. В строке 14 адрес XFER (001016) помещается в первое слово TRAP-вектора (ячейку 000034), а второе слово этого вектора очищается в строке 15, поскольку ничего другого сделать невозможно. Проследим детально действие инструкции TRAP4 в строке 49.

Когда PC содержит 0000436, ЦП извлекает инструкцию TRAP (а именно 104404) и увеличивает его значение до 000440. Инструкция TRAP помещает в стек содержимое PSW (каким бы оно ни было) и содержимое ячейки 00034 (001016) помещает в PC, а содержимое ячейки 000036 (000000) – в PSW. Таким образом, выполнение возобновляется с ячейки, имеющей метку XFER. Инструкция MOV (SP),R3 пересылает (но не выталкивает) содержимое слова на вершине стека в R3 так, что (R3) становится равным 000440. Источником инструкции MOVB 2(R3), R3 является содержимое байта памяти, адрес которого равен (R3) – 2-000436, т. е. младшему байту инструкции TRAP4. Поскольку этот байт содержит число 004 и приемником инструкции MOVB является R3, к моменту полного завершения инструкции (R3) будет равно 000004. Следующая инструкция (в строке 92) удваивает это значение, так что теперь (R3) равно 000010. В инструкции JMP в строке 93 адрес (приемника) вычисляется следующим образом. Индекс DSPTCH-2 (001110) прибавляется к R3 (000010) и в результате появляется значение 001120. Поскольку в инструкции JMP используется косвенная адресация (@), адресом приемника будет содержимое ячейки 001120, а именно 001436-MOD4.

Таким образом, число 4, которое мы назовем TRAP-кодом, привело к передаче управления подпрограмме, точка входа которой является четвертым элементом в векторе DSPTCH (так называемом векторе диспетчеризации). Выполнение подпрограммы MOD4 завершается инструкцией RTT. Верхний элемент стека, а именно число 000440, выталкивается из стека в PC так же, как и "старое" содержимое PSW , и, таким образом, выполнение программы возобновляется с ячейки 000440, т. е. со следующей инструкции в стержневой ветви программы. Аналогично можно проанализировать инструкцию TRAP1 в строке 64.

Данная команда нередко используется при написании различных модулей, составляющих операционную систему. Короче говоря, инструкция TRAP никогда не предназначалась для обычного каждодневного программирования.

Программирование ввода-вывода в МП 1810ВМ86. В МП 1810ВМ86 ввод-вывод осуществляется по командам IN и OUT . Эти команды только инициируют передачу данных, а ВУ само должно распознать обращение к нему. Можно назначить несколько портов одному ВУ (например, для обмена управляющими сигналами). Операторы IN и OUT имеют две формы:

1) в поле операнда можно указать регистр DX . Число допустимых портов в этом случае 65536 (64 К). Например:

```
OUT DX, AX; содержимое AX передается в порт по адресу DX;
IN AL, DX;  содержимое в AL передается из порта по адресу DX;
```

2) адрес порта задается однобайтовой непосредственной величиной. Число допустимых портов 256. Если указан регистр AL , то обмен идет байтами, если регистр AX – то словами. Например:

```
IN AX, 0D0;  содержимое порта по адресу D0 (8 бит) передается в AX;
OUT 0A1, AL; содержимое по адресу AL передается в порт A1.
```

В общем случае идентификация порта производится следующим образом: а) номер порта может содержаться в восьми битах команды в виде непосредственных данных (число портов до 256); б) номер порта заносится в регистр DX длиной в слово (число портов до 65 536). При вводе-выводе обмен может идти словами (через регистр AX) или байтами (через регистр AL); в) порты ввода-вывода, выполняющие обмен байтами, могут быть связаны с шиной данных по младшим (0–7) или старшим (8–15) разрядам. Если адрес порта четный, то передача идет по младшим разрядам, если нечетный – по старшим.

В МП 1810ВМ86 возможна обработка двух видов прерываний – внешних и внутренних. Внешние прерывания инициируются периферийными устройствами, выставляющими запрос на линию прерываний. Внутренние прерывания возникают при выполнении программ. В МП реализована система векторов прерываний, которые используются для указания адресов подпрограмм, обслуживающих прерывания. Обработка сигналов прерываний (внешних и внутренних) заключается и в том, что сначала в стек засылается содержимое регистра F , а затем – регистра сегмента CS и счетчика команд IP. Потом осуществляется косвенный вызов подпрограммы обработки прерывания через соответствующий вектор обработки прерываний, расположенный по абсолютным адресам от 0 до 3FFH . Каждый вектор прерываний включает четыре байта, в первых двух которого содержится смещение для подпрограммы, а во вторых – адрес сегмента, в котором находится подпрограмма. Смещение посылается в счетчик команд IP, адрес сегмента в CS.

В МП существует 255 векторов прерываний (от 0 до 255). Векторы с номерами 0–61 (адреса 0H–3FH) зарезервированы для системных аппаратных и программных средств. Например, 0 – деление на нуль, 1 – пошаговый режим, 2 – немаскированное прерывание, 4 – прерывание по переполнению и т. д.

При обработке прерывания сбрасываются биты условий TF и IF . После выполнения программы обработки прерывания по команде IRET управление передается по адресу возврата. Для этого из стека восстанавливается содержимое программного счетчика IP , сегментного регистра CS и регистра битов условий F .

В качестве примера рассмотрим формирование векторов прерываний и вызов подпрограмм, обслуживающих прерывания. Векторы прерывания находятся в абсолютном сегменте, расположенном по адресу 0:

```
          INT2  ;ВЫЗОВ TYPE2
          ...
          INT6  ;ВЫЗОВ TYPE6
 INVECTORS SEGMENT AT 0
       ORG 08H
       DD TYPE 2        ; ПРЕРЫВАНИЕ ПО ВЕКТОРУ 2
       ORG 18H
       DD TYPE 6        ; ПРЕРЫВАНИЕ ПО ВЕКТОРУ 6
INVECTORS END S
 INTPROC SEGMENT
 TYPE 2 PROC FAR
    ...                 ; ОПЕРАТОРЫ ПОДПРОГРАММЫ
    IRET
 TYPE 2 END P
 TYPE 6 PROC FAR
    ...                 ; ОПЕРАТОРЫ ПОДПРОГРАММЫ
    IRET
 TYPE 6 END P
 INTPROC END S
```

По оператору DD (резервирование двойного слова) будут сформированы адреса процедур TYPE 2 и TYPE 6 (адрес сегмента и смещение). Сами процедуры обработки прерывания могут появиться в другом сегменте, определенном пользователем. Далее в программе могут встретиться операторы INT2, INT6.

Ввод-вывод в режиме ПДП. В быстродействующих ВУ (например, в накопителях на дисках и лентах) необходимо передать большой объем данных за короткий временной интервал. Это достигается посредством передач данных непосредственно между памятью и ВУ. Так как МП в передаче не участвует, таким способом обмена является ПДП. Когда ВУ инициирует операцию ПДП (рис. 1.23), передачей блока данных между памятью и ВУ управляет модуль ПДП без участия МП. Физически контроллер ПДП представляет собой отдельный модуль, который подключается к системной шине и может быть частью интерфейса ввода-вывода. Контроллер ПДП занимает цикл памяти у текущей программы, и слово данных передается в/из памяти по адресу, определяемому специальным регистром адреса в модуле ПДП, называемом регистром-указателем буфера. Далее автоматически проводится инкремент этого регистра при каждой передаче данных, при этом блок данных записывается в последовательные ячейки памяти. Когда счетчик слов достигает нуля, модуль ПДП извещает МП о завершении передачи.

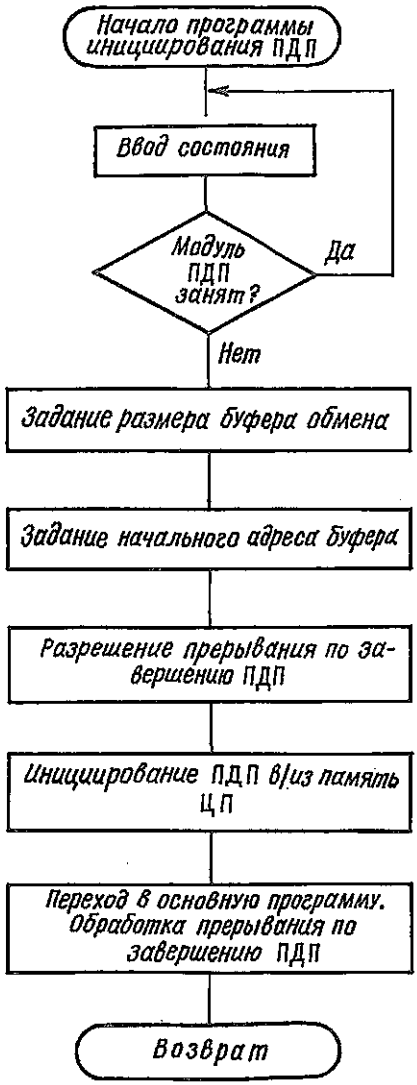


Рис. 1.23

В процессе инициализации контроллера ПДП в его регистры заносится начальный адрес блока данных и его длина.

Рекомендуемая литература: 1, 3–5, 13–15.

2. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА ДЛЯ МИКРОЭВМ

2.1. ЯЗЫК АССЕМБЛЕРА ДЛЯ 8-РАЗРЯДНОГО МП КР580ИК80А

Основные понятия. В операторах языка ассемблера для 8-разрядного МП КР580ИК80А используются следующие символы: 26 букв латинского алфавита A–Z , цифры от 0 до 9, специальные знаки, изображенные на клавишах + – ' */, ( ) : @? = < >%!. – ; ВК, ПС (здесь ВК – возврат каретки, ПС – перевод строки).

Оператор языка включает четыре поля: имени, операции, операндов, комментария. Список разделителей, распознаваемых ассемблером, следующий: ␣ — разделение полей оператора; , — разделение операндов; '...' – ограничение строки символов; (...) – ограничение выражения; ВК – ограничение оператора; ; – выделение поля комментариев; : – ограничение метки.

Поле имени оператора может содержать либо метку, либо имя. Метка включает от одного до шести символов, начинается с буквы, отделяется двоеточием. Значение метки определяется счетчиком адреса при ассемблировании. Поле метки может быть пустым. Имя записывается по тем же правилам, что и метка, используется для псевдокоманд SET и EQU , но не отделяется двоеточием.

Поле операции содержит мнемонику кода машинной команды либо псевдокоманды или операторы макроопределения или макровызова.

В поле операндов определены данные, над которыми будет выполнена операция, либо к которым будет применена псевдокоманда.

Поле комментариев содержит произвольную информацию, поясняющую текущий оператор.

Приведем пример оператора на языке ассемблера:

M1:	MOV	B, C ;	Пересылка содержимого C в B
Метка	Операция	Операнды	Комментарии

Информация в поле операндов может быть представлена регистрами, парами регистров, непосредственными 8- или 16-разрядными данными. адресами. Она определяется шестнадцатеричными, десятичными, восьмеричными и двоичными данными, счетчиком команд, строкой символов, именами, метками операторов, выражениями.

Шестнадцатеричное число начинается с цифры и заканчивается символом H (если оно начинается с буквы, то впереди ставится 0). Например, 0FFH, 12H, 0ABH. *Десятичное число* в конце может определяться буквой D или может быть записано без нее. Например, 40D, 10, 123D, 123. *Восьмеричное число* определяется буквой Q в конце: например, 72Q, 10Q. *Двоичные данные* специфицируются буквой B в конце числа. Например, 10101100B, 00000001B.

Счетчик команд идентифицируется символом $ ($). Оператор ассемблера JMP $ + 10 образует машинную команду, которая выполнит переход по ад-

ресу на 10 больше адреса данной команды. Для представления символьной строки ее заключают в апостроф. Оператор определения строки имеет вид:

```
L1:  DB  'строка символов'
```

Метки, именующие операторы, имеют значение адреса первого байта команды

```
L1:  JMP N;   Переход на метку N
```

Типы имен. Имена в ассемблере могут содержать от одного до шести символов, включая буквы, цифры и знаки ? , @ , и начинаться с буквы. К зарезервированным именам относятся имена команд, псевдокоманд, регистров (A, B, C, D , E, H, L, SP, PSW, M) и счетчика команд $. Важным свойством имен является перемещаемость. Имена размещаются, начиная с нулевого адреса, или перемещаются в памяти с заданным смещением.

Имена с изменяющимся значением при перемещении являются перемещаемыми, а с неизменяющимся – абсолютными. Абсолютные и перемещаемые имена определяются в кодовых сегментах, задаваемых соответствующими псевдокомандами ASEC, CSEC.

Определение выражений. Выражение представляет собой комбинацию имен и данных, объединенных знаками операций. Для определения выражений используется пять групп операций: арифметические, логические, сдвига, сравнения и выделения части байта.

В языке ассемблера определены следующие арифметические операции: + , – , * , / , MOD (остаток от деления). Например, 27 MOD 8 равно 3.

Логические операции следующие: NOT (отрицание), AND (И), ORA (ИЛИ), XOR (сложение по модулю два). Эти операции выполняются над младшими битами операндов.

Операция сдвига включает сдвиг вправо U SHR X и сдвиг влево U SHL X. В первом случае операнд U сдвигается на X позиций вправо, во втором – на X позиций влево. В процессе сдвига значения, выходящие за границы байта, теряются. Например, если U = 10101101B, то U SHR 1 = 01011010, а если U SHL 2, то 00101011B. Сдвиг влево на позицию эквивалентен умножению на два, вправо на позицию – делению на два.

Операции сравнения таковы: EQ (равно), NE (не равно), LT (меньше), LE (не больше), GT (больше), GE (не меньше). В результате выполнения операций сравнения получается нулевой или единичный результат. Сравнение выполняется побитно, истинному значению результата соответствуют все единицы, ложному – все нули.

Сравнение двух операндов F1 и F2 выполняется в псевдокоманде IF P1 EQ P2. Если значение сравнения истинно, то выполняется следующая команда, иначе все команды блока IF < команды > ENDIF пропускаются.

В языке используются операции выделения байта: HIGH – выделение старшего байта 16-битового числа, LOW – выделение младшего байта.

Ассемблер рассматривает каждый оператор выражения как 16-битовое значение, изменяющееся в пределах от 0 до FFFFH. Арифметические операции выполняются с использованием дополнительного кода.

Вычисление значения выражения происходит слева направо с учетом приоритетов операций и скобок. Приведем последовательность операций в поряд-

ке убывания их приоритетов: 1) выражение в скобках; 2) HIGH, LOW; 3) * / , MOD , SHL , SHR; 4) + , – ; 5) EQ , LT , LE , GT, CE , NE; 6) NOT; 7) AND ; 8) OR, XOR. Все операции, кроме * , / , + , – , должны отделяться от операндов одним пробелом.

Рассмотрим примеры правильно записанных выражений:

1) MOV T –9, R1. Если T = 12 и R1 = 2, то выполнится пересылка данных из регистра D в регистр E;

2) SHLD BS – 3*(100–K1). Если BS и K1 имеют значения 1000 и 50, то значения регистров H , L засылаются в ячейки памяти 850 и 851 соответственно;

3) CPI A,(NOT(M1 AND 0FH)SHR 4);

4) при M1 = 0FH A сравнивается с 0.

Поле комментариев содержит любой пояснительный текст, который ассемблером не воспринимается, а появляется только в распечатках, начинается с символа.

Псевдокоманды. Операторы языка ассемблера, используемые для управления программой, разделяются на группы: определения имен (EQU, SET), определения данных (DB, DW), определения памяти (DS), условного ассемблирования (IF , END IF , ELSE), прекращения ассемблирования (END), управления программными сегментами (ORG, ASEC, CSEC, DSEC). Рассмотрим их назначение.

Псевдокоманда EQU (равно) присваивает расположенному в поле имени значение выражения в поле операнда и имеет вид

< метка > EQU выражение

Например, псевдокоманда ONE EQU 0F1H присваивает значение имени ONE OF1. Псевдокоманда SET имеет то же назначение, что и псевдокоманда EQU , за исключением того, что одно и то же имя может быть многократно переопределено по SET в программе.

Псевдокоманда DB (определить байт) резервирует некоторое число байт (не более 255) и определяет их начальное значение. Ее формат

метка : DB < список >

Например, M1: DB 01H, M1+5.

Псевдокоманда DW (определить слово) определяет в памяти 16-битовые слова и задает их начальное значение. Ее формат

метка : DW < список >

Оператор DW обычно используется в программе для запоминания адреса. Пусть метке M1 присвоено значение 1600, метке K2 – значение 1000H. Тогда выражение M1: DW K2, K2+5 ассемблируется в виде 1600–00; 1601–10 ; 1602–05; 1603–10 (в памяти адрес начинается с младшего байта).

Псевдокоманда DS (определить память) позволяет зарезервировать одну или несколько ячеек памяти (до 0FFFFH ячеек) и имеет формат

метка : DS < выражение >

Например, операторы M1: DS 10; M2: DS 0AH резервируют по 10 ячеек памяти. Все метки, используемые в выражении, должны быть заранее определены.

Условное ассемблирование. Псевдокоманды IF, ELSE,ENDIF позволяют ассемблировать фрагменты программы в случае выполнения заданного условия в виде

```
имя :  IF < выражение >
       группа команд 1
имя :  ELSE
       группа команд 2
имя :  ENDIF
```

Ассемблер анализирует выражение, если младший бит в значении выражения равен 1, ассемблируются все команды, расположенные между псевдокомандами IF и ELSE (если ELSE отсутствует, то между IF и ENDIF). Если нулевой бит в значении выражения равен 0, все команды между IF и ELSE игнорируются, а между ELSE и ENDIF ассемблируются:

```
C1: IF X1 EQ 0      ; АССЕМБЛИРУЮТСЯ, ЕСЛИ ВЫРАЖЕНИЕ
    ...             ; X1=0 ИСТИННО,
    ELSE            ; АССЕМБЛИРУЮТСЯ, ЕСЛИ ВЫРАЖЕНИЕ
    ...             ; X1=0 ЛОЖНО
    ENDIF
```

Псевдокоманда END фиксирует конец исходной программы и завершает просмотр ассемблером текста программы. Ее формат

```
< имя > : END<выражение>
```

Значение выражения в этой псевдокоманде определяет пусковой адрес программы. Если он опущен, то предполагается, что он равен нулю.

Управление программными сегментами. Псевдокоманда ORG устанавливает счетчик команд в соответствии со значениями выражения в поле ее операнда и имеет формат

```
< имя > : ORG < выражение >
```

Значение выражения должно лежать в диапазоне 0—FFFFH. Метка в псевдокоманде обязательна. При обработке выражения счетчик адреса принимает значение выражения. Следующая команда будет помещаться по этому адресу.

Например, пусть необходимо сложить два десятичных 12-разрядных числа, которые занимают 6 байт (разряд — 4 бита) и расположены в памяти с младших разрядов. Программа имеет вид

```
ADDD: ORG 1000
      LXI H, ADR1   ; ЗАНЕСЕНИЕ АДРЕСА 1 ЧИСЛА В H,L
      LXI B, ADR2   ; ЗАНЕСЕНИЕ АДРЕСА 2 ЧИСЛА В B, C
      XRA A         ; СБРОС АККУМУЛЯТОРА
      MVI D, 6      ; ЗАГРУЗКА СЧЕТЧИКА ЦИКЛА
M1:   LDAX B        ; ЗАНЕСЕНИЕ ДВУХ ЦИФР 2 ЧИСЛА В АК
      ADC M         ; СЛОЖЕНИЕ ИХ С 2 ЦИФРАМИ 1 ЧИСЛА С УЧЕТОМ
                    ; ПЕРЕНОСА
```

```
        DAA             ; ПРЕОБРАЗОВАНИЕ РЕЗУЛЬТАТА В ДЕСЯТИЧНЫЙ КОД
        STAX B          ; ЗАПОМИНАНИЕ ДВУХ ЦИФР РЕЗУЛЬТАТА
        INX  H          ; УВЕЛИЧЕНИЕ АДРЕСА 1 ЧИСЛА
        INX  B          ; УВЕЛИЧЕНИЕ АДРЕСА 2 ЧИСЛА
        DCR  D          ; УМЕНЬШЕНИЕ НА 1 СЧЕТЧИКА ЦИКЛА
        JNZ  M1         ; ПЕРЕХОД ПО НЕ НУЛЮ СЧЕТЧИКА ЦИКЛА
ADR1:DB 64,52,46,32,44,33 ; ПЕРВОЕ ЧИСЛО
ADR2:DB 55,44,32,22,11,18 ; ВТОРОЕ ЧИСЛО
        END             ; КОНЕЦ ПРОГРАММЫ.
```

Макросредства. Эти средства позволяют размещать указанные в них наборы команд в программе несколько раз, модифицируя параметры. Повторяющейся группе команд, называемой макроопределением, присваивается некоторое имя. Употребление этого имени при написании программы называется макровызовом. Первый оператор макроопределения MACPO имеет вид

< имя > MACPO < список формальных параметров >

В качестве списка формальных параметров допустимо использование любых имен, разделенных запятыми. Этот список может быть также пустым. В макроопределение могут быть включены любые машинные команды и некоторые псевдокоманды.

Для завершения макроопределения записывается оператор ENDM без полей имени и операторов. Если использовать одно макроопределение, содержащее метку, многократно, то ассемблер выдает сообщение об ошибке. Для устранения этой ситуации вводится оператор LOCAL, который при каждом использовании одного макроопределения создает в нем уникальное значение имен меток. Например, для сдвига аккумулятора на несколько разрядов можно использовать следующее макроопределение.

```
SHL MACRO R1, N1    ; ЗАГОЛОВОК
    LOCAL M1        ; ЛОКАЛЬНОЕ ОПРЕДЕЛЕНИЕ МЕТКИ
    MVI   R1, N1    ; ЗАГРУЗКА СЧЕТЧИКА
M1: RLC             ; СДВИГ АК ВПРАВО
    ANI   FEH       ; ОЧИСТКА МЛАДШЕГО БИТА
    DCR   R1        ; ДЕКРЕМЕНТ СЧЕТЧИКА
    JNZ   M1        ; ПЕРЕХОД ПО НЕ НУЛЮ СЧЕТЧИКА
    END   M         ; КОНЕЦ МАКРООПРЕДЕЛЕНИЯ
```

Если теперь в программе встретится макровызов SHL B, 4 , то вместо него будут подставлены соответствующие команды, которые сдвинут содержимое аккумулятора на четыре разряда влево:

```
LDA A1          LDA A1
SHL B,Y         MVI B,Y
STA A1      M1: RLC
                ANI 0FEH
                DCR B
                JNZ M
                STA A1
```

При написании макроопределения могут использоваться псевдокоманды условного ассемблирования IF и ENDIF. Ассемблер вычисляет выражение, стоящее в поле операнда псевдокоманды IF . Если результат равен 0, то ко-

манды между IF и ENDIF в результирующую программу не включаются (включаются в противном случае).

Псевдокоманды перемещаемого ассемблера. Перемещаемый ассемблер позволяет отлаживать программы в одной области памяти (ОЗУ), а затем перемещать их в другую область памяти (ОЗУ или ПЗУ).

Перемещаемые программные модули используют три счетчика адресов, управляемых псевдокомандами ASEC, CSEC, DSEC. Перед началом ассемблирования значение этих счетчиков равно нулю. При появлении одной из указанных псевдокоманд включается соответствующий счетчик. Это приводит к размещению команд и данных в соответствующем сегменте. Если появляется псевдокоманда другого сегмента, значение предыдущего счетчика запоминается и в работу включается счетчик нового сегмента.

Сегменты команд CSEC, DSEC являются перемещаемыми, их абсолютные адреса определяются программой-загрузчиком. Сегмент команд размещается в ПЗУ, а сегмент данных – в ОЗУ. Содержимое абсолютного сегмента ASEC не перемещается. При отсутствии псевдокоманд CSEC, DSEC в начале программы оно размещается в абсолютном формате.

При разделении задачи на отдельные программные модули в них могут применяться метки и переменные, определяемые в других модулях. Поэтому все метки и переменные, описанные в других модулях, к которым есть обращение в данном модуле, должны быть в нем описаны псевдокомандой EXTRN (внешние). Например, EXTRN A1, A2, A3; определение внешних меток.

Все метки, определяемые в данном модуле, к которым есть обращение из других модулей, должны быть описаны в псевдокоманде PUBLIC (общие). Например, PUBLIC SIN, COS; определение общих меток. Если в данном модуле метки определены в псевдокоманде PUBLIC, то в других модулях они должны появиться только в псевдокомандах EXTRN.

2.2. ОСНОВЫ ЯЗЫКА АССЕМБЛЕРА ДЛЯ МП К1810ВМ86

Виды операторов. Допустимые символы языка ассемблера состоят из прописных и строчных латинских букв, цифр, специальных знаков: + – * / = ( ) [ ] ; ' " . ; , @ & ¤ ? < > % и символов: перевод строки ПС (0АН), возврат каретки ВК (0DH), табуляции (09Н). Любой другой символ воспринимается как пробел. Наименьшей конструкцией модуля является идентификатор – последовательность букв и цифр не более 31, начинающаяся с буквы, @ , ?, включая табуляцию. Например, A1, B1234, TAB_12, MAX_SIZE . К зарезервированным именам относятся имена регистров, операций, псевдокоманд.

Модуль представляет собой последовательность операторов языка, записанных на одной строке и заканчивающихся возвратом каретки и переводом строки. Если в первой позиции оператора стоит символ & , то оператор является продолжением предыдущего. Ассемблер воспринимает операторы в свободном формате, между операндами может быть любое число пробелов. Операторы разделяются на командные и директивы. Первые порождают одну машинную команду. Директивы (псевдокоманды) содержат управляющую информацию для ассемблера. Оператор имеет вид:

{метка:} {префикс} {мнемоника} {операнд (ы)}; {комментарии}

Значение метки является текущим значением счетчика в данном сегмент кода, т. е. представляет собой адрес команды. Префикс позволяет сформирс вать байты блокировки LOOK или повторения REP. Мнемоника идентифиц рует тип генерируемой команды. В зависимости от функции команды може быть один оператор, два или ни одного. Более двух операторов указываютс в макрокоманде. Комментарии поясняют смысл команды.

Директивы ассемблера имеют несколько другой формат

{имя} директива {операнд (ы)} {;комментарий}

Имя директивы имеет другой смысл по сравнению с меткой и не заканчи вается двоеточием. В ряде директив имя отсутствует. Директивы используют ся для распределения памяти, связей между модулями, манипуляции с симво лами и т. д. В отдельных директивах допускаются списки операндов. Операн ды могут быть ключевыми словами в директивах PROC, SEGMENT. Для опре деления макрокоманды используется оператор вида

MACROCODE имя {операнд (ы)} ; {комментарии}

Операторы модуля состоят из ключевых слов, идентификаторов, кон стант, символьных цепочек и специальных символов. Переменная – это едини ца данных, имеющая имя. Она имеет три атрибута: сегмент, смещение и тип. Сегмент SEG определяет сегмент, содержащий переменную, смещение OFFSET – расстояние от начала сегмента до перем ной, тип – число байт пе ременной (1, 2 или 4). Метка, представляющая имя ячейки памяти, имеет атрибутами сегмент, смещение и расстояние. Последнее определяет возмож ность достижения метки командой передачи управления с помощью смещений двухбайтового NEAR или четырехбайтового (сегмент: смещение) FAR. Кон станта отличается от переменной и метки тем, что она определяет толькочисло. Числовые константы представляют с основаниями 2, 8, 10 и 16. Символьные цепочки заключаются в апостроф.. и обычно имеют длину до 255 знаков.

Директивы ассемблера. Для определения и инициализации данных предна значены директивы: DB – определить байт, DW – определить слово, DD – определить двойное слово, которые имеют формат

имя DX < начальное значение > ,[начальное значение]

Рассмотрим пример определения переменных:

```
F0 SEGMENT AT 50H
Z  DB 0              ; ОДИН БАЙТ /0/
Z1 DW 1              ; СЛОВО /0001H/
Z2 DD 2              ; СТАРШЕЕ СЛОВО 0050H; МЛАДШЕЕ 0002H
N1 DB                ; РЕЗЕРВИРОВАНИЕ БАЙТА
F0 END S

Z1    ..  ...        ;ОДИН БАЙТ, РАВНЫЙ AB
Z2    DW  1000H      ;ОДНО СЛОВО, РАВНОЕ 1000
Z3    DD  1235H:1000H ;МЛАДШЕЕ СЛОВО 1000, СТАРШЕЕ 1245
```

Для указания произвольного значения ячеек памяти используется символ ? . Для распределения и инициализации нескольких ячеек памяти введена конструкция DUP . Возможности конструкции DUP рассмотрим на следующем примере:

```
      DB  100 DUP(0)     ;СТО НУЛЕВЫХ БАЙТОВ
      DW  20  DUP(?)     ;ДВАДЦАТЬ СЛОВ БЕЗ ЗНАЧЕНИЯ
ADR   DD  10  DUP(ADR)   ;ДЕСЯТЬ ПОЛНЫХ АДРЕСОВ
      DB  10  DUP(80DUP(' '));ДЕСЯТЬ СЛОВ, СОДЕРЖАЩИХ 80 ПРОБЕЛОВ
```

В директивах DW, DD допускаются символьные цепочки длиной 1 и 2 символа, например DW 'K1'; DD 'G'.

В выражениях, где запрашивается тип используемой информации, применяются операторы TYPE, LENGTH, SIZE. Оператор TYPE сообщает число байт, отведенных для переменной (1, 2, 4); LENGTH определяет число единиц памяти переменной (байт, слов, двойных слов); SIZE сообщает размер переменной в байтах, причем размер переменной равен длине, умноженной на тип.

В языке ассемблера имеется возможность изменить тип или сегмент, связанный с переменной. Оператор замены типа PTR позволяет использовать конкретный тип переменной. Например, ADD AL, BYTE PTR A1 означает к AL прибавить байт. Этот оператор позволяет также устранить неоднозначность анонимных обращений. В команде SUB [BX] , 5 неясно вычитается байт или слово. При записи SUB WORD PTR [BX] , 5 неоднозначность снимается, т. е. ясно, что вычитается слово 0005, а SUB BYTE PTR [BX] , 5 — байт 05.

В ассемблере вводится понятие логического сегмента, под которым понимается часть программы, которая может включать сегменты для машинного кода, данных и стека. Каждый логический сегмент должен начинаться с директивы SEGMENT и заканчиваться директивой ENDS . Логическому сегменту присваивается имя, данное программистом, и список параметров (атрибутов), которые необязательны, но необходимы в случае программы, включающей несколько модулей. Приведем пример определения простого сегмента:

```
DATA    SEGMENT
F1      DB ?
F2      DW ?
F3      DD ?
DATA    ENDS
```

До использования сегментного регистра в формировании адреса он должен быть инициализирован, для чего используется имя логического сегмента. В нашем примере инициализируется регистр DS:

```
MOV  AX,DATA
MOV  DS,AX
```

До использования стека необходимо инициализировать регистры SS и SP . В общем случае в программе первыми командами являются команды инициализации регистров DS , SS , SP .

В языке ассемблера допускаются так называемые вложенные сегменты. В этом случае сначала определяется некоторый логический сегмент S1 , который затем снова появляется в новой директиве SEGMENT. Размещаемые при

этом новые команды (данные) будут располагаться за операторами, находящимися в старом сегменте S1 .

Необходимая ассемблеру информация о значении сегментных регистров сообщается в директиве ASSUME , которая имеет формат

ASSUME SR: базовое значение > , [SR : базовое значение] .

Поле SR содержит имя одного из сегментных регистров CS, DS, SS, E а базовое значение указывает на начало области памяти, адресуемой через этот регистр. Таким именем является имя сегмента, например ASSUME DS : DAT Данная директива сообщает, что к переменным в сегменте DATA можно обратиться через регистр DS . Если какой-нибудь сегмент не будет использоваться в модуле, его значение в директиве будет иметь вид ASSUME NOTHING. Заметим, что при правильном определении директивы ASSUME программист почти не заботится об операторах замены сегментов.

Основной внутренней переменной ассемблера является счетчик адресов, который загружается нуль при появлении директивы < имя > SEGMENT При распределении каждого нового байта производится инкремент счетчика Директива ENDS освобождает данный счетчик, который будет дальше использоваться при появлении прежнего имени сегмента. Текущее значение счетчика ячеек может быть изменено программистом с помощью директивы ORG < выражение >. При выполнении этой директивы вычисляется выражение, результат загружается в счетчик ячеек. Ассемблирование дальше производится по полученному адресу, который может быть в диапазоне 0–65535.

В ассемблере имеется средство определения символических имен для часто используемых выражений в виде директивы

< имя > EQU < выражение >

Допустим, в программе необходимо многократно применять размер таблицы. Тогда его можно определить так: SIZE_TAB EQU 100 , а затем использовать в любой команде, где необходим этот размер.

Для организации процедур в языке имеется две директивы PROC и ENDP Первая отмечает точку входа в процедуру, а вторая – окончание процедуры. Формат этих директив имеет вид:

```
< имя > PROC < тип >
        < Тело >
< имя > ENDP
```

Имя представляет точку входа в процедуру. Тип процедуры (NEAR FAR) по умолчанию NEAR используется ассемблером для определения вида генерируемой команды CALL для вызова процедуры. В соответствии с двумя командами вызова генерируются и две команды возврата в зависимости от типа процедуры. К переменным или меткам внутри тела процедуры можно обращаться из любого места программы. Для предотвращения случайного выполнения процедуры без вызова ее рекомендуется размещать выше тех программ, которые ее вызывают, а также избегать вложенных определений PROC–ENDP. Передача параметров между процедурами может организовываться через регистры, но чаще всего через стек. Процедура, которая

возвращает одно значение, называется процедурой-функцией. Для обеспечения нескольких точек входа в процедуру используется директива LABEL, которая записывается в виде < имя >LABEL тип. Если в директиве LABEL в качестве типа используется BYTE, WORD, DWORD, то имя является переменной. Имя с типом FAR, NEAR представляет метку, на которую может быть передано управление. В примерах

```
BUF LABEL WORD; SIN1 LABEL NEAR
```

первое имя определяет переменную, а второе – метку SIN1.

Для описания переменных, которые определены в данном модуле (обращение к ним может быть из другого модуля), служит директива PUBLIC . Для определения переменной, которая для данного модуля является внешней, используется директива EXTRN. Формат этих директив

```
PUBLIC < имя > , [< имя > ]...
EXTRN < имя:тип > , [< имя :тип> ]...
```

Имя в обеих директивах является именем переменной или метки, а тип может быть BYTE, WORD, DWORD для переменной и NEAR , FAR для метки. Приведем примеры этих директив:

```
PUBLIC SIN , COS , TAN
EXTRN ENTRY : BYTE, ADR1:FAR , S1:WORD.
```

Заметим, что ставить метку в этих директивах не разрешается. Если ассемблер встречает внешнее имя, он не может присвоить значение адреса такому имени и сформировать машинную команду. Вместо этого в команде резервируется место, а в файле делается об этом отметка. При объединении нескольких объектных файлов редактор связей просматривает эти отметки и, используя информацию о глобальных именах, формирует необходимые адреса или значения.

Директива NAME < имя модуля > присваивает внутреннее имя объектному модулю, генерируемому ассемблером. Это необходимо для обращения к модулям и определения порядка их следования.

Директива END < пусковой адрес > определяет конец исходного модуля. Она используется ассемблером для окончания проходов трансляции. Пусковой адрес представляет ту метку команды, начиная с которой будет выполняться программа. Если модуль не является главным, поле операнда в директиве END пустое.

Для определения отдельных бит и битовых цепочек внутри байта, слова используется директива RECORD, которая имеет формат

```
имя RECORD < имя поля: длина > , [< имя поля: длина > ]...
```

Максимальное число бит, указываемое в директиве, равно 16. Приведем пример определения записи

```
E1 RECORD A1 : 4 , D1 : 5 , P1 : 7
```

Для обеспечения доступа к переменным со сложным типом данных используется директива STRUCTURE . Поля структуры в отличие от полей записи образуются из байт, слов и двойных слов и определяются с помощью ди-

ректив DB , DW , DD . При записи память структуре не распределяется. Вмест этого структура связывается с конкретной областью памяти, когда имя пол фигурирует в команде вместе с базовым адресом. Рассмотрим пример:

```
RAB1 STRUC          ;ИМЯ СТРУКТУРЫ
R1   DW ?           ;ОПРЕДЕЛЕНИЕ НОВОГО ТИПА
R2   DW ?           ;ДАННЫХ ИЗ ДВУХ СЛОВ
RAB1 ENDS           ;КОНЕЦ СТРУКТУРЫ
A1   RAB1 (26)      ;ОПРЕДЕЛЕНИЕ ПЕРЕМЕННОЙ ПО ШАБЛОНУ
     MOV AX,A1.R2 ;ЗАГРУЗКА В АХ ПОЛЯ R2 ПЕРЕМЕННОЙ A1
```

Выражение. Наиболее сложными видами операндов ассемблерных стро являются выражения. Важные элементы выражений – переменные и метки, также числовые выражения, определенные директивой EQU. Адресное выра жение вычисляется для получения адреса памяти и имеет три атрибута: сег мент, смещение и тип.

Максимальный диапазон чисел составляет от –FFFFH до FFFFH. Вс арифметические операции выполняются в дополнительном коде. В выражени ях могут использоваться арифметические операторы: + (сложения), – (вычи тания), * (умножения), / (деления), а также унарные операторы плюс и минус. Кроме того, используются буквенные операторы вычисления остатка MOD и сдвигов SHR, SHL. Сложение переменных и меток не допускается, а их вычитание возможно, если они находятся в одном сегменте. Умножение, деле ние, сдвиги и вычисление остатка производятся только над числами.

В выражениях могут использоваться бинарные логические операторы AND, OR и XOR , выполняющие логические операции над отдельными битами операндов, а также оператор отрицания NOT . Все логические операторы производят операции только с числами.

В языке ассемблера применяются стандартные операторы отношений GT, GE, EQ, NE , LT , LE . Результатом этих операторов является булева переменная со значениями единиц (истина) или нулей (ложь) во всех 16 разрядах. Например, запись MOV AH, 10 AND K EQ M приведет к генерированию команды MOV AH, 10 или MOV AH,0 в зависимости от равенства значений переменных K и M.

Операторы SEG , OFFSET предназначены для нахождения чисел путем выделения атрибутов: сегмента, смещения, переменных или меток. Например, SEG P1 = SEG C1 = DATA, OFFSET B1 = 10.

Операторы HICH и LOW выделяют старший и младший байты переменной типа слова. Например, HIGH D1 = 10 (D1 = 1020).

Вычисление выражения производится слева направо с выполнением в первую очередь более высокоприоритетных операторов. Операторы в порядке старшинства располагаются следующим образом: 1) логические операторы OR и XOR; 2) AND ; 3) NOT ; 4) операторы отношений GT , GE , EQ , NE , LT , LE ; 5) операторы арифметические + , – ; затем * , / , MOD, SHR , SHL; 6) операторы HIGH , LOW .

Рассмотрим законченный исходный модуль, демонстрирующий общие особенности оформления программ для МП 1810ВМ86:

```
NAME PROG1                  ;ИМЯ ПРОГРАММЫ
ASSUME CS:COD,DS:DT,SS:ST ;НАЗНАЧЕНИЕ СЕГМЕНТНЫХ РЕГИСТРОВ
   DT SEGMENT               ;СЕГМЕНТ ДАННЫХ
   V1 DB 5                  ;ОПРЕДЕЛЕНИЕ И ИНИЦИИРОВАНИЕ
   V2 DB 0                  ;ДАННЫХ
   DT ENDS
 ST    SEGMENT              ;СЕГМЕНТ СТЕКА
       DW 100DUP(?)         ;РЕЗЕРВИРОВАНИЕ ОБЛАСТИ 100
  ST_TOP LABEL WORD         ;БАЙТОВ ПОД СТЕК
 ST    ENDS
 COD SEGMENT                ;СЕГМЕНТ КОДА
 S1:   MOV AX,DT            ;ИНИЦИАЛИЗАЦИЯ СЕГМЕНТА
       MOV DS,AX            ;ДАННЫХ
       MOV AX,ST            ;ИНИЦИАЛИЗАЦИЯ СЕГМЕНТА
       MOV SS,AX            ;СТЕКА
       MOV SP,OFFSET ST_TOP;ЗАГРУЗКА ВЕРШИНЫ СТЕКА
 P1:   PUSH AX,DT           ;СОХРАНЕНИЕ АХ
       MOV AL,V1            ;ВЫПОЛНЕНИЕ
       SUB AL,2             ;АРИФМЕТИЧЕСКИХ
       MOV V2,AL            ;ДЕЙСТВИЙ
       POP AX               ;ВОССТАНОВЛЕНИЕ АХ
 COD   ENDS
       END S1               ;КОНЕЦ ПРОГРАММЫ
```

Сначала модулю присваивается имя PROG1 директивой NAME . Он включает три сегмента DT , ST , COD , каждый из которых начинается с директивы SEGMENT и кончается директивой ENDS. Имена логических сегментов присваиваются физическим. В сегменте данных определяются и инициализируются две переменные. В сегменте стека резервируется область в 100 слов, на границу которой устанавливается вершина стека ST_TOP . В сегменте данных D1 пять команд MOV инициализируют регистры DS , SS и указатель стека SP . Затем начинается программа, которая выполняет иллюстративные действия и завершается директивой END S1.

Рассмотрим программу обработки прерывания от клавиатуры. Первая команда сохраняет регистр AX в стеке. Далее производится чтение кода клавиши из порта клавиатуры 60H. Затем программно необходимо сбросить прерывание от клавиатуры. Для этого нужно в выходной порт с адресом 61H установить 1 в седьмом бите, что не только отменит запрос на прерывание, но будет сообщением для клавиатуры о посылке следующего символа:

```
CODE SEGMENT
 ASSUME CS: CODE
K_INT PROC FAR
PUSH AX;Сохранение AX
IN   AL,60;Выбираем номер клавиатуры
PUSH;AX   ;Сохранение кода клавиатуры
IN   AL,61;Ввод состояния
MOV  AH,AL;Сохранение AL
```

```
OR    AL,80;Добавление 128 к коду
OUT   61H,A;Сигнал об успешном получении
MOV   AL,AH;Восстановление AH
OUT   61H,AL;Возврат кода
            клавиши клавиатуры
MOV   AL,20;Сообщение о посылке символов
OUT   20H,AL;Сигнал об окончании прерывания
POP   AX
IRET
K_INT ENDP
CODE  ENDS
```

Для подключения печатающего устройства используется контроллер параллельного интерфейса. На программном уровне этот контроллер имеет три порта с адресами 378H–37AH (один порт ввода с адресом 379H и два порта вывода – один для вывода символа, который необходимо напечатать, и другой управляющий для формирования сигналов управления принтером). Пусть пятый бит этого порта для синхронизации передачи кода символа сначала устанавливается в 1, а затем в 0. Таким способом программно формируется код строба формой ⎍.

Рассмотрим программу драйвера, которая передает в принтер один символ для печати. Сначала устанавливается имя порта 378 H. Затем производится вывод символа из МП в порт. Прежде чем передать этот символ в принтер, проверяется его состояние. Для этого читается код из порта с адресом 379H. Если в 7-м разряде отсутствует 1, то принтер не готов к печати и опрос зацикливается до готовности. После того как принтер готов, программно формируется строб, для чего сначала в порт выдается код 0BH (00001011B), затем код 0AH (00001010B). Это связано с тем, что для установки любого бита в порту с адресом 37AH последний бит кода определяет значение установки (в нашем случае сначала 1, а затем 0), а три предпоследних кодируют номер устанавливаемого бита (в нашем случае 5). Затем восстанавливаются сохраненные в стеке регистры DX,AX и выход из программы обработки прерываний по команде IRET . Приведем текст программы:

```
BASE EQU 378;Адрес порта вывода
CODE SEGMENT
     ASSUME CS:CODE
MSG  DB  'AA'13,10,'$'
MAIN PROC  FAR
     PUSH DS;Адрес возврата
     LEA BX,MSG
LOOP:MOV AL,CS:[BX];Выбор символов
     CMP AL,'$';Сравнение с концом
     JE  M_RET;Переход
     CALL PRIN;Обращение к ПП
     INC BX   ;Увеличение BX
     JMP LOOP ;Цикл
M_RET RET
MAIN ENDP
PRIN PROC   NEAR
     MOV DX,BASE
     OUT DX,AL
     INC DX
BISY:IN  AL,DX;Опрос состояния
     TEST AL,80H;Бит 7 в 1?
     JZ  BISY;Цикл,если бит 7=0
```

```
        INC DX  ;Переход в порт 37A
        MOV AL,ODN;Установка разряда
        OUT DX,AL
        MOV AL,OCH;Сброс разряда
        OUT DX,AL
        RET
PRIN    ENDP
CODE    ENDS
        END   MAIN
```

2.3. ЯЗЫК АССЕМБЛЕРА ДЛЯ МИКРОЭВМ СЕМЕЙСТВА "ЭЛЕКТРОНИКА 60"

В стандартном языке ассемблера микроЭВМ семейства "Электроника 60" используется свободный формат записи. В этом языке за меткой следует двоеточие, а перед полем комментария находится точка с запятой:

метка:	оператор или директива	операнд$_1$,	операнд$_2$;	комментарий

Приведем несколько строк, правильно записанных на языке ассемблера:

```
START:MOV X,Y;ЭТИ ДВЕ СТРОКИ НАПИСАНЫ СЖАТО
SUB IND,COUNT
;    ОСТАЛЬНЫЕ СТРОКИ ИМЕЮТ СТАНДАРТНЫЙ ВИД
;
NEXT:  ADD Z,Y   ;ОЧЕНЬ УДОБНО,КОГДА ПОЛЕ КОММЕНТА-
       MOV Y,X   ;РИЕВ,А ТАКЖЕ ДРУГИЕ ПОЛЯ ВЫРОВНЕНЫ
;***   ИНОГДА ОТВОДИТСЯ ВСЯ СТРОКА ДЛЯ СООБЩЕНИЯ ***
```

Алфавит языка ассемблера содержит: 1) латинские буквы от A до Z; 2) цифры от 0 до 9; 3) знаки начальных символов системных программ · (точка); 4) специальные знаки:; (начало комментария); : (ограничитель метки); = (прямое присваивание);% (признак регистра); # (признак непосредственной адресации); @ (признак косвенной адресации); (,) + – / ! (ИЛИ) & (И).

Обозначения, используемые при описании команд языка: PC – счетчик команд; R_n – один из восьми общих регистров (R0, R1, R2, R3, R4, R5, R6, R7); PS – регистр состояния процессора; SP – указатель стека R6; R7 – счетчик команд PC.

Приведем важные свойства языка ассемблера: 1) целочисленные константы представляются в восьмеричной системе счисления, если константа не сопровождается точкой, которая служит для представления десятичных чисел; 2) в символьных уравнениях используется знак равенства; значение символа может присваиваться несколько раз; 3) номера регистров отличаются от адресов памяти предшествующим знаком процента (%); обычно это происходит при вводе символов в начале программы (например, SP = %6, PC = %7); 4) счетчик ячеек программы обозначается точкой; она может находиться с любой стороны от символа присвоения (=); 5) оператор .=< число > (напри-

Таблица 2.1

Название директивы	Пример	Действие директивы
.TITLE	.TITLE FL.FIX	Присвоение наименования объектному модулю
.GLOBL	.GLOBL A, FFT	Описание имен как глобальных
.ASECT	.ASECT . = 4 '	Указывает на начало абсолютной секции
.CSECT	.CSECT	Начало относительной секции
.WORD	WT:.WORD 17, –789., CAT	Размещение констант в последовательных ячейках памяти (слово за словом)
.BYTE	BT:.BYTE 17, 99., A	Размещение констант в последовательных ячейках памяти (байт за байтом)
.ASCII	AT:.ASCII/ABCDEFG/	Размещение символов кода в последовательных ячейках памяти (байт за байтом)
.EVEN	.EVEN	Присвоение счетчику программы четного значения (т.е. производится положительное приращение, если значение PC нечетно). Эта директива используется после директив .BYTE, .ASCII
.EOT	.EOT	Указывает на физический конец исходной ленты
.END	.END START	Окончание трансляции и предоставление произвольного начального адреса (указывает на физический и логический конец исходного модуля)

мер, . = 1000) устанавливает начальное значение счетчика ячеек программы; 6) оператор должен иметь вид мнемоники инструкции с одним или двумя операндами или без них; 7) имена пользователя должны быть определены хотя бы в одном из исходных модулей, входящих в программу. Имя определено в данном исходном модуле, если оно встречается один раз в поле метки некоторого оператора или находится слева от знака = в операторе прямого присваивания; 8) имена директив языка начинаются с точки. Наиболее важные из них представлены в табл. 2.1.

Структура основной программы на языке ассемблера микроЭВМ "Электроника 60" имеет следующий вид:

```
.TITLE  < имя >
LC=.
.=4+LC
.WORD 6,0,12,0
.=500+LC                ; память под стек
JMP START
```

```
        резервирование памяти под программные
                    переменные

START:  MOV PC,SP
        TST -(SP)

        команды, реализующие алгоритм задачи

            HALT

        описание программ, макроопределений,
           память под рабочие переменные

            .END START

        структура внешней подпрограммы

            .TITLE < имя >
            .GLOBL < список имен >
            JMP    < имя >

        резервирование памяти под програм-
                мные переменные

< имя >: NOP

        команды, реализующие алгоритм
              подпрограммы

            RTS PC

        память под рабочие переменные

            .END < имя >
```

Рассмотрим ряд примеров составления программ на ассемблере для микроЭВМ семейства "Электроника 60".

Несложная программа умножения путем выполнения операций сложения использует преимущество архитектуры микроЭВМ семейства "Электроника 60", заключающееся в наличии РОН, что позволяет разместить переменную в соответствующем регистре:

```
;  УМНОЖЕНИЕ ЧИСЕЛ БЕЗ ЗНАКА
       CLR    R0          ;аккумулятор
       MOV    MNZL,R1     ;множитель
       MOV    MNME,R2     ;множимое
       MOV    #16.,R3     ;счетчик
       ROR    R1          ;сдвиг множителя
    L: BCC    NADD        ;мл.бит=0
       ADD    R2,R0       ;мл.бит=1,суммир.частич.произв.
NADD:  CLC
       ROR    R0          ;сдвиг произв.(ст.слово)
       ROR    R1          ;мл.слово произведения
       SOB    R3,L
```

Благодаря этому появляется возможность избежать использования двух пар команд по загрузке и хранению данных в основном цикле программы, поскольку операции могут быть выполнены с непосредственным участием соответствующих регистров. Оптимизацию программы по объему можно продолжить, прибегнув к команде SOB.

Применение команд вызова подпрограмм и способов передачи данных иллюстрируется на примере пакета ALG.FL подпрограмм вычислений с ПЗ в формате микроЭВМ семейства "Электроника 60". Подлежащие передаче аргументы надо перечислить в директивах·WORD сразу же за обращением к подпрограммам:

```
    JSR R5  < имя >
·WORD  < адрес 1-го операнда >
·WORD <адрес 2-го операнда >
·WORD<адрес результата >
```

В подпрограмме данные принимаются через регистр связи R5 , в котором, согласно механизму выполнения команды JSR , будет находиться содержимое СК, указывающее на первый аргумент подпрограммы с помощью автоинкрементной адресации (например, MOV (R5)+ , R1).

Команда RTS R5 осуществляет возврат из подпрограммы:

```
        ПАКЕТ П/П ALG.FL
ВЫЧИСЛЕНИЯ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ
  В ФОРМАТЕ "ЭЛЕКТРОНИКА 60"
-------------------------------
  ADDFL - СЛОЖЕНИЕ
  SUBFL - ВЫЧИТАНИЕ
  MULFL - УМНОЖЕНИЕ
  DIVFL - ДЕЛЕНИЕ

  ОБРАЩЕНИЕ:
    JSR R5, < ИМЯ >
    .WORD <АДРЕС 1-ГО ОПЕРАНДА>
    .WORD <АДРЕС 2-ГО ОПЕРАНДА>
    .WORD <АДРЕС РЕЗУЛЬТАТА>
-------------------------------
  MOVFL - ПЕРЕСЫЛКА

  ОБРАЩЕНИЕ:
    JSR R5,MOVFL
    .WORD <АДРЕС ИСТОЧНИКА>
    .WORD <АДРЕС ПРИЕМНИКА>
-------------------------------
  POPFL - ЧТЕНИЕ ИЗ СТЕКА
  PUSHFL - ЗАПИСЬ В СТЕК

  ОБРАЩЕНИЕ:
    JSR PC, < ИМЯ >
  УКАЗАТЕЛЬ СТЕКА ДОЛЖЕН БЫТЬ
  В R0, АДРЕС ОПЕРАНДА ВЫБИРА-
  ЕТСЯ ЧЕРЕЗ R5 ( (R5)+ )
-------------------------------
  CMPFL - СРАВНЕНИЕ
  JSR R5,CMPFL
```

```
  .WORD ( АДРЕС 1-ГО ОПЕРАНДА )
  .WORD ( АДРЕС 2-ГО ОПЕРАНДА )
-------------------------------
  CLRFL - ОБНУЛЕНИЕ
  JSR R5,CLRFL
  .WORD ( АДРЕС ОПЕРАНДА )
===============================
            .=.+12
     STFL : 0          ;СТЕК
     STAT : 0          ;ДЛЯ ССП
    ADDFL : MOV   #75000,COP
            BR    OPFL
    SUBFL : MOV   #75010,COP
            BR    OPFL
    MULFL : MOV   #75020,COP
            BR    OPFL
    DIVFL : MOV   #75030,COP
    OPFL  : MOV   R0,-(SP)
            MOV   #STFL,R0        ;СОХРАНЕНИЕ РЕГИСТРА R0
            JSR   PC,PUSHFL       ;ЗАГРУЗКА 1-ГО ОПЕРАНДА
            JSR   PC,PUSHFL       ;ЗАГРУЗКА 2-ГО ОПЕРАНДА
    COP   : 0                     ;КОД ОПЕРАЦИИ
            MFPS  STAT            ;СОХРАНЕНИЕ ССП
            JSR   PC,POPFL        ;ВЫГРУЗКА РЕЗУЛЬТАТА
            MOV   (SP)+,R0
            MTPS  STAT            ;ВОССТАНОВЛЕНИЕ ССП
            RTS   R5
    MOVFL : MOV   R0,-(SP)
            MOV   #STFL,R0
            JSR   PC,PUSHFL
            JSR   PC,POPFL
            MOV   (SP)+,R0
            RTS   R5
    PUSHFL: MOV   R1,-(SP)
            MOV   (R5)+,R1        ;АДРЕС ОПЕРАНДА
            MOV   2(R1),-(R0)
            MOV   (R1),-(R0)
            MOV   (SP)+,R1
            RTS   PC
    POPFL : MOV   R1,-(SP)
            MOV   (R5)+,R1        ;АДРЕС ОПЕРАНДА
            MOV   (R0)+,(R1)+
            MOV   (R0)+,(R1)+
            MOV   (SP)+,R1
            RTS    PC
    CMPFL : MOV    R0,-(SP)
            MOV   #STFL,R0
            JSR   PC,PUSHFL
            JSR   PC,PUSHFL
            75010
            MFPS  STAT
            MOV   (SP)+,R0
            MTPS  STAT
            RTS   R5
    CLRFL : MOV   R0,-(SP)
            MOV   (R5)+,R0
            CLR   (R0)+
            CLR   (R0)+
            MOV   (SP)+,R0
            RTS    R5
```

Хотя данная схема несколько сложна в смысле обработки аргументов, она не налагает никаких ограничений на число аргументов, передаваемых подпрограмме.

Если регистром для перехода является СК, то ситуация не кажется столь удобной, поскольку адрес первого аргумента находится на вершине стека и доступ подпрограммы к аргументам в этом случае более обременителен. Если, например, программа передает таким образом три аргумента подпрограмме, вызванной через СК, то мы должны гарантировать подстройку числа на вершине стека, прибавляя в этом случае 6, до возврата из подпрограммы с помощью RTS PC. Если же передача аргументов осуществляется через РОН или стек, то использование СК в качестве регистра для перехода приводит к ситуации, управлять которой так же легко, как и в случае регистра общего назначения. Фактически применение СК для этих целей освобождает дополнительные регистры для использования их в подпрограмме.

На примере вычисления факториала числа N покажем эффективность стековой адресации при написании рекурсивных программ, т. е. программ, у которых есть возможность вызова программой самой себя. Если известно

$$N! = N(N-1)\,(N-2)\,\ldots\,1\,,$$

то можно записать

$$N! = N(N-1)!; \qquad 0! = 1.$$

На псевдоФОРТРАНе вычисление функции N! можно представить в таком виде:

```
      INTEGER FUNCTION FACT(N)
      IF (N.NE.0 ) GO TO 1
      FACT=1
      RETURN
    1 FACT=N*FACT(N-1)
      RETURN
      END
```

Анализ приведенной программы показывает, что большинство имеющихся версий компиляторов с ФОРТРАНа не могли бы сформировать правильный объектный код программы, потому что реализация рекурсивного вызова требует обязательного использования стека для хранения текущих значений FACT и адресов возврата либо в саму подпрограмму, либо в вызывающую программу, что не все компиляторы могут делать. В связи с этим мы и назвали данный язык псевдоФОРТРАНом. Однако на языке ассемблера данная функция может быть запрограммирована с использованием стекового механизма. В таком случае накладываются некоторые ограничения на организацию вызова подпрограммы и соглашения по передаче параметров. Не следует хранить адреса возврата, параметры и локальные переменные в специально выделяемых фиксированных ячейках памяти, поскольку эти данные будут стерты при первом же рекурсивном вызове подпрограммой самой себя. Для устранения такой ситуации необходимо назначать при каждом рекурсивном вызове новую область памяти для адреса возврата, параметров и локальных переменных, а затем после возврата из подпрограммы перераспределять эту область

памяти исходя из имеющихся потребностей. Подходящей структурой для хранения данных является стек. Положим, что N и N! записываются в стек. Тогда функция FACTOR рекурсивно может быть запрограммирована следующим образом:

```
FACT:  MOV    N,-(SP)          ;ЗНАЧЕНИЕ N ЗАНОСИТСЯ
                                ;В СТЕК
       JSR    PC,FACTOR        ;ВЫЧИСЛЕНИЕ N!
       MOV    (SP)+,NFACT      ;ИЗВЛЕЧЕНИЕ ИЗ СТЕКА ОТВЕТА
       RTS    PC               ;ВОЗВРАЩЕНИЕ В ВЫЗЫВАЮЩУЮ
                                ;ПРОГРАММУ
       ;
       ;
FACTOR:       MOV    2(SP),R1        ;R1<-N
       BEQ    ZERO             ;N=0?
       DEC    R1               ;НЕТ, R1<-(N-1)
       MOV    R1,-(SP)         ;R1=N-1 ЗАНОСИТСЯ В СТЕК
       JSR    PC,FACTOR        ;РЕКУРСИВНЫЙ ВЫЗОВ
RET:   MOV    (SP)+,R1         ;R1<-(N-1)!
       MUL    2(SP),R1         ;R1<-(N-1)!*N
       MOV    R1,2(SP)         ;N!=(N-1)!*N
       RTS    PC
ZERO:  MOV    #1,2(SP)         ;0!=1
       RTS    PC
```

Программа рекурсивно вызывает сама себя при выполнении команды JSR PC, FACTOR. При этом она каждый раз записывает в стек уменьшенное значение N и адрес возврата, имеющий метку RET . Далее команда MUL умножает содержимое R1 на число, представляющее собой промежуточный результат (сначала 1), хранимый в стеке. После выполнения команды возврата RTS возможен переход в программу, вызвавшую FACTOR, либо – на метку RET в зависимости от содержимого стека. Если вычисление факториала закончено, то управление передается в вызывающую программу, а результат остается в стеке.

В соответствии с концепцией модульного программирования программы должны быть стандартны и доступны различным пользователям. Не всегда заранее известно, куда будет загружена такая программа, поэтому требуется реализация возможности ее выполнения в произвольном месте памяти. Для этой цели широко применяются позиционно независимые программы, так называемые PIC-программы, которые используют методы адресации, приведенные в табл. 1.3. Возможность создания таких программ определяется структурой микроЭВМ, на которой они реализуются. В частности, в микроЭВМ семейства "Электроника 60" допускается позиционно независимое кодирование PIC-программ. Для этого служат те методы адресации, в которых исполнительный адрес формируется относительно СК. Следовательно, при любом перемещении команды меняются и другие связанные с ней элементы программы, а расстояние между ними остается постоянным. Такое постоянное относительно СК значение смещения можно вычислять для всех адресов программы.

При написании PIC-программ предлагается руководствоваться следующими рекомендациями: 1) начинать программу со стартового адреса 0 для более удобного использования абсолютного загрузчика; 2) установки начальных значений должны иметь форму

определенные внутри PIC-программы; 3) в программе не следует устанавливать начальные значения адресов. При загрузке PIC-программы абсолютным загрузчиком в память все данные размещаются с соответствующим загрузочным смещением, а те из них, которые должны располагаться в ячейках с абсолютными адресами, пересылаются во время выполнения программы.

Встречаются ситуации, когда несколько программ должны находиться в режиме активного взаимодействия, при этом управление неоднократно передается от одной программы к другой. Программа, получающая управление, производит некоторое действие, "продвигающее" вперед весь процесс. Отношения между этими программами симметричны, поэтому их называют сопрограммами.

Идея сопрограмм в основном является развитием концепции подпрограмм. Однако сопрограмма не находится в подчиненном отношении к вызывающей программе. Соответственно подпрограммы и сопрограммы различаются по способам передачи управления. Механизм взаимодействия сопрограмм основан на использовании аппаратных возможностей – стека микроЭВМ семейства "Электроника 60". Так, с помощью специального режима команды JSR:JSR PC, @(SP)+, при выполнении которой последняя заполненная ячейка стека и СК обмениваются содержимым, две программы могут попеременно передавать управление друг другу и каждый раз возобновлять работу с того места, где осуществлялась передача управления.

Рассмотрим такое взаимодействие между программами P1 и P2. Полагаем, что стек должен быть инициализирован командой MOV # PC2, –(SP). Работает программа P1. Для передачи управления программе P2 она выполняет команду JSR PC, @ (SP) + , в ней при этом происходят следующие действия: 1) PC2 выбирается из стека и SP = SP + 2; 2) SP = SP–2 и (SP) ← PC1, т. е. старое значение PC помещается в стек; 3) PC ← PC2 и управление передается в ячейку PC2 программы P2. Далее работает программа P2 до команды JSR PC, @ (SP) + , которая поменяет местами в стеке PC2 и PC1, и управление снова передается в программу P1.

Сопрограммы используются в процессах ввода-вывода (например, для двойной буферизации ввода-вывода, совмещаемого с вычислением) и представляют собой одну из основных операций, выполняемых современной операционной системой.

Особенно явно преимущества стековой организации проявляются при совмещении выполнения фоновой задачи с вводом-выводом, в сложных мультипрограммных системах, управляющих комплексом пользовательских и системных программ. При этом от программного обеспечения требуются гибкость, экономия времени выполнения программы и используемой памяти. Наряду с применением приемов написания рекурсивных программ, сопрограмм применяются реентерабельные программы, которые отличаются от обычных тем, что могут использоваться в очередной задаче, не ожидая завершения работы программы в другой задаче, т. е. в одной и той же реентерабельной программе могут находиться несколько задач разной степени завершенности.

Реентерабельная программа характеризуется тем, что не содержит ничего, кроме чистых кодов, т. е. кодов команд и констант. В такой программе не

должны использоваться области собственной памяти для записи или чтения. При этом программы характеризуются следующими качествами: простотой отладки; защитой от записи (такие программы можно загрузить в постоянную память); допустимостью прерывания программы, а затем продолжения ее выполнения. Применение реентерабельных программ позволяет выделять функции, часто используемые различными задачами. Так могут быть оформлены программы перевода из кода ASCII в двоичный, обслуживания устройств и др.

Рекомендуемая литература: 3–5, 12, 13, 15.

3. СЕКЦИОНИРОВАННЫЕ МИКРОПРОЦЕССОРЫ И МИКРОПРОГРАММИРОВАНИЕ

3.1. ОРГАНИЗАЦИЯ МИКРОПРОГРАММИРУЕМОГО ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

Как уже отмечалось, микроЭВМ — это модульная система, в которую входят микропроцессор, а также память и устройства ввода-вывода. Так как длина слова и система команд фиксированы, то исключается использование микроЭВМ, когда важно высокое быстродействие или введение специальных команд, например, в системах сбора и обработки данных в реальном времени (выполняются операции умножения, деления и операции с ПЗ), и также в контроллерах диска, где требуются совершенно другие функции (поиск дорожки, выделение и введение символа синхронизации, считывание, запись и т. д.). Хотя специальные команды можно реализовать с помощью макрокоманд и подпрограмм, такой подход обычно не удовлетворяет требованиям системы по быстродействию.

По мере появления обладающих более широкими возможностями стандартных блоков стоимость разработки программного обеспечения возрастает (по сравнению со стоимостью разработки аппаратных средств). Это обстоятельство обусловливает использование большого объема уже существующего программного обеспечения и является основным стимулом для эмулирования путем микропрограммирования существующих машин. Эмулирование машины означает , что ее машинный язык будет реализован надлежащим образом в машине с микропрограммным управлением.

Рассмотрим кратко организацию архитектуры микропрограммируемого ЦП, принципы микропрограммирования и эмулирования архитектуры машины посредством микропрограммирования.

Для достижения большей гибкости МП разбивается на следующие узлы: операционный, управления (состоит из блока микропрограммного управления (БМУ), управляющей памяти (УП) и сопряжения (рис. 3.1)). Операционный узел содержит РОН, АЛУ и селектор условий. РОН и АЛУ образуют модуль обработки данных. Обычно секционированные МП выполняются по биполярной технологии. При этом часто применяется секционированный модуль обработки данных шириной 2 или 4 бита (реже 8 бит). В отличие от однокристального МП для получения нужной длины слова можно соединить несколько секций модуля обработки данных. Основу узла управления составляет БМУ, определяющий порядок выполнения инструкций микропрограммы, хранящейся в УП. Каждая микрокоманда имеет несколько полей: A — поле управления логикой микроуправления, B — поле управления блоком обработки данных, C — поле микроопераций (рис. 3.2).

При проектировании ЦП необходимо разработать модуль управляющей логики, модуль обработки данных и микропрограммы. Все данные задачи тесно

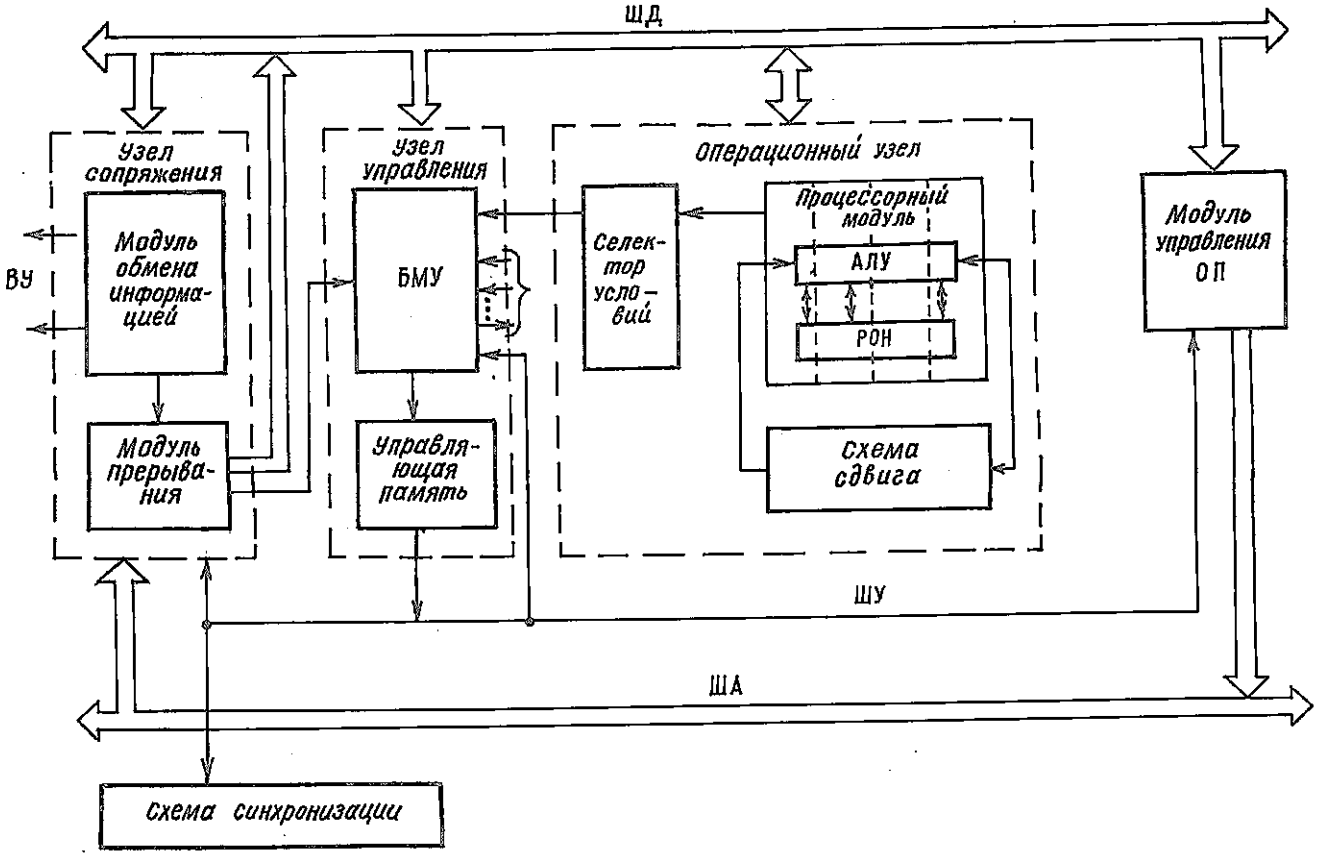

ШД
Узел сопряжения
Модуль обмена информацией
ВУ
Модуль прерывания
Узел управления
БМУ
Управляющая память
Операционный узел
Селектор условий
Процессорный модуль
АЛУ
РОН
Схема сдвига
Модуль управления ОП
ШУ
ША
Схема синхронизации

Рис. 3.1

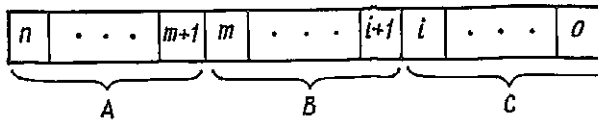


Рис. 3.2

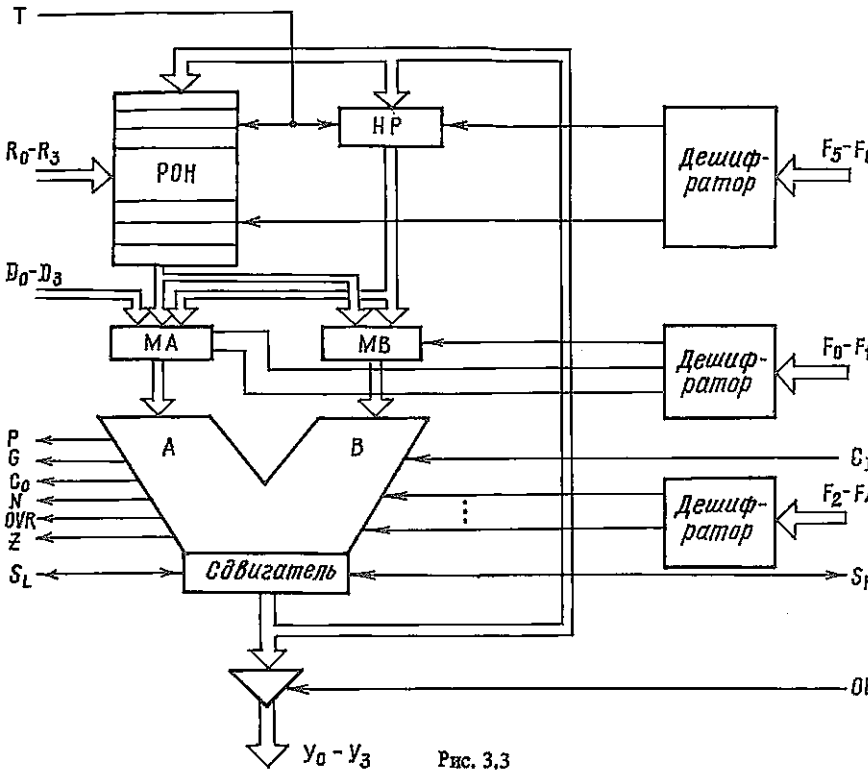


Рис. 3.3

связаны, и ни одну нельзя решать без определения базовых структур двух других частей. Из-за разнообразия типов секционированных МП (например, К589, К1804, К1802), способов реализации машины с заданными характеристиками может возникнуть необходимость оценить несколько вариантов для достижения оптимальных производительности и стоимости.

Ядро операционного узла составляет процессорный модуль (ПМ), в котором выполняются арифметические и логические операции над данными, хранящимися в регистрах внутренней памяти этого элемента или поступающими из внешних источников (ОП, ВУ и др.). Для организации ускоренного переноса при объединении нескольких ПМ используются схемы ускоренного переноса. Признаки результата операции в ПМ, определяющие его состояние в каждом такте выполнения МК, фиксируются в селекторе кода условия.

Процессорный модуль — второй базовый элемент секционированных МП — представляет собой функционально законченный элемент обработки данных (рис. 3.3).

Основой ПМ является регистровое АЛУ, которое содержит две входные (A и B) и выходную (Y) (например, 4-разрядные) шины. Для выбора источника данных на входе АЛУ используются входные мультиплексоры МА и МВ. Данные на входе D, связанном с внешними источниками информации, поступают через мультиплексор МА непосредственно на АЛУ и могут использоваться в качестве операндов АЛУ или информации для записи в РОН. В РОН одновременно хранится счетчик команд, указатель стека и другая информация. Микропроцессорная секция содержит также накапливающий регистр (НР) и линии условий (C_I – входного переноса, на которую подается младший разряд АЛУ; C_O – выходного переноса; N – знака результата операции; OVR – переполнения; Z – признака нулевого результата в сумматоре). Набор сигналов на этих линиях устанавливается по результату операции в АЛУ и соответствует его текущему состоянию. Указанные признаки используются в блоке микропрограммного управления для ветвления в микропрограмме. Можно сказать, что эти сигналы определяют вектор состояния ПМ.

Рассмотрим назначение остальных линий управления (см. рис. 3.3). Линии S_L и S_R предназначены для выполнения операций сдвига; P – линия распространения и G – линия генерации переносов с использованием внешней схемы ускоренного переноса; R – линия адресной магистрали; F – линии управления ПМ, определяющие операции, которые выполняются в АЛУ (табл. 3.1), источник данных для АЛУ и приемник результата операций (табл. 3.2).

Таблица 3.1

Входы			Операция АЛУ
F_2	F_3	F_4	
0	0	0	$A + B + C_i$
1	0	0	$A - B - C_i$
0	1	0	$A \vee B$
1	1	0	$A \wedge B$
0	0	1	$A \oplus B$
1	0	1	Сдвиг A влево 2A АЛУ$_0$ = S_R; $S_L = a_3$
0	1	1	Сдвиг A вправо A/2 АЛУ$_3$ = S_L; $S_R = a_0$
1	1	1	A

Таблица 3.2

Вход		Источник АЛУ		Вход		Приемник результата
F_0	F_1	A	B	F_5	F_6	
0	0	D	НР	0	0	Нет
1	0	D	РОН	1	0	РОН
0	1	РОН	НР	0	1	НР
1	1	НР	РОН	1	1	РОН, НР

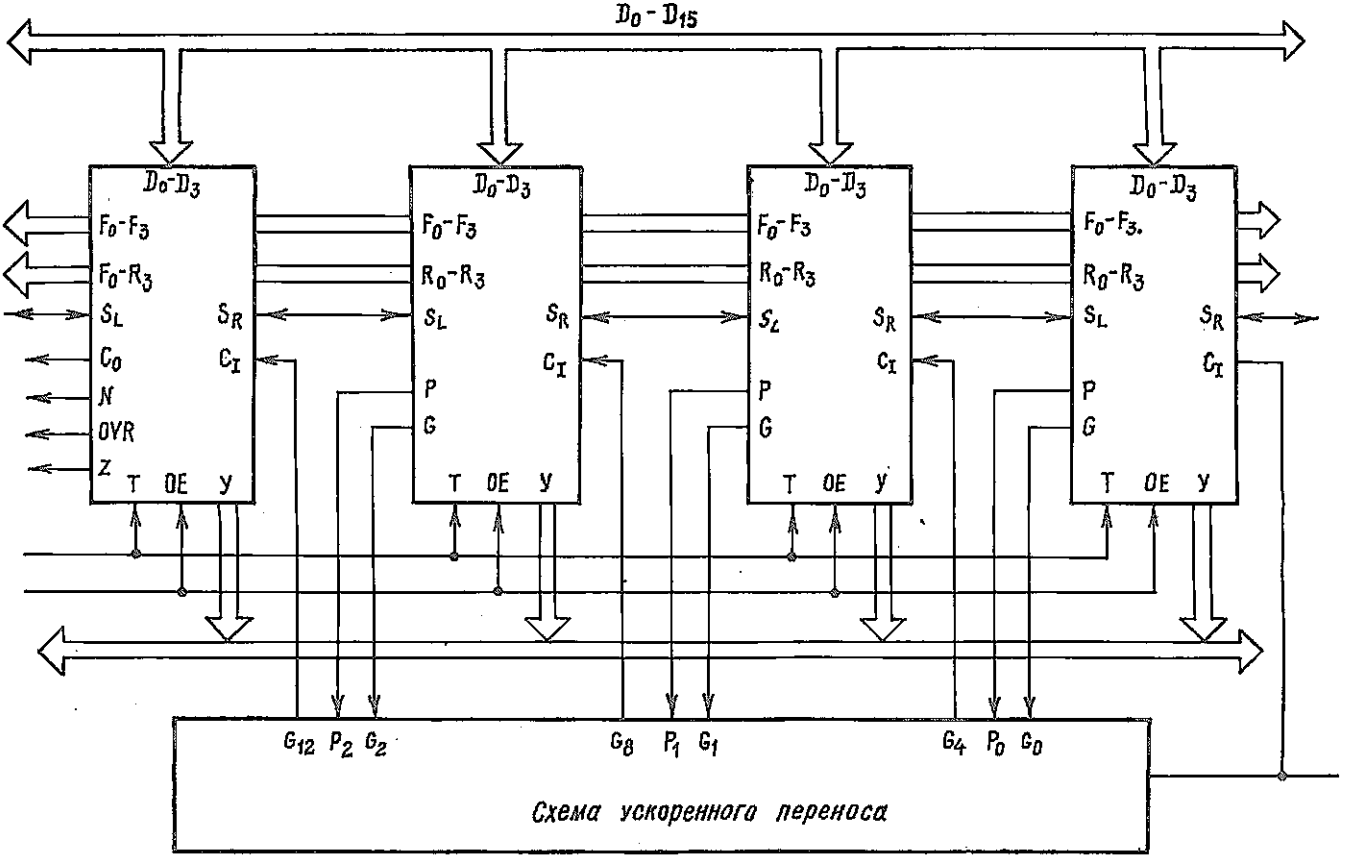

D0 - D15
D0-D3
F0-F3
F0-R3
SL
SR
C0
CI
N
OVR
Z
T
OE
y
R0-R3
P
G
G12 P2 G2
G8 P1 G1
G4 P0 G0
Схема ускоренного переноса

Рис. 3.4

Схема наращивания процессорных секций показана на рис. 3.4. Здесь адресная магистраль R и линии управления F параллельно соединены со всеми модулями. Информация в АЛУ обрабатывается в соответствии с набором сигналов на управляющих линиях F , которые можно разделить условно на три группы: F_0–F_1 (в сочетании с адресной магистралью R выбирают операнды); F_2–F_4 (задают тип выполняемой операции); F_5–F_6 (задают приемник результата операции). Сигнал на входе OE разрешает передачу результата на выходную шину Y.

Размещение процессорной секции в одном 40–48-выводном корпусе в виде функционально полной БИС обработки данных позволяет использовать эти схемы для построения гибкого высокопроизводительного микропроцессорного средства различного функционального назначения. Несмотря на варианты архитектур, процессорные модули (КР589ИК02, КР1804ВС1, КР1802ВС1) характеризуются следующими общими для всех признаками: 1) универсальностью (набором элементарных арифметических и логических операций, реализуемых в ПМ, обеспечивающих эффективную обработку информации); 2) программируемостью (все операции, реализуемые в ПМ, должны выполняться под воздействием микрокоманд узла управления); 3) наращиваемостью структуры; 4) индикацией состояния.

Доступ проектировщика и программиста к "сердцу" системы – памяти микропрограмм – позволяет создать оптимальную структуру, наилучшим образом отвечающую специфике конкретного применения. В то же время спроектированная система обладает наибольшей гибкостью, так как расширение функциональных возможностей может быть легко достигнуто путем изменения отдельных микрокоманд или замены всей памяти микропрограмм.

В использовании секционированных МП можно выделить три основных направления. Первое – использование секционированных МП вместо традиционных электронных схем с жесткой логикой. Применение МП позволяет значительно сократить количество интегральных схем, схемных плат для их размещения, уменьшить габариты и мощность процессора. Второе направление – создание на основе секционированных МП высокопроизводительных универсальных мини-ЭВМ, которые на микропрограммном уровне эмулируют систему команд другой ЭВМ, имеющей развитое программное обеспечение.

Третье направление – разработка ЭВМ, выполняющих без промежуточной трансляции программы, которые написаны на языках высокого уровня. Секционированные МП перспективны для построения мощных вычислительных систем, в которых несколько одновременно работающих процессоров реализуют параллельную обработку информации.

3.2. ПРИНЦИП МИКРОПРОГРАММИРОВАНИЯ

Эффективность решения задач во многом определяется привязкой алгоритма к архитектуре ЦП. На современном этапе развития вычислительной техники – это его микропрограммная реализация. Возможны два принципа управления последовательностью микроопераций: 1) последовательное схемное управление, т. е. управляющие сигналы генерируются некоторой логической схемой; 2) микропрограммное управление. Рассмотрим общие характеристики микропрограммного управления, которые сопоставим с последова-

тельным схемным управлением на примере выполнения команды сравнения двух двоичных чисел без знака. При этом будем использовать язык функционального микропрограммирования CDL (Computer Design Language), позволяющий определять функциональную организацию модуля обработки данных, алгоритмы и последовательности выполнения микроопераций в ЦП.

Описание функциональных узлов машины в терминах языка CDL включает название элемента, мнемоническую запись и комментарий. Так, для описания регистра управления, содержащего 6 бит, используется запись

```
Register, F(0–5), $ регистр управления.
```

Для описания памяти емкостью 1 К 16-разрядных слов применяется запись

```
Register, C(0–9), $ адресный регистр памяти,
Memory, M(C) = M(0–1023, 0–15).
```

Ключи ручного управления на пульте описываются следующим образом

```
Switch, POWER(ON), $ ключ питания,
        START(ON), $ ключ пуска машины.
```

Для отображения состояния машины служат индикаторы:

```
Light, FINI(ON, OFF), $ индикатор окончания,
       ERROR(ON, OFF), $ индикатор ошибки.
```

Синхронизация всех устройств осуществляется генератором синхроимпульсов

```
Clock, P(1–2), $ двухфазный генератор синхросигналов.
```

Синхропоследовательность P(2) генератора сдвинута относительно синхропоследовательности P(1) на время τ.

Для обозначения выполняемой функции используются операторы, которые даны в табл. 3.3 (могут применяться и другие операторы). В основе элементарных вычислительных действий лежат микрооперации. Так, для пересылки содержимого регистра R в регистр A вводится запись A ← R. Микрооперация приращения счетчика на 1 запишется таким образом: D ← incD. Микроопе-

Таблица 3.3

Оператор	Обозначение	Комментарий
AND	*	Логическое И
OR	+	Логическое ИЛИ
Циклический сдвиг влево	cil	Сдвиг влево на 1 или несколько битов
Приращение счетчика	inc	Увеличение содержимого счетчика на 1
Сложение	add	Сложение двух двоичных чисел без знака

рация, используемая только при истинности некоторого условия, записывается с помощью условного оператора

IF (выражение) THEN (выражение) ELSE (выражение).

Если условие, записанное после оператора IF , истинно, то выполняются микрооперации, записанные после оператора THEN , в противном случае — микрооперации, записанные после оператора ELSE . Если оператор ELSE отсутствует, то не предпринимается никаких действий.

Последовательность действий может задаваться с помощью многофазного генератора синхросигналов:

```
Clock, P(1-2), $ двухфазный генератор синхросигнала,
/P(1)/C ← D,
/P(2)/R ← M(C), D ← incD,
    END
```

С помощью однофазного генератора и управляющего регистра эта же последовательность будет сформирована так:

```
Register, F(0-1), $ регистр управления,
Decoder, K(0-3) = F , $ дешифратор, подключенный к выходу регистра.
Clock, P , генератор синхросигнала,
/K(0) * P/C ← D,
/K(1) *P/R ← M(C), D ← incD,
     END.
```

Для определенности положим, что операционный автомат для данной команды состоит из схемы поразрядного сравнения и двух регистров операндов. Команда выполняется путем просмотра слева направо двух сравниваемых чисел. При этом результат может быть определен сразу же, как только обнаружится несовпадение значений в каком-либо бите. Тогда при равенстве двух чисел придется просматривать все разряды этих чисел.

Модуль обработки данных и последовательное схемное управление определяются следующими предложениями-описаниями и операторами исполнения:

```
      Register, EQ , # 1 соответствует условию "равно"
                GT , # 1 соответствует условию "A>B"
                LT , # 1 соответствует условию "A<B"
              SIGN , # регистр для хранения знака
           A(0-7) , # регистр первого числа;A(0)-знак.бит
           B(0-7) , # регистр второго числа;B(0)-знак.бит
           C(0-2) , # счетчик
           T(0-3) , # регистр управления
 Decoder, K(0-8)=T , # дешифратор
 Switch, START (ON), # ключ запуска
Light, FINI(ON,OFF), # индикация работы
Clock, P ,           # тактовый генератор

  /START(ON)/ EQ(-1,LT(-0,GT(-0,C(-0,
              FINI(-OFF,T(-0,
    /K(0)*P/ IF(GT+LT=1) then (T(-2) else (T(-1),
```

```
/K(1)*P/ IF(A(0)-B(0)=A(0)*B(0)) then (LT<-1,EQ<-0),
         IF(A(0)-B(0)=A(0)*B(0)) then (GT<-1,EQ<-0),
         T<-2,
/K(2)*P/ C<-incC,A<-cilA,B<-cilB,
         T<-3,
/K(3)*P/ IF(C=8) then (T<-4) else (T<-0),
/K(4)*P/ FINI<-ON
         END
```

Здесь каждой комбинации регистра Т соответствует управляющий сигнал. Например, при Т = 2 вырабатывается операция, инкрементации счет-циклов С и осуществляется циклический сдвиг регистров А, В.

При переходе к микропрограммному управлению используются те же восемь микроопераций, выполняющих в обоих случаях функцию выявления большого числа:

```
IF(GT+LT=1) then (...) else (...),
IF(A(0)-B(0)=A(0)*B(0)) then (...),
IF(A(0)-B(0)=A(0)*B(0)) then (...),
C<-inc C,
A<-cil A,
B<-cil B,
IF(C=8) then (...) else (...),
FINI<-ON
```

Теперь вместо регистра управления Т(0–3) и дешифратора К(0–8) = Т в схему включены регистр адреса микрокоманды Н(0–2), управляющая память (память микрокоманд) СМ(0–7, 0–12) и регистр микрокоманды F(0–12). Кроме того, используется двухфазный генератор синхросигналов Р(1–2) (рис. 3.5). При использовании уйравляющей памяти к восьми микро-

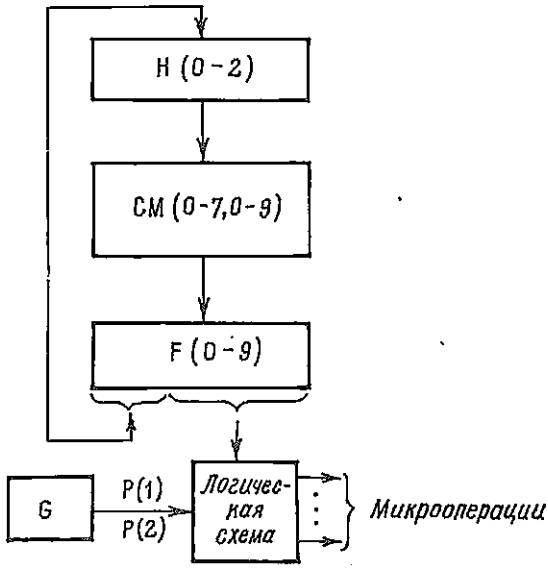


Рис. 3.5

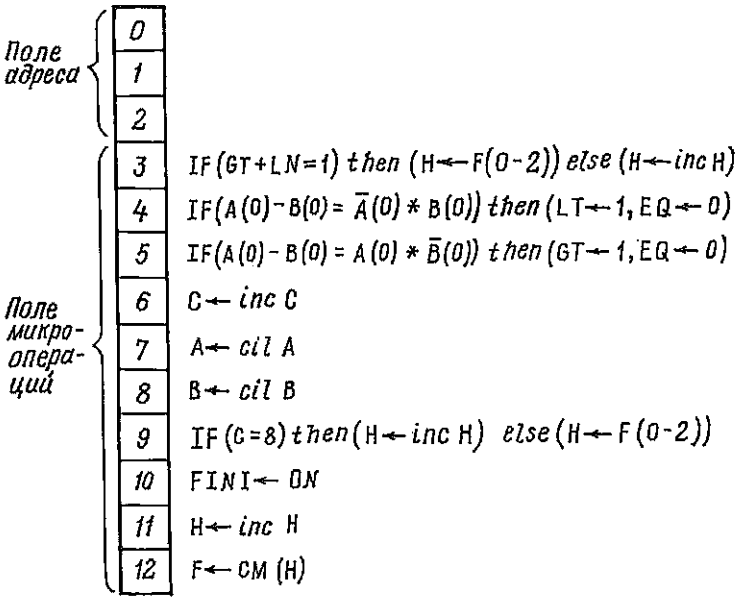


Рис. 3.6

операциям необходимо добавить еще две:

F ← CM(H) $ загрузка микрокоманды в регистр F ,
H ← inc(H) $ инкрементация регистра адреса микрокоманды H.

Команды

IF(GT + LT = 1) then (H ← F(0–2)) else (H ← incH),
IF(C = 8) then (H ← incH) else (H ← F(0–2))

замещают соответственно первую и седьмую команды. Таким образом, для решения данной задачи необходимо 10 микроопераций. На рис. 3.6 показан формат микрокоманды. Когда бит F (j) (j = 3–12) содержит 1, то формируется управляющий сигнал, запускающий *j*-ю микрооперацию. Так как выполняется одновременно несколько микроопераций, единицы могут стоять более чем в одном разряде микрокоманды.

Теперь последовательности управляющих сигналов определяются содержимым соответствующих битов в слове управляющей памяти, содержащем микрокоманду.

Запуск всех микроопераций инициируется синхросигналом P(1), а микрооперация выборки очередного слова из УП – синхросигналом P(2). Значения некоторого бита микрокоманды F (*i*) в совокупности с фазой P(1–2) определяют управляющий сигнал.

Операторы исполнения, описывающие 10 микроопераций, имеют вид

```
/F(3)*P(1)/  IF(GT+LN=1) then (H(-F(0-2)) else (H(-incH)
/F(4)*P(1)/  IF(A(0)-B(0)=A(0)*B(0)) then (LT(-1,EQ(-0
/F(5)*P(1)/  IF(A(0)-B(0)=A(0)*B(0)) then (GT(-1,EQ(-
```

```
/F(6)*P(1)/  C<-incC
/F(7)*P(1)/  A<-cilA
/F(8)*P(1)/  B<-cilB
/F(9)*P(1)/  IF(C=0) then (H<-incH) else (H<-F(0-2))
/F(10)*P(1)/ FINI <-ON
/F(11)*P(1)/ H<-incH
/F(12)*P(2)/ F<-CM(H)
```

Описание последовательности микроопераций на языке CDL для команды сравнения следующее:

```
C: начало последовательности сравнения чисел;
C: инициализация устройства

/START(ON)/   EQ<-1,LT<-0,GT<-0,C<-0,FINI<-0,H<-0,F<-1
/F(12)*P(2)/  F<-CM(H)

C: проверка регистров условий

/F(3)*P(1)/   IF(GT+LN=1) then (H<-F(0-2)) else (H<-incH)
/F(12)*P(2)/  F<-CM(H)

C: формирование значений регистров условий на i-ом шаге

/F(4)*P(1)/   IF(A(0)-B(0)=A(0)*B(0)) then (LT<-1,EQ<-0)
/F(5)*P(1)/   IF(A(0)-B(0)=A(0)*B(0)) then (GT<-1,EQ<-0)
/F(11)*P(1)/  H<-incH
/F(12)*P(2)/  F<-CM(H)

C: циклический сдвиг, увеличение счетчика на 1

/F(6)*P(1)/   C<-incC
/F(7)*P(1)/   A<-cilA
/F(8)*P(1)/   B<-cilB
/F(11)*P(1)/  H<-incH
/F(12)*P(2)/  F<-CM(H)

C: проверка счетчика и условный переход

/F(9)*P(1)/   IF(C=0) then (H<-incH) else (H<-F(0-2))
/F(12)*P(2)/  F<-CM(H)

C: формирование признака завершения

/F(10)*P(1)/  FINI<-ON
END
```

Содержимое управляющей памяти в данном случае приведено в табл. 3.4.

При каждом проходе по циклу выполняется одна или несколько микроопераций, указанных в микрокоманде (их реализация происходит одновременно и инициируется первым синхросигналом P(1)); с приходом второго синхросигнала P(2) выбирается очередная микрокоманда. Адрес микрокоманды, подлежащей выполнению, получается либо в результате H ← incH, либо H ← F(0–2).

Таблица 3.4

Адрес памяти	Микрокоманда													Представление микрокоманды в восьмеричной системе счисления
	поле адреса				поле управления									
	0	1	2	3	4	5	6	7	8	9	10	11	12	
0 0 0	0	1	0	1	0	0	0	0	0	0	0	0	1	05001
0 0 1	X	X	X	0	1	1	0	0	0	0	0	1	1	00603
0 1 0	X	X	X	0	0	0	1	1	1	0	0	1	1	00163
0 1 1	0	0	0	0	0	0	0	0	0	1	0	0	1	00011
1 0 0	X	X	X	0	0	0	0	0	0	0	1	0	0	00004

Такт любой микрокоманды состоит из трех основных фаз: 1) вычисления адреса очередной микрокоманды; 2) выборки по данному адресу микрокоманды из управляющей памяти; 3) исполнения микрокоманды в операционном узле. Порядок выполнения фаз зависит от способа соединения БМУ и управляющей памяти.

Таким образом, операторное описание находится в соответствии с микропрограммой, представленной в табл. 3.4. Быстродействие существующих машин с микропрограммным управлением ограничивается временем прохода по циклу управления. Здесь быстродействие по сравнению с последовательным схемным управлением ограничивается временем обращения к управляющей памяти. Однако стоимость управляющих схем примерно пропорциональна сложности реализуемой схемы управления, а стоимость управляющей памяти практически не зависит от сложности логики управления.

Основным элементом модуля логики микроуправления является логическая схема выбора адреса очередной микрокоманды, которая в соответствии с функцией управления и внешними управляющими сигналами подключает один из источников адреса ко входу счетчика адреса микрокоманд. Для организации подпрограмм и циклов в состав модуля вводят стек и счетчик циклов, для хранения адреса микрокоманды — выходной буферный регистр.

В зависимости от разрядности формируемого адреса все БМУ могут быть разделены на два типа: разрядно-модульные и с фиксированной разрядностью адреса. К первому типу БМУ относятся 4-разрядные модули КР1804ВУ1/ВУ2, К1800ВУ1. Разрядно-модульные БМУ позволяют наращивать разрядность адреса управляющей памяти (УП) простым каскадированием отдельных модулей. Так, при использовании двух последовательно соединенных модулей КР1804ВУ1 можно адресовать память объемом до 256 слов, трех — до 4096 слов и т. д. Ко второму типу БМУ относятся модули К589ИК01, КР1804ВУ4, обеспечивающие непосредственную адресацию УП объемом 512—4096 слов в зависимости от типа элемента. Увеличение объема УП при использовании БМУ второго типа может быть достигнуто путем страничной организации памяти, где размер страницы соответствует разрядности адреса БМУ, а текущая страница выбирается с помощью внешнего регистра, содержимое которого задается микропрограммно.

Таким образом, с точки зрения пользователя микропрограммное управление позволяет: 1) иметь список команд независимо от физического воплоще-

ния модели вычислительного процесса; 2) использовать для одной ЭВМ несколько списков команд, т. е. на новых ЭВМ решать задачи, написанные для старых ЭВМ.

3.3. ЭТАПЫ РАЗРАБОТКИ МИКРОПРОГРАММ ДЛЯ СЕКЦИОНИРОВАННЫХ МИКРОПРОЦЕССОРОВ

Процесс разработки микропрограммного обеспечения секционированных МП включает восемь этапов, каждый из которых реализуется с помощью различных средств.

Этап 1. Разработка символического языка микропрограммирования начинается после того, как описан алгоритм функционирования устройства. Здесь используются языки невысокого уровня, которые позволяют минимизировать объемы памяти и время исполнения микропрограмм.

Этап 2. Разработка транслятора с символического языка микропрограммирования.

Этап 3. Кодирование (написание) микропрограммы на созданном языке. В зависимости от принятой технологии разработки программ этот этап реализуется с помощью диалоговых средств редактирования.

Этап 4. Трансляция микропрограммы, осуществляемая ручным (для небольших микропрограмм) или автоматическим(с помощью транслятора) способом в машинный код. При ручной трансляции, как правило, получают готовую к выполнению микропрограмму в абсолютных адресах – переход к этапу 6. В случае автоматической трансляции, когда получается объектный модуль, – переход к этапу 5.

Этап 5. Определение адресов связей объектного модуля и его настройка на выполнение в заданном месте памяти. При этом используются программы-редакторы связей.

Этап 6. Отладка микропрограммы проводится с помощью программных средств и специальных аппаратно-программных комплексов (стендов), работающих по принципу внутрисхемного эмулирования. Программные средства представляют собой моделирующие системы, настраиваемые на архитектуру и язык отлаживаемого микропроцессорного устройства.

Аппаратно-программный эмулирующий стенд позволяет управлять работой отлаживаемой микропроцессорной системы в реальном масштабе времени, не нарушая ее функционирования. При этом осуществляется возможность остановить микропрограмму в заданных точках, индицировать всю информацию, передаваемую в систему и выводимую из нее, реализовать пошаговое выполнение микрокоманд и т. д.

Этап 7. Запись в ПЗУ отлаженной микропрограммы с помощью специальных программ управления программатором ПЗУ.

Этап 8. Выпуск документации.

Для повышения производительности процесса микропрограммирования секционированных микропроцессоров отдельные его этапы автоматизируются путем создания средств разработки – перекодировщиков, трансляторов, дизассемблеров, отладчиков и т. д. Согласованные по форматам входных и выходных данных и интегрированные в единое целое средства разработки называются инструментальным комплексом.

3.4. ЭМУЛИРОВАНИЕ АРХИТЕКТУРЫ ПОСРЕДСТВОМ МИКРОПРОГРАММИРОВАНИЯ

Основным следствием развития техники интегральных схем явилось изменение соотношения стоимости разработки аппаратных средств и программного обеспечения. Это обстоятельство обусловливает стремление к использованию большого объема уже существующего программного обеспечения на новых машинах и является основным стимулом для эмулирования путем микропрограммирования существующих машин. Следует отметить, что процесс создания машины на базе эмулирования может потребовать значительно меньше времени, чем при непосредственной аппаратной реализации архитектуры; при эмулировании реализовать в одной машине можно несколько различных архитектур; расширение исходной или эмулируемой архитектуры может быть произведено более легким способом в системах с микропрограммным управлением.

Существует несколько методов организации эмулирования требуемой машины. Первый метод состоит в том, что машинная команда адресует подпрограмму микропрограммы, которая выполняет соответствующую операцию посредством микроопераций. Простейший способ организации доступа к микропрограмме заключается в использовании разряда кода операции машинной команды непосредственно в качестве адреса соответствующей подпрограммы микропрограммы. Недостаток данного способа заключается в отсутствии необходимой корреляции между кодами операций машинной команды и адресами подпрограмм микропрограммы, реализующими эти операции. Попытка же приспособления структуры микропрограммы к таким произвольным кодам операций команд приводит к выполнению лишних операций перехода в микропрограмме, а это снижает быстродействие устройства управления и усложняет отладку микропрограмм.

В другом методе решения данной задачи используется косвенная адресация путем включения в микропрограмму таблицы переходов, строки которой адресуются разрядами кода операции команды. Содержимое строки в свою очередь используется для организации перехода на выполнение требуемой подпрограммы микропрограммы. Однако применение в микропрограмме таблицы переходов приводит к неоправданному снижению скорости обработки.

Решение данной проблемы облегчается применением третьего метода, который предусматривает перемещение таблицы переходов из управляющего ПЗУ в отдельное – ПЗУ кода машинной команды, поступающей из регистра команд, в адрес подпрограммы микропрограммы.

Рекомендуемая литература: 12–15.

4. ПОСТРОЕНИЕ ТРАНСЛЯТОРОВ

4.1. ВИДЫ И СТРУКТУРЫ ТРАНСЛЯТОРОВ

Ассемблер. Для написания пользователем программ на различных языках программирования необходимо иметь средства их перевода в машинный язык. Это выполняют специальные программы-переводчики, которые называются *трансляторами.* Трансляторы для микроЭВМ преобразуют исходную программу на одном из языков программирования в некоторую стандартную форму, называемую объектной программой. Существует три вида трансляторов: ассемблеры, компиляторы и интерпретаторы.

Транслятор, преобразующий программу, написанную на языке ассемблера, в машинный код, называется *ассемблером.* При написании программы на языке ассемблера программист использует мнемонические обозначения машинных команд и адресов (имена и метки). В процессе трансляции ассемблер заменяет все мнемонические обозначения (коды команд и имена) их двоичными кодами. Для сокращения текста при повторении идентичных частей программы в отдельные языки ассемблера введены средства написания и обработки макрокоманд. Это позволяет программисту определить некоторую последовательность команд как макроопределение, которое обрабатывается макроассемблером (макропроцессором). Последний подставляет тексты макроопределений с соответствующими фактическими параметрами макровызова вместо макрокоманд в программе.

Простейший ассемблер является однопроходным и преобразует исходную программу за один просмотр. Но при этом возникают трудности, связанные с вычислением адресов имен, которые определены позже. Например,

```
JMP A1...A1 : ADD B
```

В данном случае в команде перехода не определен адрес A1, который появляется далее. Этого можно избежать, потребовав, чтобы все имена данных были объявлены заранее, а неопределенные адреса заносились в специальную таблицу, в которую вводятся ссылки вперед (в нашем примере A1). Эта таблица либо используется для модификации команды (в нашем примере JMP), либо загрузчиком, который может модифицировать данный адрес во время загрузки. Однако это не всегда удобно, поэтому большинство ассемблеров выполняют два прохода.

Основная идея двухпроходного ассемблера проста. На первом проходе все символы (имена и метки) собираются в таблицу символов с соответствующими значениями адресов, на втором генерируется машинная программа на основании построенной на первом проходе таблицы символов. Пусть в нашем примере A1 на первом проходе получил адрес 0120. Тогда на втором проходе вместо команды JMP A1 сгенерируется код C32001.

Если язык ассемблера включает средства макрорасширений, то макропроцессор реализуется различными способами. Он может быть добавлен к ассемблеру как препроцессор для выполнения просмотра исходного текста перед первым проходом ассемблера. В результате препроцессирования получается программа на языке ассемблера, не содержащая макросов. При этом тексты макроопределений, если они есть в исходной программе, сохраняются, а вместо макровызовов подставляются ассемблерные команды из макроопределений. Возможно также объединение макропроцессора с первым проходом ассемблера, что сокращает время трансляции, но удлиняет текст ассемблера.

Рассмотрим пример трехпроходного ассемблирования фрагмента программы, написанной на ассемблере для МП КР580ИК80. В этом фрагменте на первом проходе вместо макровызова подставляется макроопределение, на втором составляется таблица имен, на третьем генерируется машинный код:

```
    ORG 100H ; Счётчик адреса 100H
    LDA A1   ; Загрузка аккумулятора
    SHF B,3  ; Обращение к макрокоманде
    STA A1   ; Запись в память результата
A1: DS  1    ; Резервирование памяти
    RET      ; Возврат
    END      ; Конец ассемблирования

    SHF MACRO R1,N
    MVI   R1,N
M1: RRC
    ANI   7FH
    DCR   R1
    JNZ   M1
    ENDM
```

```
 Первый проход        Второй проход

Адрес   ORG 100H     Таблица символов

 100    LDA A1       ------------------
 103    MVI B,3       Метка  ! Адрес
 105M1: RRC           ------------------
 106    ANI 7FH         A1    ! 010F
 108    DCR B         ------------------
 109    JNZ M1          M1    ! 0105
 10C    STA A1        ------------------
 10FA1: DS  1
```

```
          Третий проход

           100 3A 0F 01
           103 06 03
           105 0F
           106 E6 7F
           108 05
           109 C2 05 01
           10C 32 0F 01
           10F
           110 C9
```

Компилятором называется системная программа, которая воспринимает на входе текст программы на языке высокого уровня, а генерирует на выходе программу на языке ассемблера или машинном. На вход компилятора поступает исходная программа, а на выходе получается объектная. Объектная программа реализует те же действия, что и исходная, каждый оператор которой заменяется одной или несколькими машинными командами.

В состав любого компилятора входят три основных компонента: лексический анализатор (сканер), синтаксический анализатор и генератор машинных команд. Принцип действия анализаторов можно описать формальными моделями, но для генерации не удается создать формальное представление.

На фазе лексического анализа читается строка оператора, которая разбивается на единицы, называемые *лексемами*. К ним относятся ключевые слова (IF, DO, END и т. д.), имена переменных, знаки операций (+, –, /, *), специальные знаки (: , ; и т. д.).

Например, строка программы на ФОРТРАНе A1=A1+1 будет разбита на следующие лексемы: A1 – имя, 1 – литерал, = , + знаки операций.

После разбиения программы на лексемы следует фаза синтаксического анализа, называемая *грамматическим разбором*, на которой проверяется правильность построения операторов. Например, для оператора IF A1=5 THEN M1 грамматический разбор состоит в следующем. Надо убедиться, что вслед за лексемой IF идет правильное выражение A1=5, за которым следует снова лексема THEN . За последней тоже появляется правильное предложение. В процессе синтаксического анализа структура программы преобразуется во внутреннее представление (матрицу, список, польскую запись).

Последним выполняется процесс генерации кода, который создает программу на машинном языке. При этом возможна сначала генерация ассемблерной программы, транслируемая затем в машинный код. В последнем случае вместо каждой строки промежуточной формы после грамматического анализа могут подставляться стандартные заготовки на ассемблере. Например, для выражения A1=A1+1 промежуточная форма может иметь вид +A1 1 M1; =M1 A1, что потребует следующих заготовок на ассемблере МП КР580:

```
LDA  A1          LDA  M1
ADI  1           STA  A1
STA  M1
```

Простой анализ этих команд показывает, что часть из них является лишней (STA M1 и LDA M1). Это может выполняться в процессе машинной оптимизации. Окончательно ассемблерный и машинный фрагменты будут иметь вид

```
LDA  A1          A1–0100      3A  0001
ADI  01                       C6  01
STA  A1                       32  0001
```

Таким образом, исходная фортрановская строка A1=A1+1 скомпилируется в три машинные команды.

Взаимодействие трех описанных компонентов компилятора может быть организовано различным образом. Наиболее общим является случай много-

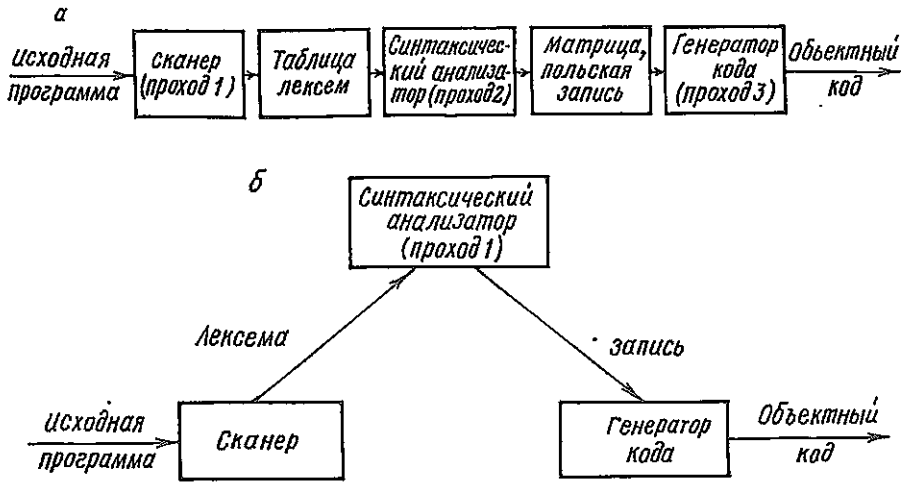


Рис. 4.1

проходного компилятора (рис. 4.1, *а*). Сканер на первом проходе читает исходную программу и представляет ее в форме файла лексем. Этот файл на втором проходе считывается синтаксическим анализатором, который выдает новое представление программы, например в виде постфиксной записи. Наконец, этот файл читается генератором кода на третьем проходе, который создает объектный код программы.

Хотя такая структура компилятора является относительно низкоскоростной в связи с работой с файлами, она обладает рядом преимуществ. Во-первых, каждая фаза компиляции относительно независима, при этом каждый проход реализуется самостоятельно. Во-вторых, структура отличается гибкостью. Если компилятор, созданный для ЭВМ типа x, изменяется для машины типа y, то переделке подвергается блок генерации кода. При изменении входного языка необходимо переделать блоки анализаторов.

Для увеличения быстродействия компилятора можно сократить (до одного) число проходов (рис. 4.1, *б*). В этом случае синтаксический анализатор играет роль основной программы, обращается к сканеру, получая от него лексему за лексемой, пока не построит постфиксную запись. Затем он обращается к генератору кода, который создает объектный код для этого фрагмента. Но в этом случае труднее осуществлять оптимизацию кода.

Интерпретатором называется системная программа, которая транслирует каждый оператор исходной программы в промежуточный код, интерпретирует его посредством одной или нескольких команд и выполняет эти команды. В отличие от компилятора интерпретатор не генерирует объектный код, а выдает результаты работы выполняемых операторов исходной программы. Например, для предложения A = B+C вместо построения объектного кода, который вычисляет значение B+C и помещает его в ячейку с адресом A, интерпретатор выбирает значения B и C, складывает их и записывает результат в область, определенную для A.

Структура интерпретатора похожа на структуру компилятора, поскольку

она содержит блоки лексического и синтаксического анализа. Но вместо генератора кода интерпретатор включает специальную программу интерпретации внутренней формы.

4.2. ДВУХПРОХОДНАЯ АССЕМБЛИРУЮЩАЯ ПРОГРАММА

Рассмотрим проектирование транслятора с языка ассемблера для МП КР580, который является двухпроходным. Исходные символы, которые используются в программах, задаются в коде КОИ-7. Операторы представляют собой основную структурную единицу программы и заканчиваются кодом перевода строки LF (0АН). Если начальным символом является символ ;, то этот оператор является комментарием. Прочие операторы состоят из следующих полей: метки, операции, операндов, комментариев, причем присутствие всех полей необязательно. Имя состоит из символов (от 1 до 6). Первым может быть один из следующих символов: ? , @ A–Z , вторым – буква или цифра и т. д. Если последним символом имени является символ : , то имя описывается как метка. После метки может следовать один или несколько пробелов. В поле операции записываются либо псевдокоманды, либо машинные команды. В поле операндов может ничего не записываться или находиться один или два операнда, причем если требуется более одного операнда, то они разделяются запятыми. Используются три вида констант: десятичные, шестнадцатеричные, символьные. В качестве управляющих используются команды ORG, END, EQU, DB, DW, DS (см. § 2.1).

Распределение памяти для ассемблера приведено на рис. 4.2: программа ввода-вывода, подпрограммы ассемблера, таблица кодов команд и псевдоопераций, список исходной программы, программа печати и таблица меток (рис. 4.2).

Программа ввода-вывода располагается непосредственно перед ассемблером. В различных системах она разная и должна считывать символ и печатать его.

Порядок работы ассемблера следующий: 1) считывается исходная программа; 2) выполняется первый проход до карты END; 3) считывается второй раз копия исходной программы; 4) печатается текст программы с возможными ошибками.

В проектируемом ассемблере распознаются следующие ошибки: 1) метка описана дважды; 2) метка не определена; 3) ошибка в поле команды (несуществующий код); 4) ошибка в операнде (команда отличается по имени регистра, по формату, по используемому символу; подряд следуют знаки + и – и т. д.); 5) ошибка в формате. При обнаружении перечисленных ошибок первые две не влияют на ход ассемблера, а при остальных ассемблирование данного оператора прекращается.

Рассмотрим способ реализации внутренних средств. Ассемблер реализует три функции: обработку меток, создание машинного кода команд, печать исходной программы. Двухпроходный ассемблер выполняет следующие функции: 1) регистрирует метки в таблице меток; 2) генерирует команды с подстановкой значения метки в операнды.

Порядок выполнения ассемблера такой: 1) считывается первый операнд и помещается в поле памяти; 2) если есть метка, то она обрабатывается;

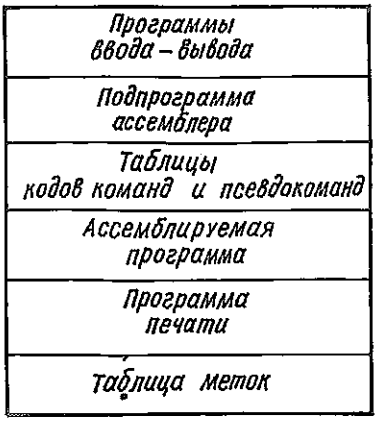


Рис. 4.2

F1	1200
ABCDEF F1	FFFF
⋮	
XY	FFFE
00	

Рис. 4.3

3) получается машинный код и тип команды, соответствующий мнемонике; 4) анализируется первый операнд, получают его тип, значение или имя регистра; 5) в соответствии с видом команды вызывается подпрограмма ее обработки, которая выполняется; 6) печатается текст программы; 7) переход к п.1.

В таблице меток (рис. 4.3) на одну метку отводится 8 байт (6 байт – ее код, 2 байта – ее адрес). Здесь отмечается двойное определение (FFFF) и отсутствие определения метки в команде EQU(FFFE). Конец таблицы указывается нулевым значением очередного байта.

Таблица команд. Таблица команд состоит из двух таблиц OPR и INSTR (рис. 4.4). Один элемент таблицы INSTR можно получить из мнемонического кода команды, типа команды и ее машинного кода. Все команды в соответствии с мнемоническим кодом записываются по алфавиту: первая группа начинается с символа A и т. д. Вторая таблица содержит смещение начального адреса каждой из групп, собранных в алфавитном порядке в первой таблице. Например, для команд группы C смещение равно 6. Сначала поиск производится в таблице OPR (определяется группа), а затем во второй таблице находятся тип и код команды. Причем поиск здесь ведется по второму и третьему символам мнемоники, т. е. если ACI , то по CI и т. д.

Выделяются следующие группы команд (типы):

1) не имеющие операторов (PCHL, SPHL, XCHG, RLC, DAD и т.д.);
2) имеющие исходный регистр (ADD, ADC, SUB, ANA, XRA и т.д.);
3) имеющие регистр места назначения (INR, DCR);
4) пересылки (MOV r_1 , r_2);
5) имеющие 1 байт в операторе (ADI, ACI, SUI, SBI, XRI и т.д.);
6) непосредственная (MVI r, b);
7) имеющие 2 байта в операнде (STA , LDA, JZ, CALL и т.д.);
8) считывающие регистры B , D , H , SP (INX, DCX, DAD);
9) работающие со стеком (PUSH, POP);
10) считывающие регистры B , D (STAX, LDAX);
11) загрузки слова (LXI rp, W);

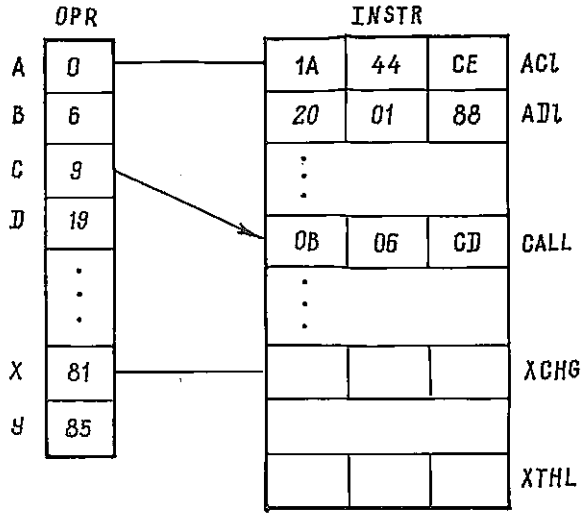


Рис. 4.4

12) прерывания (RST n);

13–18) соответственно ORG, END, DB, DW, DS, EQU.

Поскольку вся мнемоника состоит из 26 символов, для изображения одного из них отводится пять разрядов. Например, в случае команды ACI для второго символа C формат будет 00011, для третьего символа I формат соответствует коду 01001 (тип 000100(4), машинный код CE), т. е. три байта таблицы будут содержать код 1A 44 CE.

При распознавании команд в соответствии с их начальным символом по таблице OPR сначала ищется смещение для этого символа и количество таких символов. Затем в соответствии со вторым и третьим символами получаем из найденной группы машинный код и тип команды, согласно введенной мнемонике. Такое формирование машинного кода из мнемоники за два шага приводит к экономии времени и памяти.

Анализ операндов. В ассемблере МП КР580ИК80 имеются три типа форматов операндов: отсутствуют, имя регистра, выражение. В ассемблере эти типы различаются по номерам. Имена регистров имеют цифровые коды B–0, C–1, PSW–9. Из этих трех типов первый и второй легко анализируются, анализ выражения сложен.

Анализ операндов начинают с выяснения возможности определения имени из заглавия операнда: 1) если ее нет, то проверить наличие конца операнда. Если конца операнда нет, то операнд не существует; 2) если из заглавия определяется имя, то установить, совпадает ли оно с именем регистра. Если совпадает, то операнд является именем регистра, если не совпадает, то операнд является выражением. При обнаружении выражения члены в нем могут быть связаны знаками + , – . Член выражения может быть меткой, десяти- или шестнадцатеричным числом, символьной константой или текущим адресом ($). Значение члена выражения либо уменьшается, либо увеличивается в выражении. После вычисления операнда его использование зависит от конкретного

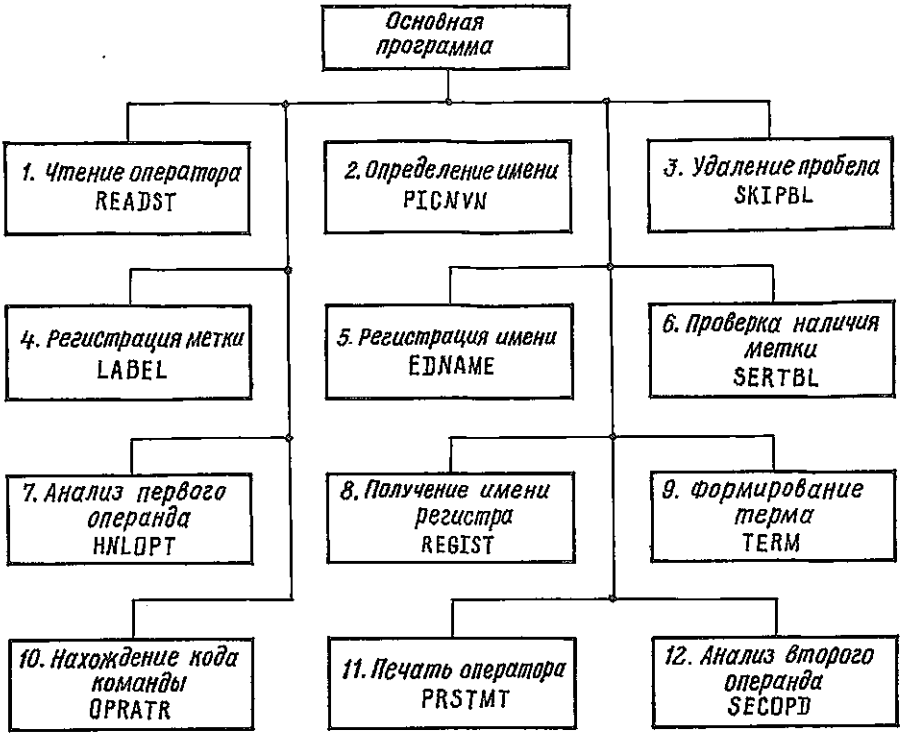


Рис. 4.5

типа команды.

Структура ассемблера. Программа включает головной модуль и 12 основных подпрограмм. Рассмотрим их назначение (рис. 4.5):

1) чтение оператора. Эта подпрограмма из области ввода ассемблируемой программы считывает один оператор. В конце оператора стоит код возврата ОА, количество символов в нем максимум 40, остальные игнорируются;

2) определение имени. Подпрограмма выделяет из строки оператора имя, длина которого не более 6 символов. Первыми могут быть символы ?, @, A–Z, последующими A–Z, 0–9;

3) удаление пробела из слова. Подпрограмма из области, адрес которой указан в HL, удаляет пробел;

4) регистрация метки. Подпрограмма определяет наличие считанного имени в таблице меток. Если имени нет, то записывается его адрес. При наличии адреса в таблице меток вместо него записывается код FFFF. Во втором проходе имя существует обязательно и выясняется только, что оно равно кодам FFFF (повторение имени) или FFFE (отсутствие имени в псевдокоманде EQU). В этих случаях устанавливается код ошибки;

5) регистрация имени. Если имя зарегистрировано в таблице меток, то оно в подпрограмме определяется через регистр DE. Если имя не зарегистрировано, то указывается ошибка отсутствия определения;

6) проверка наличия метки. Подпрограмма выясняет, существует ли данное имя в таблице меток. Если существует, то регистр С устанавливается в 0, а HL указывает адрес вхождения. Если не существует, то в регистр С устанавливается код шести;

7) анализ первого операнда. В подпрограмме анализируется один операнд. Результат анализа вводится в регистр С и области V , W . Когда С=0, операндов нет; при С=1 операнд является именем регистра; при С=С операнд является выражением;

8) получение имени регистра. Если вводимое имя – имя регистра, то в подпрограмме устанавливается признак 1, в противном случае – признак 0. Адрес регистра вводится в область V ;

9) формирование терма. Подпрограмма запрашивает значение членов, формирующих выражение. Начальный адрес членов устанавливается в паре HL;

10) нахождение кода команды. Эта подпрограмма вызывает тип и код команды, соответствующей введенной мнемонике. Тип и код команды заносятся в специальные области;

11) печать оператора. Эта подпрограмма печатает один оператор исходной программы вместе с машинным кодом;

12) анализ второго операнда. В этой подпрограмме анализируются операнды, начиная со второго. Перед операндом ставится запятая.

Рассмотрим работу головной программы. Сначала производится занесение очередного операнда в буферную область. Затем выделяется имя (не более 6 символов), которое начинается с символов ? , @ , A–Z . Производится анализ, т. е. определяется, это метка или нет. Если это метка (завершается символом :), то начинается ее обработка. На первом проходе проверяется совпадение введенной метки с уже имеющимися в таблице. Если такая метка есть, то для нее в адресной части записывается код 0FFFFH. Если такой метки нет, то она записывается в конец таблицы меток с ее адресом из счетчика. Если обрабатывается команда EQU и метка не определена, то в таблицу меток в адресное поле записывается код FFFEH. На втором проходе производится анализ таблицы меток. Если в адресной части записан код FFFF (двойное определение метки) либо код FFFE (отсутствие определения EQU), то формируется сообщение об ошибке.

Затем производится выделение кода команды. Для этого определяется мнемоника операции, по таблице операций находится соответствующая строка. Из этой строки выделяются тип команды (0–17) и ее машинный код. Обработка команды производится в зависимости от ее типа. Если операндов нет, то записывается машинный код команды. Однобайтовые команды формируются путем логического сложения кода команды и номера (номеров) регистра. Например, для команды ADD r код в таблице 80H, если это команда ADD C, то окончательный код будет 1000 0000V00000001=10000001, т.е. 81H.

Для двухбайтовых команд первый байт берется из таблицы команд, второй формируется при обработке первого операнда. Например, для команды ADI 10H будет сформирован код C610. Для команды MVI r , B первый байт формируется аналогично однобайтовой команде, т. е. код команды из таблицы складывается с кодом регистра.

Для трехбайтовой команды код из таблицы команд объединяется с двумя

байтами адреса. При этом если в качестве адреса перехода указана метка, то из таблицы меток (на втором проходе) выбирается значение адреса и подставляется в код команды. Например, для команды LDA 1020H будет сформирована последовательность 3A2010. Для команд, работающих с парами регистров B , D , H , PSW , код команды логически складывается с кодом регистра. Например, для команды INX H будет сформирован код 0000 0011∨00100000=0010 0011 (23), для команды PUSH PSW – код 1100 0101∨00110000=1111 0101 (F5H). Для команды RST n код складывается с номером перехода, сдвинутым влево на три бита. Например, для команды RST 5 код складывается с 5–1100 0111∨0010 1000=1110 1111 (EFH).

При обработке псевдокоманды ORG значение адреса из поля операнда заносится в начальное состояние адресного указателя. По псевдокоманде END происходит переход ко второму проходу (для первого прохода) либо на печать (для второго прохода). По псевдокоманде DS указатель адреса увеличивается на значение операнда. По псевдокомандам DB , DW с указанных адресов в последовательные байты памяти заносятся значения данных. Например, по оператору 100 DB 1, 2 сформируется код 0102; по оператору DW 1, 2 – соответственно код 0100 0200. По псевдокоманде EQU в таблицу меток заносится значение из поля операнда.

4.3. ПОНЯТИЕ О ФОРМАЛЬНЫХ СИСТЕМАХ

Основы описания языков. Прежде чем перейти к рассмотрению компиляторов, познакомимся с основами формального описания языка.

Для построения языка достаточно задать набор правил (грамматику), на основании которых могут быть построены все допустимые предложения языка. На стадии анализа в компиляторе выполняется процедура для определения соответствия данного предложения грамматике языка программирования. Введем несколько основных понятий:

1) алфавит представляет собой конечное множество символов. Для языка программирования алфавит включает 26 латинских букв, 10 цифр и ряд специальных знаков;

2) предложение над словарем V есть последовательность символов, возможно повторяющихся и принадлежащих V . Если V = (a, b), то объекты a, ab, bbb , aabbaaba являются цепочками. Длину цепочки обозначим |w| . Под V^* будем понимать все возможные цепочки над словарем V;

3) языком L над словарем V называется подмножество из V^* . Для его построения и используется набор правил – грамматика. Для описания этих правил применяется так называемый метаязык. К одному из них относится форма Бэкуса–Наура (БНФ), названная по именам двух ученых, принимавших участие в разработке языка АЛГОЛ 60, где впервые было использовано данное описание. При его применении символ, расположенный слева от знака = , заменяется выражением, стоящим справа, причем альтернативные символы разделяются вертикальной чертой.

Различают терминальные и нетерминальные символы. Терминальные не могут быть разделены на более мелкие. Нетерминальные символы могут состоять как из терминальных, так и нетерминальных символов.

Пример. Рассмотрим предложение: Студентка (студент) сдает экзамен (зачет). Представим его в БНФ следующим образом:

1) < Предложение > : = < подлежащее > < сказуемое > < дополнение >
2) < Подлежащее > : = студентка | студент
3) < Сказуемое > : = сдает
4) < Дополнение > : = экзамен | зачет

Используя данные правила, можно построить четыре предложения.

Более точным определением грамматики является следующее: грамматика – это четверка объектов (N , T , P , S), где N – нетерминалы; T – терминалы; P – правила грамматики; S – начальный элемент.

Для рассмотренного случая имеем N = {П, С, Д}, где П – подлежащее; С – сказуемое; Д – дополнение; T = {студент, студентка, сдает, экзамен, зачет}; P = {S = П С Д; П – студент, студентка; С – сдает; Д – зачет, экзамен}. Можно построить различные цепочки: типа П С зачет, П сдает Д, но смысл имеют лишь цепочки, состоящие из терминальных символов. Таким образом, для данной грамматики G = (N, T, P, S) язык L есть множество всех цепочек, состоящих из T и выводимых с помощью P.

Для классификации языков американским лингвистом Хомским предложено четыре класса в зависимости от способов задания правил вывода (продукций).

1. Регулярная грамматика (класс 3). В ней все продукции из P имеют вид $A \to BC$ или $A \to C$, где A, B∈N, C∈T. Этот вид грамматик удобен для теоретического описания лексического анализа в компиляторах.

2. Контекстно-свободная грамматика (класс 2). В этом случае все продукции из P имеют вид $A \to w$, где $w \in NUT$ и $A \in N$. Этот класс грамматик описывает большинство языков программирования.

3. Контекстно-зависимая грамматика (класс 1). В этом случае все продукции из P имеют вид $\alpha A \beta \to \alpha X \beta$, где $A \in N, \alpha, B, X \in TUN$.

4. Грамматика без ограничений (класс 0). В этом случае не накладывается никаких ограничений на правила подстановок. К этому классу относятся естественные языки.

Рекурсия. В приведенном примере на основании записанных правил порождается конечное число предложений. Таких правил было бы недостаточно для описания всех целых чисел, а тем более для описания языка программирования. Для расширения языка допускаются рекурсивные определения.

Грамматика называется непосредственно леворекурсивной, если содержит правило вывода $<A> \to <A>B$, непосредственно праворекурсивной, если содержит правило вывода $<A> \to B<A>$, и с непосредственным самовставлением, если содержит правила вывода $<A> \to B<A>C$, где A – нетерминал, B, C∈TUN.

Грамматика для описания всех целых чисел могла бы принадлежать любому из трех типов, однако в большинстве случаев используется леворекурсивная грамматика:

< Целое > :: = < знак > < целое без знака > | < целое без знака >
< Целое без знака > : < целое без знака > < цифра > | < цифра >
< Цифра > : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
< Знак > : : = + | –

Анализ предложений. Рассмотрим грамматику G = ((A), (01), P, S), где P = {S → 0S , S → 1S, S → 0, S → 1}. Язык, порождаемый G, состоит из всех

возможных бинарных цепочек. Например, цепочка 1010 выводится следующим образом: S → 1S → 10S → 101S – 1010. Этот процесс можно представить с помощью дерева вывода, узлы которого соответствуют нетерминальным символам, а листья – терминальным.

Дерево вывода будем рассматривать как инструмент решения двух разных задач. Первая задача, связанная с перечислением цепочек (предложений), которые можно образовать в соответствии с правилами данной грамматики, называется выводом. Вторая задача связана с распознаванием данной цепочки (предложения) с целью проверки ее принадлежности данному языку. Этот процесс распознавания цепочки называется синтаксическим анализом или грамматическим разбором.

В алгоритмах синтаксического анализа цепочки всегда просматриваются слева направо. Первый символ слева, идентифицирующий либо нетерминальный или терминальный символ языка, будет первой частью синтаксического дерева. Таким образом, чтобы для примера задания грамматики выяснить, допустимо ли предложение "студент сдает зачет", необходимо с левой стороны применить правила, ведущие к выражению "предложение": 2, 3, 4, 1. Поскольку синтаксическое дерево найдено, предложение является допустимым в данной грамматике.

Постфиксная запись. Программист обычно описывает операции, используя так называемую инфексную форму записи, в которой знак операции ставится между операндами. Например, (A + B) * C. Вычисление выражения в этой форме является непростой задачей, поскольку операцию нельзя выполнить до вычисления второго операнда, который в свою очередь может быть сложным выражением. Эту трудность можно обойти, если использовать так называемую постфиксную запись, в которой знак операции стоит за операндами. Например, инфексной записи A + B соответствует постфиксная запись AB+.

Постфиксная запись обладает двумя свойствами: для записи любого выражения не нужны скобки (A+B)*C → AB+C*; к моменту считывания очередного оператора операнды уже прочитаны. Благодаря этим свойствам выражение в постфиксной форме может быть вычислено с помощью простого алгоритма:

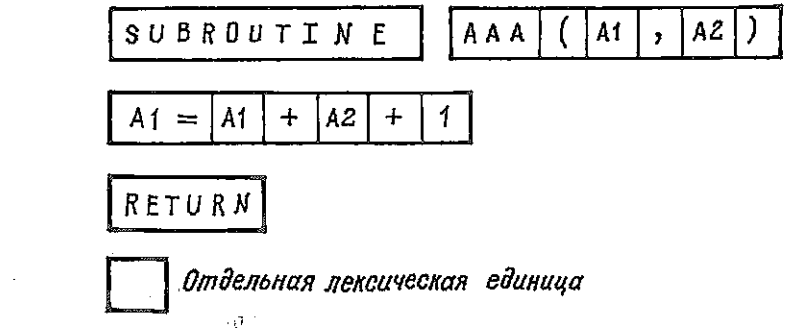
```
do while   (пока в выражении есть лексемы);
read       (считать следующую лексему);
if         (лексема-операнд) then записать ее в стек;
if         (лексема-оператор) then выполнить операцию над последними
           элементами, записанными в стек,и заменить эти элементы результатом.
```

4.4. РАЗРАБОТКА КОМПИЛЯТОРОВ (АНАЛИЗ)

Анализ языка. Разработка начинается с определения языка программирования. Если он задан (например, ФОРТРАН), то формат входных данных определен. Если же целью проекта является реализация языка для решения задач из теории графов, то появляются большие возможности по реализации средств языка. Назначение языка должно быть указано пользователем, после чего формируется описание языковых конструкций.

Лексический анализ. Первый шаг компиляции заключается в чтении элементов исходной программы и разделении их на синтаксические классы. Для облегчения построения других блоков компилятора лексический анализатор вырабатывает однородные символы фиксированного размера. При этом основные элементы исходной программы помещаются в таблицу стандартных символов. Для каждого элемента указывается его синтаксический класс (ключевое слово, ограничитель, идентификатор, литерал). С помощью указателя задается таблица, в которой хранится его исходная форма. При этом строятся таблицы идентификаторов и литералов.

Во время лексического анализа различные представления стандартизируются, а комментарии отбрасываются. Из исходной программы, содержащей комментарии и пробелы, будет получена таблица стандартных символов. Приведем лексический анализ на примере простого фрагмента на языке ФОРТРАН:



В процессе лексического анализа в первой строке будут выделены ключевое слово, идентификатор, разделитель, идентификатор, разделитель, идентификатор, разделитель. Аналогично проанализируем остальные строки.

Стандартные символы имеют фиксированную длину и состоят из указателя синтаксического типа и указателя элемента таблицы, соответствующего базовому элементу.

Таким образом, в лексическом анализе используются следующие наборы данных:

1) исходная программа (воспринимается компилятором как одна строка символов);

2) таблица терминальных символов — постоянный набор, в каждой строке которого находятся символ и указатель его класса (операция, ключевое слово, разделитель);

3) таблица литералов (создается при лексическом анализе для описания литералов, использованных в программе). Каждому литералу соответствует одна строка таблицы в виде:

Литерал	Основание	Формат	Точность	Адрес

4) таблица идентификаторов создается для описания всех идентификаторов программы. Каждому идентификатору соответствует одна строка таблицы в виде:

Имя	Атрибуты	Адрес

5) таблица стандартных символов создается для представления программы строкой лексических единиц. Элемент этой таблицы имеет вид

Значение	Индекс

Алгоритм лексического анализа начинается с разбора входной строки и выделения соответствующих лексических единиц. Затем происходит заполнение соответствующих таблиц. Входная строка разделяется на лексические единицы разделителями. Поэтому каждый символ сравнивается с разделителем. Стоящие подряд символы, которые не являются разделителями, объединяются в лексические единицы.

Сначала все лексические единицы сравниваются с элементами таблицы терминальных символов (ТЕР). В случае совпадения этот символ помечается как терминальный и формируется стандартный символ типа ТС, который помещается в таблицу стандартных символов. Остальные лексемы рассматриваются как идентификаторы (ИД) или литералы. Если лексема удовлетворяет правилам описания идентификатора (первый символ буква, длина не более 8 байт), то она классифицируется как идентификатор, в противном случае выдается ошибка определения идентификатора.

Распознанный идентификатор заносится в таблицу идентификаторов, если он ранее не был туда записан. Остальная информация об идентификаторе записывается на следующих фазах. Независимо от того, был ли записан распознанный идентификатор в таблицу идентификаторов, он в любом случае записывается в таблицу стандартных символов с типом ИД.

Числа и другие самоопределяемые данные классифицируются как литералы. Происходит просмотр таблицы литералов. Если рассматриваемого литерала там нет, то создается новый элемент. При этом в таблицу литералов записываются все его атрибуты – длина, точность и т.д. Независимо от того, записан элемент в таблицу литералов или нет, он заносится в таблицу стандартных символов типа ЛТ.

Лексический анализ производится либо за один просмотр исходной программы, либо путем вызова лексического анализатора при запросе очередной лексемы.

Для приведенного фрагмента программы на языке ФОРТРАН в результате лексического анализа используются табл. 4.1–4.4. Заполняются таблицы в следующем виде: таблица терминальных символов (табл. 4.1), таблица идентификаторов (табл. 4.2), таблица литералов (табл. 4.3) и таблица стандартных символов (табл. 4.4).

Синтаксический анализ. После фазы разделения программы в процессе лексического анализа на основные элементы следует фаза распознавания операторов, составленных из этих элементов. Компиляторы на данной стадии вырабатывают промежуточную форму, на основании которой генерируется код (возможно, оптимизированный). Интерпретаторы получают промежуточную форму, готовую для выполнения.

Рассмотрим два основных метода синтаксического анализа: сверху вниз с возвратом и снизу вверх. Второй метод основан на использовании грамматик

Таблица 4.1

Индекс	Символ	Разделитель
1	SUBROUTINE	Нет
2	(	Да
3	,	"
4	)	"
5	=	
6	+	
7	RETURN	Нет

Таблица 4.2

Имя	Атрибуты
AAA	
A1	Заполняется позже
A2	

Таблица 4.3

Литерал	Основание	Точность	Адрес
1	DECIMAL	1	Заполняется позже

Таблица 4.4

Тип	ТЕР	ИД	ТЕР	ИД	ТЕР	ИД	ТЕР	ИД	ТЕР	ИД	ТЕР	ИД	ТЕР	ЛТ	ТЕР
Индекс	1	1	2	2	3	3	4	2	5	2	6	3	6	1	7
Лексема	SUBROUTINE	AAA	(	A1	,	A2	)	A1	=	A1	+	A2	+	1	RE-TURN

с операторным предшествованием.

Анализатор, работающий сверху вниз, на каждой стадии определяет, является ли рассматриваемая строка допустимым символом языка, а предложение – допустимой формой. Если рассматриваемые символы являются нетерминальными, для них устанавливаются новые подцели. Процесс проверки продолжается для определения реализации новой подцели. Приведем пример распознавания десятичных чисел.

Пример. Для разбора используются следующие правила подстановки:

а :< Десятичное число > ::= < целое > | < дробное >
б < Целое > ::= < целое без знака > | +< целое без знака >|–<целое без знака >
в < Дробное >::=< целое без знака >
г < Целое без знака > ::= < цифра > < целое без знака > | < цифра >
д < Цифра > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Таблица 4.5

Шаг	Строка цели, подцели		Цель, подцель	Правило	Справедливость правила	После возврата	
	старая	текущая				правило	справедливость правила
1		+	ДЧ	а1	Да		
2		+	Ц	б1	Нет		
3		+	Ц	б2	Да		
4	+	1	ЦБЗ	г1	Да	г2	Да
5	+	1	ЦФ	д2	Да		
6	+1	.	ЦБЗ	г1	Нет	г2	Да
7	+1	.	ЦБЗ	г2	Нет		
7а	+1	.	Ц	б1	Да		
8а	+1	.	ДЧ	а2	Да		
9	.	2	ЦБЗ	г1	Да↑	г2	Да
10	.	2	ЦФ	д3	Да↑		
11	+2		ЦБЗ	г1, г2	Нет		

Правила помечаются буквами, а альтернативы в них – цифрами, т. е. целое без знака идентифицируется б2. В качестве примера рассмотрим распознавание числа +1.2. Последовательность шагов дана в табл. 4.5.

Первая цель заключается в доказательстве, что предложение является десятичным числом. Для этого надо показать, что оно начинается с целого. Правило б2 допускает это (шаг 3). Следующая подцель – показать, что символы, следующие за +, являются целым без знака. К символу 1 применено правило д2 (цифра). Для шагов 6, 7 правила г1, г2 не подходят, поскольку за цифрой 1 следует точка. Поэтому необходим возврат к ближайшему альтернативному правилу подстановки. В результате правило, примененное на шаге 4, заменяется на правило г2 (подцель – целое уже достигнуто).

Ни одно правило не порождает целое вида +1. Устанавливается новая подцель (а2), по которой за целым следует дробная часть. Правило а2 выполняется (шаг 8а), поскольку за точкой должно следовать целое без знака. Это распознается на шагах 9, 10. В конце для распознавания конца строки осуществляется возврат к шагу 9. Все цели достигнуты.

Недостатком анализа сверху вниз является невозможность с помощью первого метода распознавать левую рекурсию и его невысокая скорость.

В отличие от анализаторов сверху вниз распознаватели снизу вверх выполняют разбор с помощью многократного поиска самого левого простого выражения – основы формы. Применяя правило грамматики, такие анализаторы сворачивают ее в нетерминальный символ.

Если А является частью основы, а В нет, или наоборот, то А(В) предшествует (старше) В(А). Если существует однозначное отношение предшествования для любой пары символов, справедливое для всех форм, то эти отношения будут указывать начало и конец основы. Если это условие выполняется, грамматики называются с предшествованием. Рассмотрим это положение на примере грамматик с операторным предшествованием. Такими грамматиками могут быть определены арифметические выражения. Конструкции, отличные от арифметических выражений, могут быть изменены с помощью лексического анализа таким образом, что вход анализатора будет подчиняться правилам грамматики с операторным предшествованием.

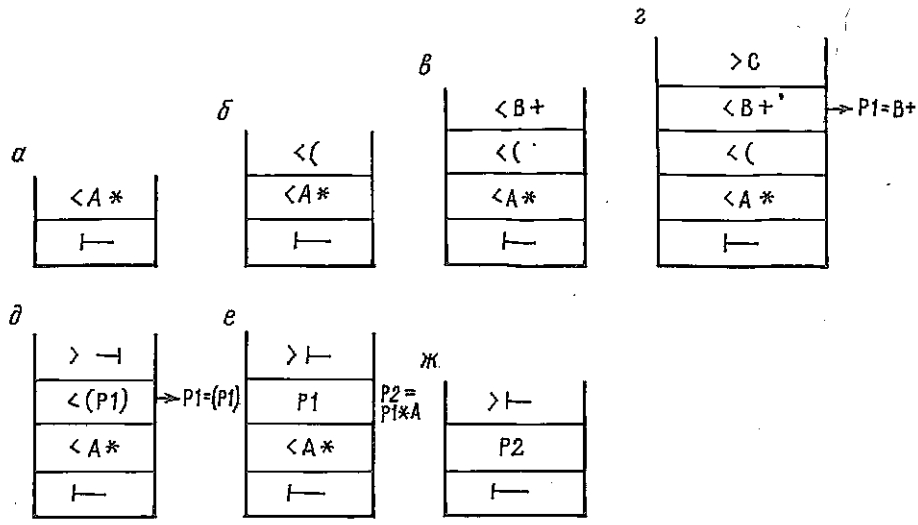


Рис. 4.6

Рассмотрим пример с операциями * , + , (,) , ⊢ ⊣ , где последние два символа представляют начало и конец арифметического выражения. Обозначим начало и конец основы соответственно $<$, $>$. Тогда правила предшествования для описанного набора операторов могут быть записаны так, как показано в табл. 4.6. В строке указан первый оператор, в столбце – второй, а элемент таблицы задает их отношение предшествования. Пустые элементы таблицы обозначают невозможность следования операторов друг за другом (за „(" не может следовать „)"). При встрече такой комбинации компилятор должен выдавать ошибку.

Алгоритм разбора на основании данной таблицы следующий. Форма просматривается слева направо, пока для пары операторов не будет найдено отношение $>$ (конец основы). Теперь форма просматривается влево от этой точки до символа $<$ (начало основы), а затем основа сворачивается. При этом используется стек. В качестве примера рассмотрим анализ выражения ⊢ A * (B+ +C) ⊣ , приведенного на рис. 4.6.

Сначала в стек загружается символ ⊢ (рис. 4.6, *а*). Следующий оператор * (рис. 4.6, *б*) по таблице предшествует ⊢ (основа не обнаружена), и в стек заносятся символы A * (рис. 4.6, *в*). Следующий оператор (. Поскольку $* < ($, стек опять пополняется (рис. 4.6, *г*). Это происходит до тех пор, пока не появится символ). Из табл. 4.6 имеем $+ <$), поэтому идентифицируется конец основы. Новых элементов в стек не добавляется и обрабатывается выражение P1 = B+C , поскольку последний символ $<$ стоит перед B (рис.4.6, *д*). На вершине стека находится (P1). Следующий оператор ⊣ , поэтому скобки убираются (рис. 4.6, *е*). Затем основой является выражение P2 = P1 * A . На этом завершается разбор выражения (рис. 4.6, *ж*).

Рассмотрим основные наборы данных синтаксического анализатора. На его вход подается таблица стандартных символов, которая является входной для стека. Каждый символ таблицы помещается в стек один раз.

Таблица 4.6

Оператор	Оператор				
	+	*	(	)	—|
|—	<	<	<	—	Выход
+	>	>	<	>	>
*	>	>	<	>	>
(	<	<	<	—	—
)	>	>	—	>	>
—|	—	—	—	—	—

Стек содержит набор стандартных символов, обрабатываемый фазами синтаксического анализатора. Редукции — синтаксические правила исходного языка.

Общая форма редукций

метка: старая верхушка стека (программа интерпретатора)
новая верхушка стека (следующая редукция)

Фаза синтаксического анализа выполняется следующим образом: 1) правила проверяются последовательно, сравнивается старая верхушка стека с фактической до тех пор, пока они не совпадут; 2) управление передается программам, определенным в поле программ интерпретации, которые выполняются слева направо; 3) после возврата управления анализатору последний изменяет верхушку стека так, как это записано в поле новой верхушки стека; 4) затем следует шаг 1, начиная с редукции, определенной в поле "Следующая редукция".

Рассмотрим действие трех редукций

```
1: (xxx)
2: < идентификатор > : SUBROUTINE  (начало процедуры)
     S1  (xxxx) 4
< любой > < любой > < любой >  (ошибка)  S2  S1 * 12
4 идентификатор > < любой > (параметр)  S4  S1  xx  14
```

При интерпретации выполняются следующие шаги:

1) первые три стандартных символа помещаются в стек из таблицы стандартных символов;

2) проверяется соответствие трех элементов верхушки стека полю редукции

< идентификатор > : SUBROUTINE.

Если такое соответствие есть, вызывается подпрограмма, вычеркиваются метка и символ : , помещаются следующие 4 стандартных символа в стек и выполняется переход к редукции 4. Если проверка окажется неудачной, вызывается программа ОШИБКА, удаляется третий символ из стека, берется еще один символ из таблицы стандартных символов и осуществляется переход к редукции 2. Эти действия будут происходить до тех пор, пока не найдено ключевое слово SUBROUTINE или не будут исчерпаны все стандартные символы.

Синтаксический разбор завершается фазой интерпретации, которая выполняется соответствующим набором программ. Назначение последних заключается в создании промежуточной формы исходной программы и в введении информации в таблицу идентификаторов. В качестве промежуточной формы может быть матрица, каждый элемент которой содержит тетраду: номер строки, операцию, операнд 1 , операнд 2.

4.5. ПОСТРОЕНИЕ КОМПИЛЯТОРОВ (СИНТЕЗ)

Промежуточная форма, полученная в процессе синтаксического анализа, может быть оптимизирована следующим образом: 1) повторное вхождение подвыражения в один и тот же арифметический оператор исключается, подвыражение вычисляется один раз;

2) все операции, операнды которых константы, выполняются во время компиляции и заменяются найденным выражением;

3) из цикла выносятся выражения, не зависящие от переменной цикла;

4) сложные условные операторы упрощаются с помощью правил булевой алгебры.

Пусть в результате синтаксического анализа арифметического выражения A = B + C + (B + C) * D получена матрица (табл. 4.7).

После оптимизации, которая является машинно-независимой, получена матрица, представленная в табл. 4.8. После оптимизации матрицы, созданной синтаксическим анализатором, выполняется генерация кода. Простейшее решение заключается в использовании так называемых кодовых заготовок для каждой строки матрицы. При этом просматриваются строки матрицы и для каждой из них генерируется код, определенный для оператора этой строки. Это аналогично макровызову, в котором имя макроса определяется операцией, а параметрами являются операнды и отведенная под результат промежуточная память.

Таблица 4.7

Результат	Операция	Операнд 1	Операнд 2
S1	+	B	C
S2	+	B	C
S3	*	S2	D
S4	+	S1	S3
S5	=	A	S4

Таблица 4.8

Результат	Операция	Операнд 1	Операнд 2
S1	+	B	C
S3	*	S1	D
S4	+	S1	S3
S5	=	A	S4

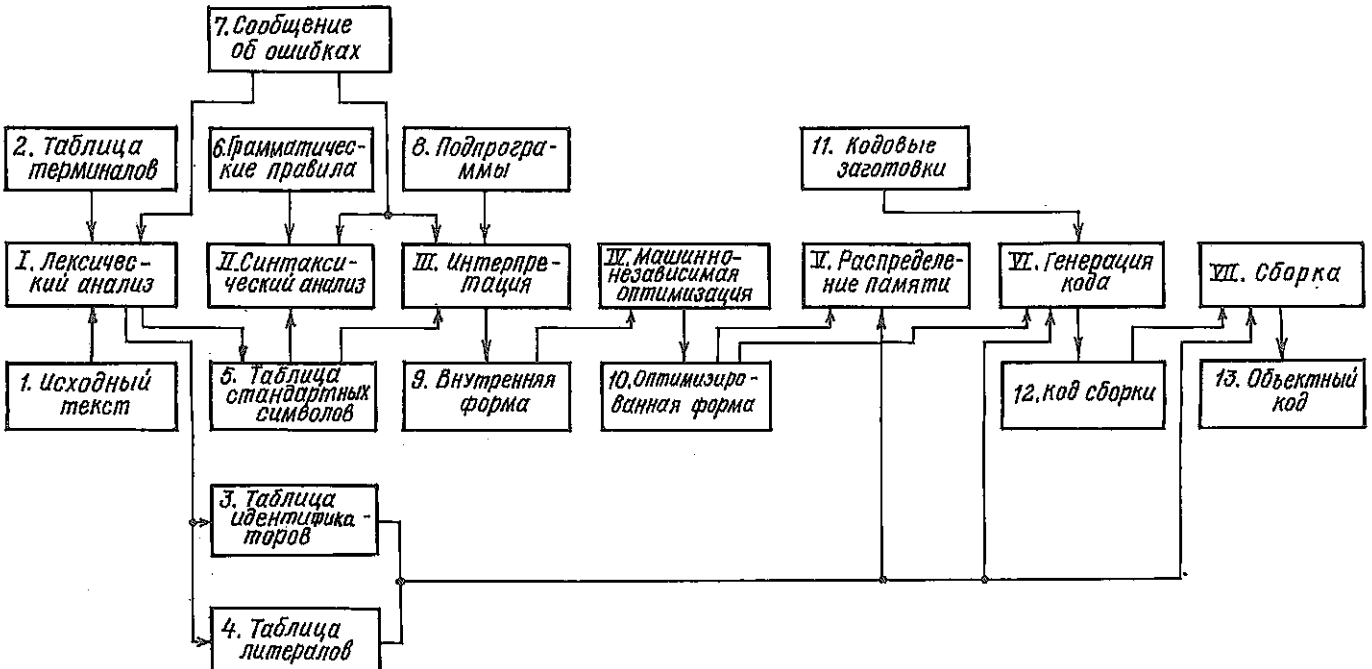

7. Сообщение об ошибках
2. Таблица терминалов
6. Грамматические правила
8. Подпрограммы
11. Кодовые заготовки
I. Лексический анализ
II. Синтаксический анализ
III. Интерпретация
IV. Машинно-независимая оптимизация
V. Распределение памяти
VI. Генерация кода
VII. Сборка
1. Исходный текст
5. Таблица стандартных символов
9. Внутренняя форма
10. Оптимизированная форма
12. Код сборки
13. Объектный код
3. Таблица идентификаторов
4. Таблица литералов

Рис. 4.7

Таблица 4.9

Номер команды	Строка матрицы				Неоптимальный код	Длина, байт	Оптимизированный код	Длина, байт
	операция	операнд	операнд	результат				
1					LDA Z	3	LDA Z	3
2					LXI H, Y	3	LXI H, Y	3
3	+	Z	Y	T1	ADD M	1	ADD M	1
4					STA T1	3	MOV B, A	1
1					LDA Y	3	LDA Y	3
2					LXI H, T1	3	SUB B	1
3	–	Y	T1	T2	SUB M	1	MOV B, A	1
4					STA T2	3		
1					LDA X	3	LDA X	3
2					LXI H, T2	3		
3	+	X	T2	T3	ADD M	1	ADD B, A	1
4					STA T3	3	MOV B, A1	1
1					LDA D3	3	LDA D3	3
2					LXI H, T3	3	ADD B	1
3	+	D3	T3	T4	ADD M	1	STA T4	3
4					STA T4	3		

В качестве примера рассмотрим генерацию кода для выражения X + Y – – (Z + Y) +D3(табл. 4.9).

На основании кодовых заготовок для матрицы был сгенерирован неоптимизированный код. При его рассмотрении становится очевидно, что он далек от оптимального. Во-первых, промежуточные результаты T1, T2, T3 можно сохранять не в памяти, а в регистрах (например, в регистре B). Во-вторых, отдельные команды, например, LXI H, Ti становятся лишними и могут быть исключены. В результате в правом столбце табл. 4.9 получается оптимизированный код. Более эффективная версия кода была получена за счет лучшего использования регистровой памяти, что уменьшит число команд записи в память с 4 до 1, исключения трех команд занесения адреса в пару HL , уменьшения объема памяти, занимаемого фрагментом программы (с 40 до 25 байт).

В данном примере генерации кодов выходная программа представлена на языке ассемблера, поскольку символические имена команд и символические адреса нагляднее для читателя. На самом деле компилятор генерирует вместо переменных ссылки на фактические ячейки оперативной памяти. Однако меткам не могут быть присвоены значения, пока не сгенерирована последняя команда. Поэтому стадия генерации состоит из генерации кода и сборки. Вторая фаза логически аналогична второму просмотру ассемблера.

Общая структура компилятора. При анализе компиляции выявлены следующие логические этапы, которые и определяют структуру компилятора (рис. 4.7).

I. Лексический анализ – распознавание базовых элементов и построение таблицы стандартных символов.

II. Синтаксический анализ – распознавание базовых синтаксических конструкций на основе набора грамматических правил (редукций).

III. Интерпретация – построение внутренней формы (матрицы, польской записи) после синтаксического разбора на основании программ интерпретации.

IV. Машинно-независимая оптимизация – получение более оптимальной внутренней формы.

V. Распределение памяти – модификация таблиц идентификаторов и литералов на основании описания данных (DIMENSION в ФОРТРАНе).

VI. Генерация кода – использование кодовых заготовок для замещения строк матрицы и их оптимизация.

VII. Сборка – разрешение символических адресов и генерирование программы на машинном языке.

Этапы I–IV являются машинно-независимыми и определяются только языком, этапы V–VII будут машинно-зависимы и не зависят от языка. В практических реализациях компиляторов этапы могут быть не так четко разделены.

Для связи логических этапов компилятора используется ряд наборов данных, которые имеют следующее назначение:

1) исходный код – вход компилятора;

2) таблица терминалов – постоянная таблица, в которой записаны все ключевые слова и специальные символы языка;

3) таблица идентификаторов – содержит все переменные компилируемой программы. Строится на этапе I, модифицируется на этапах III и V, используется на этапах VI, VII;

4) таблица литералов – содержит все константы исходной программы, создается и используется аналогично таблице идентификаторов;

5) таблица стандартных символов – заполняется на этапе I, состоит из списка лексем, расположенных в том порядке, в котором они встречаются в программе, используется на этапах II, III;

6) грамматические правила – постоянная таблица для хранения правил построения синтаксических конструкций, используется на этапе II;

7) сообщения об ошибках – постоянная таблица, содержащая диагностические сообщения при обнаружении ошибок, используется на этапах I–III;

8) подпрограммы – инициируются синтаксическим анализатором. С их помощью генерируется внутренняя форма исходной программы.

9) внутренняя форма – чаще всего матрица тетрад. Создается на этапе III, оптимизируется на этапе IV, используется для генерации кода;

10) оптимизированная форма – является объектом машинно-независимой оптимизации, используется на этапе VI;

11) кодовые заготовки – постоянная таблица определений, элемент которой определяет код для каждой возможной операции матрицы;

12) код сборки – версия программы на языке сборки, созданная этапом генерации;

13) объектный код – окончательный выход этапа сборки, используется как входная информация для редактора связей (загрузчика).

Интерпретатор транслирует исходную программу во внутреннюю форму, а затем выполняет программу в этой форме. В качестве внутренней формы чаще всего применяют обратную польскую запись. Этапы I–III компилятора (см. рис. 4.7) входят в состав интерпретатора, а вместо этапов IV–VII используется этап интерпретации внутренней формы. Этот этап реализуется программой интерпретации. Например, считав постфиксную запись XY+ , интерпретатор выполняет следующее: помещает в стек X, затем Y , потом выбирает X и Y , суммирует их и заменяет два верхних элемента стека значением суммы X + Y.

Интерпретатор не производит оптимизации, поэтому программа, полученная после интерпретации, выполняется более медленно, чем после компиляции. В связи с этим интерпретаторы используются при разработке и отладке программ. Кроме того, если в процессе вычислений необходимо многократное обращение к подпрограммам, удобнее обращаться к интерпретатору.

В ряде случаев при генерировании программ возможно использование элементов компиляции и интерпретации (например, для разработки трансляторов с проблемно-ориентированных языков). В этом случае трансляция с исходного языка во внутренний происходит методом компиляции, а с внутреннего в машинный — интерпретацией.

Рекомендуемая литература: 1–3, 13, 15.

5. СИСТЕМНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

5.1. ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММ

Современные микроЭВМ представляют собой довольно сложное сочетание аппаратных и программных средств, предназначенных для решения конкретных прикладных задач. Для ускорения и упрощения процесса разработки и отладки программ необходимо соответствующее программное обеспечение. Именно оно превращает физические аппаратные средства в виртуальную машину, более удобную для пользователей.

Жизненный "цикл" программы включает следующие этапы: 1) формирование требований, предъявляемых к программе; 2) составление спецификаций; 3) проектирование; 4) разработка документации; 5) кодирование; 6) тестирование и отладка; 7) сопровождение программы.

Первые шесть этапов составляют традиционный "цикл разработки" программы. На этап кодирования приходится весьма малая доля цикла разработки (рис. 5.1), хотя начинающие программисты зачастую основное внимание уделяют кодированию или записи алгоритма на языке программирования. Еще более неожиданным является распределение трудозатрат между отдельными этапами "жизненного цикла" большой программы. Как показано на рис. 5.2, этап сопровождения таких программ может вызвать бо́льшие трудозатраты, чем весь цикл разработки. Высокая стоимость сопровождения программ определяет необходимость построения цикла их разработки, обеспечивающего возможность снижения объема работ по сопровождению программ.

Этап формулирования требований присутствует в той или иной форме в любой деятельности, направленной на разработку какой-либо системы. На этом этапе вырабатываются критерии, определяющие характеристики проектируемого изделия.

Входы и выходы программы и их связи задаются на этапе *составления внешних спецификаций*. Например, спецификации для редактора текстов определяют формат текстовых файлов и содержат все команды редактирования и функции каждой из них. Однако внешние спецификации не дают описания того, как в программе будут реализованы такие функции. Это является задачей проектирования.

На этапе *проектирования* определяется структура программы в соответствии с формулой Программа = Алгоритм + Структура данных. На данном этапе часто осуществляется декомпозиция общей задачи, которую необходимо решить, на отдельные фрагменты путем выделения некоторого набора взаимодействующих программных модулей высокого уровня. Подобный подход требует дополнительных работ по проектированию и созданию внутренних спецификаций, так как каждый модуль и его взаимосвязи с другими модулями

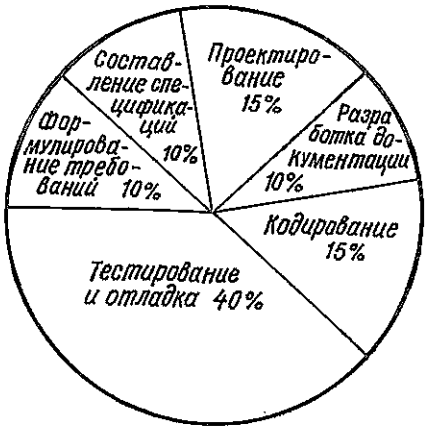


Рис. 5.1

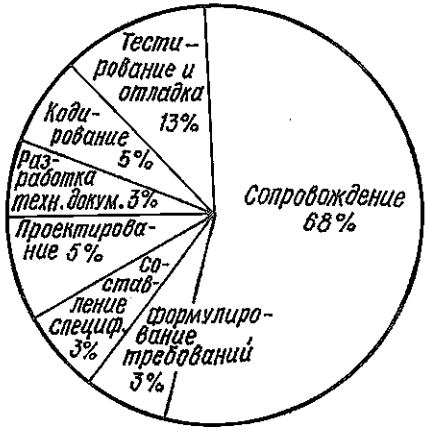


Рис. 5.2

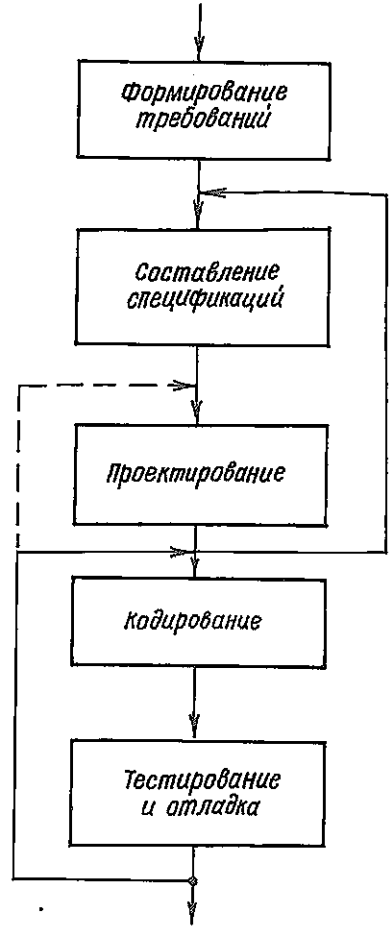


Рис. 5.3

должны быть определены. Затем следует разработать внутреннюю структуру модулей. В зависимости от сложности модуля производится его декомпозиция на подмодули.

В процессе разработки системы нередко требуется осуществить несколько итераций этапов составления спецификаций и проектирования (рис. 5.3). Такая необходимость может возникнуть, когда при проектировании программы вскрываются неопределенности, противоречия или другие недостатки, свойственные внешним спецификациям. Однако в основном потребность в итерациях связана с применением метода нисходящего проектирования программ. После того как программа в соответствии с внешней спецификацией будет спроектирована как комплекс взаимосвязанных модулей, каждый модуль и все его связи с другими модулями должны быть описаны с помощью внутренних спецификаций. Затем каждый модуль проектируется отдельно. Этот процесс повторяется, если осуществляется декомпозиция модуля на подмодули.

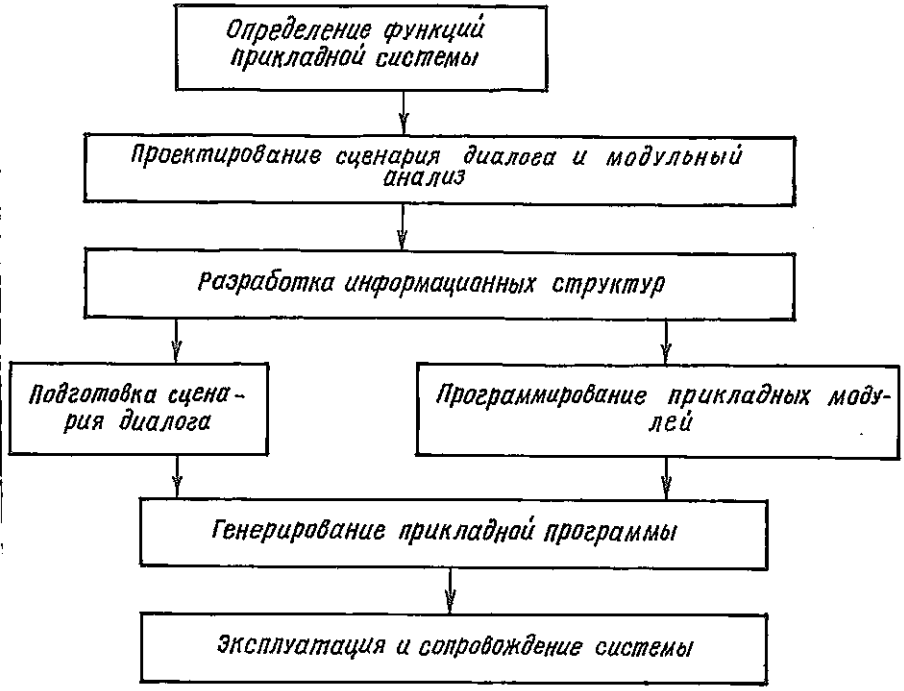


Рис. 5.4

Основная часть документации должна быть создана на этапах составления спецификаций и проектирования. Документация необходима для того, чтобы идеи, заложенные в проект разработки программы, могли быть донесены до разработчиков, пользователей и групп сопровождения программы.

На этапе кодирования проект записывается на языке программирования конкретной вычислительной системы. Анализ разработки программного обеспечения показал, что ошибки при проектировании совершаются гораздо чаще, чем при кодировании, так что качественное проектирование является важнейшей предпосылкой успешного выполнения работ.

Отладка и тестирование предполагают устранение выявленных в программе ошибок, проверку работоспособности программы, а также ее соответствие внешним спецификациям.

Для больших программ необходима организация их сопровождения во время эксплуатации, что вызвано двумя причинами: во-первых, в программах, как правило, остаются ошибки, не выявленные при тестировании; во-вторых, пользователи в ходе эксплуатации программы настаивают на внесении в нее изменений и ее дальнейшем совершенствовании (этого избежать практически невозможно).

В настоящее время главным в любой управляющей системе является программное обеспечение. Современные условия требуют перевода на промышленную основу процессов создания и сопровождения программного обеспечения, при этом необходимо решение комплекса проблем, в том числе по надеж-

ности и качеству. Одним из путей решения указанных задач является совершенствование технологии разработки программных средств, создание инструментальных систем разработки программ. Инструментальные системы должны состоять из средств макетного обмена и монитора ведения диалога, входящих в разрабатываемые прикладные программы, а также из средств подготовки макетов ввода-вывода и сценариев диалога, согласно которым функционируют прикладные программы. Очередность этапов разработки прикладной системы показана на рис. 5.4. Главным в таких инструментальных системах является управляющий программный комплекс, который построен в соответствии с моделью управления, основанной на представлении алгоритма любой задачи направленным графом и на обработке его по единой методике. При этом процесс программирования задачи сводится к соответствующему описанию и кодированию служебной информации о программах, необходимых для решения задачи, и связях между ними.

Функционально управляющий комплекс состоит из интерпретатора приказов, управляющей программы, программы обработки прерываний, средств взаимодействия с системными программами ввода-вывода. Связующим звеном между интерпретатором приказов и управляющей программой является база данных, содержащая в закодированном виде информацию о структуре задачи, множестве программ, необходимых для решения задачи, и связях между ними. Управляющая программа должна допускать рекурсивное обращение, что позволяет строить многоуровневые диалоги. В соответствии с этим база данных имеет многоуровневую структуру.

5.2. РЕДАКТОРЫ ТЕКСТА

Редактор текста представляет собой средство для создания и модификации исходных текстов программ и других текстовых материалов. С помощью редактора, используя вводимые с терминала команды, можно записывать и удалять символы, строки или даже группы строк, содержащиеся в буфере. Во всех системах редактор работает в режиме on-line, его ответ на команды является немедленным и динамичным.

Редакторы обеспечивают высокую эффективность труда программиста. Они позволяют осуществлять поиск новых символов, делать исправления и читать или записывать блоки данных. Одни редакторы работают со строками, другие — с отдельными символами. В первом случае необходим специальный символ конца строки, например возврат каретки. Предполагается, что строка начинается с символа, следующего за возвратом каретки, и заканчивается символом, предшествующим возврату каретки.

Применение видеотерминалов и специальных интерактивных дисплейных процессоров положило начало созданию ряда экранных редакторов. Пользователь может видеть и редактировать сразу целый блок, так как программы имеют определенную структуру, выражающуюся в корреляции между строками, и обладают другими особенностями. Экранные редакторы позволяют пользователю одновременно видеть блок символов от 1 до 25 строк. Курсор выделяет позицию, куда может быть введен новый символ, и такой символ сразу же однозначно отображается в поле экрана.

Рассмотрим построение учебного экранного редактора SEDBK для микро

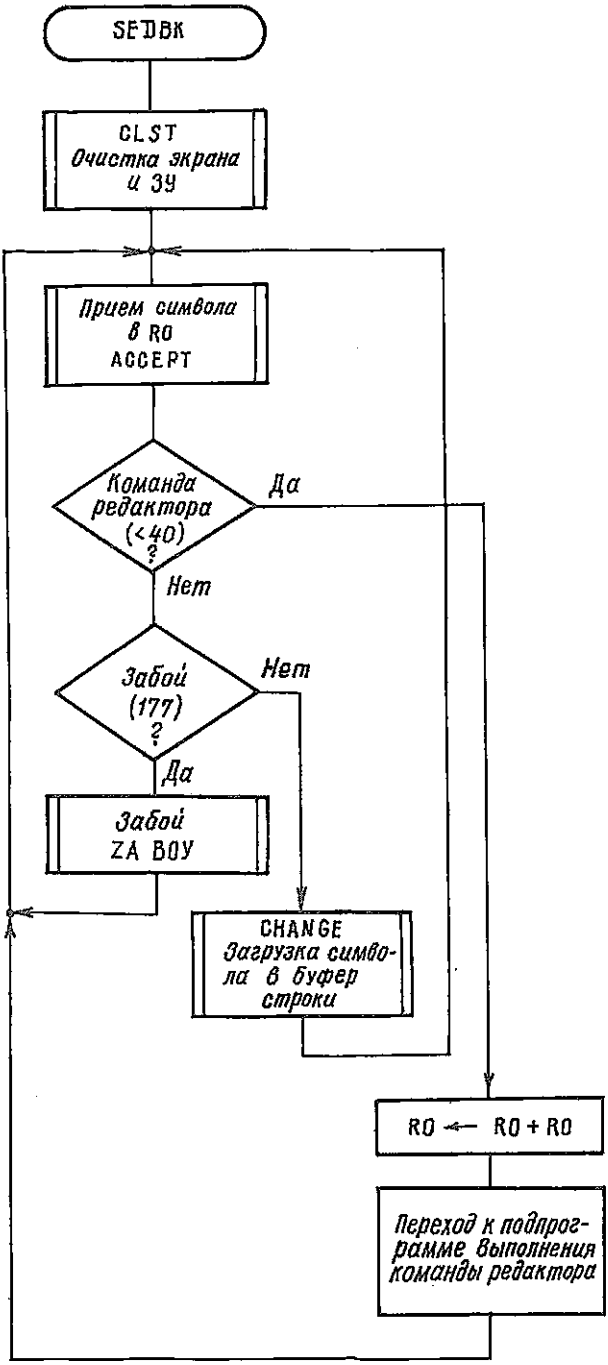

SEDBK
CLST
Очистка экрана
и ЗУ
Прием символа
в R0
ACCEPT
Команда
редактора
(<40)
?
Да
Нет
Забой
(177)
?
Нет
Да
Забой
ZA B0У
CHANGE
Загрузка симво-
ла в буфер
строки
R0 ← R0 + R0
Переход к подпро-
грамме выполнения
команды редактора

Рис. 5.5

ЭВМ ДВК. На рис. 5.5 показана схема редактора, основу которого составляют модули приема символа от клавиатуры терминала; анализа его; выполнения команды редактора; записи в буфера строки и текста; вывода символьного файла на терминал. Каждый символ кодируется байтом согласно коду ASCII . Прием символа от терминала и обратный вывод символьного файла осуществляются соответственно подпрограммами ACCEPT и TYPE:

```
;-----------------------------------
; ПОДПРОГРАММЫ ВВ/ВЫВ С ТЕРМИНАЛА
;-----------------------------------
; ВВОД С ТЕРМИНАЛА  JSR R5,ACCEPT
; ВВЕДЕННЫЙ СИМВОЛ В РЕГИСТРЕ R0
;-----------------------------------

ACCEPT : TSTB   @#177560        ;ОЖИДАНИЕ ГОТОВНОСТИ
         BPL    ACCEPT
         MOVB   @#177562,R0      ;ПРИНЯТЬ СИМВОЛ
         BIC    #177600,R0       ;МАСКИРОВАТЬ БИТЫ 15-7
         RTS    R5

;-----------------------------------
; ВЫВОД НА ТЕРМИНАЛ: JSR R5,TYPE
;                    .BYTE <СИМВОЛЬНЫЙ ФАЙЛ>,200
;-----------------------------------

TYPE   : TSTB   @R5              ;НЕ КОНЕЦ ФАЙЛА ?
         BMI    L5               ;ЕСЛИ ДА - НА ВЫХОД
  L6   : TSTB   @#177564         ;ОЖИДАНИЕ ГОТОВНОСТИ
         BPL    L6               ;УСТРОЙСТВА
         MOVB   (R5)+,@#177566   ;ПЕРЕСЛАТЬ СИМВОЛ
         BR     TYPE             ;ПОВТОРИТЬ
  L5   : TST    (R5)+            ;ВЫРАВНИВАНИЕ R5 НА ГРА-
         BIC    #1,R5            ;НИЦУ  СЛОВА
         RTS    R5               ;ВЫХОД
```

Для редакторов текста типичны следующие команды: BEGTXT – курсор в начало текста; CLEAR – стирание текста; DL – удаление строки; IL – вставка строки; команды чтения и записи (например,с перфоленты); команды управления маркером терминала и др.

Положим, что максимальный размер буфера строки 64 символа, а начальный его адрес LINE. Буфер текста ограничен размером LIMIT и состоит из последовательности строк, каждая из которых завершается признаком конца строки 000Q (пусто). Строку в буфере текста, к которой идет обращение, назовем текущей. Адрес текущей строки в буфере текста определим переменной ALIN , а длину – переменной LEN. Начало буфера текста зададим адресом BUF , конец – адресом ENDBUF. Признак конца буфера – код 000Q.

Все множество адресов подпрограмм команд редактора сгруппируем в таблице CTRL таким образом, чтобы по процедуре JSR R5, @CTRL(R0) осуществлялся переход к соответствующей подпрограмме, обслуживающей команду редактора. Содержимое R0 – это удвоенное значение управляющего кода (< 40) клавиатуры, являющегося смещением по отношению к базе CTRL. Ведущая программа редактора и таблицы CTRL при этом могут выглядеть следующим образом:

```
START : JSR     R5,CLST
ST1   : JSR     R5,ACCEPT
EDIT  : CMPB    R0,#40
        BMI     ST2
        CMPB    R0,#177
        BNE     ST0
        JSR     R5,ZABOY
        BR      ST1
ST0   : JSR     R5,CHANGE
        BR      ST1
ST2   : ADD     R0,R0
        JSR     R5,@CTRL(R0)
        BR      ST1

CTRL  : .WORD   EMP,IL,BEGTXT,CLEAR
        .WORD   EMP,ENDTXT,IL,READ
        .WORD   BEGSCR,HT,LFCR,DLE
        .WORD   CLEAR,CR,RUS,LAT
        .WORD   DL,READ,PPAGE,IS
        .WORD   DS,ENDSCR,NPAGE,WRITE
        .WORD   ENDLIN,RIGHT,LEFT,CTRL.U
        .WORD   UP,DOWN,WRITE,DTXTE
```

Например, для признака перевода строки (код ASCII равен 12), согласно программе START (RO-24), будет осуществлен переход к подпрограмме команды редактора LFCR через таблицу CTRL. Код ASCII символа команды удваивается, так как таблица CTRL организована с помощью директивы WORD, следовательно, необходим четный адрес.

Работа программы редактора начинается с его инициализации (программа CLST): запись кода 000_8 (признак конца строки) – в первую ячейку буфера текста BUF и признака конца буфера текста 000Q – в ячейку ENDBUF = = BUF + 1 (фактическая длина LEN текущей строки равна 0); заполнение буфера строки кодом пробела (040Q); очистка экрана; установка курсора на первую строку и первую позицию. Заполнение буфера строки пробелами облегчает ее обработку: следуя от конца к началу до первого кода, отличного от пробела, так как внутри строки пробелы могут быть и иметь определенную смысловую нагрузку.

Главным в работе редактора является обеспечение правильного взаимодействия между буфером строки и буфером текста, что поддерживается подпрограммами: LOAD (загрузка текущей строки в буфер строки из буфера текста); UNLOAD (запись строки из буфера строки в буфер текста); SQ.EX (коррекция буфера текста). Если длина текущей строки после обработки отличается от ее первоначального размера, то необходима коррекция буфера текста перед записью в него текущей строки. Загрузка символа в буфер строки осуществляется подпрограммой CHANGE:

```
CHANGE : CMP     MP,#63.
         BLE     CHAN2
         JSR     R5,TYPE
         .BYTE   7,200
         BR      CHAN0
CHAN2  : MOV     MP,R1
```

```
            MOVB    R0,LINE(R1)
CHAN1 :     TSTB    @#177564
            BPL     CHAN1
            MOVB    R0,@#177566
            INC     MP
CHAN0 :     RTS     R5
```

Чтение текущей строки из буфера текста в буфер строки осуществляется подпрограммой LOAD в такой последовательности:

```
LOAD  :  MOV  R2,-(SP)       ;СОХРАНЕНИЕ
         MOV  R3,-(SP)       ;РЕГИСТРОВ
         MOV  R4,-(SP)       ;ОБЩЕГО НАЗНАЧЕНИЯ
         MOV #64.,R4         ;ЗАДАНИЕ РАЗМЕРА СТРОКИ 64 СИМ.
         MOV  ALIN,R3        ;АДРЕС ТЕКУЩЕЙ СТРОКИ В R3
         MOV #LINE,R2        ;АДРЕС БУФЕРА СТРОКИ В R2
LOAD1:   TSTB @R3            ;ПРОВЕРКА НА ПРИЗНАК 0 КОНЦА
         BEQ  LOAD0          ;СТРОКИ
         MOV  (R3)+,(R2)+    ;ЗАГРУЗКА БУФЕРА ТЕКУЩЕЙ СТРОКИ
         SOB  R4,LOAD1       ;ИЗ БУФЕРА ТЕКСТА
LOAD0:   MOV #64.,R3         ;ОПРЕДЕЛЕНИЕ ДЛИНЫ
         SUB  R4,R3          ;ТЕКУЩЕЙ СТРОКИ
         MOV  R3,LEN         ;LEN - ДЛИНА СТРОКИ
         TST  R4
         BEQ  LOAD3          ;ОБМЕН ЗАКОНЧЕН
LOAD2:   MOVB #40,(R2)+      ;ЗАПОЛНЕНИЕ ПРОБЕЛАМИ ТЕКУЩЕЙ
         SOB  R4,LOAD2       ;СТРОКИ ДО 64 СИМВОЛОВ
LOAD3 :  MOV  (SP)+,R4       ;ВОССТАНОВЛЕНИЕ
         MOV  (SP)+,R3       ;РЕГИСТРОВ
         MOV  (SP)+,R2       ;ОБЩЕГО НАЗНАЧЕНИЯ
         RTS  R5
```

По выходу из подпрограммы LOAD строка окажется в буфере строки, который будет отображен на экране терминала (подпрограмма TYPE), а также будет определена ее фактическая длина LEN.

Дальнейшая обработка буфера экрана осуществляется согласно алгоритму, приведенному на рис. 5.5. Курсор отмечает текущую строку. Отредактированная строка с помощью подпрограммы UNLOAD записывается обратно в буфер текста:

```
UNLOAD:  MOV    R2,-(SP)
         MOV    R3,-(SP)
         MOV    R4,-(SP)
         MOV    #64.,R4
         MOV    #LINE,R3
         ADD    R4,R3
UNLO1:   CMPB   #40,-(R3)
         BNE    UNLO0
         SOB    R4,UNLO1
UNLO0:   MOV    R4,SEDATA
         SUB    LEN,SEDATA
         JSR    R5,$0.EX
         MOV    #LINE,R3
         MOV    ALIN,R2
         TST    R4
         BEQ    LOAD3
```

```
UNLO2: MOVB   (R3)+,(R2)+
       SOB    R4,UNLO2
       BR     LOAD3
```

При этом перед записью определяется разница DELTA между длиной SEDATA обработанной текущей строки, находящейся в буфере строки, и длиной LEN текущей строки, находящейся в буфере текста (DELTA=SEDATA=LEN). Механизм коррекции буфера текста иллюстрируется подпрограммой SQ.EX:

```
SQ.EX : MOV   R2,-(SP)
        MOV   R3,-(SP)
        MOV   R4,-(SP)
        MOV   ENDBUF,R4
        MOV   SEDATA,R3
        ADD   R3,ENDBUF
        CMP   ENDBUF,LIMIT
        BLOS  SE4
        JSR   R5,TYPE
        .BYTE 7,15,12,12
        .ASCII /?SEDVK - БУФЕР ТЕКСТА ЗАПОЛНЕН/
        .BYTE 15,12,200
        .EVEN
  SE4 : TST   R3
        BEQ   SE0
        BMI   SE1
        MOV   R4,R2
        ADD   R4,R3
        SUB   ALIN,R4
        BEQ   SE0
  SE2 : MOVB  -(R2),-(R3)
        SOB   R4,SE2
        BR    SE0
  SE1 : MOV   ALIN,R2
        NEG   R3
        ADD   R2,R3
        SUB   R3,R4
        BEQ   SE0
  SE3 : MOVB  (R3)+,(R2)+
        SOB   R4,SE3
  SE0 : CLRB  @ENDBUF
        MOV   (SP)+,R4
        MOV   (SP)+,R3
        MOV   (SP)+,R2
        RTS   R5
```

При этом возможны три ситуации: в буфер текста помещается новая строка; текущая строка после обработки стала длиннее текущей строки до обработки (SEDATA > LEN), текущая строка после обработки стала короче текущей строки до обработки (SEDATA < LEN). В первых двух случаях необходимо увеличение длины буфера текста, а в третьем — его сжатие. Сокращение длины буфера текста осуществляется путем переноса признака конца буфера 000_8 по адресу ENDBUF=DELTA , но при этом осуществляется сдвиг содержимого участка буфера, ограниченного адресами ALIN+DELTA и ENDBUF, по адресу ALIN.

Зак. 5404

Расширению длины буфера текста предшествует операция проверки на ее допустимость (⩽ LIMIT). В противном случае пользователю выдается на терминал сообщение "? SEDBK – БУФЕР ТЕКСТА ЗАПОЛНЕН". Если новая длина буфера текста лежит в заданных пределах, то осуществляется перезапись (ENDBUF–ALIN)+ DELTA байтов информации, начиная с адреса ENDBUF в область памяти, начальный адрес которой ENDBUF + DELTA.

Таким образом, мы рассмотрели приемы построения экранного редактора текста, динамику взаимодействия его буферов. Вопросы, связанные с конкретными командами редактора (например, вставка строки, удаление строки, сдвиг строк и т. д.), не представляют затруднения для их решения, так как ядром этих подпрограмм будут подпрограммы LOAD и UNLOAD . Читателю предлагается самостоятельно дописать редактор на определенное множество его команд.

5.3. ЗАГРУЗЧИКИ

Задача системы заключается в том, чтобы ввести программу в память и начать ее выполнение с нужной команды. Данную операцию выполняют начальные загрузчики (предзагрузчики), функция которых состоит в том, чтобы ввести и запустить более длинную, более сложную программу загрузки. Предзагрузчик сначала устанавливает связь с ЗУ, а затем запускает программу, которая в свою очередь устанавливает связь с терминалом и, возможно, будет выполнять самые примитивные функции монитора.

Рассмотрим организацию и работу наиболее характерного предзагрузчика для перфоленточной операционной системы. Предзагрузчик использует специальный формат перфоленты и может самомодифицироваться. Буфер устройства чтения с перфоленты PRB-177552, регистр состояния считывающего устройства PRS-177550. Первый бит регистра есть разряд "готовности к чтению", и всякий раз перед чтением очередного байта его необходимо программно устанавливать. В момент считывания он автоматически сбрасывается, и для чтения следующего байта его нужно установить снова. Исходная программа предзагрузчика имеет следующий вид:

```
077744  016701  START:  MOV     77776,R1
        000026
077750  012702  S1:     MOV     #352,R2
        000352
077754  005211          INC     (R1)
077756  105711  LOOP:   TSTB    (R1)
077760  100376          BPL     LOOP
077762  116162          MOVB    2(R1),77400(R2)
        000002
        077400
077770  005267          INC     .77752
        177756
077774  000765          BR      S1
077776  177550  .WORD   PRS
```

Допустим, что мониторная программа должна храниться в памяти с адр са 77600. При работе начальный загрузчик размещает прочитанные данные последовательные байты, расположенные за адресом 77600. Образец перф

ленты, загружающий данные с адреса 77 600 по адрес 77 742, выглядит следующим образом:

```
351
351
351     Ракорд
 .
 .
 .
351
177     Младший байт начального смещения минус 1
 .
 .      Данные, которые должны быть загружены
 .
301
035
026     Код, эквивалентный MOV77776, R1
000
302
025 ←   Код, эквивалентный MOV #352, R2
373
XXX     Действительный адрес смещения между 77 600 и 77 742, по которому
        загруженная программа должна начать выполнение
```

Наличие специального ракорда, который представляет собой повторение одного и того же байта 351 на протяжении нескольких сантиметров, снижает риск физического повреждения самой программы и четко обозначает начало программы – место, где кончается ракорд. Предзагрузчик начинает свою работу с того, что загружает адрес регистра состояния устройства PRS и регистр R1. В цикле с меткой LOOP регистр PRS проверяется уже известным способом, и в случае поступления байта последний пересылается из PRS в память. Отметим, что перед этим командой INC(R1) бит готовности был установлен в PRS . Затем программа передает управление назад на метку S1 для ввода следующего байта.

Так как первоначально в регистре R2 находится число 352. первый байт данных будет помещен в ячейку с адресом 77400 (R2) = 77400 + 352 = = 77752 затем произойдет увеличение содержимого этой ячейки на единицу (INC 77753) и выполнится переход BRS1. Увеличенные на единицу данные (353) становятся непосредственным значением, которое загрузится в регистр R2 при очередном выполнении команды, помеченной меткой S1.

Код ракорда, увеличенный на единицу, дает значение числа, находящегося в регистре R2 в момент начала загрузки, т. е. 353, поэтому код ракорда не оказывает влияния на программу предзагрузчика. Каждый раз, когда читается код ракорда, процессор выполняет один и тот же цикл, и поэтому программа не модифицируется. Первый код, отличный от кода ракорда, заменит данные в регистре R2 величиной, которая будет восприниматься как указатель начала загружаемой программы с адресом загрузки, равным 77400(R2) = = 77400 + (177+1) = 77600. Значит, следующие байты будут загружаться уже не по адресу непосредственного операнда, а в ячейки памяти 77600, 77601, 77602, ..., 77742. Таким образом, будет введена вся программа.

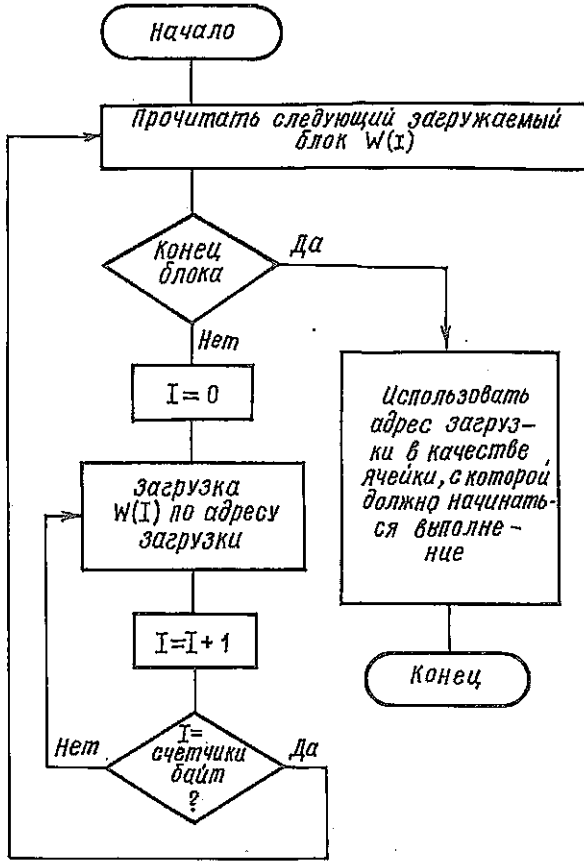


Рис. 5.6

В конце загрузки происходит наложение данных с перфоленты на две команды предзагрузчика. Первая из них (команда MOV 77776,R1) остается при этом без изменений, в то время как вторая модифицируется. Модификация осуществляется так, чтобы поместить очередной байт, вводимый с перфоленты (смещение для перехода), в младший байт команды перехода BR. Для этого по команде MOV в регистр R2 заносят число 374 (т. е. введенный байт, 373, увеличенный на 1). В результате выполнения команды MOVB 2(R1), 77 400(R2 адрес 77 400 + 374 = 77 774 совпадает с младшим байтом команды BR S1. Благодаря занесению в эту команду нового смещения XXX предзагрузчик может передать управление на начало загруженной команды или на другой адрес внутри нее.

Основное требование к начальному загрузчику связано с минимизацией занимаемой им памяти. Кроме предзагрузчика, операционная система содержит абсолютный загрузчик, т. е. системную программу для чтения записей, содержащих команды машинного языка и привязанных к абсолютным адресам ячеек памяти.

Обычно абсолютный загрузчик загружается начальным загрузчиком в самую верхнюю область памяти. Таким образом, предотвращается пересечение областей памяти загрузчика и других системных и прикладных программ. Это следует учитывать при их написании.

Любая программа, загружаемая в память микроЭВМ абсолютным загрузчиком, должна состоять из нескольких блоков данных, каждый из которых включает: индикатор начала блока; число слов (байт) (N), которые должны быть загружены; адрес загрузки (L); загружаемую информацию; контрольную сумму. В принципе можно обойтись без первого и последнего элементов, но, как правило, они являются необходимым условием со стороны загрузчиков микроЭВМ.

На рис. 5.6 показана схема простого абсолютного загрузчика. Здесь W(I)– массив вводимых данных. Заметим, что последний адрес загрузки в некоторых случаях применяется как адрес передачи управления по окончании процесса загрузки.

Альтернативой выбору адреса загрузки из вводимого блока может быть задание адреса расположения программы в памяти с помощью консольного переключателя. Это позволяет размещать PIC-программы в области памяти, отличные от указанных в загружаемом блоке, т. е. PIC-программы могут быть введены в произвольное место памяти путем формирования действительного адреса загрузки в виде суммы двух указанных адресов.

5.4. РЕДАКТИРОВАНИЕ СВЯЗЕЙ И ЗАГРУЗКА

На практике большая программа состоит из определенного числа подпрограмм, написанных отдельно друг от друга и, возможно, различными разработчиками. Результат трансляции каждой из подпрограмм называется объектным модулем. Программа, которая соединяет объектные модули в единую программу на машинном языке и которая может быть передана загрузчику, называется редактором связей, а программа на его выходе называется загрузочным модулем. В начале операции редактирования связей и загрузки используется следующая информация перемещаемого ассемблера: 1) объектный код программы; 2) данные о свойствах перемещения отдельных полей в объектном коде; 3) относительный адрес первой команды или элемента данных в загружаемом модуле; 4) глобальные точки входа и ссылки на внешние символы; 5) длина загружаемого модуля.

Данная информация необходима связывающему загрузчику для создания схемы загрузки. Она детализирует, какие программы должны быть загружены, какова их длина, где они размещаются в памяти и какие другие программы требуются для их выполнения. На рис. 5.7 показан возможный вариант для объектных модулей: таблица внешних символов (словарь внешних символов (СВС)); текст программы на машинном языке, словарь перемещений (СП) – список местоположения адресных констант, которые должны быть перемещены загрузчиком.

Положим, программы A и B оттранслированы раздельно. В программе A определены две ячейки памяти DATAWORD1 и DATAWORD2, на которые есть ссылки из обеих программ, а также программа B является подпрограм-

мой для программы А, которая должна ссылаться на точку входа SUBRB программы В. На рис. 5.8 приведен возможный формат представления этих программ для редактора связей. В каждой строке таблицы СВС имеется признак для идентификации. В первой строке, обозначенной через Р, указаны длина программы вместе с ее именем. Во второй и третьей строках СВС для программы А описаны DATAWORD1 и DATAWORD2 как внешние символы. Они являются адресами ячеек памяти 120 и 121 этой программы. Четвертая строка SUBRB предназначена для ссылки на адрес в программе В. Эта ссылка содержится в ячейке памяти 180 программы А. Адрес SUBRB должен быть задан в СВС программы В. Содержимое ячейки памяти 180 программы А для входа в программу В через метку SUBRB определяется во время выполнения процесса связывания. Тот факт, что содержимое ячейки должно быть перемещено во время ввода окончательного загрузочного модуля в ОП, также подчеркивается включением ячейки памяти 180 в СП (аналогично для модуля программы В). В данном модуле имеется также дополнительная локальная ссылка, которая представлена внутренней адресной константой LOCALBR, равной 7. Ссылка производится в слове по адресу 74, что отмечается в СП. Другие два элемента СП указывают местоположение адресных констант DATAWORD1 и DATAWORD2.

Рис. 5.7

Таким образом, в СВС и СП объектных модулей А и В содержится вся информация для редактора связей, для построения загрузочного модуля, состоящего из 375 слов программ А и В. Предположим, что редактору связей последовательно были переданы объектные модули программ А и В. Структура загрузочного модуля в данном случае будет следующая (рис. 5.9): в начале находится описание длины программы, затем коды программ и составной СП. Редактор связей устанавливает, что SUBRB занимает 251-ю ячейку памяти составного загрузочного модуля, поэтому адресная константа 251 помещается в слово 180 тела программы загрузочного модуля. Аналогично адресная константа 7 в 74-й строке программного модуля В заменяется на адресную константу 257 = = 250 + 7, когда помещается в 324-ю строку (250 + 74) окончательного загрузочного модуля. После связывания объектных модулей для программ А и В загрузочный модуль может передаваться программе загрузчика. Он загружает программу длиной 375 слов в ОП, начиная с заданного начального адреса S . В этом случае загрузчик прибавляет значение S–1 к константам 251, 257, 120 и 121 в ячейках памяти 180, 324, 330 и 340 загрузочного модуля.

Описанное редактирование адресов называется методом векторов передачи.

Итак, в результате работы редактора связей объектные модули настраиваются на абсолютные адреса; различные модули объединяются вместе и устанавливается взаимосвязь по глобальным идентификаторам между теми моду-

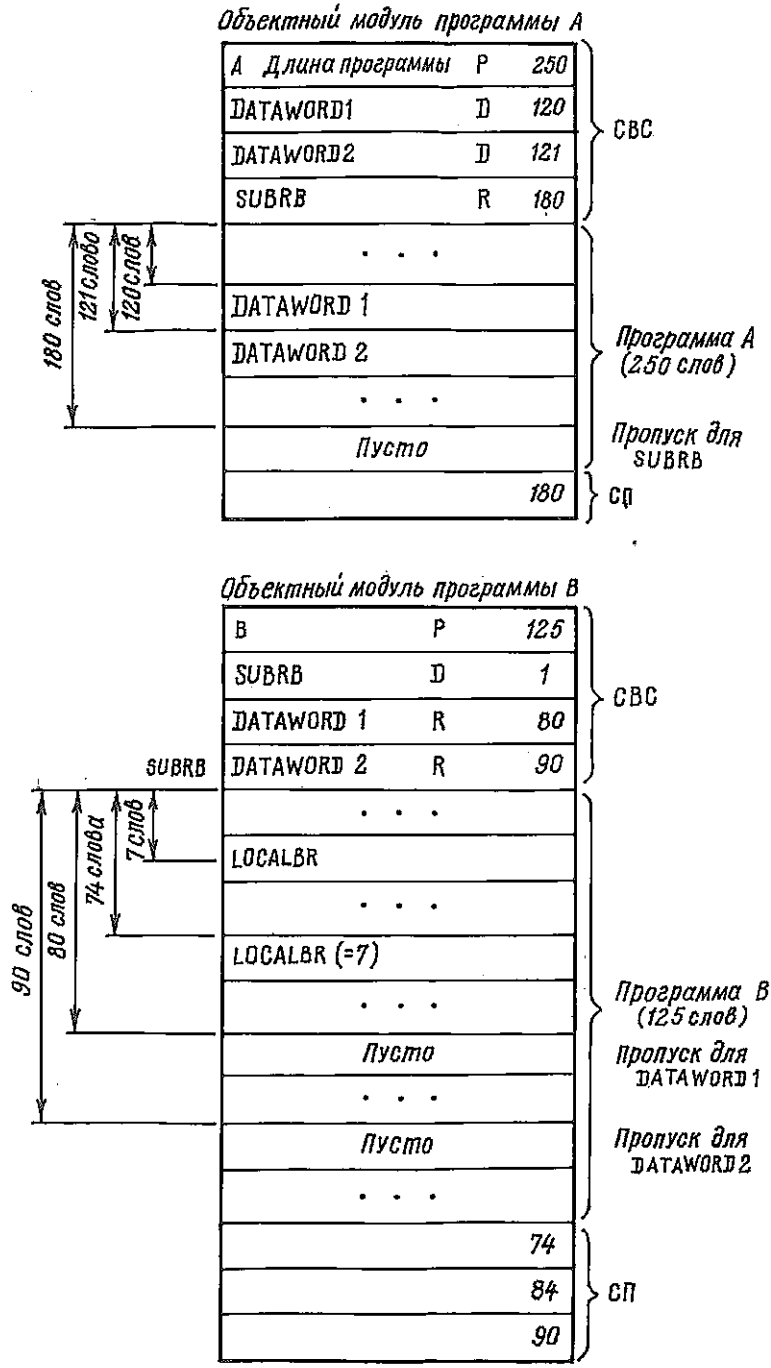

Объектный модуль программы A
A Длина программы P 250
DATAWORD1 D 120
DATAWORD2 D 121
SUBRB R 180
СВС
180 слов
121 слово
120 слов
DATAWORD 1
DATAWORD 2
Пусто
Программа A (250 слов)
Пропуск для SUBRB
180
СП
Объектный модуль программы B
B P 125
SUBRB D 1
DATAWORD 1 R 80
DATAWORD 2 R 90
СВС
SUBRB
90 слов
80 слов
74 слова
7 слов
LOCALBR
LOCALBR (=7)
Пусто
Пусто
Программа B (125 слов)
Пропуск для DATAWORD 1
Пропуск для DATAWORD 2
74
84
90
СП

Рис. 5.8

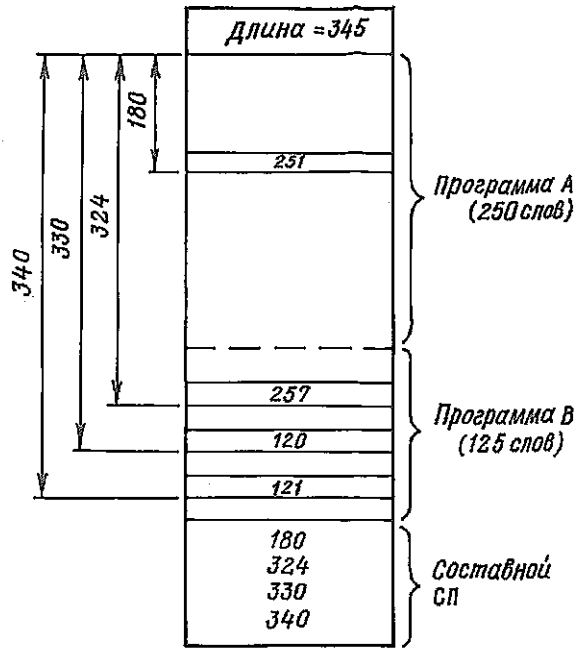


Рис. 5.9

лями, которые их используют; формируется карта распределения памяти, иллюстрирующая присвоение абсолютных адресов. Карта загрузки начинается с печати пускового адреса, нижнего и верхнего пределов (младшего и старшего адресов) относительной секции модуля. Затем печатаются данные о распределении памяти для каждого отдельного модуля: наименование модуля, наименования секций, адрес загрузки и длина секций в байтах, имена входа в алфавитном порядке с адресами.

Все это позволяет пользователю микроЭВМ ассемблировать программы и подпрограммы как отдельные модули, но выполнять их как одну программу. Следует отметить, что, как правило, функции редактора связей и загрузчика совмещаются в единой системной программе, которая называется связующим загрузчиком (компоновщиком). В этом случае генерация загрузочного модуля является промежуточным шагом при выполнении связующего загрузчика.

5.5. ОТЛАДЧИКИ

Проверка корректности программ, т. е. проверка соответствия их внешним спецификациям, осуществляется тестированием. Программы проверяются на функционирование с различными исходными данными, а затем сравниваются с эталонными значениями.

Отладка программ включает следующие этапы: планирование отладки; составление тестов и задания на отладку; исполнение программ; информирование о результатах исполнения программ по исходным данным; анализ результатов; обнаружение ошибок и локализация неисправностей.

Существуют два способа начального тестирования программ: 1) пошаговый режим, когда программа выполняется по одной команде за один раз, а пользователь сопоставляет содержимое памяти, регистров с ожидаемыми результатами; 2) трассировка программ. Результатом трассировки программы будут данные на экране дисплея или длинная распечатка с ходом выполнения программы (в случае использования в качестве терминала печатающего устройства). Отслеживание программы продолжается автоматически до тех пор, пока не будет остановлено программистом, который, анализируя эти данные, может обнаружить ошибки.

Таким образом, для отладки прикладных программ необходима служебная программа, обеспечивающая удобное и точное выполнение отладочных операций. Такая программа называется динамической отлаживающей программой. Она обычно реализуется в виде диалоговой системной программы, предоставляющей программисту удобные средства отладки и управления ходом выполнения программы. По существу данная программа представляет собой двоичный редактор и супервизор одновременно. Посредством команд программы отладчика пользователь с терминала должен суметь: 1) запустить программу; 2) прервать ее выполнение в заранее определенной точке; 3) проанализировать и изменить содержимое ячеек памяти и регистров; 4) сделать дополнения или коррекции в работающей программе, используя восьмеричный или символический код.

В состав команд отладчика входят команды: 1) открытия и изменения содержимого слов памяти (регистров); 2) пошагового выполнения, когда программа пользователя реализуется (команда за командой) с возвратом управления отладчику после каждого шага; 3) прогона – программа пользователя начинается с произвольного адреса и осуществляется без прерываний; 4) прогона с используемыми контрольными точками, в которых управление передается программе отладчика при выполнении команд, адреса которых указаны пользователем в списке контрольных точек; 5) поиска определенного набора бит, например адреса, константы, команды.

В семействе микроЭВМ "Электроника 60" используется отладочная программа, типичная для программ такого рода. Как и редактор, отладчик имеет командный режим, в котором выполняются запросы пользователя, обозначаемые на терминале звездочкой или точкой.

Основной набор команд программы отладки следующий: 1) *n*/ (открыть слово в ячейке *n*); 2) *n* (открыть байт в ячейке *n*); 3) ↵ (закрыть ячейку); 4) *n* ; G (запустить программу с адреса *n*); 5) *n* ; B (задать точку останова *n* (устанавливает точку прерывания в ячейке *n*)); 6) P (продолжить выполнение с точки останова); 7) ¤*n* (открыть регистр общего назначения *n* , *n* = = 0,..., 7); 8) ¤ S (открыть регистр состояния программы).

Приведем простейшую программу на языке ассемблера для микроЭВМ "Электроника 60", которая даст представление о динамической отлаживающей программе и ее работе:

```
          .1000
START:    MOV     #1,R0
          CLR     R1
          CMP     R0,R1
          HALT
          .END
```

В процессе работы отладчика, в частности, возможен следующий диалог (для наглядности приведены комментарии):

```
*1004/005001          ;ПРОВЕРЯЕТ КОМАНДУ
*R1/000000  123456    ;МЕНЯЕТ СОДЕРЖИМОЕ R1
*1004;P               ;УСТАНОВЛЕНЫ ТОЧКИ ПРЕРЫВАНИЯ
*1010;P               ;В ЯЧЕЙКАХ 1004 И 1010
*R0;001004            ;ВЫПОЛНЯЕТСЯ ПРЕРЫВАНИЕ
*R0/000001            ;ПРОВЕРЯЕТСЯ R0
*R1/123456            ;И R1
*;P                   ;ПРОДОЛЖЕНИЕ ВЫПОЛ. ПРОГРАММЫ
R1;001010             ;ВЫПОЛНЯЕТСЯ ВТОРОЕ ПРЕРЫВАНИЕ
*R1/000000            ;ПРОВЕРКА СОДЕРЖИМОГО
*R0/000001            ;РЕГИСТРОВ R1 И R0
```

При построении программы отладки широко применяются команды специальных прерываний (см. § 1.7). Команда BPT организует специальное прерывание через ячейку 14, разрешая отладчику в оперативном режиме производить вставку контрольных точек останова программы путем замены командами BPT команд пользователя. Если команда устанавливает в PSW процессора разряд T в состояние 1, то специальное прерывание будет организовано через ячейку памяти 14 после выполнения следующей команды. Тогда отладчик сможет организовать пошаговое выполнение программы пользователя. Например, предположим, что ЦП обеспечивает функционирование отладчика, а текущее значение счетчика команд и PSW для программы пользователя занесены в вершину стека. Тогда можно выполнить одну команду программы пользователя следующим образом: 1) устанавливается разряд T в копии слова состояния процессора программы пользователя в стеке в состояние 1; 2) выполняется команда RTT , при этом происходит возврат управления программе пользователя и извлечение из стека слова PSW программы пользователя; разряд T устанавливается в 1; 3) выполняется одна команда программы пользователя, а затем устанавливается режим трассировки, что возвращает управление отладчику.

Следует обратить внимание, что в данном случае используется команда RTT вместо команды RTI . Различие между RTT и RTI заключается в реакции на состояние бита трассировки, или T-бита в PSW. Если он установлен в 1, то после выполнения каждой команды (включая и RTI) будет осуществляться переход к ловушке по вектору 14—16, т. е. команда RTT позволит выполнить одну команду программы пользователя.

5.6. КРОССАССЕМБЛЕРЫ

Кроссассемблеры осуществляют те же функции, что и ассемблер: трансляцию мнемонических команд в машинный код; выполнение псевдоопераций; обнаружение синтаксических ошибок; обработку макрокоманд. Вместе с тем кроссассемблер позволяет обрабатывать программы в операционной системе ЭВМ, имеющей систему команд, отличную от команд микроЭВМ, на которой будет выполняться ассемблированная программа.

Кроссассемблеры представляют собой многоуровневый программный комплекс, предназначенный для трансляции (ассемблирования) текстов про-

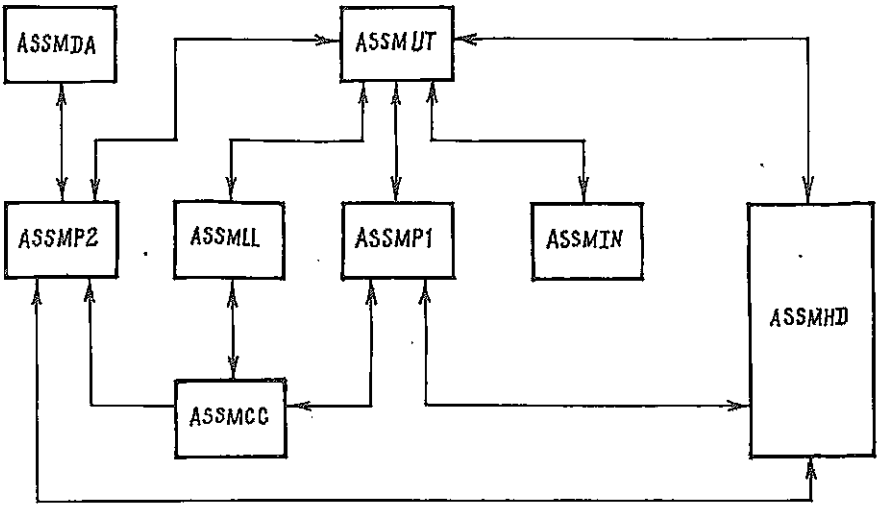


Рис. 5.10

грамм, написанных на языке ассемблера проектируемой микроЭВМ, формирования файлов листинга программ и выходных объектных кодов. Основными языками программирования, используемыми при разработке кроссассемблера, являются языки высокого уровня (например, ПАСКАЛЬ). Часто с целью достижения компромисса по быстродействию и объему памяти отдельные модули записываются на ассемблере.

Структурно кроссассемблер (рис. 5.10) состоит из головного программного модуля ASSMLL, осуществляющего инициализацию устройств, рабочих и выходных файлов, организацию необходимого взаимодействия между составляющими системы, а также интерфейса с пользователем; основных модулей – процедур ASSMP1, ASSMP2 и ASSMIN , выполняющих кроссрансляцию; служебных модулей ASSMCC (подпрограммы для преобразования чисел из одной системы счисления в другую и анализа выражений и операндов), ASSMUT (подпрограммы работы с таблицами, обработки ошибок и т. д.), ASSMDA (подпрограммы обращения к системной дате и преобразования ее в стандартную символьную форму). Связь и взаимодействие между отдельными программными модулями осуществляются с помощью общей области памяти, формируемой посредством модуля ASSMHD.

Модуль-процедура ASSMIN осуществляет инициацию имен и кодов в таблице команд микропроцессора. Такой модуль выделяется особо для упрощения возможной модификации кодов и (или) команд МП. Первый проход обработки исходного текста программы осуществляется модулем-процедурой ASSMP1, который выполняет лексический и синтаксический анализ текста, классификацию и формирование таблицы символов программы, определение внешних и глобальных ссылок и т. п. В результате получаем промежуточный временной файл отображения текста программы. На выходе второго прохода обработки текста программы (модуль ASSMP2) формируется объектный файл и создается файл листинга.

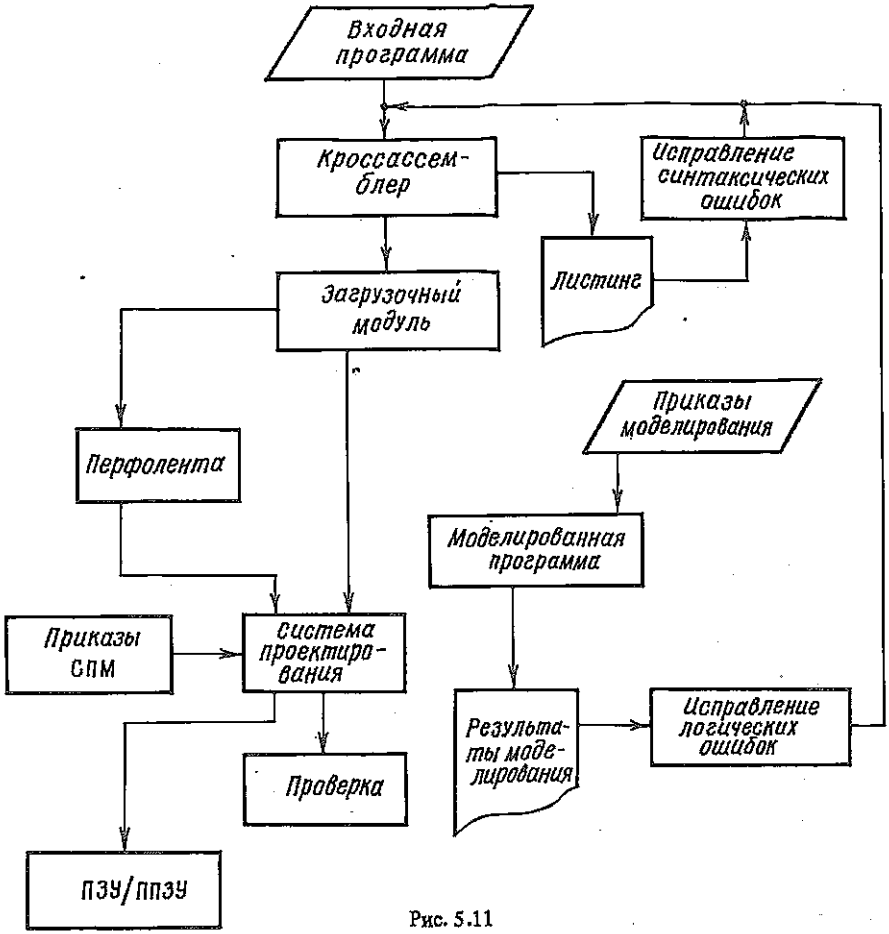


Рис. 5.11

В процессе работы кроссассемблер взаимодействует с файлами текста программы, объектных кодов и листинга. Файл с текстом программы является входным файлом кроссассемблера, который может быть создан с помощью редактора текста операционной системы. Объектный файл и файл листинга – выходные файлы кроссассемблера.

С помощью кроссассемблерных программ можно моделировать и отлаживать различные МП-системы, предсказывать производительность системы на самом раннем этапе проектирования архитектуры.

При проектировании программ часто кроссассемблеры используются совместно с моделирующими программами (рис. 5.11). Загрузочный модуль – результат работы кроссассемблера – вводится в моделируемую программу, которая имитирует выполнение каждой команды. Отлаженный загрузочный модуль записывается в ПЗУ или ОЗУ проектируемой ЭВМ.

Кроссассемблер и схемные эмуляторы включаются в инструментальные отладочные комплексы МП-систем.

Рекомендуемая литература: 1, 13, 15.

6. ОСНОВЫ ПОСТРОЕНИЯ ОПЕРАЦИОННЫХ СИСТЕМ

6.1. ОПРЕДЕЛЕНИЯ И ПОНЯТИЯ ОПЕРАЦИОННЫХ СИСТЕМ

Набор программ, обеспечивающих функционирование центрального процессора и периферийных устройств как единого комплекса, служащего для разработки и выполнения программ, называется *операционной системой* (ОС).

Большинство ОС разрабатывается и реализуется в расчете на конкретные ЭВМ или семейства ЭВМ. Исключением являются мобильные ОС – UNIX и ее отечественный аналог ИНМОС.

По функциональному назначению ОС делятся на следующие типы: 1) одно- и многопользовательские для разработки и отладки программного обеспечения (инструментальные); 2) прикладные и встроенные реального масштаба времени (РВ); 3) информационно-поисковые, использующиеся для организации банков данных, каталогов и т.д.; 4) для сетевой обработки информации, обеспечивающие маршрутизацию, коммуникацию информации в сети в соответствии с конкретными кодами и протоколами.

Обычно ОС, работающие на микроЭВМ, выделяют в самостоятельный класс. Ограниченность ресурсов, которые доступны системам, работающим на микроЭВМ, приводит к ограничениям самих ОС по сравнению с системами для больших машин. Принципы, лежащие в основе и тех и других, одни и те же, главное различие состоит в том, что в системах разного масштаба главное внимание уделяется различным системным компонентам.

Для систем автоматического управления объектами все большее применение находят встроенные ОС РВ. В настоящее время появляются прикладные ОС РВ, содержащие функциональный набор системных вызовов. Поскольку подобные ОС невозможно применить для решения задач, круг которых не определен на стадии проектирования, такие системы целесообразно называть специализированными (в отличие от систем общего назначения). Отметим, что появление специализированных операционных систем (СОС) явилось насущной необходимостью, поскольку ОС общего назначения ориентированы, как правило, на решение задач вычислительного и инструментального характера и многие из них вообще не способны поддерживать режим РВ.

Ориентация ОС зависит от распределения приоритетов между различными системными компонентами. Среди системного программного обеспечения обычно выделяют управляющие и обрабатывающие программы. Управляющие программы контролируют распределение системных ресурсов, обеспечивают простой доступ к устройствам внешнего хранения информации и их эффективное использование, а также управляют данными. Управляющие программы, как правило, включают супервизор (планировщик и монитор), который распределяет ресурсы процессора между задачами, активизирует, приостанавливает и завершает задачи, управляет памятью. Эта программа обычно связана с

механизмами обработки прерываний, вызываемых как внутренними, так и внешними причинами. Программа управления вводом-выводом обеспечивает хранение и передачу файлов с одного внешнего устройства на другое, защиту информации, доступ к ней и сохранение ее секретности. Часто эти программы управления объединены в одну.

Обрабатывающие программы состоят из прикладных и обслуживающих. К ним относятся редакторы текстов, трансляторы с языков программирования, программы диагностирования и т. д. Состав этих программ весьма различен и определяется ориентацией ОС.

Можно выделить следующие компоненты, характерные для любой ОС: монитор, файловая система, система ввода-вывода.

Конструкции для управления доступом конкурирующих процессов к разделяемым ресурсам называются *мониторами.* Монитор непосредственно не инициирует каких-либо действий в системе. Это пассивный модуль, который может быть активизирован только посредством вызова процедур. Монитор реализуется таким образом, что одновременное выполнение нескольких процедур невозможно. Следовательно, процесс сохраняет монопольный контроль над ресурсами монитора при выполнении одной из его процедур.

Вся информация о наборе разделяемых ресурсов, о том, как с ними работать, хранится в области данных, называемой областью монитора, куда включен и ряд процедур, определяющих операции над разделяемыми ресурсами. Эти процедуры доступны для всех процессов в системе. Если процессу необходимо обратиться к разделяемому ресурсу, он должен выполнить одну из процедур соответствующего монитора.

Монитор может синхронизировать конкурирующие процессы и передавать данные между ними через разделяемые ресурсы. Для этого в мониторе организуются очереди запросов к ресурсам. Дисциплина планирования, т. е. порядок, в котором процессы помещаются и обслуживаются в очереди, может быть произвольной и определяется пользователем.

Связь между процессом и монитором обычно осуществляется с помощью системных вызовов. Для данной цели используются определенные команды программных прерываний (EMT, SVC, IOT, TRAP). Это обеспечивает независимость процесса от места расположения монитора.

Файл представляет собой набор связанных записей или элементов данных с которым можно работать как с одним информационным модулем. Для хранения файлов используются магнитные ленты и диски, которые имеют свою файловую структуру, определяющую методы записи, связь между элементами и каталогизацию данных, т. е. организацию файлов на устройстве и методы доступа к ним. Организационное структурирование играет важную роль в связи тем, что файл может быть эффективно использован только в том случае, если выбранная структура соответствует требованиям пользователя.

Существуют три вида распределения памяти для файла: непрерывное, списковое и индексное. Методы доступа (последовательный или прямой) определяются типом устройства. Для накопителей на магнитной ленте характерен последовательный метод доступа и к самим файлам, и к отдельным записям них. При этом организуется непрерывное распределение памяти для файлов На дисковых устройствах обычно реализуется прямой или произвольный метод доступа к файлам, опирающийся на списковый или адресный механизм

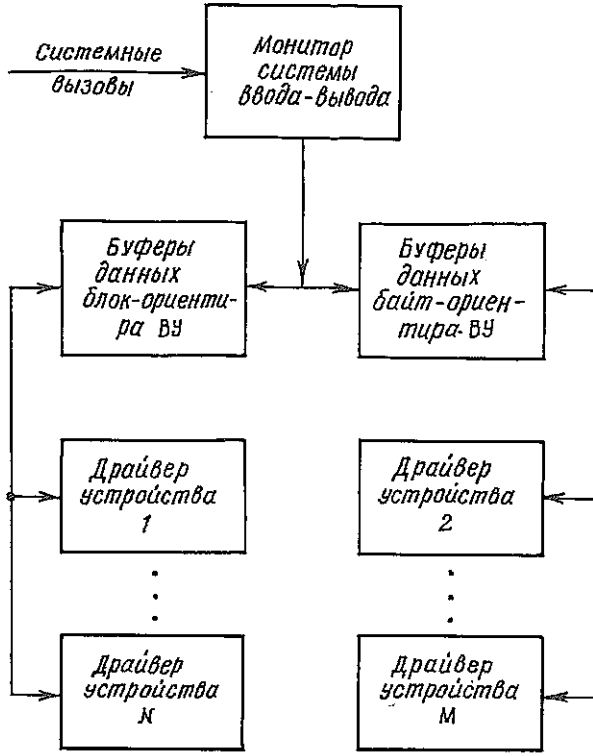


Рис. 6.1

Для организации адресации к файлам используют оглавление (каталог) — таблицу, являющуюся частью файловой системы и содержащую информацию о каждом файле.

Основные требования к файловой системе следующие: минимальное время доступа; высокий процент полезного объема хранения (т. е. минимум системной информации); удобная и понятная каталоговая структура, обеспечение идентичного механизма доступа к файлам различных устройств; защита от несанкционированного доступа.

Система ввода-вывода — это средства, доступные программисту (прикладной задаче) в виде системных вызовов, подпрограмм и функций, позволяющие унифицированно взаимодействовать с различными устройствами ввода-вывода.

К системе ввода-вывода любой ОС можно предъявить ряд требований, основные из которых следующие: обеспечение эффективной передачи данных между памятью процесса и внешней средой; поддержание синхронизации работы процесса и внешних устройств; предоставление удобного и "понятного" интерфейса для обращения к системе ввода-вывода, который максимально унифицирован для доступа к различным по своим характеристикам внешним устройствам, файлам, оперативной памяти.

Выполнение этих требований обычно осуществляется с помощью иерархической структуры программных средств (рис. 6.1), большинство из которых невидимы для прикладной задачи.

При работе с внешними устройствами программист оперирует с объектами–файлами независимо от физической структуры устройства.

Согласование интерфейса для доступа к различным по своим физическим характеристикам внешним устройствам выполняет программа, именуемая драйвером этого устройства. Обычно система ввода-вывода осуществляет следующий набор операций над файлами: открытие (допустимо только к существующему файлу; производится установка внутреннего указателя системы на начало файла); создание (создается файл нулевой длины); чтение из файла (читается N очередных структурных единиц данных (байтов, блоков)); запись в файл (выполняется аналогично чтению; если указатель дошел до конца файла, файл расширяется). Специфика отдельных устройств может накладывать ограничения на эти функции. Кроме того, система ввода-вывода имеет ряд библиотечных функций по обслуживанию устройств, каталогов и файлов.

При проектировании любой ОС основное требование к ней – соответствие условиям применения. Исходные данные к ОС можно сформулировать следующим образом: используемые аппаратные средства; набор реализуемых прикладных программ; режим реализации прикладных программ (выполнение или разработка); необходимая степень динамичности распределения ресурсов; критерии распределения ресурсов; удобства языковых интерфейсов и т. д. При разработке ОС главным является широко известная концепция аппаратно-программного единства средств вычислительной техники. Суть ее заключается в том, что любой процесс можно рассматривать как результат совместного функционирования аппаратных и программных ресурсов. Иными словами, программа является продолжением аппаратуры.

6.2. ОРГАНИЗАЦИЯ ДИСКОВОЙ ОПЕРАЦИОННОЙ СИСТЕМЫ

Системную среду микроЭВМ составляет дисковая операционная система (ДОС), включающая несколько секций: NUC, DIO, BIO, CINT. Главная секция ДОС – ядро NUC , распределяющее ресурсы системы и взаимодействующее с интерфейсами ввода-вывода. Секция DIO управляет доступом к дисковым накопителям со стороны пользователя. Секция байт-ориентированного ввода-вывода BIO содержит подпрограммы низкого уровня для управления консолью, ленточным накопителем, печатающим устройством и т. д. Секцию интерпретатора приказов CINT можно считать прикладной программой, так как при управлении системой она использует ресурсы через ядро. Структурная иерархия ДОС показана на рис. 6.2.

Применяются два варианта размещения ДОС в памяти. Вариант размещения ДОС в начале памяти удобнее, так как вся операционная система занимает единую область. Область прикладной программы АРА расширяется в направлении старших адресов памяти. При этом АРА можно увеличивать за счет области, занимаемой секцией CINT (транзитная секция, т. е. находится в памяти только при необходимости), которая не нужна при выполнении прикладной программы.

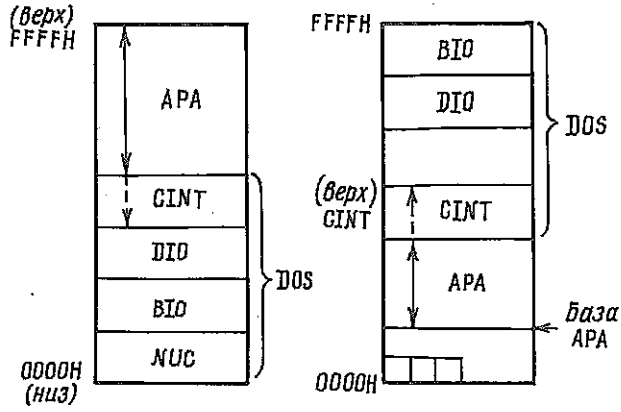


Рис. 6.2

Основной недостаток данного варианта состоит в том, что секции BIO и DIO трудно расширить при введении дополнительных функций, потому что "захватывается" APA. Если программы скомпилированы по абсолютному адресу, то их придется компилировать повторно. Однако данный недостаток легко устраняется, если все прикладные программы ассемблируются или компилируются в перемещаемый объектный код.

Размещение ДОС в конце памяти (второй вариант)позволяет все программы загружать по базовому адресу APA BASE и при необходимости они могут расширяться до CINT.База OG – база секции NUC. Если секции BIO и DIO увеличиваются в размере, то резервируется дополнительно обычно 1000 байт в верхней части памяти для системного стека и других интерфейсных подпрограмм. Недостатком второго варианта является то, что при увеличении емкости памяти необходимо перемещать всю ОС, т. е. все внутренние адреса суммируются с константой смещения. Однако четких рекомендаций по размещению ОС в памяти пока нет.

Независимо от задачи, которую предстоит выполнить прикладной программе, она должна обратиться к ядру и запросить обслуживание, обеспечивающее управляемый доступ ко многим устройствам системы. Для этих целей предусматривается подпрограмма – запрос системного ресурса SRR, – обычно размещаемая в начале памяти. При вызове SRR необходимо сообщить о запрашиваемом обслуживании, передать данные или указатель обрабатываемых данных. Тип функции определяется числом, загружаемым в регистр (можно в аккумулятор) процессора. Приведем типичные функции SRR:

1) выполнить горячий старт;
2) считать символ с консоли;
3) выдать символ на консоль;
4) считать символ с устройства пользователя 1 (лента);
5) записать символ на устройство пользователя 1 (лента);
6) считать символ с устройства пользователя 2;
7) записать символ на устройство пользователя 2;
8) выбрать диск;

9) создать файл;
10) открыть файл;
11) закрыть файл;
12) удалить файл;
13) считать блок из файла (последовательный);
14) записать блок в файл (последовательный);
15) считать произвольный блок (находит блок);
16) записать произвольный блок (находит блок);
17) считать байт из файла (последовательный);
18) записать байт в файл (последовательный);
19) переименовать файл;
20) защитить диск от записи.

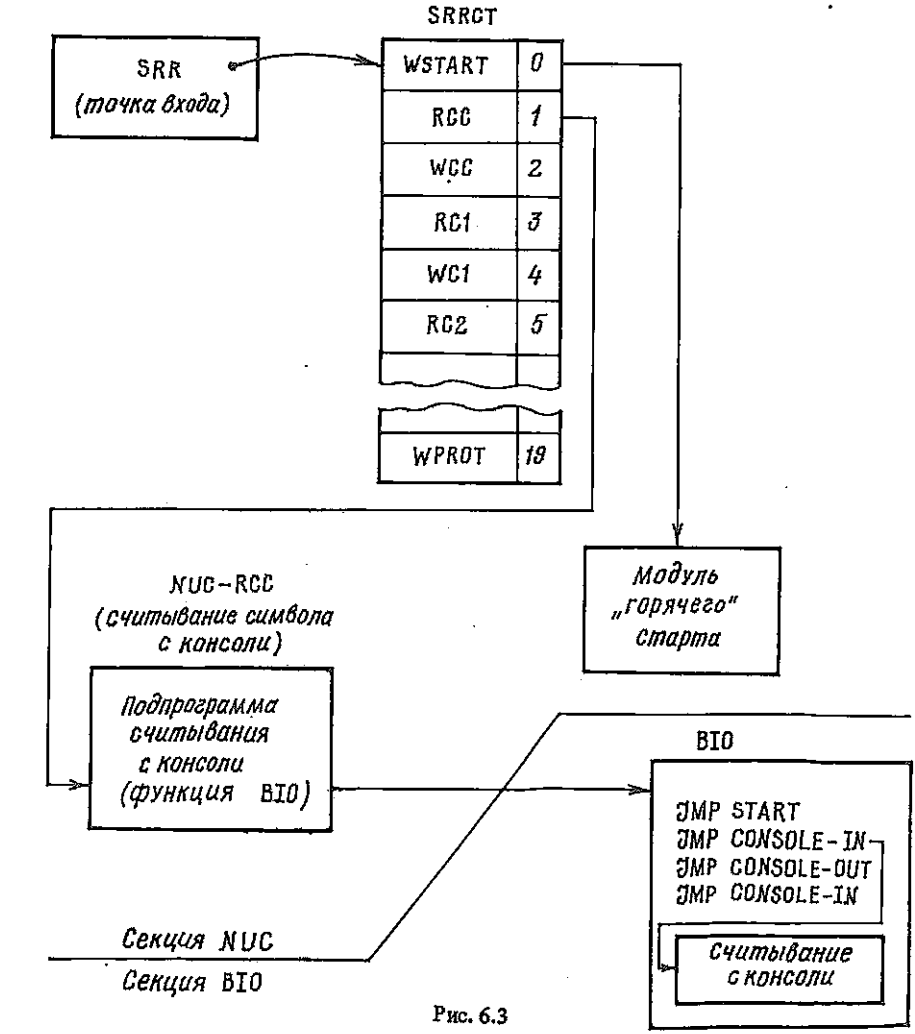


Рис. 6.3

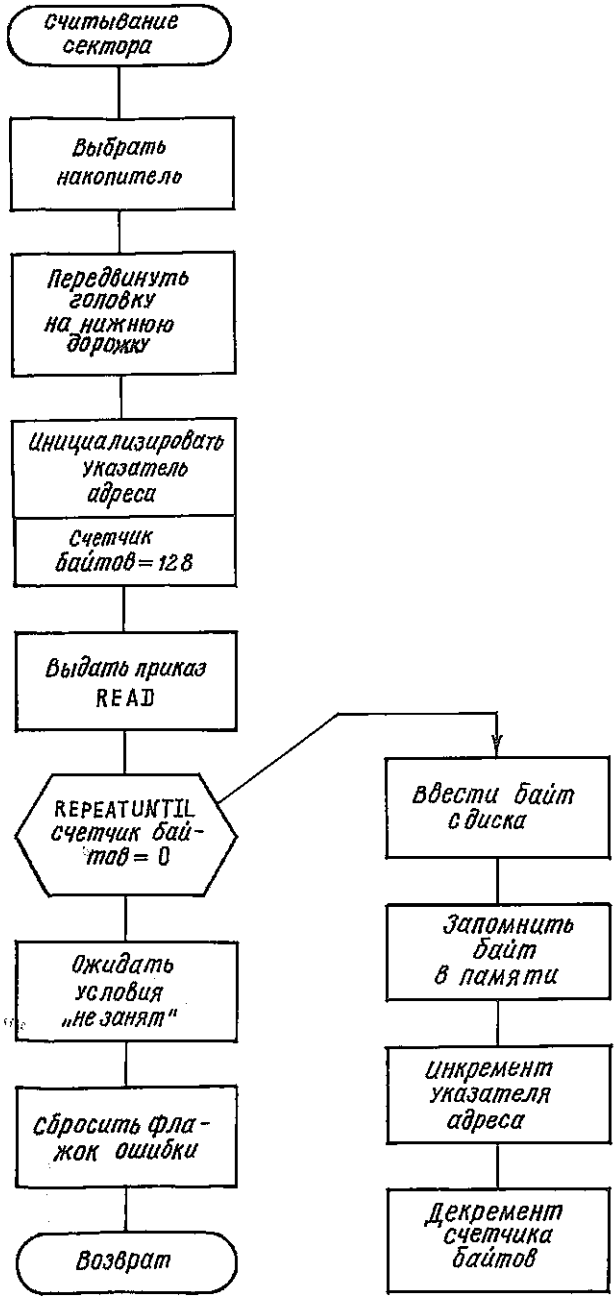

Считывание сектора
Выбрать накопитель
Передвинуть головку на нижнюю дорожку
Инициализировать указатель адреса
Счетчик байтов = 128
Выдать приказ READ
REPEAT UNTIL счетчик байтов = 0
Ожидать условия „не занят"
Сбросить флажок ошибки
Возврат
Ввести байт с диска
Запомнить байт в памяти
Инкремент указателя адреса
Декремент счетчика байтов

Рис. 6.4

Векторы переходов расположены в начале памяти, что связано с тем, что у многих микропроцессоров некоторые функции ассоциируются с началом памяти. Например, в микропроцессорном комплекте К580 адреса восьми ячеек используются как векторы рестарта или прерывания, а ячейки 170, 174, 270 и 274 в МП К588 – как немаскируемые постоянные вектора прерываний. Вход в ядро реализуется как вызов подпрограммы SRR. После запоминания всех регистров процессора SRR осуществляет индексирование в таблице приказов SRRCT , чтобы найти адрес подпрограммы, которая будет обслуживать запрос прикладной программы AP (рис. 6.3). По завершении выполнения функции подпрограмма из таблицы SRRCT должна перейти к общему модулю возврата, который восстанавливает регистры ЦП и передает управление прикладной программе.

Ядро, приняв запрос на обслуживание, сканирует таблицу функций SRR по номеру функции и определяет подпрограмму управления операцией ввода-вывода, которая находится в секции BIO . Размещается в начале BIO набор векторов переходов, соответствующих всем подпрограммам BIO. Внутренняя структура BIO включает следующие функции: 1) состояние консоли; 2) ввод с консоли; 3) вывод на консоль; 4) ввод с устройства 1 пользователя; 5) вывод на устройство 1 пользователя; 6) ввод с устройства 2 пользователя; 7) вывод на устройство 2 пользователя; 8) ввод состояния устройства 1 пользователя; 9) ввод состояния устройства 2 пользователя.

Две наиболее важные функции управляющей программы секции DIO – считывание сектора и его запись. Структурная схема типичной подпрограммы считывания сектора показана на рис. 6.4. Операция записи сектора почти аналогична операции считывания. Разница состоит лишь в том, что данные берутся из памяти по целевому адресу и передаются в контроллер через выходной порт. Следует отметить, что директивы выбора диска, инициирование поиска указанной дорожки и позиционирование на ней головки записи-считывания представляют собой самостоятельные подпрограммы. Первая директива прежде всего проверяет, не выбран ли уже запрошенный накопитель, вторая – контролирует текущее положение головки, а также проверяет действительность адресов запрошенных дорожки и сектора (т. е. находятся ли они в диапазоне 0–34 для мини-диска (накопитель СМ-6225) и 0–76 для накопителей ГМД-7012).

Приведем функции секции DIO: 1) выбор диска; 2) установка дорожки; 3) установка сектора; 4) установка адреса памяти; 5) считывание сектора; 6) запись сектора; 7) разметка дорожки.

6.3. УПРАВЛЕНИЕ ДАННЫМИ НА ГИБКОМ ДИСКЕ

Метод хранения данных на гибком диске существенно влияет на эффективность системы. Диск можно рассматривать с различных точек зрения: как более сложный кассетный интерфейс, как устройство с последовательным распределением файлов и их оглавлением, как устройство с произвольным доступом и с оглавлением или как устройство с произвольным доступом и несколькими оглавлениями (иерархией оглавлений).

Широко применяются следующие методы организации данных на диске: последовательное распределение; отображение секторов; произвольное рас-

пределение. При *последовательном распределении* программно адресуются каждая дорожка и каждый сектор. Это метод кассетного интерфейса, не требующий обязательного наличия оглавления файлов. При более совершенном способе последовательного распределения на первой дорожке диска размещается оглавление файлов. Любой элемент в оглавлении содержит имя NAME и тип файла TYP , начальные дорожку TRK и сектор SCTR, а также длину файла в секторах LENGTH и диспозицию DISP: 0 – обычный, 1 – защищенный, 2 – системный, 99 – удаленный. Каждый элемент оглавления представлен в форме управляющего блока файлов (FCB). В нашем случае FCB составляет 16 байт. В секторе емкостью 128 байт размещается восемь FCB. На рис. 6.5

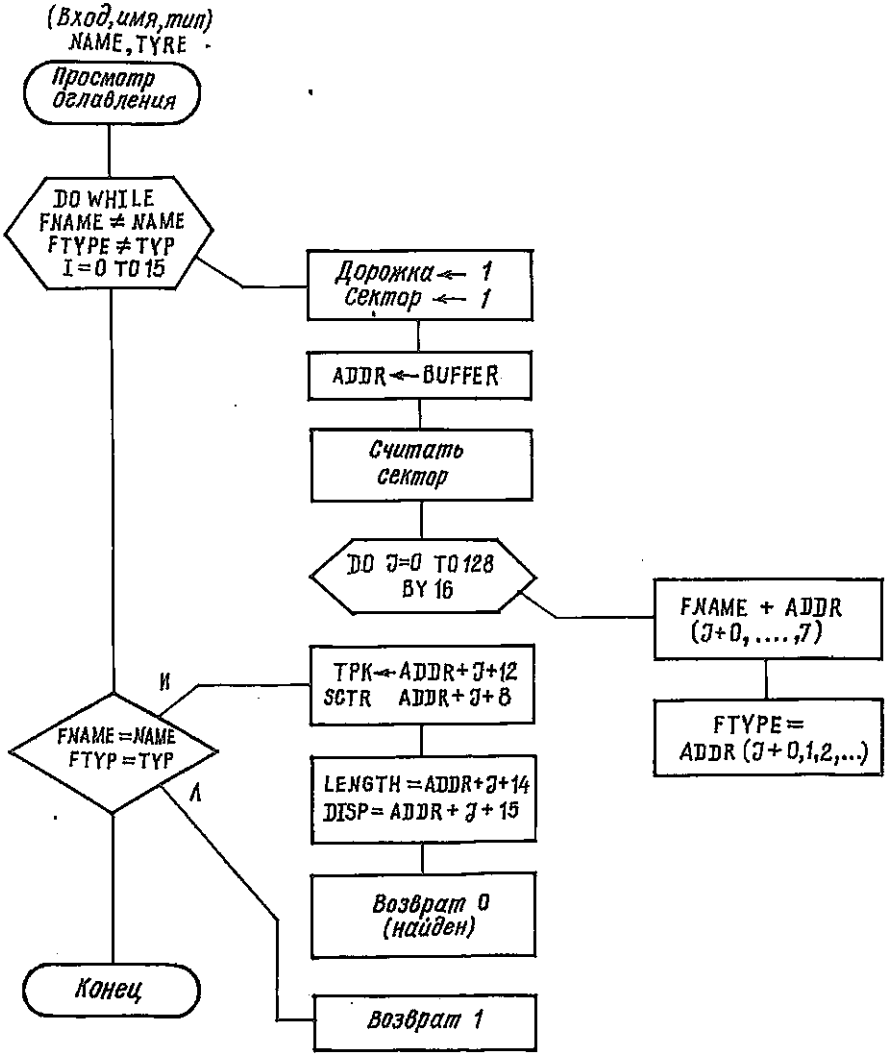


Рис. 6.5

приведена схема программы просмотра оглавления. Здесь в цикле DO WHILE проверяется имя и тип файла, которые находятся индексированием в каждом FCB. Начальный адрес каждого FCB вычисляется с помощью зацикливания от 0 до 128 шагов, равных 16 (длина FCB). Базовый адрес буфера сектора находится в ADDR, а адрес ADDR + J + K, где K – байт индекса, дает соответствующие поля в просматриваемом FCB.

Основной недостаток медленных НГМД состоит в том, что после локализации и считывания сектора диск "проскакивает" начало следующего сектора, прежде чем программа начнет считывать. Таким образом, каждое считывание-запись сектора занимает целый оборот диска. Для устранения данного недостатка применяется отображение секторов, т. е. любой логический сектор отображается через три сектора по часовой стрелке от предыдущего логического сектора. Данный прием дает программе время на фиксацию сектора (не происходит ожидания целого оборота диска). Однако данный способ неэффективен при произвольном распределении секторов.

Недостатком последовательного распределения секторов является фрагментация при записи новых файлов на диск. Реализация же произвольного распределения секторов на диске дает возможность любую единицу пространства предоставить файлу, т. е. файл может быть "разбросан" по всему диску. В этом случае FCB должен содержать список распределенных секторов, чтобы файл можно было считывать последовательно. При данном способе записи файла на диск необходимо решать такие проблемы, как учет доступных секторов, расширение файла, уплотнение пространства диска.

Файл не распределяется по секторам до тех пор, пока в него не производится запись. Когда идет запись, ядро находит в оглавлении нераспределенный сектор и добавляет его в список секторов, выделенных файлу. Для решения этой проблемы широко используются два способа. Первый состоит в организации списка "дыр" (промежутков), для сокращения потери на фрагментацию. При распределении новых секторов производится модификация списка, отображающая эти изменения. Недостатком способа является необходимость хранить на диске список "дыр" и загружать его в память, когда диск вставляется в накопитель.

Второй способ основан на том, что для каждого активного и вставленного диска в памяти хранится двоичная карта секторов, причем нуль означает, что сектор не распределен. Карта секторов строится из находящегося в оглавлении управляющего блока FCB.

Если управляющий блок распределенного файла FCB имеет длину 32 байта (имя файла – 8 байт; тип – 3 байта; адресация (дорожка, сектор, длина в секторах) – 3 байта; диспозиция – 1 байт и 1 байт не используется; секторная карта – 16 байт), то он допускает файлы максимум из 16 секторов (2048 байт при емкости сектора 128 байт), что не всегда бывает достаточно. Для того чтобы длина FCB не увеличивалась при расширении файлов, после первого FCB добавляют фиктивные блоки (экстенты), каждый из которых, например, соответствует 16 секторам файла. Число экстентов помещается в первый FCB группы в поле EX. Ядро NUC должно быть информировано о достижении конца первого FCB, чтобы иметь возможность отыскать следующий FCB. Для экономии пространства оглавления и упрощения адресации распределение осуществляется блоками секторов, а не отдельными секторами. Если секторы рас-

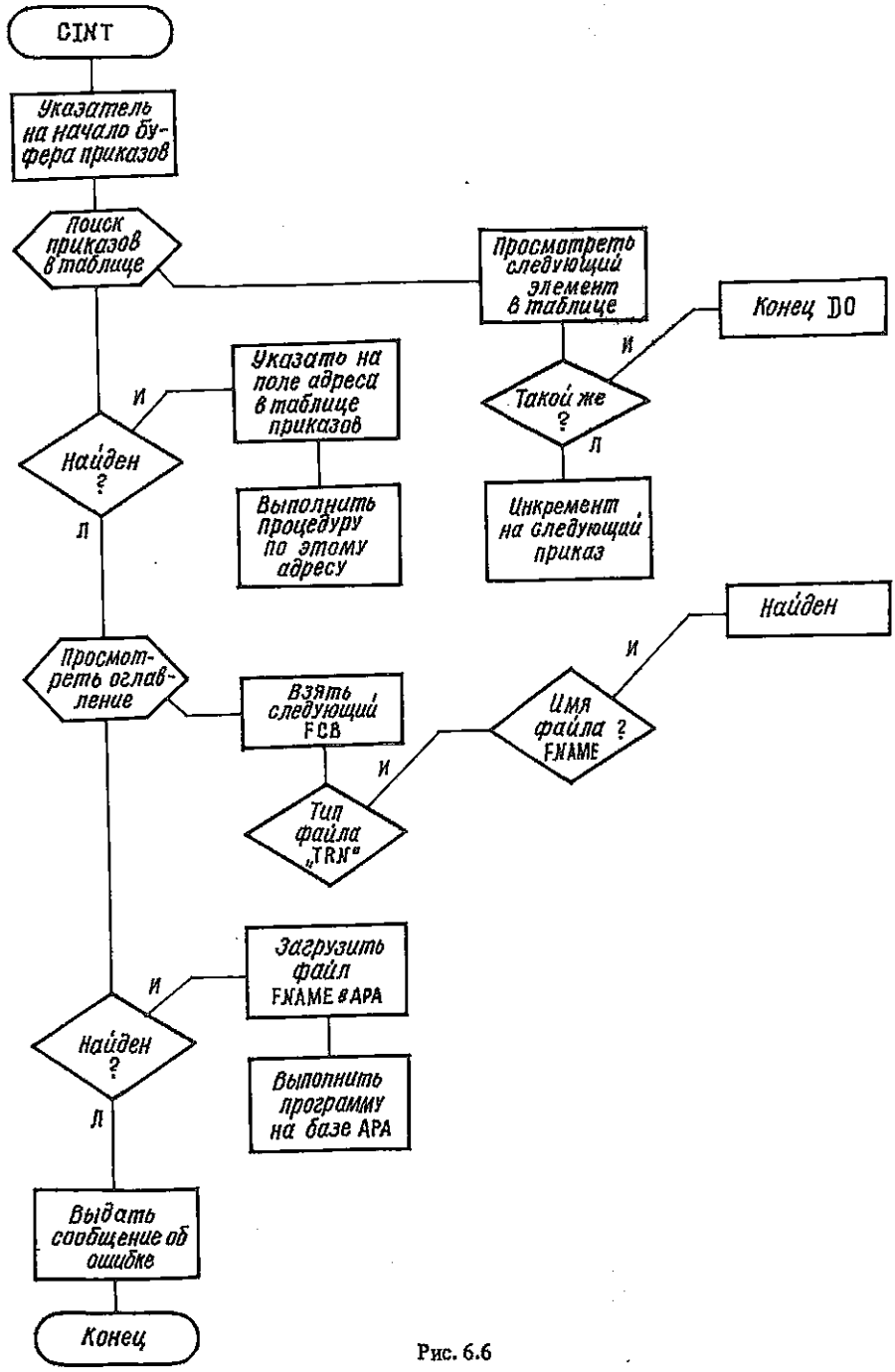

CINT
Указатель на начало буфера приказов
Поиск приказов в таблице
Просмотреть следующий элемент в таблице
Конец DO
И
Такой же ?
Л
Инкремент на следующий приказ
И
Указать на поле адреса в таблице приказов
Найден ?
Выполнить процедуру по этому адресу
Л
Найден
И
Просмотреть оглавление
Взять следующий FCB
Имя файла ? FNAME
И
Тип файла „TRN"
Загрузить файл FNAME в APA
И
Найден ?
Выполнить программу на базе APA
Л
Выдать сообщение об ошибке
Конец

Рис. 6.6

пределяются блоками (по восемь в каждом), то в любом FCB будет место для 16 блоков по восемь секторов, т. е. 16 (блоков) × 8 (секторов) × 128 (байт) = 16 К байт пространства файла.

Таким образом, при произвольном распределении файлы расширяются посредством выделения дополнительных секторов и добавления адресов секторов в список секторов FCB. Если FCB будет исчерпан, то ядро должно создать экстент FCB и изменить первый FCB для учета числа экстентов. Этот метод реализуется несложной процедурой.

В системе с последовательным распределением расширять файлы нелегко, так как "на пути" встречаются другие файлы. В общем случае придется передвигать все файлы, имеющиеся на диске, на требуемое число секторов, а затем соответствующим образом модифицировать оглавление. При последовательном распределении необходимо объединить промежутки и образовать один большой блок свободного пространства в конце диска. Это достигается перемещением всех файлов к нулевому сектору и последующей модификацией оглавления, что поможет учесть новое положение файлов. При произвольном распределении задача становится тривиальной, поскольку автоматически действует механизм распределения, учитывающий доступные секторы.

Для обеспечения сервисных функций (распределения пространства для нового файла, удаления старых файлов, выполнения прикладных программ, распечатки оглавления, копирования файлов и дисков и распечатки файлов) создается интерпретатор приказов. Различают приказы резидентные COM (внутри секции CINT) и транзитные TPN , которые можно вызывать с диска по мере необходимости. Последовательность поиска приказа приведена на рис. 6.6.

6.4. МОНИТОР ОПЕРАЦИОННОЙ СИСТЕМЫ

Основой программного обеспечения интерактивного взаимодействия "человек–машина" является управляющая программа — монитор. Функции монитора — поддержание диалога с оператором, инициирование и контроль за выполнением прикладной программы. Для обеспечения универсальности монитор должен быть программируемым. Программирование монитора заключается в создании набора в данных, содержащих информацию о приказах (командах) оператора и соответствующих им прикладных программах.

Рассмотрим построение монитора для СОС, который специально ориентирован на программируемые базы данных, что позволяет обеспечивать большую гибкость управления. Положим, что операционная среда СОС состоит из двух частей – монитора и системных программ.

Функциональные возможности монитора, как наиболее важной программы, определяют возможности системы в целом. Более детально функции монитора в СОС можно сформулировать следующим образом: поддержание диалога с оператором; обработка наборов данных; загрузка (при необходимости) и инициирование прикладных программ; управление ходом выполнения прикладной программы; обработка программных прерываний.

Для обеспечения максимальной гибкости процедуры управления естественно потребовать, чтобы монитор был оформлен в виде подпрограммы и допускал рекурсивное обращение. Это позволит строить многоуровневый диалог ("диалог в диалоге"), причем на различных уровнях монитор должен опериро-

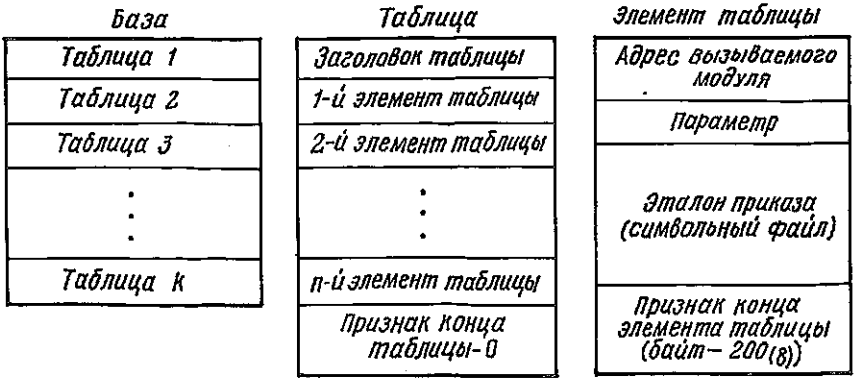


Рис. 6.7

вать с различными наборами данных.

Управление ходом выполнения прикладной программы может заключаться в экстренном завершении по инициативе оператора и при останове выполнения программы. После завершения выполнения прикладной программы управление должно быть передано монитору.

Структурно монитор состоит из интерпретатора приказов и наборов данных. В зависимости от использования в управляющей системе внешних запоминающих устройств прямого доступа (например, гибких дисков) возможны три модели организации СОС: дисковая, резидентная и сверлейная.

Настройка СОС на конкретное применение осуществляется посредством создания наборов данных монитора. Так как все программное обеспечение одновременно находится во внутренней памяти, наборы данных должны содержать информацию о расположении точек входа в отдельные программные модули.

Как отмечалось, монитор должен допускать рекурсивное обращение. При этом его можно рассматривать как прикладной модуль. Но монитор требует передачи ему при обращении как минимум одного параметра – адреса обрабатываемого набора данных. Чтобы упростить управляющую программу и структуру наборов данных, необходимо унифицировать процедуру вызова прикладного модуля: вызывать его как подпрограмму без параметров, а рекурсивное обращение к монитору сделать косвенным, т. е. вызывать его не непосредственно управляющей программой, а из инициированного прикладного модуля, который и передаст монитору адрес нового набора данных. В целях достижения универсальности СОС структура данных должна быть количественно недетерминированной, регулярной и иметь списковую структуру.

Структура данных (рис. 6.7) состоит из заголовка, последовательности (списка) элементов данных, ограниченной признаком конца структуры. Заголовок содержит информацию для интерпретатора приказов, определяющую форму проведения диалога. Длина и строение заголовка фиксированы.

Элемент данных представляет собой пару "идентификатор приказа (или ответа) – адрес точки входа прикладной программы." Идентификатором приказа может быть символьный файл, содержащий эталон приказа и представля-

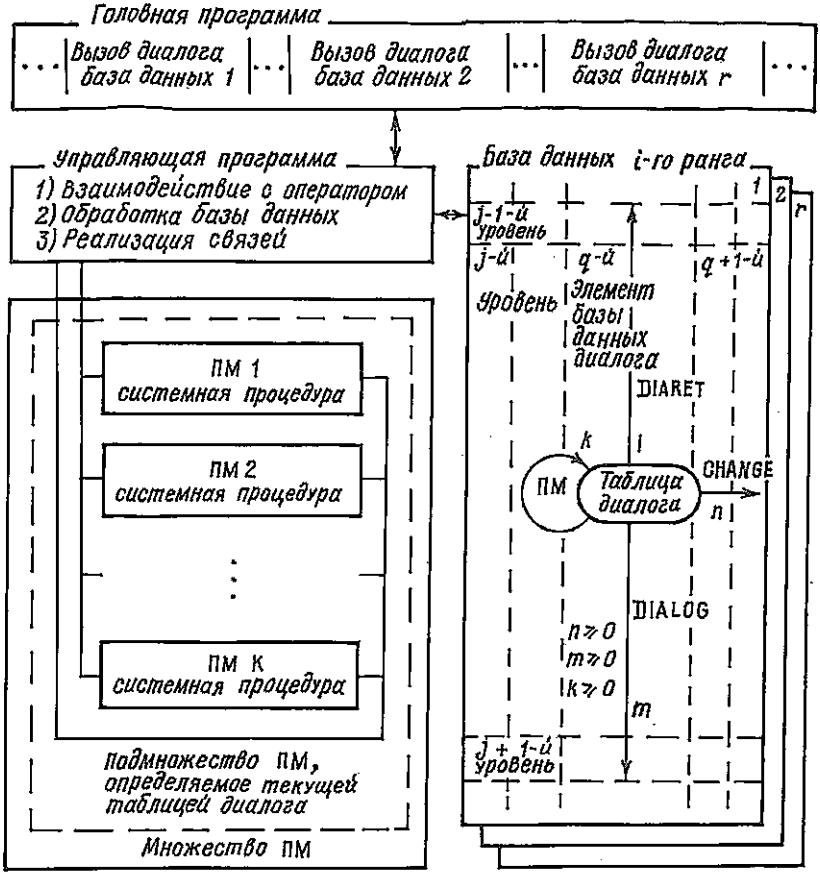


Рис. 6.8

ющий собой строку символов, ограниченную признаком конца. Поскольку в элементе данных содержится только один идентификатор, признак конца символьного файла можно использовать для разграничения элементов в списке.

Описанная структура набора данных монитора позволяет производить просмотр (сканирование) набора при идентификации приказа. Кроме того, возможна организация процедуры динамического формирования наборов данных, что расширяет возможности системы.

Связь между уровнями диалога осуществляется с помощью системных процедур: DIALOG, CHANGE, DIARET (рис. 6.8). Процедура DIALOG (сама управляющая программа) организует собственно процедуру диалога и переход из j-го уровня в $(j + 1)$-й. Процедура CHANGE обеспечивает последовательный переход от одного диалога к другому без изменения уровня вложенности. Процедура DIARET осуществляет возврат из j-го уровня диалога в $(j-1)$-й. При этом переход происходит всегда в одну и ту же точку $(j-1)$-го уровня независимо от переходов внутри j-го уровня по процедуре CHANGE.

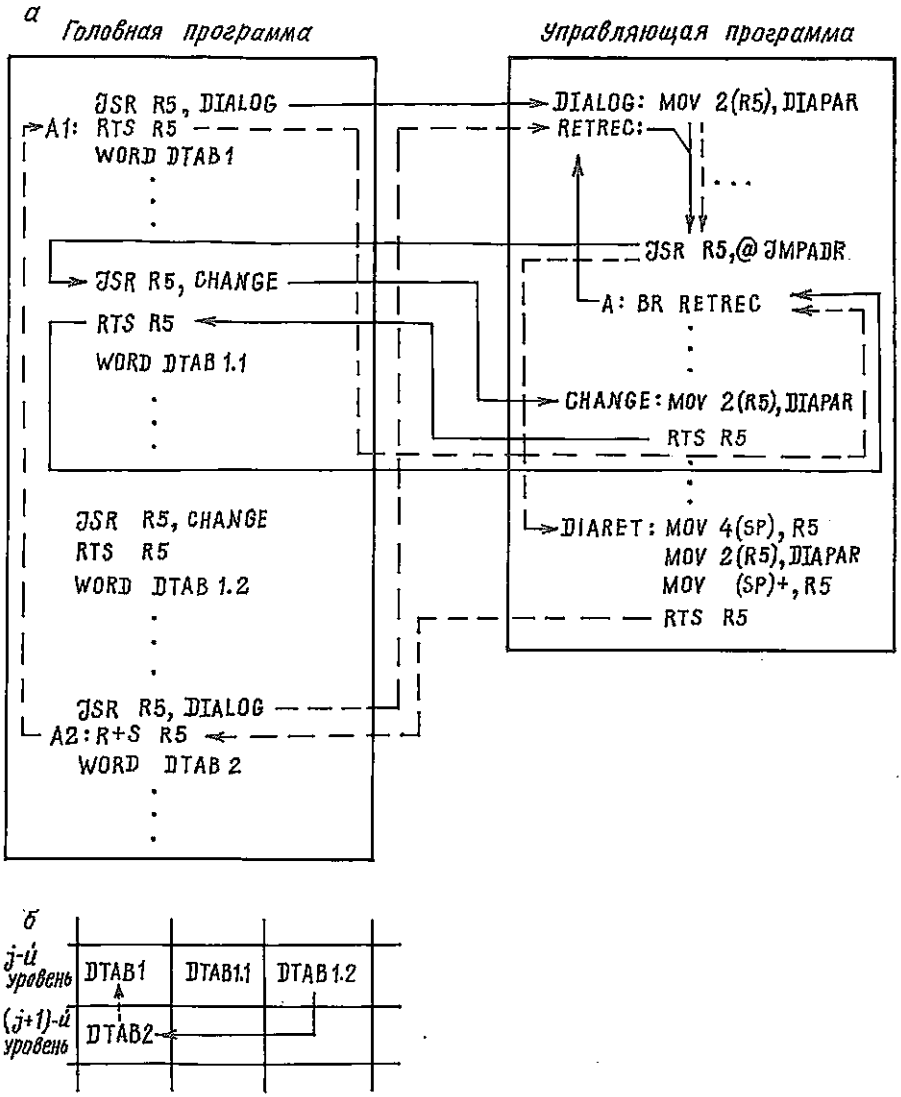


Рис. 6.9

На рис. 6.9, *а* показана диаграмма выполнения последовательности процедур (DIALOG–CHANGE–CHANGE–DIALOG–DIARET). Вложенность управляющей программы DIALOG равна двум. Переход от одного элемента набора данных к другому в *j*-м уровне (рис. 6.9, *б*) будет осуществляться после каждого обращения к процедуре CHANGE в головной программе. Сплошными линиями на рис. 6.9, *а* показана передача управления при действии процедуры CHANGE . Переход из управляющей программы в головную осуществляется через ячейку @ JMPADR, в которой должен быть подготовлен адрес перехода.

На рис. 6.9, *а* штриховой линией обозначены переходы в головной и управляющей программах при переходе к (*j*+1)-му уровню набора данных (вторая процедура DIALOG) и возврате в *j*-й уровень набора данных (процедура DIARET). При обработке данных DTAB2 управляющая программа получила системную процедуру DIARET, на основании которой она сформировала адрес перехода к программе DIARET и осуществила передачу управления на обработку набора данных DTAB1 (возврат в *j*-й уровень).

Для генерации рабочей версии системы необходимо сформировать наборы данных монитора (программирование монитора), выбрать необходимые системные модули из библиотек СОС и скомпоновать их с недостающими прикладными программами. Полученный в результате компоновки загрузочный модуль и будет рабочей версией специализированной системы.

Рассмотрим пример программирования монитора СОС, а именно набор данных под именем DIAP5:

```
DIAP5 : .WORD   SFIL9,TPAR1
;ОБРАЩЕНИЕ К СПРАВОЧНОМУ ФАЙЛУ
        .WORD   INFORM,1        ;НОМЕР СПРАВОЧНОГО ФАЙЛА
        .ASCII  /СПРАВКА/
        .BYTE   200
        .EVEN
;УПРАВЛЕНИЕ ЭКСПЕРИМЕНТОМ
;
        .WORD   UPSTIS,0
        .ASCII  /1/
        .BYTE   200
        .EVEN
;СПЕКТРАЛЬНЫЙ АНАЛИЗ ПРОЦЕССОВ
        .WORD   SPANPR,0
        .ASCII  /2/
        .BYTE   200
        .EVEN
;НЕОПОЗНАННЫЙ ОТВЕТ
        .WORD   RETDIA,0
        .BYTE   200
        .EVEN
        0
```

Первая строка является заголовком, содержащим адрес символьного файла выводимого сообщения SFIL9 (в приведенном листинге не представлен) и таблицы параметров TPAPI , содержащей информацию для управления режимом работы конкретного дисплея. Далее идет последовательность элементов набора данных. Структура элемента следующая: адрес точки входа в вызываемый программный модуль; передаваемый ему параметр (если параметр не требуется, место для него все равно должно быть зарезервировано); идентификатор ответа; ограничитель элемента – код 200 (восьмеричное число).

На адрес точки входа вызываемого модуля наложено ограничение: он не должен быть равен 0. Если в наборе данных на месте адреса точки входа встречается 0, то программа обработки структуры данных рассматривает его как признак конца текущего набора данных. При этом управление передается модулю, описанному в последнем обработанном элементе данных.

Рекомендуемая литература: 1, 6, 7, 11.

7. ОСНОВЫ ОРГАНИЗАЦИИ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

7.1. ОСНОВНЫЕ ПОНЯТИЯ РЕСУРСА И ПРОЦЕССА

Режимы ОС. При проектировании современной ОС микроЭВМ наиболее подходящей единицей декомпозиции является процесс. В самом общем смысле вычислительный процесс предполагает два момента: он является носителем данных и одновременно выполняет операции. Концепция процесса длительное время использовалась в языках программирования для моделирования дискретных событий.

После того как индивидуальные характеристики любого устройства микроЭВМ абстрагированы, его можно рассматривать как некоторый ресурс. К ресурсам относятся также программы и наборы данных. Основная единица, претендующая на ресурсы внутри микроЭВМ, – это процесс. Поскольку внутри ОС может существовать произвольное число процессов, между ними возможна конкуренция за обладание тем или иным ресурсом, в первую очередь временем процессора. Это связано с необходимостью организовать управление процессами и их планирование.

Работа микроЭВМ протекает под управлением ОС, которая включает совокупность программ для управления процессами, распределения ресурсов, организации ввода-вывода и интерфейса с пользователем. С точки зрения организации вычислительных процессов существует несколько режимов ОС. Простейшим является однопрограммный однопользовательский, в котором вычисления носят последовательный характер, а ресурсы не разделяются. В режиме пакетной обработки ОС содержит, кроме основного задания, дополнительные (фоновые), которые реализуются в моменты времени, когда процессор свободен от выполнения основной программы.

Мультизадачная ОС обеспечивает одновременное нахождение в памяти нескольких программ, которые последовательно используют время процессора, прерываясь на выполнение ввода-вывода. При этом возможно разделение как аппаратных, так и программных ресурсов. В режиме разделения времени (многопользовательская) ОС обеспечивает выделение каждому пользователю квант времени процессора, при этом усложняется распределение ресурсов, в первую очередь времени процессора и основной памяти. В режиме реального времени ОС работает в "линию" с управляемым объектом и имеет жесткие ограничения на время ответа, в которое могут быть выделены максимально возможные ресурсы.

Мультипроцессорная ОС управляет работой системы, включающей несколько неоднородных процессоров (обрабатывающих, периферийных, функциональных). В такой ОС выполняются реальные параллельные процессы, а распределение ресурсов носит наиболее сложный характер. Возможны также комбинированные вычисления в ОС – разделение времени и реального времени.

Управление процессами. Многопроцессная система для эффективного использования ресурсов организует к ним очереди процессов. Для этого в памяти находится несколько процессов, ожидающих ресурс и готовых его использовать после освобождения.

Необходимо различать процессы пользователя и системные. Первые не могут прямо порождать другой процесс и инициировать ввод-вывод и выдают запросы на системное обслуживание. Вторые управляют ресурсами системы и не могут выдать запрос на системное обслуживание.

Процессы находятся в различных состояниях — готовности, выполнения, ожидания. Процесс, стоящий в очереди на выполнение процессором, находится в состоянии *готовности.* Процесс, ожидающий ресурса, отличного от ЦП, или сообщения другого процесса, является *ожидающим.* Наконец, процесс, получивший ресурс процессора от диспетчера, находится в состоянии *выполнения.*

За время своего существования процесс может переходить из одного состояния в другое, что обусловлено последовательностью команд, взаимодействием с другими процессорами и состоянием ресурсов. В многопроцессорной системе из-за взаимодействия процессов возможны четыре ситуации: взаимное исключение, синхронизация, взаимоблокировка и коммуникация. Рассмотрим кратко эти задачи.

Взаимное исключение возникает из-за предотвращения использования несколькими процессами одного ресурса. Оно реализуется на нижнем уровне применением семафоров, на верхнем уровне — механизма мониторов. Под *синхронизацией* процессов понимают связь между двумя процессами по установленному алгоритму. При этом вводится понятие переменной типа "событие" у передающего процесса, которая вызывает у получающего процесса соответствующие действия. Под *взаимоблокировкой* понимается состояние, возникающее вследствие конкуренции за ресурсы. Возможна ситуация, когда все процессы в системе не в состоянии продолжить работу. Например, процесс P_1 владеет ресурсом V_1 и требует ресурса V_2, а процесс P_2 владеет ресурсом V_2 и требует ресурса V_1. Для предотвращения этой ситуации предложено несколько решений: предотвращать, автоматически обнаруживать, устранять тупиковые ситуации.

Между процессами возможен обмен информацией, длиной бо́льшей, чем слово. Этот обмен сообщениями имеет сложную структуру, обеспечивающую защищенность процессов при *коммуникации.*

Структура данных процессов и ресурсов. Для отражения состояния процессов и ресурсов используются определенные структуры данных — дескрипторы, которые обрабатываются системными программами и определяют состояние ОС.

Дескриптор процесса может быть задан пятеркой компонентов вида $D = (I, V, T, P, R)$, где I — идентификатор процесса, V определяет виртуальную машину процесса (процессор, память и другие ресурсы); T включает состояние процесса (выполняемый, готов, ожидающий); P указывает на "отца" и "потомков" процесса; R определяет данные для планирования (приоритет).

Для управления вычислительной системой (ВС) системное программное обеспечение должно включать следующие компоненты: управление процессами, распределение ресурсов, файловую систему, программу обработки прерываний. Модуль управления процессами выполняет функции по созданию, уда-

лению, активизации, связи и планированию процессов. Распределение ресурсов отвечает за управление основной и внешней памятью, устройствами ввода-вывода, а также процессорами. Файловая система ответственна за создание, разрушение, модификацию и восстановление информации на внешней памяти. Эти компоненты вызывает четвертая составляющая – программа обработки прерываний (например, создания, удаления процессов, ввода-вывода и т. д.).

ОС может рассматриваться как расширение аппаратуры, т. е. как виртуальная машина для управления вычислительными процессами. В основной памяти всегда находятся программы управления процессами, распределения ресурсов, программы (драйверы) ввода-вывода и обработки прерываний. Для микроЭВМ имеется устойчивая тенденция аппаратной реализации перечисленных функций.

С каждым классом ресурса связаны процедуры распределения и освобождения. При этом указываются класс ресурса, идентификатор и число. Основными примитивами управления ввода-вывода являются операции: ЧИТАТЬ, ПИСАТЬ, ОПРОСИТЬ СОСТОЯНИЕ.

При обработке внешние и внутренние прерывания могут рассматриваться как сигналы "пробуждения" блокированных процессов, сообщающие об одном из событий: завершении аппаратных операций внешнего устройства или программного процесса; затребовании обслуживания пользователем; ошибке ввода-вывода, адресации; обнаружении переполнения и т. д. Программный обработчик прерываний должен сохранить состояние прерванного процесса, определить, какой процесс необходимо запустить, после его завершения восстановить прерванный процесс.

Пользовательский интерфейс. Компоненты ОС непрерывно вызываются пользовательскими программами через запросы на ресурсы, ввод-вывод, а также в ответ на прерывания. На более высоком уровне информация о программах и файлах связывается посредством командного языка ОС. Последний включает команды для выполнения разнообразных функций: создания, редактирования, удаления файлов, запуска программы на выполнение, показа списка файлов и т.д. Традиционная форма предложений командного языка имеет вид: команда < список операндов >< режимы >.

Командный язык определяет для пользователя виртуальную машину, которая соответствует управлению процессами, файлами на внешнем уровне. Существует процесс, читающий и интерпретирующий команды пользователя (интерпретатор команд).

Методология проектирования ОС. Операционную систему можно рассматривать как динамический набор взаимодействующих процессов, которые связаны системными структурами данных. Существует несколько подходов к проектированию ОС.

Первый подход основан на размещении процессов ОС в иерархическом порядке. При этом каждый уровень процессов определяет абстрактную машину. Чем выше уровень иерархии, тем больше выполняется задач по управлению ресурсами, но определенные классы ресурсов могут не рассматриваться на каждом уровне. Уровни бывают следующие: нулевой – распределение процессоров между процессами; первый – управление основной и внешней памятью; второй – связь между виртуальными процессами и терминалом оператора и т. д. Процессы выше нулевого уровня не "интересуются" числом до-

ступных физических процессоров, выше первого "работают" только с сегментами информации независимо от ее размещения в памяти и т. д. Синхронизация в системе достигается за счет аппаратного механизма прерываний и программными средствами. Так, прерывания от таймера связаны с нулевым уровнем, при работе с внешней памятью – с первым. Данная методология разделяет функции ОС и устанавливает каналы связи между уровнями, что снижает сложность проектирования. Основные трудности связаны с определением структуры и функций уровней.

Второй подход заключается в том, что основные усилия при проектировании сосредоточиваются на организации ядра, которое строится универсальным для нескольких модификаций ОС. При этом стратегия распределения ресурсов может расширяться при изменении требований новых задач. На основании этих двух подходов сформировалась последовательность шагов проектирования ОС, которая может включать: 1) определение виртуальной машины пользователя (состав команд), разработку командного языка и определение функций, выполняемых командами; 2) установление путей прохождения команды через ОС и аппаратуру; 3) определение процессов, необходимых для реализации п. 1, 2, уточнение функций каждого процесса, его взаимосвязи с другими процессами, а также разработку структур данных, требуемых для связи; 4) размещение процессов в иерархии, определение компонентов ядра системы и их структур данных, разработку стратегии распределения ресурсов; 5) исследование правильности проектирования и анализ поведения системы с помощью аналитических методов и моделирования.

На практике моделирование требует значительных затрат времени и ресурсов, а аналитические методы разработаны недостаточно. Поэтому реальной методологией является объединение этапов проектирования, моделирования и реализации. При этом часто функции ядра и ввода-вывода фиксируют, а затем их расширяют для реальных приложений.

7.2. ВЗАИМОДЕЙСТВУЮЩИЕ ПРОЦЕССЫ

Семафоры и сигналы. Связь между процессами осуществляется обычно с помощью общей области памяти, куда заносятся и откуда считываются данные. ОС предоставляет системные процедуры для доступа к этим взаимно исключающим областям. Кроме того, процессы требуют синхронизации, для чего используются системные процедуры. Чтобы лучше понять программирование параллельных систем, рассмотрим проблему ПОСТАВЩИК–ПОТРЕБИТЕЛЬ.

Пусть имеется процесс, генерирующий поток данных (ПОСТАВЩИК), а другой процесс получает эти данные и является ПОТРЕБИТЕЛЕМ. В общем случае скорость процессов может быть различной, поэтому между процессами помещается буфер. Элемент данных x заносится в буфер обращением ЗАНЕСТИ x, а выбирается из буфера с помощью обращения ВЫБРАТЬ x. Поскольку буфер ограничен, необходимы две функции ПОЛОН и ПУСТ.

Приведем решение данной задачи, причем каждый процесс представляется программой, а буфер находится в общей области памяти:

```
program поставщик;   program потребитель;
 var x:данные;          var x:данные;
```

```
  begin                    begin
  do                        do
  выработать(x);       while пуст do
while полон do             ждать
  ждать                  end do
end do;                      выбрать(x);
  занести(x)                 потребить(x)
end do                     end do
 end                        end
```

Процесс ПОСТАВЩИК выполняет бесконечный цикл: выработка x , ожидание неполного буфера, занесение x в буфер. Процесс ПОТРЕБИТЕЛЬ выполняет дополнительный цикл: ожидание непустого буфера, выбор из него x и его обработка. При этом возможно одновременное обращение к буферу, что может привести к неправильному занесению или выборке данных.

Для исключения взаимного обращения к общим ресурсам (буферу) используется механизм семафоров. *Семафором* называется неотрицательная двоичная переменная, которая принимает значение 1, когда ресурс доступен. Для семафора $s = 0$ буфер будет недоступным и процесс вынужден ждать. Для снижения затрат процессорного времени приостановленный процесс помещается в очередь, связанную с данным семафором. Проверка семафора осуществляется с помощью обращения к двум системным процедурам:

оградить (s) if $s = 1$ then $s := 0$ else приостановить процесс, поместив в очередь (s)

освободить (s) if очередь (s) пуста then $s := 1$ else возобновить первый процесс в очереди (s).

С использованием семафора доступ к буферу обеспечивается блокировкой обращений к нему процедурами ОГРАДИТЬ и ОСВОБОДИТЬ. Для исключения затрат машинного времени на ожидание полноты и пустоты буфера необходимо ввести еще один примитив, называемый сигналом. Сигнал используется процессом для сообщения другому процессу о совершении некоторого события. Для работы с сигналами имеются две процедуры: ПОСЛАТЬ (e) для посылки сигнала e и ЖДАТЬ (e) для ожидания поступления сигнала e. Сигнал также является двоичной переменной, процедуры имеют вид:

послать (e) if очередь (e) пуста then e = 1 else возобновить первый процесс в очереди;

ждать (e) if e = 1 then e : = 0 else приостановить процесс, поместить его в очередь (e).

Заметим, что определение сигнала похоже на определение семафора. Разница в том, что семафор применяется для взаимного исключения обращения к ресурсу, а сигнал — для синхронизации процессов. С использованием введенных примитивов решение задачи о ПОСТАВЩИКЕ и ПОТРЕБИТЕЛЕ имеет вид:

```
program поставщик;          program потребитель;
 var x:данные;               var x:данные;
  s:семафор;                 s:семафор;
нэполон,непуст:сигнал;      неполон,непуст:сигнал
  begin                        begin
   do                           do
выработать(x);                 оградить(s);
   оградить(s);                if пуст then
```

```
if полон then                освободить(s);
  освободить(s);              ждать(непуст);
    ждать(неполон);           оградить(s);
     оградить(s)               end if;
    end if;                    выбрать(x);
   занести(x);                освободить(s);
   освободить(s)               послать(неполон)
   послать(непуст)               end do
     end do
  end                                    end
```

В данном фрагменте процесс ПОСТАВЩИК выполняет последовательность:

выработать (x), оградить (s), проверить, не полон ли буфер,
занести (x), освободить (s) .

При этом критический участок огражден обращением к процедурам ОГРАДИТЬ и ОСВОБОДИТЬ. В конце каждого цикла ПОТРЕБИТЕЛЬ посылает сигнал Неполон, по которому поставщик, ожидая свободного места в буфере, продолжит свою работу. В свою очередь процесс ПОТРЕБИТЕЛЯ синхронизируется сигналом Непуст.

Рассмотренному взаимодействию процессов с использованием примитивов сигналов и семафоров присущи несколько недостатков. Во-первых, при их использовании можно допустить ряд ошибок: передать управление критическому участку, минуя процедуру ОГРАДИТЬ, что приведет к порче данных; забыть обратиться к процедуре ОСВОБОДИТЬ, что приведет к тупиковой ситуации; забыть воспользоваться семафором. Аналогичные ошибки возможны и при использовании сигналов. Таким образом, примитивы семафоров и сигналов имеют низкий уровень. При их применении можно ошибиться, но даже при правильном использовании получаются неясные программы. Кроме того, каждая программа транслируется отдельно и транслятор может не знать о параллельной среде и не сумеет проверить правильность использования параллельных примитивов. Поэтому при описании параллельных систем программист должен использовать более мощные средства для определения взаимодействующих процессов.

Мониторы. Из двух рассмотренных примитивов наибольшие трудности связаны с семафорами. Поэтому для решения проблемы взаимного исключения ресурсов предложены различные конструкции более высокого уровня, одной из которых является монитор. Монитор – модуль, в котором ресурс объединяется с процедурами для раздельного использования процессов. При этом только один процесс может выполнять мониторную процедуру в данном мониторе. Монитор является конструкцией более высокого уровня для предотвращения одновременного доступа к разделяемым ресурсам, однако он не предоставляет никаких средств синхронизации процессов. Поэтому монитор используется совместно со средствами для обработки сигналов.

Рассмотрим использование примитива монитора для решения задачи ПОСТАВЩИК–ПОТРЕБИТЕЛЬ, опустив описание интерфейса монитора:

```
program постав-потреб;
 monitor буферизация;
  var b:буфер;неполон,непуст:сигнал;
 procedure entry передать(in x:данные);
  begin
```

```
if полон then ждать(неполон) end if;
   занести(x);
   послать(непуст)
 end;
procedure entry получить(out x:данные);
  begin
if пуст then ждать(непуст) end if;
   выбрать(x);
   послать(неполон)
  end;
begin
   инициализировать b
 end;{monitor}
 process поставщик    process потребитель
  var x:данные;        var x:данные;
   begin do             begin do
 выработать(x);     буферизация.получить(x);
 буферизация.передать(x)  потребить(x)
    end do                 end do
  end;                   end;
        begin
 init поставщик,потребитель
        end
```

Структура процессов ПОСТАВЩИК–ПОТРЕБИТЕЛЬ значительно проще, чем в первом случае, так как каждый процесс состоит из двух обращений к процедурам. Ответственность за обращения к буферу берет на себя монитор.

В мониторе содержится описание данных, входные процедуры для обеспечения доступа к данным и тело, выполняемое при инициализации данных. Монитор работает следующим образом. Процесс ПОСТАВЩИК передает данные в буфер, обращаясь к процедуре ПЕРЕДАТЬ. Процесс ПОТРЕБИТЕЛЬ получает данные, обращаясь к мониторной процедуре ПОЛУЧИТЬ. При обращении любого процесса к монитору, в котором находится другой процесс, выполнение первого приостанавливается, и он заносится в очередь. При освобождении монитора первый процесс из очереди активизируется.

Синхронизация процессов в мониторе реализуется следующим образом. Когда процесс получает доступ к процедуре монитора, а затем выдает операцию ждать сигнала, следующему процессу разрешается войти в монитор и послать сигнал, что он ждет. Для исключения этого последней является операция монитор ПОСЛАТЬ.

Однако при использовании монитора возникают две трудности. Во-первых, выходящий из монитора процесс возобновляет не более одного процесса, т. е. невозможно синхронизировать несколько процессов с помощью одного монитора, если не вывести из него сигналы. Во-вторых, если из монитора А вызывается процедура монитора В, то возможна тупиковая ситуация в случае приостановления процесса в В (приостанавливается процесс в А).

7.3. УПРАВЛЕНИЕ ПРОЦЕССАМИ И РЕСУРСАМИ

Планирование и диспетчеризация процессов. При управлении процессами необходимо осуществлять их запуск и остановку, поддерживать связь и выполнять коммуникацию. Для распределения ресурсов, в первую очередь процессора (ов), используются планировщики и диспетчеры. Рассмотрим базовые

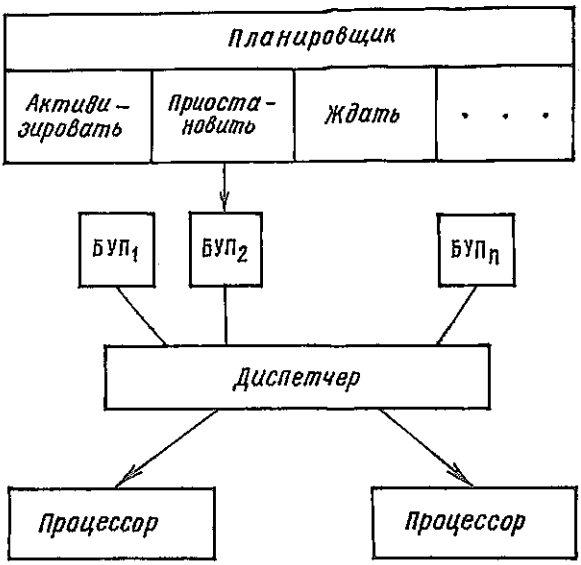


Рис. 7.1

концепции планирования и диспетчеризации ОС и связанные с ними средства и действия.

Хотя в большинстве систем не делается разницы между планированием и диспетчированием, в системах РВ их необходимо различать. Планирование – это построение процессов по определенной стратегии в некоторую последовательность, реализуемое программой ПЛАНИРОВЩИК. Диспетчирование – это выбор процессов из очереди по определенной стратегии, перевод их в активное состояние и передача на выполнение процессорам. Эти действия осуществляются программой ДИСПЕТЧЕР.

Базовая структура планировщика-диспетчера приведена на рис. 7.1. Планировщик представляет собой набор модулей, выполняющих операции управления процессами: создать, активизировать, приостановить и т.д. Возможны два варианта реализации планировщика. Первый предполагает использование главного (единственного) планировщика, встроенного в ядро ОС и осуществляющего разделение времени процессора. Второй вариант предполагает использование раздельного планировщика, который помещается в адресную часть каждой программы пользователя. Процесс осуществляет вызов планировщика для постановки себя в очередь. Это позволяет каждой программе использовать различную стратегию планирования.

Планировщик выбирает следующий, готовый к исполнению процесс – либо отложенный, либо выбранный из очереди на выполнение. Если отложенного процесса нет, выбирается либо следующий по порядку из очереди (в бесприоритетной системе), либо процесс с высшим приоритетом. Если очередь на выполнение пуста, выбирается пустой процесс.

Планировщик начинает работать по истечении заданного кванта времени (прерывание по таймеру), а также при поступлении внешнего прерывания с пользователя либо внутреннего прерывания из системы. После обслуживани

прерываний процессор считается доступным для всех готовых к выполнению процессов.

Алгоритмы планирования. Различают планирование циклическое, приоритетное, адаптивное. При *циклическом* планировании процесс, использовавший выделенный ему квант времени, помещается в конец очереди на выполнение. Возможно организовать несколько очередей, в каждой из которых величина кванта времени меняется.

При *приоритетном* планировании каждому процессу присваивается приоритет, который определяет его положение в очереди на выполнение. Процесс с высшим приоритетом ставится в начало очереди, с низшим – в ее конец. Приоритет может быть статическим и динамическим.

При *адаптивном* планировании предполагается контроль над реальным использованием памяти. Каждому процессу устанавливаются значения объема памяти и виртуального времени процессора, которое обратно пропорционально максимальному объему памяти, необходимому процессу. Процессу выделяется необходимый временной интервал только при наличии достаточного объема свободной памяти. Планирование ориентировано на систему процессов с минимальными запросами, т. е. отдается предпочтение процессам небольшого размера с малым временем выполнения.

Данные для управления процессами. Процесс планирования базируется на двух структурах данных – блоке управления процессом (БУП) и очереди выполнения. БУП содержит информацию описания процесса. Очередь на выполнение включает список процессов в порядке их обработки.

Различные ОС микроЭВМ содержат различную информацию в БУП. Рассмотрим основные составляющие БУП. Каждый БУП включает базовую область, которая неизменна для всех процессов и связей с другими блоками управления процессом.

В течение существования процесса его БУП находится в системной очереди. Процесс может иметь блок данных для хранения локальных данных процесса. Большинство процессов имеют "прародителя", который информируется о результатах выполнения "потомка" кодом завершения.

Для обслуживания запроса на ввод-вывод порождается системный процесс. Это позволяет основному процессу продолжить свою работу независимо от реализации ввода-вывода. Необходимые для запроса параметры содержатся в области запроса ввода-вывода. Для каждого запроса требуется канал, связанный с файлом либо устройством. При передаче данных указываются адрес памяти передачи, адрес файла, адрес буфера для передачи. Вся эта информация содержится в области запросов ввода-вывода БУП.

Для управления записями в БУП вводится соответствующая область, которая содержит элементы, отвечающие за обеспечение целостности записей. Элементы области запросов к устройствам БУП содержат параметры для создания и обслуживания таких запросов. Например, для диска требуются следующие параметры: физический адрес данных, адрес памяти обмена, число передаваемых слов, ключ сортировки в области запросов к диску.

Системным процессам необходима область памяти для создания стека процесса, который располагается в БУП и занимает место в области запросов к системным службам. Последняя включает поля активных событий, поля завершенных событий и очередь всех процессов.

Блок данных процесса (БДП) содержит данные самого процесса и располагается в младших адресах пространства памяти процесса. Он включает адресуемые данные процесса, адрес БУП процесса – "отца", передаваемый код событий, размер сообщения порождаемому процессу и процессу – "отцу", указатель стека пользователя, а также поля для хранения статуса.

Таким образом, БУП процесса может включать следующие области: базовую, запросов ввода-вывода, управления записями, запросов к устройствам, стека процесса, запросов к системным службам и блок данных.

Функции планировщика-диспетчера. В многопроцессных ОС имеются функции, позволяющие пользователю инициировать и останавливать процессы, создавать или уничтожать их, а также пересылать сообщения. В основе функций лежит вызов модулей, которые ответственны за управление процессами. Рассмотрим отдельные из этих модулей.

В ходе выполнения системного процесса он может установить отсутствие требуемого для него ресурса. Для освобождения процессора он временно себя приостанавливает, записывая в стек свой БУП. При этом не происходит записи в очередь на выполнение. Для активизации такого процесса другой процесс обращается к процедуре АКТИВИЗИРОВАТЬ с указанием адреса процесса. Процедура АКТИВИЗИРОВАТЬ помещает БУП в очередь на выполнение

Выполнение программ инициализации, необходимых для обработки файлов и запросов ввода-вывода, реализуется модулем Запуск подпроцесса. Для хранения адреса подпроцесса вводится элемент Статус события. По окончании работы подпроцесса возвращается его код завершения, а для остановки подпроцесса используется модуль Конец подпроцесса.

Для параллельного выполнения основного процесса и подпроцесса применяют модуль, который проверяет код события, определяющий действия, реализуемые после завершения работы подпроцесса.

В работе планировщика используются модули для обработки очередей процессов. БУП помещается в очередь на выполнение программой АКТИВИЗИРОВАТЬ, а удаляется программой ПЛАНИРОВЩИК. Последние программы используют модули работы с очередями: ВСТАВИТЬ в очередь и УДАЛИТЬ из очереди. Эти программы различны для циклического и приоритетного планировщиков, поскольку для первого вставка производится в конец очереди, а удаление выполняется из начала очереди, для второго вставка производится в соответствии с приоритетом.

Функции управления процессами. Различные ОС могут включать несколько функций, но основными являются: создание, удаление процесса, его идентификация, задержка, изменение приоритета и планирование.

Операция создания нового процесса может выполняться на двух уровнях – пользователя и системном. На уровне пользователя в запросе указывается код события (для сообщения порождающему процессу статуса созданного процесса), начальный адрес нового процесса, длина сообщения, которое порожденный процесс может передать породившему, адрес сообщения и идентификатор нового процесса. Создание системного процесса производится только ОС и включает запрос на распределение блока управления процессом из пула БУП. Адрес БУП передается процессу – "отцу".

В ОС включается набор средств для завершения работы процесса: благополучно по окончании задачи или неблагополучно из-за ошибок. Процесс мо

жет быть удален директивно с указанием идентификатора или остановлен другим процессом; кроме того, он может быть также самоуничтожен. При удалении процесса он стирается из всех системных очередей, а также освобождаются и возвращаются в системные пулы все занимаемые им ресурсы. Идентификация процесса включает два момента – выяснение собственного идентификатора и определение статуса другого процесса по известному идентификатору. В первом случае входным параметром является адрес БУП процессора, во втором – идентификатор процесса.

В приоритетных системах планирования управление приоритетами включает: изменение процессом своего собственного приоритета с последующей обработкой планировщиком; изменение приоритета класса процессов.

Большинство ОС содержит функции для приостановки и возобновления работы процесса, что требует меньшего количества операций, чем создание и уничтожение процесса. Кроме того, сохраняются блок данных процесса и отведенные ему ресурсы. Далее процесс может быть обработан планировщиком и позволяет обойти ограничения на ресурс памяти.

В отдельных ОС реального времени добавляется возможность планирования работы процесса заранее. Основой для такого планирования является инфомация о процессах, упорядоченная во времени. Системный планирующий процесс просматривает очередь и выявляет процесс, который должен быть активизирован. При обнаружении такой информации она удаляется из очереди, а требуемый процесс создается и помещается в очередь на выполнение.

7.4. УПРАВЛЕНИЕ ПАМЯТЬЮ

Функции управления памятью. В ОС микроЭВМ могут различаться понятия логической и физической памяти. *Физическая память* соответствует реальной памяти, а *логическая* – набору адресов, с которыми может работать программа. Очевидно, что процессы не могут одновременно находиться в одном месте физической памяти, поэтому первой функцией управления является *отображение* адресов логической памяти на физическую.

Для обеспечения эффективной работы нескольких программ ОС должна обеспечить *разделение* памяти между несколькими процессами, что усложняет работу планировщика. Это является второй функцией управления памятью. Разделение памяти создает еще одну проблему, связанную с *защитой* процессов друг от друга. Кроме того, для межпроцессорного обмена необходимы общие наборы данных или файлы, которые также требуют защиты.

Отображение памяти. Перевод логических адресов в физические производится несколькими способами: абсолютной, статической и динамической трансляцией. Первая осуществляется транслятором при генерировании ими абсолютных адресов (псевдокоманда ORG в ассемблерах). Статическая трансляция выполняется для загрузочного модуля, который помещается в память относительно базового адреса. Динамическая трансляция реализуется в процессе загрузки программы средствами ОС. При прерываниях возможно перемещение программы в другую область памяти.

Статическая и динамическая трансляции отличаются трактовкой двух основных аспектов выполнения программы: предсказания доступности памяти и адресных ссылок процесса. При статической трансляции считается, что после

компоновки ресурс памяти определен, невозможно расширение адресного пространства процесса путем выдачи запроса.

При динамической трансляции ресурс памяти не может быть распределен изначально, а пространство памяти увеличивается и уменьшается в ходе выполнения процесса.

Разделение и распределение памяти. Разделение памяти между процессами осуществляется несколькими способами. Программный модуль может быть подсоединен к каждой из нескольких программ в виде отдельной копии. Кроме того, этот модуль может быть переадресован в адресное пространство каждого из процессов путем аппаратной поддержки (использованием встроенного стека). Программы пользователя могут поочередно обращаться в ОС к сервисным программам (утилитам), используя для этого механизм с семафорами.

Распределение памяти в многопроцессной системе является функцией ОС и заключается в выделении каждому процессу блоков физической памяти с поддержанием контроля за этим распределением. Если система имеет виртуальную организацию памяти, то необходимо следить за виртуальной адресацией и взаимодействием с программами файловой системы.

Защита памяти. В многопроцессной системе процессы должны быть защищены друг от друга, а ОС защищена от процессов пользователей. Защита включает два аспекта: выявление ошибки проникновения процесса в адресное пространство другого процесса и устранение последствий такого проникновения. Реализация защиты чисто программными средствами трудоемка из-за проверки обращений, распределения памяти и т.д. Защита легче реализуется применением аппаратных средств и зависит от выбранного способа разделения памяти – на блоки фиксированной длины (страничная организация) или блоки переменной длины (сегментная организация). При этом каждый блок назначается процессу пользователя или ОС и защищается блоком управления памяти, контролирующим все адресные ссылки, по которым разрешаются чтение и запись данных.

Расширение памяти. Физическое пространство памяти микроЭВМ ограничено длиной адресной части команды (16 разрядов для МП 1810ВМ86). Адресное пространство – это диапазон адресов, с которым может работать программист. Оно может быть расширено как программными, так и аппаратными средствами при логической организации и только аппаратными при физической. В последнем случае к адресной части прибавляется дополнительный код (содержимое регистра сегмента для МП 1810ВМ86). Без применения аппаратных средств пространство расширяется путем использования оверлейного механизма, при котором программа в память загружается по частям с возможностью перекрытия, т. е. с использованием одинаковых адресов.

Другими средствами расширения памяти является страничная и сегментная организация памяти. В первом случае адресное пространство разбивается на участки фиксированной длины, во втором – на участки переменной длины (обычно равной длине используемых программ). Страничная организация накладывает ограничения на возможность разделения программ и данных, а также динамического расширения и сжатия программы в процессе ее работы. Сегментирование позволяет решить данные проблемы, однако при этом значительно возрастает сложность ОС и аппаратной поддержки памяти.

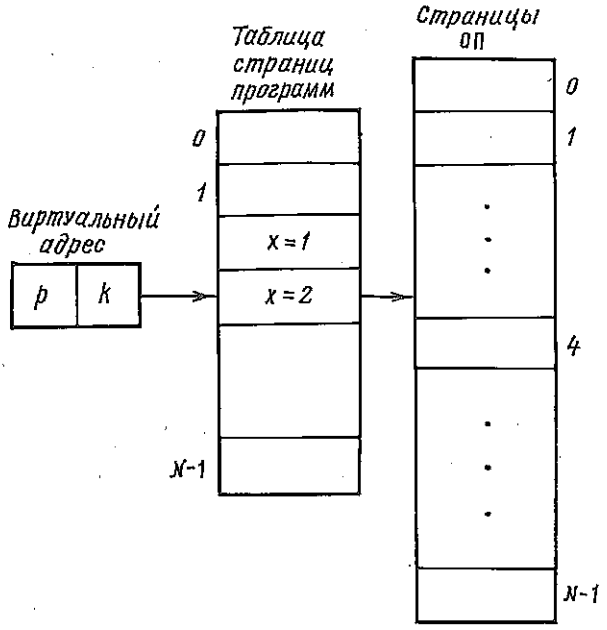


Рис. 7.2

Методы управления страницами. При страничной организации основная память делится на блоки фиксированного размера, а программа пользователя — на такие же блоки, называемые страницами. При этом возможно различное соотношение между логическим и физическим адресным пространством. При их равенстве увеличивается эффективность использования памяти. При большем логическом пространстве реализуется виртуальная память (память, реализуемая скрытым от пользователя механизмом отображения логических адресов на реальную оперативную и внешнюю память). Рассмотрим управление страницами в последнем случае.

При страничной организации различают виртуальный и физический адреса. Все ссылки к виртуальному пространству должны быть преобразованы в физический адрес памяти. Для этого используется идентификатор страницы и смещение внутри нее. Каждый виртуальный адрес представляет собой пару (p, k), где p — номер страницы процесса, k — индекс в странице. Для преобразования виртуального адреса в физический используется поиск в таблице страниц идентификатора x для основной памяти. Реальный адрес образуется умножением x на размер страницы и прибавлением к результату индекса k. Эта процедура выполняется блоком управления памяти, для чего в него загружается копия таблицы страниц процесса. На рис. 7.2 показано отображение страницы на основную память.

С каждым элементом таблицы связывается набор контрольных битов, наиболее важными из которых являются биты присутствия и активности. Первый бит указывает, находится ли страница в данный момент в основной памяти, а второй — на использование данной страницы процедурами обмена за последнее время.

В управлении страницами существенными считаются методы замены страниц, определяющие, какие страницы должны быть удалены из памяти, а также методы выборки, определяющие, какие страницы надо загрузить в память. Эти методы взаимно дополняют друг друга. Рассмотрим наиболее распространенные методы: 1) случайный выбор – удаляется случайная страница; 2) FIFO – удаляется самая старая страница, которая дольше всех находилась в основной памяти; 3) FIFO с приоритетом. Каждому процессу выделяется r страниц, в пределах которых применяется стратегия FIFO; 4) удаление по давности – удаляется страница, к которой дольше всех не было обращения, используется бит активности; 5) предписанные приоритеты. Страницам назначается приоритет, а система удаляет страницу с минимальным значением приоритета.

Реализация механизмов виртуальной памяти выполнена в 32-разрядных микроЭВМ (PS/2), а также в 16-разрядной (ЕС 1842, PC AT).

7.5. ФАЙЛОВАЯ СИСТЕМА

Основные понятия. Файлом является поименованная совокупность логических записей, однотипных по структуре и методу доступа, оформленных как единое целое. Файл состоит из записей, которые включают отдельные поля. Файлы выполняют долговременное хранение данных для процессов. Физическая организация данных иногда не устраивает пользователей, поэтому файл как логическое понятие может по-разному рассматриваться каждой программой. Файловая система представляет различные услуги для удобной работы пользователей, обеспечивает защиту данных, организует разделение данных.

Для работы с файловой системой выполняется ряд функций как логических (установка имени), так и физических (обмен блоков информации). Эти функции образуют иерархию уровней системы управления файлами. Каждый из уровней реализует свои возможности, используя функции нижележащего уровня. Работа физических устройств ввода-вывода координируется системой ввода-вывода, процессы которой считывают и записывают блоки (сектора) информации при обмене между основной и внешней памятью. Для работы с файлом используется блок управления файлом (БУФ), включающий адрес файла, его длину, перечень сегментов, диспозицию и т.д. Связь уникального имени файла с БУФ и реализацию команд расчленения файла на сегменты выполняет базовая подсистема. Уникальное имя файла определяется с помощью логической подсистемы по символическому имени, с которым работает пользователь. Подсистема выполняет команды открытия и закрытия, чтения и записи файлов.

Для реализации логических методов доступа к данным, отличных от их физического размещения (доступ к записям по значению ключа и др.), используется подсистема методов доступа.

Расширением системы управления файлами является система управления базой данных (СУБД). Разные программы могут потребовать от файлов, содержащих одинаковую информацию, различной логической структуры. Но дублировать информацию нецелесообразно, поэтому СУБД обеспечивает гибкий программный интерфейс для организации доступа к физическим данным при различных логических структурах.

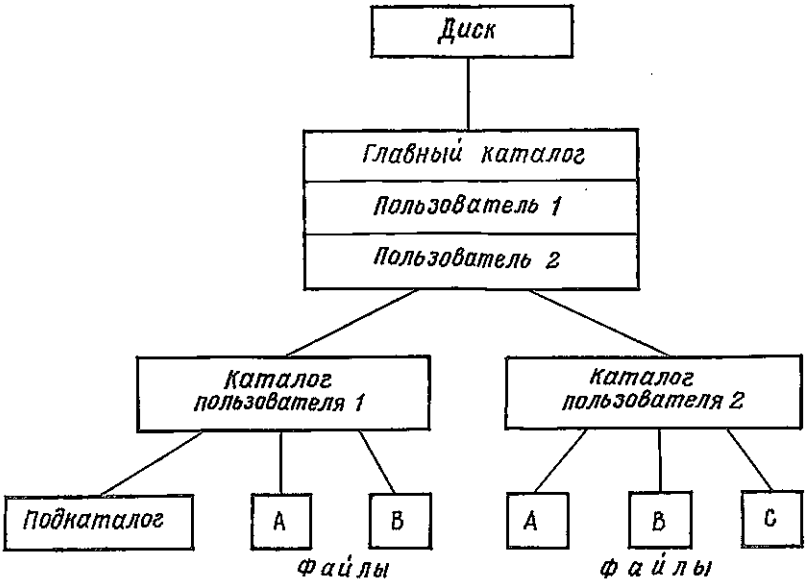


Рис. 7.3

Структура файловой системы. Наибольшее распространение для файловых истем микроЭВМ получила иерархическая организация файлов (рис. 7.3). Iверху расположен главный справочник, включающий справочники файлов. Справочник – список файлов, принадлежащий одному пользователю. Справочник и главный справочник, называемые каталогами, также являются файлами. Каталог пользователя может содержать подкаталоги, которые в свою очередь ключают как файлы, так и подкаталоги. В каталоге имена всех файлов являются уникальными.

Каждый файл имеет имя (от 1 до 8 символов), начинающееся с буквы. Для различия исходной, транслированной, загружаемой программы и файла каталога вводят тип файла, являющийся расширением имени. Например, DIR – файл каталога (подкаталога), EXE – выполняемая программа и т.д. *Расширение* файла называется его типом и отделяется от имени точкой.

Файл также отличается набором атрибутов, определяющих его тип, защиту, способ буферизации. Структурно различают последовательные файлы, непрерывные и сегментированные. Последовательные файлы имеют последовательную организацию секторов, непрерывные включают непрерывный блок памяти, сегментированные обеспечивают структуру с произвольным доступом.

ОС обеспечивает возможность обработки файлов несколькими процессами с соответствующей защитой. Система защиты файлов содержит: типы доступа к файлам, возможных пользователей и процедуры, осуществляющие защиту файлов. Тип доступа включает чтение, запись, обновление, удаление и изменение. Для различия пользователей может вводиться система паролей. Структура диска позволяет системе управления файлами оранизовать файлы тремя способами: последовательным, непрерывным и сегментированным.

Последовательный файл состоит из связанных последовательно блоков Для связи блоков используются указатели от одного блока к другому. В фай ле можно считывать либо предыдущий, либо следующий блоки (начиная от те кущего).

Непрерывный файл включает ряд физических блоков, которые располо жены в одной сплошной области диска. Размер непрерывного файла фиксиру ется при его создании. Доступ к записям в блоке осуществляется по адрес блока, который вычисляется по формуле A = LR * NR/LB (LR – длина запи си, NR – номер записи, LB – длина блока). Включение новой записи очен сложно, в среднем необходимо переместить 1/2 записей.

Сегментированный (индексный) файл состоит из последовательного ин декса, содержащего адреса соответствующих блоков данных. Индекс строится как множество блоков указателей сегментов и может быть организован после довательным или непрерывным способом. В первом случае размер файла фик сирован, во втором неограничен. Для получения записи из такого файла тре буется два обращения к диску: сначала считывается индексный блок, из кото рого извлекается адрес блока данных, а затем сам блок.

Структура управления файлами. Возможны три подхода для управления файлами: с ожиданием выполнения запроса результатов обращения к файлу; с параллельным выполнением процесса и обращения к файлу; комбинирован ный – одни функции выполняются параллельно, а другие ожидаются запраши вающим процессом. Наибольшее распространение получил второй подход (во многих ОС микроЭВМ).

Управление файлами существует как независимый прикладной процесс ОС. Он создается и инициируется при загрузке системы. В системе с большим числом устройств ввода-вывода работают несколько процессов.

Пользовательский процесс запрашивает функцию управления файлами через командный интерфейс ОС. Каждый запрос преобразуется в элемент системной очереди (ЭСО), который помещается в общую очередь процесса управления файлами (ПУФ). ПУФ выбирает элементы системной очереди и выполняет функции, содержащиеся внутри них, либо путем вызова нужной подпрограммы, либо породив подпроцесс для исполнения запроса. По завершении функции управления файлами результат посылается вызвавшему процессу.

В ЭСО содержится вся необходимая информация для выполнения функции управления файлами: код функции, адрес БУП вызывающего процесса, код события завершения функции, размер ЭСО, тип файла, число и размер выделяемых буферов, индекс карты памяти пользователя, аргументы запроса. Размер ЭСО меняется в зависимости от типа запрашиваемой функции.

ПУФ работает следующим образом. Он запрашивает первый ЭСО из своей очереди, выбирает требуемые значения из него и ищет адрес функции. Затем запрашивает загрузку найденной программы, вызывает функцию как подпрограмму. Функция в процессе выполнения устанавливает адрес события. По возвращении в основную программу результат посылается в программу пользователя, ЭСО возвращается в пул памяти, а ПУФ переходит к следующему элементу своей очереди. В течение обработки запроса ПУФ требуется рабочая память, которая выделяется в блоке данных процесса для сохранения реентерабельности процедур управления файлами. ПУФ для поиска адреса запрашиваемой функции обращается к таблице управления адресов всех модулей

управления файлами. В процессе работы ПУФ может использовать программы инициализации ЭСО, завершения ЭСО, обработки ошибок при инициализации ЭСО. Кроме того, применяются программы предварительной обработки функций управления файлами, которые выполняют: 1) распределение блока ЭСО; 2) инициирование основной информации в ЭСО; 3) копирование аргументов для функции в ЭСО из стека процесса; 4) постановку блока ЭСО в очередь ПУФ; 5) при необходимости обработку ошибок с освобождением блока ЭСО.

Функции управления файлами. Число функций и их действия зависят от среды, для которой проектируется система. Минимальный набор функций управления файлами следующий: СОЗДАТЬ – добавляется новый файл в системный каталог; УДАЛИТЬ – удаляется файл из системного каталога; ПЕРЕИМЕНОВАТЬ – изменяется имя файла в системном каталоге; АТРИБУТЫ ФАЙЛА – установить атрибуты файла; ОТКРЫТЬ – создать логический путь от пользователя к файлу или устройству; ЗАКРЫТЬ – закрыть логический путь между пользователем и файлом; ЧИТАТЬ КАТ – прочитать информацию каталога о данном файле; СОЗДАТЬ СВЯЗЬ – создать логический путь от одного файла к другому; УДАЛИТЬ СВЯЗЬ – удалить логический путь к файлу; КОПИРОВАТЬ – копирование вводного файла в указанный выводной; ПЕЧАТЬ – распечатать содержимое файла на указанном терминале.

Создание файла включает распределение пространства под файл и спецификацию характеристик файла. Удаление файла – это действие, посредством которого система по желанию пользователя получает обратно дисковое пространство. Удаление связи – уничтожение логического пути к файлу-цели. Прежде чем файл удаляется, он должен быть закрыт всеми процессами. Процедуры удаления различны для разных типов файлов.

Программа переименования проверяет список активных файлов, системный каталог на наличие файла с новым именем. Если такой файл найден, то возвращается ошибка. При переименовании файл должен быть закрыт.

Прежде чем процесс может выполнить операцию ввода-вывода с файлом или устройством, необходимо установить канал связи, по которому будут перемещаться данные. Этот канал представляет собой логическое соединение между блоком управления каналом и БУФ. Процедура открытия включает проверку номера канала, его связи с файлом и наличия ошибок.

При выполнении закрытия файла разрывается его логическая связь с каналом, а в каталоге обновляются соответствующие параметры файла.

7.6. АППАРАТНАЯ ПОДДЕРЖКА ФУНКЦИЙ ОПЕРАЦИОННОЙ СИСТЕМЫ

Управление задачами. В связи со значительными успехами в области микроэлектроники в последнее время появляются ОС, в которых часть функций реализуется аппаратно либо микропрограммно. Рассмотрим функции такой поддержки на примере БИС 80130, которая может совместно работать с МП 1810ВМ86. Микросхема обеспечивает следующие возможности: управление задачами, прерываниями, синхронизацию и обмен сообщениями, управление памятью. Схема работает с различными типами данных – заданиями, задачами, сегментами, почтовыми ящиками. Рассмотрим основные функции процессора ОС.

Каждое задание является средой, в которой выполняются прикладны программы. В состав задания входит одна или несколько задач (программ Задания разделяют память на пулы. Каждый пул содержит: область памя в которой ОС размещает информацию о состоянии задачи, и область для зад ОС обеспечивает также несколько задач в задании, управляя ресурсами ка дой из них. Задача соответствует процессу, поэтому в дальнейшем будем п пользовать термин процесс. Когда процесс создается, ему выделяется памя под данные, стек, а также определяется его приоритет, являющийся целы числом от 0 до 255. Причем приоритеты до 128 назначаются процессам, рабо ющим с прерываниями.

Процессор ОС распределяет и планирует процессорное время между пр цессами на основании приоритета. Когда готов процесс с более высоким при ритетом, он получает управление процессора. При этом сохраняется состоян процесса с более низким приоритетом в его образе, загружаются регистры М из образа процесса с более высоким приоритетом, управление передается н вому процессу. Процесс будет выполняться до тех пор, пока не произойд прерывание. Он не потребует недоступных в данный момент ресурсов или создаст ресурсы, необходимые процессу с более высоким приоритетом, ожид ющему их.

Процесс может быть переведен в состояние готовности после получени сообщения, управления, прерывания. Для организации временных останово используется таймер, по которому процесс может устанавливать промежуто времени для ожидания события, ресурсов или сообщения в почтовом ящик (области памяти для обмена информацией между процессами).

Для работы с процессами обеспечиваются следующие примитивы работы Создать задание (выделяются системные ресурсы и создается среда для вы полнения задач); Создать задачу (создается образ задачи и устанавливаютс ее атрибуты); Удалить задачу (уничтожается образ задачи, прекращается обр ботка ее команд, перераспределяются ресурсы и удаляется стек задачи); О ложить задачу (откладывается указанная задача или увеличивается глубина о ложенности с установкой состояния Отложена); Возобновить задачу (вычит ется 1 из глубины отложенности, при равенстве нулю последней устанавлив ется состояние Готова или Приостановлена (если было Приостановлена – О ложена)); Приостановить задачу (перевод задачи в состояние Приостановлен на заданное число единиц времени); Изменить приоритет (изменить или уст новить приоритет задачи).

Управление прерываниями. Подсистема прерываний обеспечивает два ме ханизма их обработки при использовании Обработчика прерывания и Задач прерывания. Обработчик прерываний блокирует все маскируемые прерыва ния, работает со следующими функциями обработки: Блокировать, Принят прерывание, Выдать прерывание, Принять уровень, Возобновить прерывание Задача прерывания дает возможность работы всех функций. Взаимодействи между Обработчиком прерывания и Задачей прерывания одного уровня регу лируется примитивами Выдать и Ожидать прерывание. Взаимодействие явля ется асинхронным. После того как Обработчик выдает Задаче прерывания, о начинает работу со следующим прерыванием, помещая его в очередь. Разме этой очереди задается в примитиве Установить прерывание. Когда примити Задача прерывания заканчивает работу, он обращается к очереди, использу

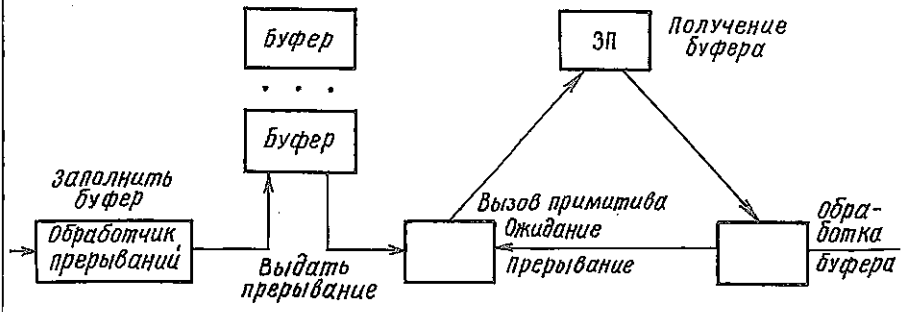


Рис. 7.4

примитив Ожидать прерывание. Если очередь прерываний пуста, Задача прерывания переходит в состояние Отложена. В микросхеме поддерживаются следующие примитивы управления прерываниями: Блокировать–Разблокировать (прерывание – блокируется/разблокируется) – внешний уровень прерываний; Принять прерывание – обработчику прерываний выделяется сегмент для данных; Возобновить прерывание – используется обработчиком, когда он не возбуждает задачи прерывания (ЗП); Принять (Сбросить) прерывание – устанавливает и отменяет величину уровня прерывания; Выдать (Ожидать) прерывание – используется для синхронизации работы Обработчика и Задачи прерывания.

На рис. 7.4 показан пример обработки прерываний с использованием буферов.

Управление памятью. Каждому заданию для его выполнения выделяется память Администратором свободной памяти. Как только в задании создаются задачи, почтовые ящики и семафоры, для них из ресурсов задания выделяется память. Для выделения памяти не требуется отдельного вызова программ, поскольку в следующих примитивах Администратор памяти вызывается по умолчанию: Создать задание, Удалить задание, Создать (Удалить) почтовый ящик, Создать (Удалить) семафор. Для явного указания размеров запрашиваемой памяти используются примитивы: Создать сегмент (выделяется сегмент указанной длины из пула Задания), Удалить сегмент (освобождается память, занятая сегментом и возвращается в пул Задания).

Связь и синхронизация процессов. Процессор ОС имеет встроенные средства, позволяющие задачам выполнять синхронизацию и обмениваться информацией на основе механизма почтовых ящиков. Каждый почтовый ящик состоит из двух связанных очередей Задачи и Сообщения. Обеспечение связи и синхронизации реализуется следующими примитивами: Создать (Удалить) почтовый ящик – создается (удаляется) средство обмена; Получить (Послать) сообщение – задача посылает (получает) сообщение через почтовый ящик. Если ящик пуст, Задача может либо продолжаться, либо перейти в состояние ожидания.

Взаимное исключение. В данной ОС взаимное исключение поддерживается с помощью механизма семафоров, использующих системные данные Регион. Регион представляется очередью задач, ждущих обращение к разделяемому ресурсу. Для работы с такими ресурсами и взаимоисключающими данными используются четыре примитива взаимного исключения:

Создать (Удалить) регион – создать (удалить) структуру данных;
Передать (Получить) управление – покинуть (получить доступ) регион;
Запросить управление – требует доступа к региону, не ожидая его освобождения.

В процессе работы возможно динамическое изменение приоритета задач во время их нахождения в регионе. Это происходит, если задача с более высоким приоритетом запрашивает ресурс, защищенный регионом (с помощью примитива Получить управление), в котором находится задача с меньшим приоритетом. Последняя автоматически поднимает свой приоритет до запрашивающей задачи, до освобождения региона, а затем использует примитив Передать управление.

Работа ОС. Если в состав микроЭВМ включается микросхема 80130, система расширяется новыми объектами: Заданиями, Задачами, Сегментами, Почтовыми ящиками и Регионами. В процессе работы эти объекты могут создаваться и уничтожаться. Когда объект создан, система возвращает его той за-

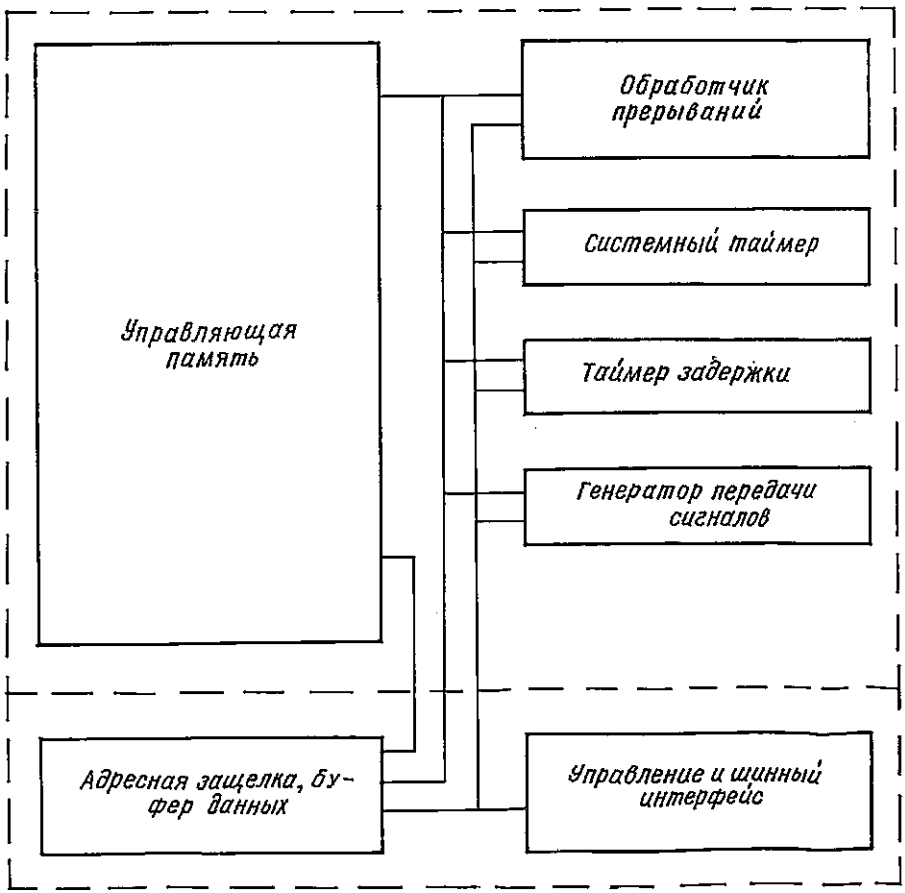


Рис. 7.5

даче, которая его запрашивала. В дальнейшем на имя объекта производятся ссылки. Это имя является исключительным средством для доступа к диапазону адресов МП размером 1 М байт. Например, для обращения к объекту Сегмент производится загрузка 16-го адреса в один из сегментных регистров процессора. Затем этот регистр используется для адресации в любом месте памяти. Аналогично при создании объекта Задача указывается приоритет, сегмент для расположения задачи, ее стек и начальный адрес программы задачи.

Чтобы в полной мере использовать преимущества многопроцессорной работы, ОС для каждой прикладной программы предоставляет тип задания, в рамках которого создаются процессы (задачи) и выделяется память. Для синхронизации и обмена задачи используют механизм примитивов работы с почтовыми ящиками, а для взаимного исключения – семафоры (регионы).

Структура кристалла ОС. Структурная схема поддержки ОС состоит из узла ОС и узла управления (рис. 7.5). Узел ОС включает средства для поддержки функций ядра (примитивы, находящиеся в памяти), таймеры для планирования задач, устройство задержек и контроллер прерываний для управления прерываниями. Узел управления содержит адресную защелку и буферный регистр данных для принятия адреса и данных по шине МП 1810ВМ86, а также блок управления. Последний дешифрирует сигналы состояния МП (S_0–S_2) и вырабатывает сигналы управления ОС и шиной MULTIBUS . К МП 1810ВМ86 БИС ОС подключается через шины адреса и данных, синхронизации, прерываний и выбора кристалла.

Рекомендуемая литература: 7, 9.

8. АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

8.1. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

Исходным материалом для создания имитационной модели является содержательное описание моделируемой системы, сделанное на основе детального изучения ее характеристик.

Разработка имитационной модели осуществляется в несколько этапов.

На *первом* этапе элементы и процессы описываются словесно. Приводятся исходные данные — начальные условия, характеристики и параметры; формулируется цель моделирования, которая может быть определена совокупностью искомых величин с указанием требуемой точности. На основе содержательного описания процессора составляется формализованная схема протекающих в нем процессов. Эта схема представляется совокупностью операторов и условий, при которых выполняется каждый оператор. Формализованная схема содержит как математическое описание системы, так и словесное, а также таблицы и графики.

На *втором* этапе на базе формализованной схемы исследуемых процессов осуществляется представление элементов и процессов с помощью математической символики в виде алгебраических уравнений, неравенств. Процессы, которые не поддаются аналитическому описанию, переводятся в другие формы, приемлемые для анализа численными методами (аппроксимирующие выражения, таблицы и т.д.).

На этапе математического описания необходимо обратить внимание на уменьшение размерности разрабатываемой математической модели: выделить малосущественные связи, упростить или совсем исключить их описание. Вместе с тем связи, оказывающие решающее влияние на адекватность модели по искомым характеристикам, должны быть описаны наиболее полно.

Третьим этапом разработки является подготовка математической модели процессора к моделированию на ЭВМ. Приводится оценка результата функционирования системы, выделяются ее управляемые и неуправляемые переменные и параметры: $S = f(x, y)$, где S — оценка результата функционирования процессора; x — управляемые переменные и параметры; y — неуправляемые переменные и параметры процессора. На данном этапе составляются схемы алгоритмов исследования модели на ЭВМ.

Четвертый этап разработки предусматривает подготовку комплекса программ для ЭВМ, включающего все задачи исследования системы.

Испытание, проверка, отладка модели выполняются на *пятом* этапе разработки имитационной модели.

При определении состава моделирующих алгоритмов имеют место две группы алгоритмов — функционально необходимые и организующие. Функционально необходимые алгоритмы описывают процессы преобразования ин-

формации и состояние моделируемого объекта. Каждый алгоритм выполняет соответствующую функцию в составе модели и определяет организацию вычислительного процесса, обработку результатов моделирования, контроль выполнения программы, формирование выходных документов.

Требования, предъявляемые к программному обеспечению имитационных моделей, обусловливают целесообразность при разработке проекта программ использования принципов структурного программирования, в соответствии с которыми проектирование, написание и отладка программ выполняются по определенным правилам. В основу положен метод программирования сверху вниз, согласно которому процесс программирования начинается с разработки общего подхода к построению программы. Далее определяется иерархическая структура программы как совокупность подчиненных модулей.

Формирование модулей программы осуществляется при реализации модулем одной определенной законченной функции, наличии у каждого модуля одного входа и выхода, а также связи между модулями через переменные, именуемые одинаково внутри всех исходных текстов, выбора объема входного или объектного текстов программы при условии удобства работы с программой модуля с учетом характеристик ввода-вывода.

Для каждого модуля задаются следующие условия: модуль возвращает управление той программе, которая его вызвала; модуль может вызвать другой модуль уровнем ниже.

Снижение трудоемкости моделирования сравнительно простых систем достигается применением специализированных языков (СИМПАК, СЛЕНГ, СИМУЛА, АЛСИМ, НЕДИС и др.). Однако при исследовании более сложных систем этот способ малоэффективен. Поэтому использование специализированных языков с целью снижения трудоемкости моделирования должно быть дополнено другими методами и средствами подготовки задач и общения с ЭВМ.

Один из таких подходов — модульный принцип, в рамках которого все элементы системы описываются единообразной стандартной математической схемой — модулем, что позволяет получить универсальную программу, используемую для имитации различных систем определенного типа. Состав конкретных систем, параметры ее элементов, структуры сопряжения последних учитываются в исходных данных без изменения вида универсальной программы имитации.

Ввод исходных данных в ЭВМ, размещение данных на внешних носителях, преобразование описаний элементов конкретных систем и схем сопряжений реальных элементов к стандартному виду, настройка универсальной модели на реальный объект, моделирование реальных систем и анализ полученных результатов осуществляются с помощью пакета программ моделирования (ППМ).

Имитацию процесса функционирования исследуемого процессора можно представить как совокупность имитации функционирования элемента процессора, имитации взаимодействия между элементами и управления очередностью событий в процессоре.

Имитация функционирования элемента процессора сводится к настройке модуля на конкретный элемент с последующей реализацией на ЭВМ конкретной модели элемента и включает поиск во внешней памяти соответствующей

строки и ввод в оперативную память параметров имитируемого элемента. Реализация полученной модели элемента на ЭВМ осуществляется подпрограммой определения состояния элемента в обусловленные моменты времени, например в момент поступления входного сигнала.

Взаимодействие между элементами имитируется определением адреса, содержанием сигнала и передачей сигнала в соответствии с адресом. Требуемая очередность событий в имитируемом процессоре обеспечивается автоматически специальным построением алгоритма моделирования.

Реализация программы имитации функционирования процессора осуществляется следующим образом. По заданным начальным состояниям элементов модель настраивается на элементы процессора в соответствии с опорными моментами модельного времени. Последовательность системных событий обеспечивается выбором наименьшего значения модельного времени и вызовом из архива модели строки, задающей соответствующий элемент. Информация этой строки расшифровывается в блоке имитации функционирования элемента системы, в результате чего определяются имена процедур, адреса параметров этих процедур и адреса компонентов состояния элементов. По полученным адресам процедуры вызываются из библиотеки стандартных процедур и реализуются.

После реализации процедур значения координат выходного сигнала записываются в регистр. Далее работает блок имитации взаимодействия элементов, используя информацию в архиве элементов. В этих данных содержатся номера элементов, получающих сигналы от данного элемента, и другие сведения, необходимые для формирования входных сигналов элементов по компонентам выходного сигнала, сформированного в регистре выходного сигнала блока имитации взаимодействия элементов.

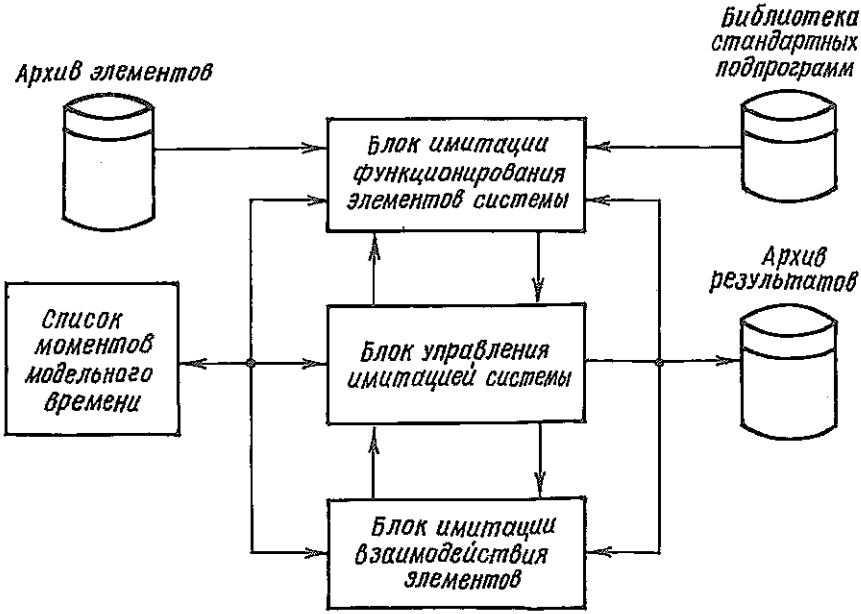


Рис. 8.1

После того как сформулированы воздействия на все элементы процессора, управление вновь передается управляющей программе для определения следующего момента модельного времени. Структура системы имитации приведена на рис. 8.1.

8.2. СЕТИ ПЕТРИ И МОДЕЛИРОВАНИЕ

Одним из математических графовых методов алгоритмического описания и моделирования параллельных процессов и взаимодействия между ними является теория сетей Петри.

Сетью Петри (PN) называется ориентированный двудольный(бихроматический) граф G = (P, T, K, S), в котором P и T – два множества вершин, причем P = $\{p_e\}$ – множество мест p_e; T = $\{t_j\}$ – множество переходов t_j (P ∩ T = = 0), соединенных между собой дугами K = P×T∪T×P по функциональным правилам S (рис. 8.2).

В общем случае функциональные правила $S = I_Q, I_R, m, b, \omega, L$, где $I_Q: K_Q \to Q \subseteq P \times T$, $I_R: K_R \to R \subseteq T \times P$ – функции инцидентности; $b: T \cup P \to L$ – функция обозначения; $\omega: K \to N (N=0, 1, 2, 3, ..., n)$ –весовая функция; $m: P \to N$ – разметочная функция; $L = X \cup Y$ – используемый алфавит.

Разложение функциональных правил S на функции и их реализация имеют важное практическое значение. С одной стороны, они являются основой построения различных интерпретаций сетей Петри, с другой – позволяют выявить особенности и основные характеристики сетей. В зависимости от функциональных правил S определяется и область применения тех или иных интерпретаций сетей Петри. Обычно в сетях Петри местам p_r соответствуют состояния z_r, а переходам t_j – операции или микрокоманды a_j.

В настоящее время известны оригинальные интерпретации сетей Петри и разрабатываются новые. Под интерпретацией сетей Петри понимают определенный набор понятий и правил, в основе которых лежит принцип построения размеченных графов, состоящих из двух типов вершин (вершин переходов T и вершин мест P), соединенных между собой дугами K по функциональным правилам S. В общем случае сети Петри PN = (P, T, K, S), как и любые графы, содержат структурные (P, T, K) и функциональные S элементы.

Известные модификации сетей Петри позволяют в основном моделировать параллельные процессы в программном (алгоритмическом) обеспечении вычислительных систем (на разных уровнях – от системного до микропрограммного). Традиционным является требование к отсутствию критических свойств в построенных моделях. В случае обнаружения критического свойства делается вывод о неработоспособности рассматриваемого алгоритма и выполняются действия по его изменению таким образом, чтобы во

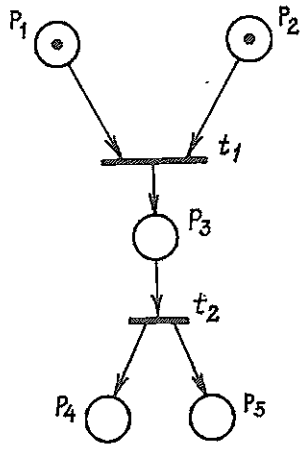


Рис. 8.2

вновь построенной адекватной модели критические свойства не были бы обнаружены. Основной недостаток такого подхода – трудоемкость процесса многократного построения моделей алгоритмов и анализ их работоспособности.

Методы, позволяющие проводить моделирование и проектирование аппаратных средств вычислительной техники с помощью сетей Петри, должны обязательно учитывать требование к бесконфликтности построенных моделей. При этом невозможно достичь требуемого уровня адекватности при моделировании аппаратных средств. Если используется обычная интерпретация компонентов сети Петри (вершины мест и метки отображают наличие или отсутствие условий, а переходы – выполнение действий по наступлению соответствующих условий), то наибольшая трудность возникает при отображении в модели процессов обработки управляющих сигналов. Так, в реально существующих технических устройствах возможны ситуации, когда предопределена возможность возникновения какой-либо критической ситуации. Например, два шинных формирователя могут быть подключены к одной внутренней магистрали. В случае правильной последовательности управляющих сигналов данные будут сниматься на внутреннюю магистраль только с одного формирователя, а в случае неправильной – с обоих. Очевидно, нельзя построить модель без критических свойств, если в самом моделируемом объекте заложена возможность возникновения "ошибочных" (критических) ситуаций при определенных условиях. Следовательно, в ходе имитационных экспериментов возникает необходимость построения моделей, с которыми при корректной последовательности внешних управляющих воздействий имитировалась бы правильная работа устройства, а при некорректной – диагностировалась и идентифицировалась бы часть устройства, в которой создавалась критическая ситуация. Результатом экспериментов могут быть как временные диаграммы срабатывания составляющих частей устройства, так и непосредственно цифровой результат – время цикла сети. Таким образом, на однократно сгенерированной модели системы возможно проведение отладки множества (программ) микропрограмм, интерпретирующих тот или иной вычислительный процесс.

Для построения таких моделей вводится формальное описание модифицированных (аппаратных) сетей Петри – двудольный граф APN = (P, T, K, Z), содержащий два непустых непересекающихся множества вершин мест $P = \{P_\varepsilon, S_\nu\}$ и вершин переходов $T = \{t_j, \tau_i\}$, связанных между собой дугами K по определенным функциональным правилам Z. Места в APN характеризуют наличие или отсутствие условий для выполнения или фактического завершения отработки какой-либо части параллельного алгоритма. Переходы моделируют выполнение отдельных элементарных частей алгоритма и включают два типа переходов – простые t_j и управляемые (макропереходы) τ_i. Макропереходы $\tau_i = \{\tau_i^1, \tau_i^2, \tau_i^3\}$ подразделяются на управляемые τ_i^1, декрементные τ_i^2 и прерываемые τ_i^3. Графически простые переходы t_j обозначаются утолщенной перпендикулярной к дуге линией, а макропереходы τ_i – двумя утолщенными линиями. Подмножество мест p_ε включает вершины с традиционными правилами построения в общих сетях Петри. Вершины аккумулирующих мест могут содержать метки ($m(S_\nu) > 0$) или не содержать их ($m(S_\nu) = 0$). Особенностью функционирования S_ν является то, что при срабатывании одного или нескольких простых переходов $t_j \in T$ разметка вершин не меняет-

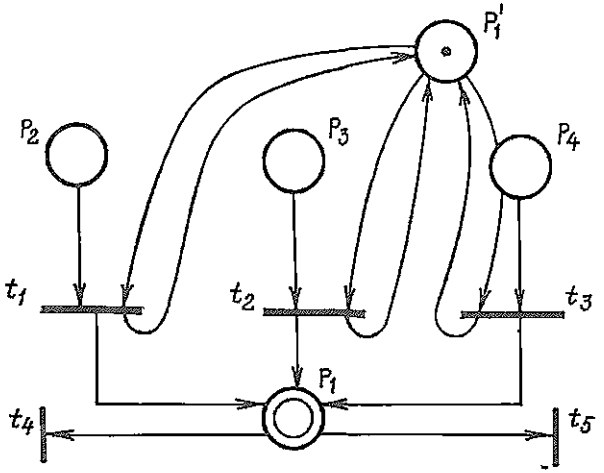


Рис. 8.3

ся ($m(S_v)$ = const). Графически макроместа S_v отображаются двойными кружками. Множество дуг K = k_1, k_2 включает подмножество простых дуг k_1 и подмножество дуг с переменным весом k_2. В оба подмножества входят дуги необходимых условий, имеющие те же свойства, что и дуги в PN и дуги дополнительных условий, отображающие условия срабатывания соответствующих переходов. Дуги дополнительных условий обозначаются направленными стрелками (рис. 8.3).

Для упрощения описания моделируемого объекта, сокращения объема и повышения адекватности модели в предлагаемый формальный аппарат вводятся понятия счетчика и очереди.

В рамках модели любому переходу t_j (или группе переходов) может быть поставлен в соответствие счетчик, увеличивающийся на 1 каждый раз при срабатывании любого перехода t_j. При достижении заранее заданного максимального значения счетчика вырабатывается управляющий сигнал во входном U и/или выходном V алфавите, либо добавляется некоторое число меток в заранее определенное место p_{ϵ}.

Счетчик CN_i определяется так:

$$CN_i = < NS_i, NMAX_i, p_{\epsilon i}, NM_i D_i > , \; i = 1, 2, ..., N_c ,$$

где NS_i – текущее число срабатываний CN_i; $NMAX_i$ – максимальное число срабатываний i-го счетчика; $p_{\epsilon i}$ – место сети, в которое помещаются метки при $NS_i = NMAX_i$; NM_i – число меток, помещаемое в место $p_{\epsilon i}$ при $NS_i = NMAX_i$; D_i – генерируемый управляющий сигнал; N_c – число счетчиков данной модели.

Например, входящее в модель описание счетчика $CN_1 = < 2,5, p_9, 3, 0 >$ говорит о том, что в соответствии с ним после пяти срабатываний переходов вместо p_9 будет помещено три метки, и управляющий сигнал не выработается (произошло два таких срабатывания).

Любому переходу сети Петри может также соответствовать входная и/или выходная списковая структура. Выходная списковая структура пополняется после каждого срабатывания перехода.

Списковая структура

$$QU_i = < LT_i , LMAX_i , MS_i > , \ i = 1, 2, ..., N_a \ ,$$

где N_a – число списков в модели; LT_i – длина списковой структуры; $LMAX_i$ – максимальная длина списковой структуры; MS_i – состав структуры QU_i, представляет собой множество $\{< KS_{ij} , SA_{ij} , DA_{ij} , LS_{ij} , PA_{ij} >\}$, $j = 1, 2, ..., LT_i$; KS_j – код j-го элемента QU_i; SA_j – код источника сообщения; DA_{ij} – код приемника сообщения; LS_{ij} – длина сообщения; PA_{ij} – прочие параметры j-го элемента QU_i.

Введение механизма списковых структур позволяет имитировать в модели генерацию, передачу и обработку команд, данных, сигналов. Кроме того, в каком-либо из списков может содержаться отлаживаемая микропрограмма.

Для реализации дополнительных средств в рамках модели создан сервисный глобальный макропереход ${}^{TS}T$, содержащий набор функциональных переходов, выполняющих действия: 1) ПОСТАВИТЬ В СПИСОК ПЕРВЫМ (сообщение, сгенерированное переходом, поставить первым в выходной список); 2) ПОСТАВИТЬ В СПИСОК ПОСЛЕДНИМ (сообщение, сгенерированное переходом, поставить последним в выходной список); 3) ВЗЯТЬ ИЗ СПИСКА (взять сообщение из соответствующего входного списка); 4) НАЧАТЬ/ПРЕКРАТИТЬ ПРЕРЫВАНИЕ (обработка одного из прерываемых переходов ${}^{i}t_v$); 5) СЧЕТЧИК (обслуживает указанный счетчик, производит обработку указанных в нем условий); 6) ОПРЕДЕЛИТЬ КОНФИГУРАЦИЮ ВХОДНОГО (ВЫХОДНОГО) ВЕКТОРА (вычисляет и заносит в модель значения весовой функции ω для дуг с переменным весом); 7) ОПРЕДЕЛИТЬ ДЛИТЕЛЬНОСТЬ РАБОТЫ ПЕРЕХОДА (по заданным параметрам определяет и устанавливает в модели длительность срабатывания перехода); 8) ОПРЕДЕЛИТЬ ВЕКТОР X (управляет вектором дополнительных условий X).

В процессе моделирования можно выделить следующие состояния переходов: 1) рабочее (происходит моделирование процесса a_v, описанного переходом t_v); 2) возбужденное (выполняется условие запуска v-го перехода, определяемое функциями ω и r, но значение вектора X_v запрещает запуск перехода); 3) активизированное (сигнал ${}^{v}X_v$ разрешает запуск v-го перехода); 4) прерванное (переход ${}^{i}t_v$ выведен из рабочего состояния запуском перехода $t_j (j \neq v)$); 5) пассивное (переход не находится ни в одном из перечисленных состояний).

В любой момент модельного времени t^m информация о каждом переходе множества T может быть представлена в виде

$$TI_{t}^{m_v} = < X_{t}^{m_v} , MP_{t}^{m_v} , TC_{t}^{m_v} , B_{t}^{m_v} , LQI_{t}^{m_v} ,$$

$$LQO_{t}^{m_v} , LCU_{t}^{m_v} >,$$

где X_{m_v} – состояние разрешающего сигнала вектора X; MP_{m_v} – состояние перехода; TC_{m_v} – текущее число срабатываний; B_{m_v} – временной интервал до окончания срабатывания перехода; LQI_{m_v} – длина входного списка; LQO_{m_v} – длина выходного списка; LCU_{m_v} – состояние счетчика, соответствующего данному переходу.

В любом сложном моделируемом объекте можно выделить систему иерархических уровней, которая представляется в виде взаимосвязанных подсистем нескольких уровней, каждая из которых построена по единому принципу (рис. 8.4). Включение или исключение из модели подсистем различных уровней детализации позволяет проводить имитационный эксперимент с необходимой точностью описания отдельных фрагментов системы.

В основе многоуправляемых моделей лежит понятие статической иерархической структуры общего вида, представляющей собой упорядоченное множество FS элементов модели J_i, $i = 1, ..., M$, каждый из которых охарактеризован степенью детализации N_i на модели:

$$FS = <<N_1, J_1>, <N_2, J_2>, ..., <N_i, J_i>, ..., <N_M, J_M>>.$$

Определив свойства и состояния элементов модели относительно временной координаты и взаимоотношений элементов внутри группы, подчиненной элементу верхнего уровня иерархии, получим многоуровневую модель:

$$FS = <<N_1, CL_1, J_1, R_1>, <N_2, CL_2, J_2, R_2>, ...,$$
$$<N_M, CL_M, W_M, R_M>>,$$

где CL_M – класс элемента модели; R_M – отношение в группе элементов одного уровня.

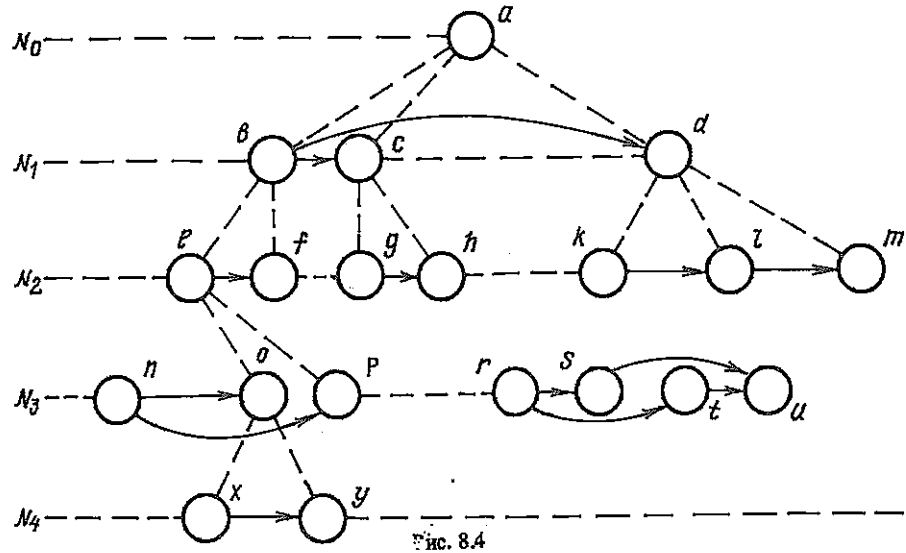


Рис. 8.4

В модели элементы группы низшего уровня локализованы в зоне дейс вия элементов высшего уровня. Применительно к модели, построенной с п мощью аппаратных сетей Петри, в качестве элементов модели J_i примем мн жество переходов T. Детализация элементов модели осуществляется путем з мены простых переходов t_ν на глобальные простые переходы ^{t}T.

Информационная многоуровневая модель системы на базе аппаратных с тей Петри может быть записана так:

$$SP = \{m, FS\},$$

где m – разметочный вектор; FS – многоуровневая структура элементс модели.

Каждый ν-й элемент структуры FS имеет вид

$$< J_\nu, CL_\nu, N_\nu, S_\nu, IT_\nu, {}^{ev}\vec{\mu}, {}^{av}\vec{\mu}, RR_\nu, \vec{\rho}_\nu, \varphi_\nu >,$$

где J_ν – порядковый номер элемента (перехода); CL_ν – класс (тип) элеме та; N_ν – номер уровня в иерархии модели; S_ν – функциональные правил перехода; IT_ν – правило определения продолжительности t_ν; ${}^{ev}\vec{\mu}$ – входно разметочный вектор; ${}^{av}\vec{\mu}$ – выходной разметочный вектор; RR_ν – приорите перехода; $\vec{\rho}_\nu$ – параметры перехода (необязательные); φ_ν – набор специал ных функций. В число ,необязательных параметров перехода входят: макси мальное число срабатываний, входная очередь, выходная очередь, счетчик, па раметры соответствующей случайной функции, число параллельных функцио

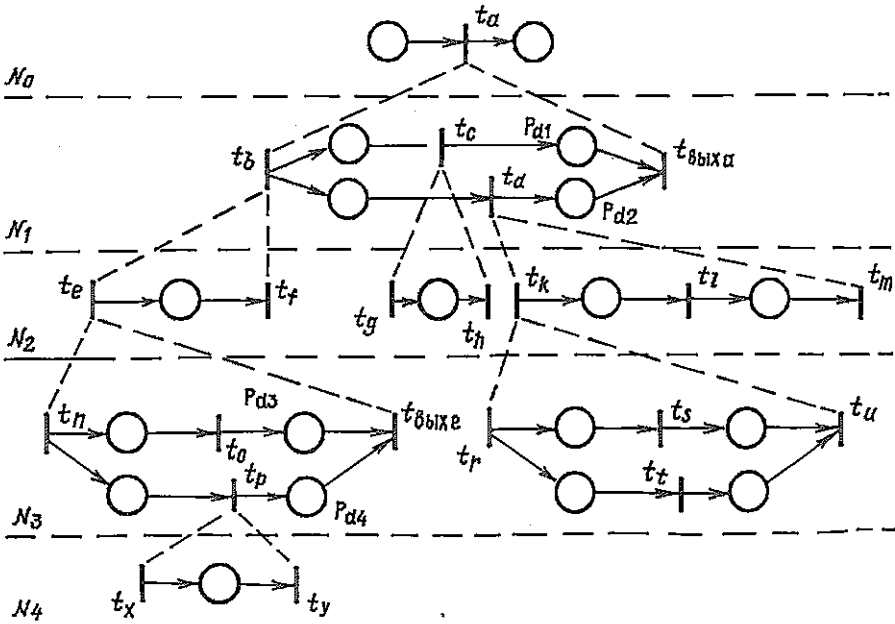


Рис. 8.5

нальных блоков, реализующих функции перехода, номера этапов моделирования, на которых производится обработка специальных функций.

Имитация функционирования подобной системы сводится к последовательной настройке моделирующих средств на конкретный элемент с реализацией полученной модели элемента и отображением результатов во временных координатах, что позволяет определить множество переходов τ сети APN в качестве множества W элементов модели. Если отношения композиции элементов модели определяются множествами мест P, дуг K и функциональными правилами S, то многоуровневой структуре, приведенной на рис. 8.4, можно поставить в соответствие сеть APN, изображенную на рис. 8.5.

Чтобы сети, локализованные переходами t_a и t_c, могли сохранять некритические свойства при детализации модели, в структуру дополнительно вводятся вершины мест $p_{d_1}, p_{d_2}, p_{d_3}, p_{d_4}$ и вершины переходов $t_{\text{вых}.a}$ и $t_{\text{вых}.e}$.

Представление в виде иерархических многоуровневых структур позволяет проводить моделирование объектов с требуемой степенью детализации отдельных фрагментов модели на разных этапах имитационного эксперимента при сохранении некритических свойств моделирующей сети.

8.3. СПЕЦИАЛЬНОЕ МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ПОСТРОЕНИЯ И АНАЛИЗА МОДЕЛЕЙ АППАРАТНЫХ СЕТЕЙ ПЕТРИ

При создании специализированного математического обеспечения (СМО) необходимо учитывать противоречия между универсальностью и специализацией программных средств, требования технологичности подготовки и решения задач, возможности организации диалога с ЭВМ и взаимодействия прикладных программ. Наиболее перспективным является построение открытого СМО с интерфейсом пользователя снизу. Основные функции системы моделирования следующие: трансляция описаний объектов с языка описаний моделей, интерпретация моделей во времени, организация интерфейсных связей с процедурами пользователя, оформление и обработка результатов моделирования. Общая схема организации такого СМО представлена на рис. 8.6.

К программным средствам системы относятся: редактор исходных моделей, транслятор моделей, модуль поиска критических ситуаций, интерпретатор моделей, редактор результатов, библиотека стандартных подпрограмм.

Приведенная структура программного обеспечения системы моделирования сетей Петри позволяет с незначительными затратами описывать и имитировать функционирование микропроцессорных систем. Наличие интерфейсных связей значительно снижает трудоемкость разработки программных средств пользователя при моделировании конкретных объектов.

Редактор исходных моделей выполняет функции по вводу, корректировке исходного текста модели. Транслятор системы предназначен для преобразования моделей, записанных на языке описания моделей на сетях Петри, в загрузочные модули фрагментов описания объекта. Основными функциями транслятора являются: ввод исходных данных трансляции, трансляция исходного текста в объектный код, вывод сообщения об обнаруженных ошибках, формирование листинга трансляции, формирование файла объектного кода модели, обработка описаний моделей агрегатного типа. Входные данные транс-

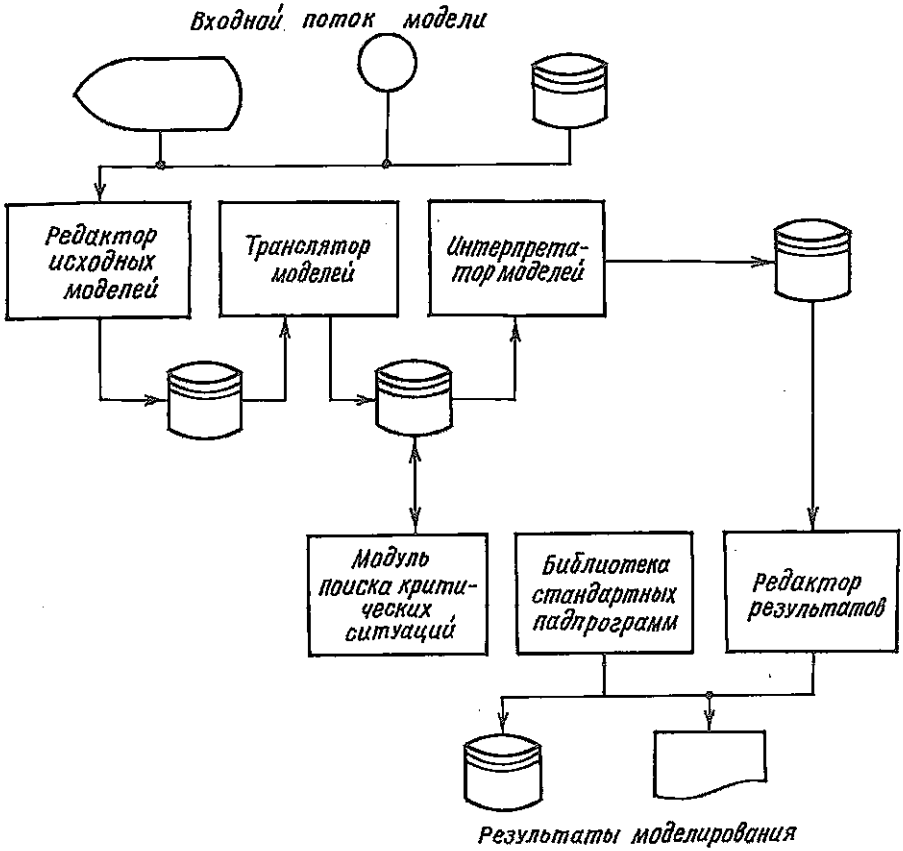


Рис. 8.6

лятора: режим трансляции, исходный текст модели на языке описания. Выходные данные: объектные коды моделей, листинг трансляции с диагностикой ошибок в тексте модели.

Синтаксис языка описания моделей на аппаратных сетях Петри может быть построен следующим образом:

```
< модель > : : = < список предложений >
< список предложений > : : = < предложение > |
                             < предложение > < список предложений >
< предложение > : : = < символ начала > < список выражений >
                      < символ конца > |
                      < символ начала > < список выражений >
                      < символ продолжения > < список выражений >
                      < символ конца >
< список выражений > : : = < выражение > | < выражение > < разделитель >
                           < список выражений >
```

< разделитель > : : = ␣ | < разделитель > ␣

< выражение > : : = ␣ < код команды 1 > < константа >
< код команды 2 > < тип длительности >
(< список временных параметров >) |
< код команды 3 > < имя файла > |
< код команды 4 > < имя элемента > |
< код команды 5 > < входной-выходной вектор > |
< код команды 6 > (< список параметров счетчика >)

< константа > : : = < цифра > | < константа > < цифра >

< список временных параметров > : : = < временной параметр 1 > ,
< временной параметр 2 >

< список параметров счетчика > : : = < константа > , < константа > ,
< константа >

< входной-выходной вектор > : : = < список дуг >

< список дуг > : : = < описание дуги > , < список дуг > | < описание дуги >|

< описание дуги > : : = < номер дуги > | < номер дуги > (< вес дуги >) |
< номер дуги > < аккумулирующий признак >
< номер дуги > : : [< ссылка на счетчик >] |
< номер дуги > < аккумулирующий признак >
(< вес дуги >) : : [< ссылка на очередь >]

< ссылка на очередь > : : = < номер очереди > , < емкость очереди >

< символ начала > : : = #

< символ продолжения > : : = *

< символ конца > : : = ;

< цифра > : : = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

< код команды 1 > : : = NT = |CF = |CL = |PR = |PC = |ET =

< код команды 2 > : : = TI =

< код команды 3 > : : = NF =

< код команды 4 > : : = NP =

< код команды 5 > : : = IV = |OV =

< код команды 6 > : : = CN =

< тип длительности > : : = NORM | FUAS | RAUN

< временной параметр > : : = действительная константа

< имя файла > : : = имя файла пользователя ОС RSX – 11 M

< имя элемента > : : = < символ > < имя элемента > | < символ >

< номер дуги > : : = константа

< вес дуги > : : = константа

< аккумулирующий признак > : : = A

< номер очереди > : : = константа

< емкость очереди > : : = константа

< символ > : : < цифра > | буквы русского алфавита | буквы латинского алфавита |

Признак определения длительности срабатывания перехода задается следующим образом: по нормальному закону – NORM; по закону Пуассона – PUAS; по равномерному закону – RAUN.

Набор и форматы команд определения параметров элементов модели приведены в табл. 8.1.

Таблица 8.1

Наименование параметра	Код команды	Формат	Примечание
Имя элемента	NM=	NM = *xx ... x*	Длина имени до 30 символов
Номер элемента	NT=	NT=*n*	$n \leqslant 999$
Класс элемента	KL=	KL=*n*	$n \leqslant 9$
Имя файла	NF=	NF=*xx ... x*	Имя файла – в соответствии со спецификацией файла пользователя OS RSX – 11 M
Длительность срабатывания	TI=	TI=*xxxx*(*a*, *b*)	*xxxx* = NORM / RAUN / PUAS
		TI=*n*	$0 \leqslant a \leqslant 9999.99999$ $0 \leqslant b \leqslant 9999.99999$
Число срабатываний	CL=	CF=*n*	$n \leqslant 99$
Номер процедуры пользователя	PC=	PC=*n*	$n \leqslant 20$
Номер этапа для подключения процедуры пользователя	ET=	ET=*n*	$n \leqslant 3$
Входной вектор	IV=	IV=*nA* (*m*) :[*k*, *l*,..., *n*1	*n* – номер входного (выходного) места; *A* – признак места аккумулятора;
Выходной вектор	OV=	OV= *nA* (*m*) :*k*, *l*,..., *n*1	*m* – вес дуги; *k* – номер очереди; *l* – емкость очереди
Уровень в иерархии	IE=	IE=*n*	$n \leqslant 10$
Ссылка	CN=	CN= (*n*1, *n*2 , *n*3)	*n*1 – номер счетчика; *n*2 – номер места; *n*3 – число меток
Приоритет	PR=	PR=*n*	$n \leqslant 10$

В процессе обработки исходного текста описания модели транслятор обнаруживает ряд ошибок. Сообщения об ошибках выводятся в файл FERR. MOA в виде

IER = zzIEL = xxxxIST = yyyyIVK = AAKPAR = BB ,

где IER – номер обнаруженной ошибки; IEL – номер элемента модели, в котором обнаружена ошибка; IST – номер строки исходной модели, в которой обнаружена ошибка; IVK – номер позиции, в которой обнаружена ошибка; KPAR – номер параметра исходного языка, при обработке которого обнаружена ошибка.

После обработки модели транслятор выдает на консоль оператора сообщение о количестве ошибок, обнаруженных в данной модели. Если ошибок нет, то созданный объектный код модели пригоден для обработки последующими компонентами СМО.

Компоновщик моделей предназначен для сборки в единое целое объектных кодов, полученных в результате работы транслятора. Компоновка моделей возможна в двух случаях: 1) при необходимости проведения эксперимента на модели более высокой степени детализации (реализуется подход, описанный выше); 2) когда в рамках модели выявлено несколько одинаковых подсетей, не удовлетворяющих требованиям к иерархическому представлению сетей Петри, однако желательно сократить объем описания за счет однократного отображения в рамках исходной модели одинаковых подсетей. Во втором случае выделяемые подсети описываются в качестве агрегатов, на которые производится ссылка из основной сети.

Например, на рис. 8.7, *а* показана сеть Петри, состоящая из двух одинаковых подсетей P_2, P_3, t_2, t_5 и P_4, P_5, t_3, t_4. Описание сети можно значительно сократить (рис. 8.7, *б*), выделив подсеть в агрегат. Первоначальный вид сети можно восстановить за счет ссылок (A, B, C), поставленных на разорванных дугах. Таким образом, фрагменты сетей (рис. 8,7, *в*) транслируются отдельно, а затем компоновщики на основании полученных объектных кодов формируют объектный код сети Петри (см. рис. 8.7, *а*).

Наиболее значительным элементом рассматриваемой СМО является интерпретатор моделей, выполняющий функции по отображению во временных координатах загрузочного кода модели и архивизации полученных результатов.

Загрузочный код модели, представляющий входные данные интерпретатора, имеет вид:

$$<<J_1, CL_1, N_1, T_1, {}^{1v}\vec{\mu}, {}^{a1}\vec{\mu}, PR_1, PC_1, KM_1, QI_1, LT_1, QO_1, LO_1, CO_1,$$

$$K_1, PA_1 >, ..., <J_v, CL_v, N_v, T_v, {}^{lv}\vec{\mu}, {}^{av}\vec{\mu}, PR_v, PC_v, KM_v, QI_v, LT_v,$$

$$QO_v, LO_v, CO_v, K_v, PA_v>, ..., <J_m, CL_m, N_m, T_m, {}^{lm}\vec{\mu}, {}^{am}\vec{\mu}, PR_m, PC_m,$$

$$KM_m, QI_m, LT_m, QO_m, LO_m, CO_m, K_m, PA_m >>,$$

где m – число переходов в модели; J_v – порядковый номер элемента; CL_v – класс перехода; N_v – номер уровня в иерархии модели; T_v – длительность (правило определения длительности) срабатывания t_v; ${}^{lv}\vec{\mu}$ – входной разметочный вектор; ${}^{av}\vec{\mu}$ – выходной разметочный вектор; PR_v – приоритет перехода t_v; PC_v – этап обработки процедуры пользователя; KM_v – максимальное число срабатываний t_v; OT_v – ссылка на входную очередь перехода; LT_v – максимальная емкость входной очереди; QO_v – ссылка на выходную очередь перехода; LO_v – максимальная емкость выходной очереди; K_v – число параллельных блоков, выполняющих функцию перехода; PA_v – ссылка на параметры случайной функции.

Упрощенная схема работы интерпретатора имеет следующий вид:

ш а г 1 – запуск программы;

ш а г 2 – ввод в интерпретатор загрузочного кода модели и создание массивов характеристик мест, очередей, счетчиков и т. д.;

ш а г 3 – создание служебного массива, содержащего набор пар $\{< P_\epsilon, t_v >\}$ и указывающего для места $P_\epsilon (1 \leqslant \epsilon \leqslant n)$ все переходы t_v, к которым направлены дуги от P_ϵ;

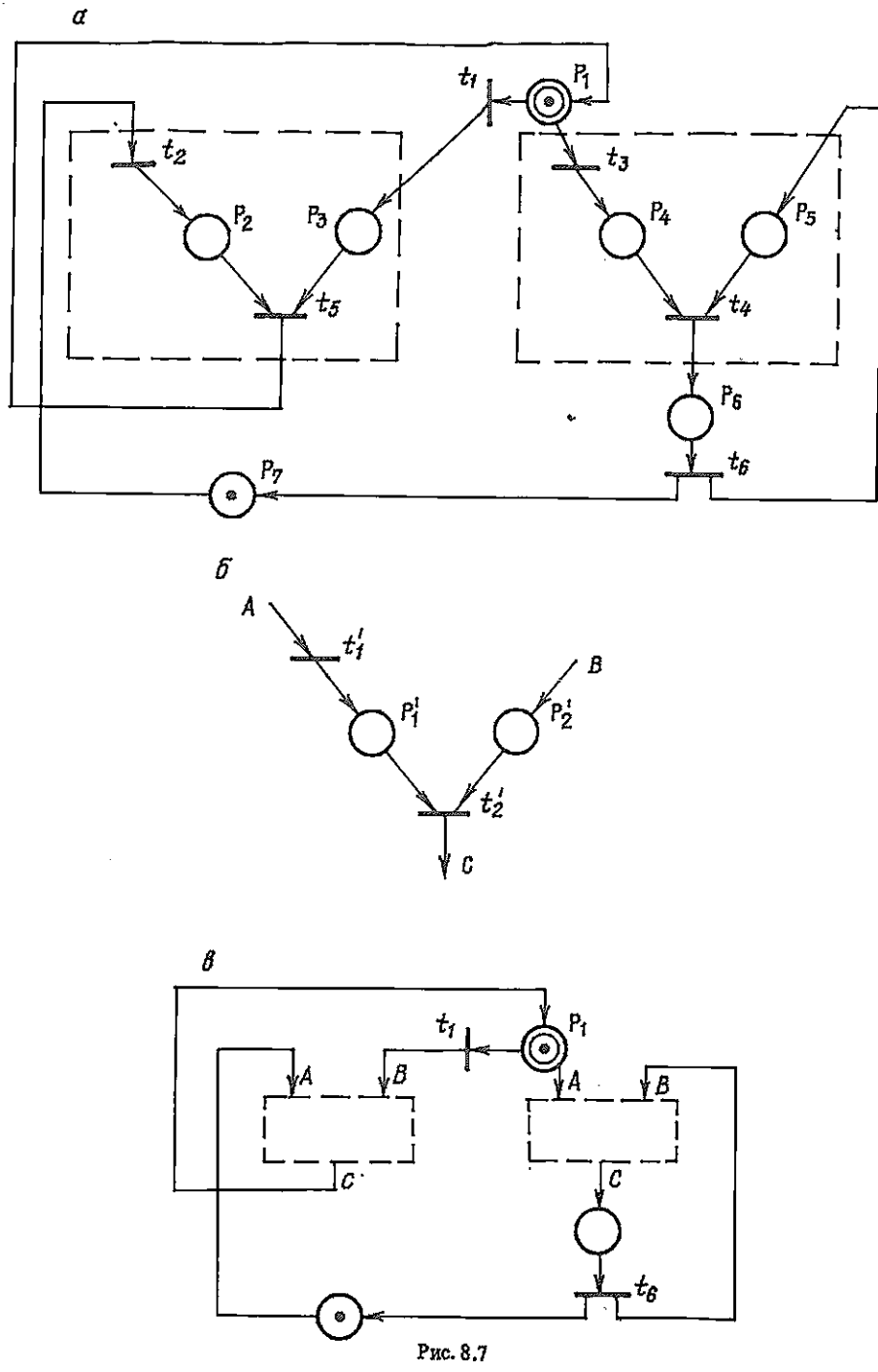

а
t_1
P_1
t_2
P_2
P_3
t_5
t_3
P_4
P_5
t_4
P_6
t_6
P_7
б
A
t_1'
B
P_1'
P_2'
t_2'
C
в
t_1
P_1
A
B
A
B
C
C
t_6

Рис. 8.7

ш а г 4 – создание инвертированного списка пар $\{<PR_\nu, J_\nu>\}$, позволяющего проводить опрос входных разметочных векторов ${}^{e\nu}\vec{\mu}$ в порядке убывания приоритетов переходов t_ν;

ш а г 5 – последовательный опрос переходов на выполнение условий задачи переходов $({}^{e\nu}\vec{\mu} \in \vec{m}^{(n)}, \nu_{xj} = 1)$. В момент модельного времени ТМ = 1 производится опрос всех переходов модели, в моменты модельного времени ТМ ≠ 1 опрашиваются только переходы множества АТ, содержащие активные и возбужденные переходы, а также переходы, к которым направлены дуги от мест, включающих в данный момент модельного времени метки. Введение множества АТ позволяет значительно сократить время, необходимое для определения множества VT переходов, запускаемых в текущий момент модельного времени. В результате выполнения шага 5 формируется множество VT и дополняется список активных и возбужденных переходов;

ш а г 6 – производится перевод в рабочее состояние переходов $t_\nu \in VT$, вычитаются ${}^{l\nu}\vec{\mu}$ из $\vec{m}^{(n)}$, определяются моменты окончания работы запускаемых событий, дополняется и сортируется календарь событий, представляющий собой набор пар чисел:

$$KALS = \{<J_\nu, TM^\nu>\},$$

где J_ν – номер перехода; TM^ν – момент окончания работы ν-го перехода. Значение TM^ν первой пары чисел принимается в качестве следующего момента модельного времени;

ш а г 7 – пассивизируются переходы t'_ν (для которых $TM^\nu = TM$), выходные разметочные векторы прибавляются к $\vec{m}^{(n)}$, производится обработка счетчиков, очередей, формируются выходные управляющие сигналы, дополняется множество АТ, изменяется календарь событий;

ш а г 8 – обработка условий завершения моделирования по времени окончания, числу срабатываний переходов, счетчиков; если ни одно из условий не выполняется, то осуществляется возврат к шагу 5;

ш а г 9 – останов.

Особенность данного интерпретатора – наличие интерфейсного блока, позволяющего пользователю описывать функционирование отдельных блоков моделируемого объекта средствами, которые не могут быть формализованы средствами математического аппарата сетей Петри. Процедуры пользователя могут обрабатываться на этапах опроса входных векторов, определения длительности и пассивизации перехода. При обращении к процедуре пользователя передаются сведения о переходе в соответствии с информационной моделью, а также о текущем состоянии t_ν в виде вектора:

$$TI^\nu_{TM} = < MP^\nu_{TM}, TC^\nu_{TM}, B^\nu_{TM}, KI^\nu_{TM}, KO^\nu_{TM}, KU^\nu_{TM}, X^\nu_{TM} >,$$

где MP^ν_{TM} – состояние перехода в момент модельного времени ТМ; TC^ν_{TM} – текущее число срабатываний; B^ν_{TM} – временной интервал до окончания срабатывания t_ν; KI^ν_{TM} – длина входной очереди; KO^ν_{TM} – длина выходной очереди; K^ν_{TM} – состояние счетчика; X^ν_{TM} – состояние управляющего сигнала.

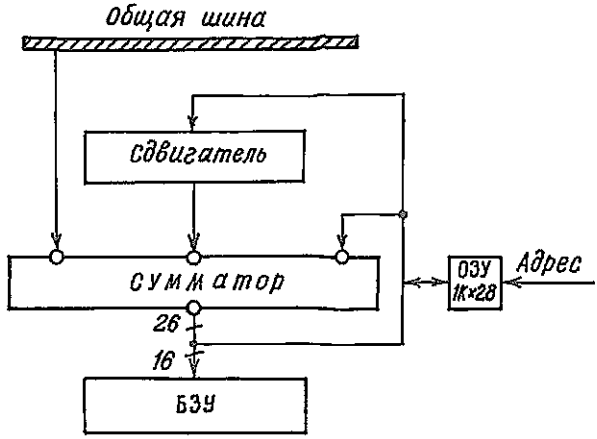


Рис. 8.8

Редактор предназначен для оформления результатов работы интерпретатора в виде графиков, таблиц, временных диаграмм работы системы в требуемых масштабах времени.

В качестве примера рассмотрим описание с помощью формализованного аппарата сетей Петри операционного автомата блока усреднения спектроанализатора, реализованного на секционированных микропроцессорах, функциональная схема которого показана на рис. 8.8.

Разряды регистра микрокоманды представлены в табл. 8.2. Каждый цикл работы микрокоманды состоит из трех микроциклов: φ_1 , φ_2 , φ_3 . Модель блока микропрограммного управления представлена на рис. 8.9. Каждому из

Таблица 8.2

Наименование разряда регистра управления	Управляющий сигнал	
	обозначение	длительность
Чтение-запись ОЗУ 1К × 28	$\overline{W}$/R ОЗУ	φ_2
Выбор ОЗУ	$\overline{CS}$ ОЗУ	φ_2
Адрес ОЗУ 1К × 28	$\overline{Адр}$	$\varphi_1, \varphi_2, \varphi_3$
Код сдвига сдвигателя	КС	$\varphi_1, \varphi_2, \varphi_3$
Строб сумматора	STBSM	φ_1
Режим сложение–вычитание сумматора	+/–	$\varphi_1, \varphi_2, \varphi_3$
Отключение выхода сумматора	$\overline{ED5}$	$\varphi_1, \varphi_2, \varphi_3$
Отключение входа В сумматора	$\overline{EDB}$	$\varphi_1, \varphi_2, \varphi_3$
Адрес БЗУ	$\overline{Адр. БЗУ}$	$\varphi_1, \varphi_2, \varphi_3$
Чтение-запись БЗУ	$\overline{W}$/R БЗУ	φ_2
Выбор БЗУ	$\overline{CS}$ БЗУ	φ_2

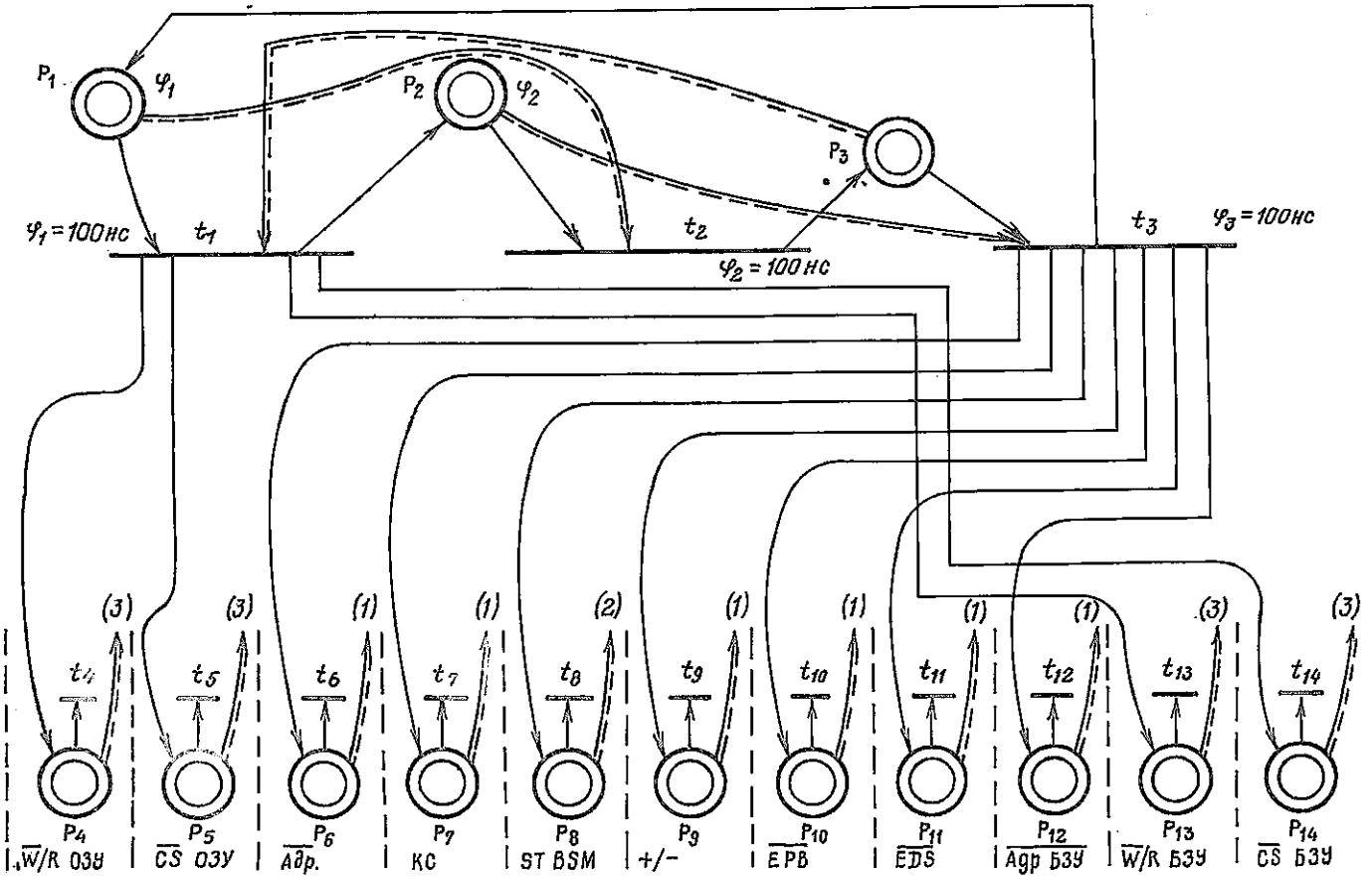

P1
φ1
P2
φ2
P3
φ1 = 100 нс
t1
t2
φ2 = 100 нс
t3
φ3 = 100 нс
(3)
(3)
(1)
(1)
(2)
(1)
(1)
(1)
(1)
(3)
(3)
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13
t14
P4
W/R ОЗУ
P5
CS ОЗУ
P6
Адр.
P7
КС
P8
ST BSM
P9
+/−
P10
EPB
P11
EDS
P12
Адр БЗУ
P13
W/R БЗУ
P14
CS БЗУ

Рис. 8.9

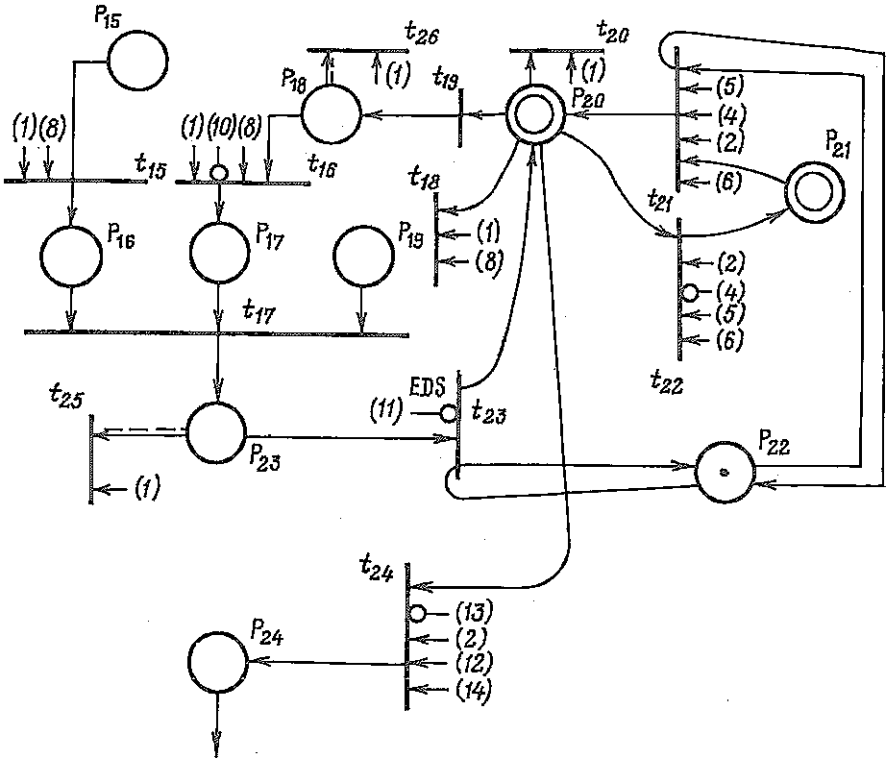


Рис. 8.10

разрядов регистра управления соответствует место аккумулирующего типа (P_4–P_{14}), причем места P_6 и P_{12} соответствуют всем разрядам адресов ОЗУ. В скобках рядом со стрелками указаны номера переходов, к которым направлены данные дуги.

На рис. 8.10 показана модель операционной части блока усреднения. В скобках представлены номера мест , от которых направлены дуги. Смысловая нагрузка мест данной сети следующая: P_{15} – внешняя общая шина, шина P_{16} моделирует наличие информации на входе А сумматора, P_{17} – наличие информации на входе В сумматора, P_{18} – наличие информации после функционирования сдвигателя, P_{19} – наличие информации на входе С сумматора, P_{20} – поступление информации на внутреннюю магистраль усреднителя, шина P_{21} соответствует ОЗУ 1К х 28, шина P_{22} – место контроля корректности пользования внутренней магистралью; P_{23} – выдача результата из сумматора, шина P_{24} соответствует буферному ЗУ (БЗУ).

Реальные действия, моделируемые переходами, распределены так (номер перехода – действие): 15 – занесение информации с общей шины на вход А сумматора; 16 – подключение входа В сумматора; 17 – функционирование сумматора; 18 – занесение данных на вход С сумматора; 19 – функционирование сдвигателя; 20 – снятие информации с внутренней магистрали; 21 –

считывание из ОЗУ на внутреннюю магистраль; 22 – запись в ОЗУ с внутренней магистрали; 23 – считывание результата из сумматора на внутреннюю магистраль; 24 – запись в БЗУ с внутренней магистрали.

Для функционирования данной сети необходимо поставить в соответствие переходам t_1 , t_2 и t_3 процедуры пользователя, которые в зависимости от содержащейся в файле микропрограммы определяют веса дуг (0 или 1), направленных от t_1 , t_2 t_3 к местам (P_4–P_{14}). Таким образом, происходит управление метками, имитирующими логические уровни разрядов регистра управления. Далее составленная сеть Петри представляется операторами описания входного языка.

Рекомендуемая литература: 17.

9. СИСТЕМНЫЕ СРЕДСТВА ПЕРСОНАЛЬНЫХ ЭВМ

9.1. ОБЩАЯ ОРГАНИЗАЦИЯ ПЕРСОНАЛЬНЫХ ЭВМ

Виды персональных ЭВМ. В последние годы широкое распространение получили персональные компьютеры, предназначенные для использования в учебном процессе (учебные), в профессиональной деятельности (профессиональные) и в быту (бытовые). К учебным относятся персональные ЭВМ (ПЭВМ), используемые в системе высшего и среднего образования (Агат, Корвет ДЗ-28, ДВК-2, ДВК-3, ДВК-3М, НЕМИГА). К бытовым ПЭВМ относятся такие, как БК 0010, Ириша, Микроша. В настоящее время в стране выпускаются такие профессиональные ПЭВМ (ППЭВМ), как ЕС 1840, ЕС 1841, Нейрон, Искра-1031 (табл. 9.1). Их зарубежными аналогами являются PC/XT, PC-AT (табл. 9.2).

Рассмотрим элементы организации и программирования ППЭВМ ЕС.

ППЭВМ ЕС – это малогабаритная настольная, но достаточно мощная машина. Она широко применяется как профессиональными программистами, так и начинающими пользователями. Сфера ее действия в значительной степени определяется программным обеспечением, которое разделяется на операционные системы, системы программирования и пакеты прикладных программ (ППП).

ППП имеют определенную профессиональную направленность, используются для решения инженерных и экономических задач, моделирования, обработки текстовой информации, а также обучения пользователей.

Системы программирования предоставляют в распоряжение пользователей средства создания программ на различных языках программирования (ассемблере, ФОРТРАНе, БЕЙСИКе, ПАСКАЛе , СИ), их трансляции, компоновки и отладки.

Структура ППЭВМ. ППЭВМ состоит из следующих блоков: центрального процессора, оперативной памяти, адаптеров для подключения алфавитно-цифрового и графического дисплеев, накопителей на гибких и твердых (винчестерских) дисках, печатающих устройств, графопостроителей и т. д. (рис. 9.1).

Центральный процессор в основном строится на базе 16-разрядного микропроцессора (К1810ВМ86) с возможностью подключения сопроцессоров. К последним относятся: сопроцессор для выполнения арифметических операций и специальных математических функций (К1810ВМ87); сопроцессор для организации ввода-вывода между памятью и периферийными устройствами и обмена память-память (К1810ВМ89). В блок центрального процессора входят буфера адреса и данных, контроллер и арбитр шины (для мультипроцессорной структуры), таймер, блок прерываний ПЗУ.

Блок оперативной памяти предназначен для приема, хранения и передачи данных на системную шину по запросу ЦП или периферийного устройства. В

Таблица 9.1

Параметр	Характеристики ППЭВМ			
	ЕС 1840.05	ЕС 1841	Искра-1031	Нейрон И9.69
Тип МП	К1810ВМ86	К1810ВМ86	К1810ВМ86	К1810ВМ86
Разрядность	16	16	16	16
ОЗУ, К байт	512–640	640–1024	1024–4096	1024
Быстродействие (опер. рег-рег), млн/с	1,0	1,5	1,5	1,5
Дисплей	Черно-белый	Цветной	Цветной	Цветной
Размер экрана:				
точки	640X200	640X200	640X200	640X200
символы	80X25		80X25	80X25
Внешняя память, объем (количество) К байт	НГМД 360 (2)	НГМД 360 (2)	НГМД 360 (2); НМД 10000 (1)	НГМД 360 (2); НМД 10000 (1)
Принтер, пр	Мозаичный, 160	Мозаичный, 160	Мозаичный, 75	Мозаичный, 75
Скорость, знак/с	160	160	75	75
Устройства машинной графики	Нет	"Мышь", графопостроитель	"Мышь", графопостроитель, плоттер	"Мышь", графопостроитель, плоттер
ОС	М86, MS-DOS	М86, MS-DOS	АДОС	Нейрон-ДОС
Система программирования	Ассемблер, Бейсик, Паскаль, Си	Ассемблер, Бейсик, Паскаль, Си	Бейсик, Паскаль, Си, ЯМБ	Бейсик, Паскаль, Си
Базовое ПО	Текст, АБАК СУБД	Текст, СУБД, графика, телеобработка	Текст, документы, графика, телеобработка	Текст, СУБД, графика, телеобработка

состав блока входят контроллер памяти и банки памяти (старший и младший).

Блок адаптера алфавитно-цифрового дисплея управляет вводом-выводом информации на дисплей. В нем используется алфавитно-цифровой метод формирования изображения, согласно которому изображение состоит из отдельных фрагментов (букв, цифр, графических элементов). Фрагмент имеет стандартный формат, определяемый 14-строчной матрицей (на 9 столбцов), элементом которой является точка экрана. Фрагменты изображения хранятся в памяти в так называемом знакогенераторе, содержащем 256 таких фрагментов. Память знакогенератора может изменяться программно, что дает возможность использовать в ППЭВМ произвольные образы знаков.

Выбор необходимых фрагментов, из которых формируется изображение на экране, определяется последовательностью кодов, хранящихся в памяти регенерации. Последняя для ЕС 1840 позволяет отображать на экране 25 строк по 80 символов в строке. Кроме памяти регенерации, адаптер содержит память атрибутов, определяющих вид выводимого символа (обычное и инверсное изображения, мигание, подчеркивание и т.д.).

Таблица 9.2

Параметр	Характеристики зарубежных ППЭВМ			
	IBM PC/XT	IBM PC/XT/370	IBM PC/AT	IBM PC/2-80
Тип МП	1 8088	1 8086	1 80286	1 80386
Разрядность	16	16	16	32
ОЗУ, К байт	640	512	640–3072	4096–8192
Состав	Дисплей; НГМД; НМД; принтер	Дисплей; НГМД; НМД; принтер; эмулятор; IBM 370	Дисплей; НГМД; НМД;	
ОС	CP/M-86, MS-DOS	CP/M-86, MS-DOS	MS-DOS	MS-DOS, OS/2
Система программирования	Бейсик, Фортран, Паскаль, Си	Бейсик, Фортран, Паскаль, Ада, Ассемблер	Бейсик, Фортран, Паскаль, Си, Ада, Пролог	"
Базовое ПО	Тексты, документы, СУБД, графика, телеобработка	Тексты, СУБД, графика, телеобработка	Тексты, графика, интегрированные ППП, телеобработка	"

Адаптер НГМД предназначен для сопряжения процессора с накопителями на гибких дисках диаметром 133 мм и позволяет подключить до 4 накопителей. Вся поверхность дискеты разбита на дорожки, представляющие собой концентрические окружности. В ППЭВМ наиболее распространены форматы записи с 40–80 дорожками на поверхности. Каждая из дорожек может состоять из 8 или 9 секторов. Все служебные функции (подвод головки к заданной дорожке, поиск, выделение и преобразование данных) выполняются аппаратурой адаптера под управлением соответствующей программы ОС. Кроме перечисленных, к служебным функциям адаптера относятся следующие: 1) программирование длины записи данных 128, 256, 512, 1024 байт на сектор; 2) программирование скорости перехода с дорожки на дорожку, времени загрузки и разгрузки головки; 3) передача данных в режиме ПДП или в режиме прерываний.

Адаптер интерфейсов предназначен для подключения к ППЭВМ периферийных устройств, использующих интерфейсы типа ИРПР-М (параллельный) и С2 (последовательный). Он включает адаптер параллельного интерфейса и два адаптера последовательного интерфейса.

Адаптер монитора цветной графики обеспечивает подключение к ППЭВМ монитора, работающего на телевизионной частоте. Можно использовать один из двух сигналов – сигнал базисных цветов или составной видеосигнал. При этом в качестве устройства отображения применяют стандартный телевизионный приемник. В режиме воспроизведения знаков предусмотрено 16 основ-

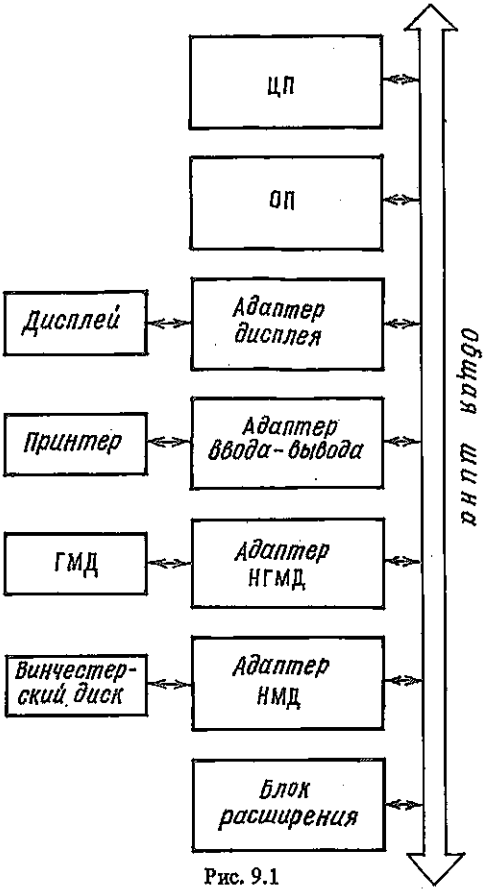


Рис. 9.1

ных и 8 фоновых цветов. Плата адаптера имеет память объемом 16 К байт, которая адресуется из процессора независимо от системной шины.

В текстовом режиме символы формируются генератором символов, который имеется в ПЗУ адаптера. Генератор содержит изображения для 256 возможных символов, графику для игр и обработки слов, международные и специальные знаки. Допускается программная перезагрузка ППЗУ новым знакогенератором.

В графическом режиме возможны два варианта: 320 × 200 точек и 640 × × 200 точек. В первом случае каждый элемент изображения имеет один из программно-задаваемых цветов и 15 вариантов фона. Режим 640 × 200 позволяет получать только черно-белое изображение, поскольку 16 К байт в этом случае используются для задания точек.

Клавиатура. С помощью клавиатуры и дисплея обеспечивается связь пользователя с ППЭВМ. С клавиатуры вводится вся необходимая информация (команды и данные), которая, как правило, отображается на экране. На экран дисплея выводятся сообщения системы, результаты работы, диагностическая информация.

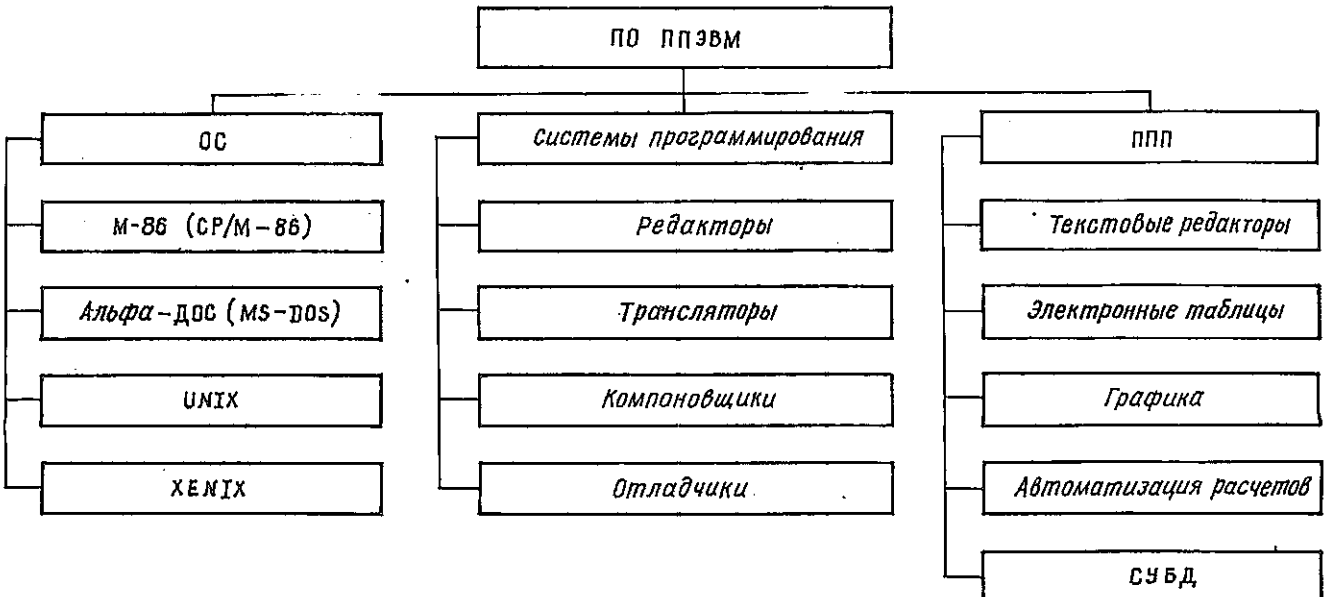

ПО ППЭВМ
ОС
М-86 (CP/M-86)
Альфа-ДОС (MS-DOS)
UNIX
XENIX
Системы программирования
Редакторы
Трансляторы
Компоновщики
Отладчики
ППП
Текстовые редакторы
Электронные таблицы
Графика
Автоматизация расчетов
СУБД

Рис. 9.2

На клавиатуре ППЭВМ ЕС имеются клавиши трех типов: ввода данных, управляющие и функциональные. Первые клавиши предназначены для ввода букв русского и латинского алфавитов, цифр и специальных знаков. На клавиатуре для ввода символов расположены четыре регистра: нижний, верхний, русский и латинский. На русском и латинском регистрах набираются буквы соответствующих алфавитов. Нижнему регистру соответствуют строчные буквы русского и латинского алфавитов, верхнему – прописные буквы.

Управляющие клавиши в комбинации с другими клавишами предназначены для осуществления следующих функций: управления вводом информации с клавиатуры, редактирования вводимой информации, управления системой и выполнением программ. Наиболее часто приходится обращаться к управляющей клавише УПР. Например, для одновременной печати и отображения на экране применяется комбинация УПР-Р, для прекращения выполнения программы – комбинация УПР-СТОП.

Функциональным клавишам (Ф1–Ф10), расположенным на клавиатуре слева, соответствуют некоторые наиболее часто используемые команды системы. Во время работы программист может изменить список команд, закрепленных за функциональными клавишами. Кроме того, функциональной клавише должно соответствовать не более 20 символов.

Структура ПО ЭВМ. Программное обеспечение ПЭВМ изображено на рис. 9.2. Базовой ОС является ОС М86. Она поддерживает функционирование четырех дисководов A, B, C, D (соответственно 1, 2, 3, 4). Система выполняет управление ресурсами ППЭВМ, обеспечение ввода-вывода информации, организует хранение информации на внешней памяти, управляет выполнением программ и др. Для взаимодействия с пользователем ОС М86 располагает набором команд. Каждая команда системы представляет собой запрос на определенные работы. Кроме ОС М86, на ППЭВМ ЕС могут использоваться ОС CP/M-86 и MS-DOS , команды которых в основном совпадают с командами ОС М86.

Основные ППП, применяемые на ПЭВМ под управлением ОС М86, – это создание и редактирование текстов, работа с электронными таблицами (АБАК), реляционная база данных (РЕБУС), автоматизация научно-технических расчетов. Возможно использование других ППП, работающих под управлением CP/M-86 (М-86) или MS-DOS – машинной графики (GSS, GSX), систем деловой графики (Lotos 1-2-3), баз данных (dBase-III).

9.2. ЯЗЫКИ ПРОГРАММИРОВАНИЯ ПЕРСОНАЛЬНЫХ ЭВМ

Технология программирования. Пользователей, работающих на ПЭВМ, условно можно разделить на следующие группы: системные программисты, программисты, непрограммисты. Каждый из них использует соответствующие языковые средства: системный программист – языки ассемблера, программисты – языки программирования высокого уровня (БЕЙСИК, ФОРТРАН, ПАСКАЛЬ), непрограммист – объективно-ориентированный язык УТОПИСТ, позволяющий создавать свои проблемно-ориентированные языки, язык логического программирования ПРОЛОГ.

В связи с изложенным возможны два пути решения задач. Первый, традиционный, заключается в том, что разрабатывается алгоритм решения задачи,

включая описательную и процедурную составляющие. Затем пишутся на одном или нескольких языках программирования модули с использованием редактора текста (edif). Эти модули транслируются и, возможно, компонуются редактором связей (linker) в один загрузочный модуль и загрузчиком (loader) помещаются в оперативную память. Перед выполнением в модуле возможны логические ошибки, которые могут быть обнаружены отладчиком (debugger).

В первом случае осуществляется разработка алгоритма решения задачи и его детализация с использованием языков программирования. Во втором случае описывается не алгоритм решения задачи, а ее постановка. При этом используются не языки программирования, а так называемые языки программных спецификаций. К последним относятся языки УТОПИСТ и ПРОЛОГ.

Ассемблеры. Различают два основных варианта ассемблеров, работающих под управлением ОС М86 (CP/M-86) и АДОС (MS-DOS). Основные конструкции диалектов данных ассемблеров рассмотрены в § 2.2. В основном эти ассемблеры отличаются набором средств псевдо- и макрокоманд.

Из одного входного файла исходная программа ассемблера обычно создает три выходных файла: листинга, шестнадцатеричного кода и символических имен. При этом файлы имеют следующие расширения (М86):. А86 – исходный текст; .ЛСТ – листинг; .Н86 – шестнадцатеричный формат; „CUM – символические имена. Перед точкой пользователь может поставить свое имя длиной не более восьми букв.

Вызов ассемблера осуществляется путем ввода следующей команды:

ASM86 имя исходного файла < SS необязательные параметры >

К необязательным параметрам относятся устройства для исходного и выходных файлов. По умолчанию принимается текущий дисковод. Например, для вызова ассемблера с дисковода А(1) необходимо выдать команду А > >ASM F0 , т. е. ассемблировать файл F0.A86, создать файлы F0.HEX, F0. LST и F0.SYM (все на дисководе А). Для вызова ассемблера в MS-DOS выполняется команда А >ASM или А >MASM . Первая запускает усеченный вариант, а вторая – макроассемблер. После запуска ассемблер просит указать, какие файлы будут использоваться при ассемблировании. Для входного файла достаточно дать только его имя без расширения. Объектному файлу присваивается то же имя, что и входному с расширением „OBJ (если не изменить его). Для файлов листинга и перекрестных ссылок по умолчанию принимается расширение NUL . Это специальный тип файла, в который информация только записывается (он недоступен для чтения). Если эти файлы в дальнейшем надо использовать, необходимо присвоить им другие имена. Приведем пример:

```
Source file name [.ASM]:B:F1
Objekt file name [F1.OBJ]:B:
Source listing „[NUL.LST]:B:F1.LST
Cross reference [NUL.CRF]:B:F1.CRF
```

В листинговом файле объединяются программа на машинном языке и исходный файл. Листинг может быть выведен на дисплей по команде

A > TYPE B: F1.LST

После листинга программы выводится таблица символических имен, в которой указаны все имена программы с их атрибутами. Имена включают метки, переменные и константы с указанием типа, адреса и сегмента:

```
B1      LBYTE     0160      CODE
M1      LNEAR     0A60      CODE
```

Чтобы воспользоваться файлом перекрестных ссылок, необходимо его преобразовать в код ASCII . Для этого вызывается программа CREF , при запуске которой указываются имена двух файлов: входного типа .CRF и выходного типа .REF . Для просмотра последнего используется команда

```
A > TYPE B: F1.REF ,
```

по которой на экране отображается таблица перекрестных ссылок, содержащая в левом столбце все символические имена, а справа – номера строк, в которых имеются ссылки на это имя. Если за номером стоит символ # , то в этой строке данное имя было определено:

```
B1  3  3  # 30  63  100.
```

Таблица перекрестных ссылок позволяет при отладке определить все команды, в которых используется ошибочное имя.

В последнее время для ПЭВМ разработаны системы программирования, компиляции, компоновки и отладки программ, что создает удобства для программиста, позволяя ему работать в одной системе (TURBO).

Особенности разработки программ. При создании прикладных систем большая часть модулей пишется на языках высокого уровня, а отдельные модули – на ассемблере, в связи с чем возникает задача организации взаимодействия программ, транслированных с различных языков. При этом решаются вопросы о составе исходного текста программ (на одном или нескольких языках), структуре исполняемой программы (единый загружаемый модуль или несколько сегментов с частичным перекрытием в памяти), способе хранения данных (в одном или нескольких файлах). Данные характеристики влияют на качество разработки, время выполнения и объем занимаемой памяти.

Самый простой способ разработки не предполагает деления программы на сегменты или модули с использованием текстового редактора ОС или встроенного в систему программирования (БЕЙСИК, TURBOPASCAL). Более сложная программа включает описание данных, структурирование на уровне процедур и функций.

БЕЙСИК. Популярность этого языка объясняется его простотой, диалоговыми возможностями, интерпретирующим механизмом трансляции. В БЕЙСИКе имеются встроенные функции для работы с экраном, клавиатурой, накопителями, принтером. Это позволяет рассматривать язык как расширение аппаратных средств.

Система программирования на языке БЕЙСИК включает редактор текста и интерпретатор. При редактировании пользователь вводит программу, корректирует ее , сохраняет в файле (SAVE) или загружает с внешней памяти старую программу для ее модификации или выполнения (COPY). Выполнение программы начинается по команде RUN, при этом между пользователем и системой возможен рабочий диалог. Процесс интерпретации сокращает цикл:

разработка–выполнение, коррекция ошибок – повторное выполнение. Однако результирующие программы работают медленнее, чем при компиляции. Это объясняется тем, что в режиме интерпретации каждый оператор языка анализируется и выполняется, а в режиме компиляции для всей программы генерируется машинный код. Поэтому отладку удобнее выполнять интерпретатором, а после ее завершения программа компилируется. Интерпретатор занимает объем от 30 до 100 К байт. Для ПЭВМ применяют версии языка BASIC, ABASIC, GWBASIC.

В БЕЙСИКе нет средств для явного определения процедур, а их роль играют участки программ, оканчивающиеся оператором RETURN. К ним обращаются из любой точки программы вводом оператора GOSUB. При структурном программировании подпрограммы располагаются в конце, а их вызовы – в начале текста.

Языки ПАСКАЛЬ и СИ. Эти языки используются опытными программистами для написания как прикладных, так и системных программ. Они отличаются сложной структурой, наличием средств работы с процедурами, компиляцией исходных программ, возможностью написания задачи по модулям с дальнейшим их связыванием. На этих языках создаются большие программные системы на ПЭВМ.

В тексте программ выделяются три компонента: 1) заголовок программы – название, список параметров, описания типов, глобальных переменных; 2) описания процедур – их заголовки с описанием параметров и тела, состоящего из выражений; 3) тело программы, включающее выражения и обращения к процедурам.

Все необходимые связи между формальными параметрами процедур и фактическими параметрами их вызовов выполняются транслятором. В разных языках способы оформления указанных компонентов различны при сохранении общего принципа. В языке СИ вместо заголовка программы используется главная процедура с именем main , вместо процедур – описание функций.

Один из приемов деления исходной программы на части заключается в применении макрогенерации. На ассемблере это вызов макроопределений вместо макрокоманд. На языках ПАСКАЛЬ и СИ включается текстовый файл M1.PAS (M1, C) соответственно с помощью выражений: $ INCLUDE: 'M1.DAS' и include 'M1.C'

После компиляции исходных модулей получаются объектные коды, которые помещаются в файл типа OBJ . Следующий этап заключается в получении загрузочного модуля типа COM или EXEC из нескольких объектных. Этот этап реализуется компоновщиком LINK, которому необходимо указать имена всех объектных модулей, имена файлов загрузочной программы и листинга. Для связывания модулей, оттранслированных с ПАСКАЛя, обращение к компоновщику имеет следующий вид:

```
LINK P1 + P2 + P3,  PP1, PP. LST, PASLIB.LIB ,
```

где PASLIB . LIB – библиотека процедур.

Используемые многократно программы помещаются в бибниотеку объектных модулей, для работы с которой служит процедура LIB . Она позволяет поместить программу в старую или новую библиотеку. Например, запись

LIB OLDLIB + P1 , NEWLIB

вызывает формирование новой библиотеки, в которую помещается содержимое старой библиотеки плюс модуль P1. Файлы с библиотеками для Паскаль- и Си-трансляторов имеют тип LIB . Обращение к библиотечным процедурам требует их упоминания в начале программы с указателем EXTERNAL.

Весь процесс разработки программ на ППЭВМ включает формирование модулей, трансляцию, компоновку. Однако в отдельных системах (TurboPascal, TurboC) этап компоновки отсутствует, так как транслятор порождает сразу загрузочный код.

Большинство трансляторов имеют несколько фаз (проходов), предназначенных для выполнения работы над исходным текстом программы. Так,в системе Pascal транслятор двухпроходный, а в системе С86 – четырехпроходный. Помимо фаз синтаксического анализа и генерации объектных модулей, часто выделяется отдельно фаза оптимизации, за счет которой результирующая программа получается более качественной.

Рассмотренные приемы позволяют при составлении программ иметь дело с несколькими автономными компонентами, которые собираются вместе либо в начале процесса трансляции (текстовое включение), либо при сборке исполняемых программ (редактирование связей).

Организация взаимодействия программ. При разработке программы, состоящей из модулей, написанных на различных языках, возникает проблема их взаимодействия. В этом случае можно воспользоваться механизмом прерываний ДОС или написать программы-связки на ассемблере.

В первом случае используется прерывание 33 (21H), через которое любая прикладная программа может получить доступ к внутренним функциям ОС. К ним относится ряд функций для взаимодействия программ.

Функция 31H останавливает данную программу и позволяет остаться ей в памяти резидентной для нового к ней обращения. Функция 4BH обеспечивает вызов с диска и запуск другой программы, после завершения которой управление снова передается вызывающей программе. Существует функция для выполнения только загрузки.

Функция 4CH завершает работающую программу с кодом возврата, который помещается в регистр AL. Этот код может быть проанализирован с помощью функции 4DH , которая выясняет причину завершения (нормальное, прерывание пользователем, аварийное, окончание по функции 31H).

Функции 48H и 49H позволяют соответственно запросить у ДОС и освободить оперативную память для работы прикладной программы. Функция 4AH изменяет размер оперативной памяти. Например, если в прикладной программе применяется прерывание 33 для вызова и запуска другой программы (функция 4BH), то в регистры заносится: AH-4B – номер вызываемой функции; AL-1 – исполнение программы; DS:DX – в этих регистрах содержится адрес строки с расширенным именем файла; ES:BX – в этих регистрах содержится адрес управляющего блока запускаемой программы, который должен быть оформлен пользователем. Кроме того, через прерывания ДОС можно непосредственно обращаться к функциям, обслуживающим различные внешние устройства.

Развитые системы программирования обеспечивают обращение к прерываниям ДОС через специальные процедуры. Например, в системе TurboPascal

обращение к прерыванию ДОС осуществляется процедурой INTR (N, R), где N – номер прерывания; R – список базовых регистров микропроцессора.

9.3. СРЕДСТВА РАЗРАБОТКИ ПРОГРАММ ПЕРСОНАЛЬНЫХ ЭВМ

Текстовый редактор. Для разработки программ в среде как ОС М86, так и MS-DOS используется ряд системных программных средств, к которым относятся текстовые редакторы, трансляторы, сборщики, отладчики, специальные и интегрированные пакеты.

Для создания и редактирования текстовых файлов на диске ОС М86 (MS-DOS) используется редактор текста РТ (EDLIN). При этом создание и редактирование файла может производиться как построчно, так и посимвольно. Для работы редактора PI используется часть ОП, называемая буфером. Редактирование текста производится относительно указателя, называемого курсором. Запуск редактора осуществляется по команде РТ. Файл, который создается или редактируется, должен быть указан в команде. Например, для создания программы на ассемблере выдается команда A > PT F.ASM.

После запуска (команда РТ) редактор принимает команды пользователя на выполнение функций. Список команд редактора и соответствующие им функции приведены в табл. 9.3. В скобках указаны мнемоники для редактора, работающего в среде CP/M-86, MS-DOS.

Таблица 9.3

Имя команды ОС М86	(CP/M-86)	Назначение
А	(Q)	Аннулировать изменения и закончить редактирование
Б	(F)	Найти в буфере цепочку символов
В	(I)	Вставить заданную цепочку символов
Г, –Г	(B, –B)	Переместить курсор в начало (конец) буфера
nД	(nA)	Добавить n строк из исходного файла в буфер
Е	(E)	Закончить редактирование и записать буфер в файл
nЗ	(W)	Записать буфер в файл для сохранения результатов
nИ, –nИ	(nK, –nK)	Удалить n строк над (под) курсором
nК, –К	(V, –V)	Установить (отменить) нумерацию строк
Л, –Л	(P, –P)	Пролистать буфер на n страниц вперед (назад)
М	(M)	Объединить и выполнить группу команд редактора
О	(O)	Отменить изменения, вернуться к исходному файлу
Н	(H)	Сохранить изменения, перейти к началу файла
nП с.д.	н.д. (S)	Заменить n раз старую цепочку (с.д.) новой (н.д.)
nС, –nС	(nC, –nC)	Переместить курсор на n символов вперед (назад)
nЭ, –nЭ	(nT, –nT)	Отобразить заданное число строк от курсора вниз (вверх)
Ф. стр.	(F)	Найти указанную строку (стр.) в буфере
nХ	(nX)	Передать n строк от курсора во временный файл
Ц	(J)	Вставить цепочку в указанное место буфера
Ч	(R)	Считать в буфер n строк, записанных по команде Х
nТ, –nТ	(nL, –nL)	Переместить курсор на n строк вверх или вниз
n, –n		Переместить курсор на строку n (–n) и отобразить ее
nY, –nY	(nD, –nD)	Удалить n символов вниз справа от курсора (слева)
nЫ, –nЫ		Вывести на печать n строк буфера перед курсором или после него

Некоторые команды редактора РТ содержат цепочки символов для поиска, занесения в буфер, замены. Для определения цепочек признак их конца устанавливается при нажатии клавиши КОН или КЛЮЧ (это равносильно вводу управляющего символа УПР-Z).

Для упрощения процесса редактирования редактор отображает номера строк в виде *nnnnn* в диапазоне 1–65536. Номера строк не содержатся ни в буфере, ни в текстовом файле, а присваиваются при выводе на экран. Для отмены или установления нумерации строк используется команда К. Редактор работает в режиме КОМАНДА или ВСТАВКА. Первый режим устанавливается после запуска редактора, после символа * редактор ожидает ввода команд. В режиме ВСТАВКА, устанавливаемом по команде В, все набираемые символы, кроме управляющих, помещаются в буфер, начиная с текущего положения указателя. Для выхода из режима ВСТАВКА необходимо нажать одну из клавиш КОН и КЛЮЧ или управляющий символ УПР-Z.

Общая команда запуска редактора имеет вид

РТ < исходный файл > < выходной файл >

Под исходным понимается файл, который надо отредактировать, если он существует, или создать, если его нет. Выходной файл задается для сохранения результатов редактирования, если пользователь не хочет заменить исходный файл отредактированным.

При введении команды РТ без параметров пользователю выдаются запросы: ИСХОДНЫЙ ФАЙЛ (возможен ответ ААА, А86); ВЫХОДНОЙ ФАЙЛ (указать имя выходного нового файла или нажать клавишу ВВОД). Если исходный файл существует, то после запуска команды РТ вводится команда Д (добавить) для чтения пакета из файла в буфер. Если исходный файл не существует, редактор создает новый файл, после чего пользователь по команде В (вставить) в режиме ВСТАВКА вводит текст.

Чаще в команде РТ указывается только исходный файл. В этом случае отредактированному файлу дается имя исходного, а исходный сохраняется как резервный и ему присваивается тип РЗВ. Можно сохранить отредактированный файл с тем же именем на другом диске, указав имя этого диска вместо имени выходного файла.

После запуска редактора РТ буфер пуст. Редактируемый текст копируется из файла в буфер по команде Д. Если файл новый, то занесение пакета в буфер осуществляется по команде В. По окончании редактирования во временный файл записывается содержимое буфера и весь текст из исходного файла, а редактор сохраняет исходный файл, присваивая ему тип РЗВ. Отредактированный файл получается из временного после замены редактором его имени на имя исходного файла, указанного в команде.

При редактировании могут быть использованы некоторые управляющие символы клавиатуры: УПР-Е – переместить курсор к началу следующей строки; УПР-U – переместить курсор влево и удалить символ; УПР-I – переместить курсор вправо на позицию, кратную 8; УПР-J , УПР-М – завершить ввод; УПР-Р – отобразить текущую строку и продолжить ввод; УПР-Х – стереть текущую строку и возобновить ввод.

Сборщик. Программа, полученная на выходе ассемблера или другого транслятора, еще не готова к выполнению и обрабатывается сборщиком или

редактором связей. Последний осуществляет связывание нескольких объектный модулей в один выполняемый. Рассмотрим работу сборщика на примере обработки ассемблерных программ.

Пусть получены два объектных модуля F1.OBJ и F2.OBJ , которые необходимо собрать. В исходных программах F1.ASM и F2.ASM имеются директивы EXTRN и PUBLIC , поскольку в модуле F1 есть обращение к модулю F2. При этом для модуля F1 имя F2 является внешним, а для модуля F2 имя F2 является общим, т. е. на него возможны ссылки из других модулей. В директиве EXTRN описываются все внешние имена с их атрибутами, а в директиве PUBLIC – все общие имена. Для нашего примера эти директивы будут иметь вид

```
EXTRN F2:NEAR
PUBLIC F2
```

Атрибут NEAR указывает, что модуль F2 будет находиться в том же сегменте, что и модуль F1 . В процессе редактирования связей устанавливается соответствие между внешними именами и операторами PUBLIC . После этого происходит запись вычисленных адресов в команды, где есть ссылки на внешние имена.

В данном случае если длина модуля F1 100 байт, а его относительный адрес загрузки в сегменте – 0000H, то для модуля F2 относительный адрес загрузки будет равен 100, т. е. 0064H. Таким образом, в операторе CALL F2 модуля F1 определится адрес внешнего имени F2.

Для объединения модулей в ОС М86 вызывается программа СБОР, а в ОС MS-DOS – программа LINK. Рассмотрим для второго случая работу редактора связей:

```
Objekt Modules [.OBJ]:B:F1+F2
Run File [F1.EXE]:B:
List File [NUL.MAP]:B:F1.MAP
Libraries [.LIB]:
```

Программист указывает без расширений имена обьединяемых модулей. Исполняемый модуль по умолчанию получает имя первого модуля с расширением .EXE (можно изменить его имя). Далее запрашивается имя для хранения карты связей. В последней содержатся адреса начальный и конечный, длина и имя сегмента. Например:

Stard	Stop	Length	Name
0000	0059	005A	CODE

Последний запрос редактора связей касается имен библиотек, программы которых могут быть включены в исполняемый модуль. Для программ на ассемблере такой необходимости нет, однако для программ, написанных на языках высокого уровня, имена библиотек можно указать.

Отладчик. Для проверки и отлаживания программ в среде ОС CP/M-86 (M86) в интерактивном режиме служит программа DDT-86. Для ее вызова необходимо ввести одну из команд: DDT-86 или DDT-86 имя файла. Первая команда загружает отладчик и выполняет его команды, вторая, кроме этого, загружает еще и указанный файл. При этом загруженный файл должен иметь тип GMD.

Таблица 9.4

Мнемоника	Назначение
AS	Транслировать введенные операторы языка ассемблера в ячейку с адреса S
B	Сравнить блоки памяти
D	Отобразить память в шестнадцатеричном формате и коде ASCII
E	Загрузить для выполнения программу
F	Заполнить константами блок памяти
G	Перейти к выполнению с дополнительными остановами
H	Выполнить сложение, вычитание с адресами
I	Ввести хвостовую часть команды (отобразить порт ввода)
L	Распечатать память, используя мнемонику
M	Переслать блок памяти
P	Читать дисковый файл в память
S	Установить новые значения памяти
T	Трассировать выполнение программы
U	Контролировать нетрассируемую программу
V	Показать размещение считанного дискового файла в памяти
O	Отобразить порт вывода
W	Записать содержимое блока памяти на диск
X	Проанализировать и изменить состояние ЦП

После готовности отладчика к работе появляется символ курсора. В ответ пользователь набирает нужную команду; для выхода из отладчика используется комбинация УПР-С. Для исправления вводимых команд возможно использование функции редактирования M86: УПР-Х, УПР-М, УПР-Р и т. д. Первый символ командной строки определяет действия команды, описание которых приведено в табл. 9.4. За символом команды могут следовать один или несколько аргументов (шестнадцатеричные значения, имена файлов), отделяющихся друг от друга запятыми или пробелами.

Большинство команд отладчика требуют в качестве операндов адреса, представляющие собой 20-разрядные значения в виде ssss: XXXX, где ssss – номер необязательного 16-разрядного сегмента; XXXX – 16-разрядное смещение.

Рассмотрим примеры работы с командами отладчика:

D 0100:1200, 1250 (производится отображение памяти в шестнадцатеричном коде и коде ASCII с адреса 2200 по адрес 2250);

F 0200:0100, 0120, 5 (по этой команде содержимое памяти с адреса 2100 по адрес 2120 заполняется значением кода 5);

G 1100, 1150, 1180 (управление передается программе с относительного адреса 1100, при этом установлены две точки останова по адресам 1150 и 1180. Значение адреса сегмента берется из регистра CS);

U 1000:0100, 0120 (производит отображение содержимого памяти на языке ассемблера с адреса 10100 по адрес 10120);

M 0100:0200, 0300, 0200:0300 (происходит пересылка блока памяти 100 байт, расположенного по адресу 1200, в область памяти, начиная с адреса 2300);

X (отображается состояние ЦП в форме

AX	BX	CX	...	SS	ES	IP	).
xxxx	xxxx	xxxx		xxxx	xxxx	xxxx	

Вначале указывается содержимое девяти флажков (1 или 0), затем содержимое всех программно-доступных регистров; X AX распечатывает значение регистра AX (по нажатии клавиши ВВОД его содержимое не меняется); если набрано значение zzzz, то оно будет помещено в данный регистр. Для работы с отладчиком в среде MS-DOS необходимо ввести команду A >DEBUG.EXE\F1

После загрузки отладчика в распоряжении программиста имеются команды (см. табл. 9.3), по обозначению в основном совпадающие с ранее рассмотренными командами отладчика DDT-86. При этом основными приемами отладки являются просмотр содержимого дампа памяти по команде D как в двоично-шестнадцатеричном, так и в символьном коде; просмотр содержимого и его изменение для любой ячейки памяти, регистров, портов ввода и вывода (используя соответственно команды E , R , I, O); покомандное или групповое исполнение отлаживаемой программы по командам T, U.

Кроме того, отладчик в среде MS-DOS позволяет осуществлять преобразование файла типа .EXE в тип COM. Хотя оба типа файлов обрабатываются командным интерпретатором ОС, они имеют различия. Главное из них заключается в форматах записи соответствующего файла на дискете. В файле типа .COM точка входа начинается со смещения 100H, и он после загрузки сразу может быть выполнен. При этом вся программа располагается в одном сегменте, поэтому ее размер не может превышать 64 К байт. Все сегментные регистры устанавливаются на начало программы, а указатель стека — на ее конец.

В файле типа .EXE , который является перемещаемым и может располагаться в различных сегментах, задаются значения регистров CS, IP, SS, SP. Значения DS, ES устанавливаются на начало сегмента, в который загружается программа. Программа в сегменте может иметь любое смещение, большее 100 H. Файл типа .EXE занимает иногда на диске больше, чем 64 К байт.

Интегрированные прикладные пакеты. Для решения профессиональных задач на ПЭВМ используется ряд прикладных пакетов (текстовые редакторы, базы данных, графические, обработки таблиц и т.д.). Однако при выполнении сложных работ может возникнуть необходимость многократного применения сначала одних средств, затем других (например, создание набора данных с помощью текстового редактора, затем занесение их в базу данных, корректировка таблиц, затем снова работа с базой данных). При этом необходимо выходить в ОС, вызывать требуемый пакет, выходить из него и т. д. Эта ситуация и послужила причиной создания интегрированных прикладных пакетов на ПЭВМ.

Цель создания прикладного пакета — объединение в одной системе наиболее распространенных программных средств (текстового редактора, электронных таблиц, графических средств, базы данных и средств коммуникации). В интегрированном пакете упрощена стыковка по данным, имеется одинаковая возможность доступа к различным компонентам путем выбора позиции меню, увеличена скорость работы. Недостатком интегрированной системы является большой объем занимаемой памяти (как правило, 200—400 К байт основной и до 1 М байта внешней). Кроме того, ограничиваются возможности отдельных

пакетов по сравнению с лучшими образцами в своем классе.

Наибольшее распространение получили пакеты Lotus 1-2-3 , Symphony и Framework . Первая из этих систем включает три компонента: пакеты электронных таблиц, графики, базу данных. Требует 192 К байт памяти и одного гибкого диска. Последние системы содержат больше компонентов, но соответственно требуют большего объема памяти. Рассмотрим основы системы Framework , на базе которой можно создавать как небольшие проблемно-ориентированные системы, так и автоматизированные рабочие места.

Пакет Framework выполняет следующие функции: работу с текстами, структурное представление документов, использование электронных таблиц, баз данных, графическую обработку информации. Пользователь может работать со специальным языком программирования Fred, на котором описываются сложные алгоритмы обработки данных. Все создаваемые пользователем структуры данных рассматриваются как фреймы. Под последним понимается универсальная структура данных, которая может содержать тоже фреймы, тексты, таблицы, записи и графики.

На экране фрейм изображается в виде прямоугольника, в котором либо название фрейма, либо содержание его части.

При вызове системы на экране появляется главное меню, которое включает девять команд верхнего уровня, соответствующих режимам работы:

Disk, Greate, Edit, Lokate, Frames, Words, Numbes, Graphs, Rint

Выбор команды осуществляется нажатием клавиши УПР и первой буквы команды. Появляется вспомогательное меню в виде столбца, строки которого поясняют отдельные операции. Выбор команды в этом меню осуществляется с помощью стрелок ↑, ↓ и клавиши "Исполнение".

Рабочая область располагается в центре экрана, в ней размещаются окна с фреймами, в которых содержится информация. Область состояния находится ниже и служит для отображения формулы, имени текущего фрейма, индикаторов нажатых клавиш и т. д. В нижней части экрана находится поле сообщений, состоящее из двух строк. В верхней отображается текст обрабатываемого имени фрейма, формулы или числа, а в нижней – сообщения об ошибках, вопросы и реакция системы. В правом верхнем углу в виде столбца располагаются имена накопителей (A:, B:, C:, D:).

Первые действия пользователя заключаются в выборе накопителя. В рабочей области возникает окно-столбец с именами фреймов данного накопителя. Выбирается один из них, его содержимое появляется в рабочем поле, с которым можно работать. Если это текст, то его редактируют, если электронная таблица, то заполняют; в базе данных производят поиск, просмотр, сортировку; графики строят, выбирая из таблицы или базы данных, а документы просматривают и модифицируют. При этом обрабатываемый фрейм имеет вложенную структуру. При работе с пакетом пользователь может обратиться в ДОС и вернуться назад.

ОС М86. Наиболее распространенными ОС для ПЭВМ серии ЕС являются М86 (аналог CP/M-86) и АДОС (аналог MS-DOS). Рассмотрим организацию и работу первой из них.

Минимальная конфигурация ПЭВМ, достаточная для функционирования М86, включает оперативную память не менее 128 К байт, дисплей с клавиатурой и НГМД.

При загрузке с системной дискеты в оперативную память помещается резидентная часть системы, которая занимает 32 К байт памяти. В оставшуюся часть памяти загружаются нерезидентные системные программы и программы пользователя. При этом функции распределения памяти реализуются системой от пользователя (требуется только указать имена загружаемых файлов).

ОС М86 в процессе функционирования выполняет следующие функции: управляет ресурсами ПЭВМ, обеспечивает ввод-вывод информации, организует работу с информацией на внешней памяти и осуществляет выполнение программ.

Структура ОС М86 включает интерпретатор команд (ССР), базовую систему ввода-вывода (BIOS) и ядро (BDOS). Вход в систему BIOS осуществляется с помощью программного прерывания МП К1810ВМ86–22Н. Код функции передается в регистр CL с параметрами байтов в DL или слов в DX. Значения выполненной функции возвращаются через регистры AL, AX, BX. Все регистры сегментов, кроме ES, сохраняются после входа в BIOS и восстанавливаются после выхода из него. Простые вызовы функций BIOS соответствуют номерам 0–12, куда входят сброс системы и односимвольный ввод-вывод. Например, функция 1 (ввод с пульта): ввод CL = 01Н, возврат AL – код символа. Функции с 13 по 52 соответствуют операциям над файлами. Например, функция 13 – сброс дисковой системы; ввод CL = 0D, возврата нет. Функции 53–59 соответствуют управлению памятью и загрузке программ. Например, 59 – загрузка программы (с диска); ввод CL=3B, DX – смещение блока FCB; возврат AX – код возврата; BX – адрес базовой страницы.

Блок BDOS осуществляет управление файлами и реализует распределение ресурсов. Интерпретатор ССР обрабатывает команды пользователя, а также выполняет группу команд (так называемые командные файлы).

Основным средством взаимодействия пользователя с ОС является система команд. Команды можно разделить на следующие группы: управление ОС, обслуживание дисков, работа с файлами, работа с ПУ (табл. 9.5). Различают команды резидентные и транзитные. Резидентным соответствуют программы, входящие в состав командного интерпретатора (СП, СС, ИМЯ, УД, ЧТ, НОМ). Каждой транзитной команде соответствует хранящийся на диске командный файл, имеющий имя этой команды. Набор команд ОС может быть расширен (например, пользователь, написав программу, имеет возможность ввести команду, выполняемую как транзитную).

Информация, введенная пользователем после приглашения ОС, обрабатывается следующим образом. Проверяется, резидентная ли это команда: если да, то она начинает выполняться, если нет, то производится поиск одноименного файла на текущем или заданном диске. Если файл не найден, выдается ошибка. Выполнение команды может быть завершено либо нормально, либо

Таблица 9.5

Команда СР/М-86	Команда М86	Назначение
ASM86	АСМ86	Преобразует программы, написанные на ассемблере МП 18086 (К1810ВМ86), в машинный код
ASSIGN	ПАРМ86	Позволяет направить вход и выход на ВХ ПЭВМ
COPYDISK	ФДСКТ	Создает копию диска
DIR	СП	Вывод списка имен файлов пользователя
DIRS	СС	Вывод списка системных файлов
ED	РТ	Создание и редактирование файлов
ERA	УД	Удаление одного или нескольких файлов
FUNCTION	ПАРМ86	Программирование функциональных клавиш
GENCMD	СБОР86	Преобразует в выполнимый командный файл выходной объектный файл ассемблера
HELP	ИНФ86	Представляет информацию о всех командах ОС
NEWDISK	ФДСКТ	Форматирует и проверяет диск
PIP	КОП	Копирует один или несколько файлов, объединяет их
REN	ИМЯ	Позволяет переименовывать файл
SUBMIT	ПАКЕТ	Выполнение группы команд (командного файла)
STAT	СОСТ	Изменяет атрибуты файла и режимы доступа к нему
TOD	ДАТА	Устанавливает дату и время суток
TYPE	ЧТ	Отображает содержимое файла на устройстве вывода
DDT86	ДДТ86	Позволяет отлаживать выполняемые модули
USER	НОМ	Устанавливает номер пользователя
PROTOCOL	НАП	Позволяет изменять протокол передачи данных для последовательных портов
SPEED	ПАРМ86	Настраивает асинхронный адаптер на связь
	ПАП	Настройка режимов печати
	ЗАЛП	Загрузка русского алфавита в принтер

может быть прервано по инициативе системы или пользователя.

Команда ОС включает имя, за которым следуют один или несколько параметров. Количество и способ задания параметров определяются конкретной функцией команды. Одна команда может иметь различные наборы параметров.

В качестве примера рассмотрим команду СОСТ – получение информации о дисках и файлах, а также изменение их атрибутов. Ее формат:

$$\text{СОСТ}\left\{\text{спецификация файла}\right\}\left\{\text{ЧТ/ЧЗ/С/П}\right\}.$$

Используя этот формат, можно ввести следующие варианты команды: СОСТ – получить информацию об используемых дисках; СОСТ B: A1.M – получить информацию о файле A1.M, расположенном на диске B_2 и режимах доступа к нему; СОСТ B: .C1 ЧТ – присвоить атрибут ЧТ всем файлам с типом C1, расположенным на диске B.

Справочную информацию о командах системы М86 можно получить по команде ИНФ86. Если в последней команде указать в качестве параметра имя любой команды, то можно получить справочную информацию о функциях и форматах этой команды.

ОС АДОС. Другой распространенной ОС для ПЭВМ является АДОС (MS-DOS). Важнейшей ее особенностью является модульность. Это позволяет собирать в одном модуле логически связанные функции, изолировать отдельные части ОС, производить реконфигурацию системы. В состав ДОС входят следующие модули: 1) базовая система ввода-вывода BIOS (находится в ПЗУ); 2) блок начальной загрузки (БНЗ) (находится в первом секторе нулевой дорожки системной дискеты); 3) модуль расширения БСВВ (_BIO .COM) и модуль обработки прерывания (_DOS.COM) (располагаются сразу за БНЗ); 4) командный интерпретатор (COMMAND.COM) (находится на системной дискете); 5) утилиты ДОС (FORMAT.COM, DISKCOPY.COM) (находятся на системной или другой дискете).

Здесь в скобках даны английские названия модулей, используемых в MS-DOS. В начале имен файлов BIO.COM и DOS.COM ставятся идентификаторы фирменной принадлежности.

Как указано, BIOS находится в ПЗУ, остальные модули – на дискете. Модули расширения BIOS и обработки прерываний хранятся в двух специальных файлах на системном диске, места размещения которых фиксированы (располагаются всегда за БНЗ). При этом оба файла располагаются только на системной дискете.

Командный интерпретатор находится в файле, размещенном в любом месте системного диска. Утилиты могут располагаться в командных файлах на дисках как на системном, так и на пользовательских. Доступ к утилитам осуществляется через файловую систему ОС.

Рассмотрим назначение и функционирование отдельных модулей АДОС.

БНЗ. Данный блок позволяет загружать в оперативную память модули ДОС. Работа БНЗ заключается в просмотре каталога системного диска и определении того, что первые два файла являются модулями ДОС. В ДОС это файлы BIO.COM и DOS.COM. Они снабжаются специальными атрибутами, которые делают невозможным вывод их имен на экран дисплея, хотя они и присутствуют в каталоге. Если БНЗ не обнаружит эти два файла, данный диск является несистемным, о чем выдает сообщение модуль БСВВ BIOS, хранящийся в ПЗУ.

БСВВ. Входящие в модуль программы выполняют ряд важных функций. Одной из них является тестирование основных компонентов ПЭВМ для обнаружения и локализации неисправностей. Набор тестов предусматривает проверку работоспособности всех устройств ПЭВМ: процессора, оперативной памяти, дисплея, принтера, НГМД, клавиатуры. Тест процессора проверяет контроллер прерываний, прямой доступ к памяти, работу шин адреса и данных и т. д. Тестирование ОП осуществляется при записи и считывании в различных режимах.

Тест клавиатуры проверяет правильность сканирования нажатых клавиш, тест дисплея – адаптер дисплея и самого монитора, тест принтера – адаптер и само устройство, тест НГМД включает проверку форматизации дискеты, записи и считывания эталонной информации.

Второй важной функцией БСВВ является вызов БНЗ, который после считывания в ОП загружает все основные модули ДОС. При такой двухступенчатой загрузке с БСВВ снимается функция поиска и настройки различных моду-

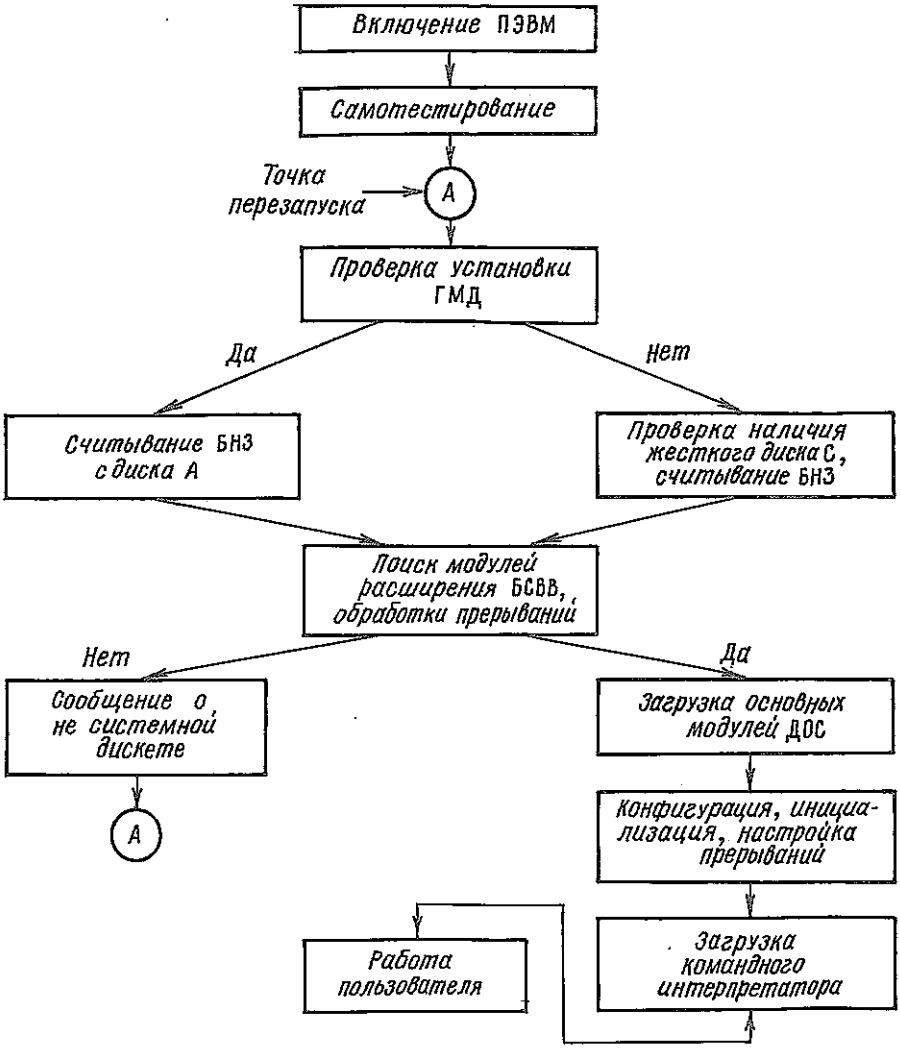


Рис. 9.3

лей ДОС. На рис. 9.3 показан алгоритм загрузки и инициализации ДОС. БНЗ располагается в первом секторе нулевой дорожки.

Третья функция БСВВ – обслуживание системных вызовов (прерываний). Прерывания разделяются на аппаратные, логические и программные. *Аппаратные* прерывания возникают из-за неисправностей работы машины, а также при поступлении сигналов от внешних устройств. *Логические* прерывания появляются в особых ситуациях в процессоре: переполнении, делении на нуль, появлении точки останова и др. *Программные* прерывания вырабатываются, если одна программа запрашивает сервис, связанный с работой аппаратуры, от другой программы.

На БСВВ возлагается обслуживание прерываний нижнего уровня (номера 0–1FH), значения которых приведены в табл. 9.6. Прерывания, обслуживаемые БСВВ, связаны с базовыми операциями ВУ: дисплеем, НГМД, принтером, клавиатурой. Отдельные прерывания обеспечивают доступ к нескольким связанным функциям. Прерывание 13H (управление диском) открывает доступ к 18 функциям с кодами 0–17 (начальная установка, выдача статуса диска, чтение, запись группы секторов на одной дорожке, верификация после чтения / записи, форматирование дорожки и т.д.). Прерывание 10H обеспечивает доступ к 24 функциям управления дисплеем. Прерывания 16H, 17H обеспечивают доступ к 10 функциям управления клавиатурой и 3 функциям управления принтером.

Расширение БСВВ придает гибкость ДОС, позволяя управлять с ее помощью таким набором ВУ, который наиболее удовлетворяет требованиям пользователей конкретной ПЭВМ. Этот модуль может быть модифицирован с учетом особенностей конкретной версии ДОС.

Появляется возможность включения в BIOS дополнительных программ (драйверов), обслуживающих новые внешние устройства. О необходимости подключения новых драйверов или изменении других параметров ДОС пользователь может сообщить через специальный файл конфигурации системы CONFIG.SYS. Этот файл обрабатывается модулем расширения БСВВ, который осуществляет необходимую подстройку прерываний или других параметров ДОС в соответствии с заданными командами конфигурации. Последние могут указывать на дополнительные драйверы, подключаемые к системе, количество одновременно открытых файлов (по умолчанию 8), буферов для обмена информации с диском. Для этого используются специальные команды ДОС. Например, для подключения дополнительного драйвера задается команда: DEVICE = MOUSE.SYS , которая включает имя драйвера; для изменения количества одновременно открытых файлов – команда FILES = 10; для изменения количества буферов для обмена с дисками – команда BUFFERS = 6 и т.д. Имеется возможность включения в ДОС драйверов не только для новых устройств, но и для существующих. Например, в файле CONFIG. SYS по команде

Таблица 9.6

Номер	Ситуация	Номер	Ситуация
0	Деление на ноль	14	Управление адаптером
1	Шаговый режим	15	Управление магнитофоном
2	Падение напряжения	16	Управление клавиатурой
3	Появление точки останова	17	Управление принтером
4	Переполнение регистров АЛУ	18	Обращение к БЕЙСИКу в ПЗУ
5	Печать графической копии	19	Перезапуск системы
6	Зарезервировано	A–D	Зарезервировано
7	"	E	Окончание обмена с НГМД
8	Сигнал от таймера	F	Обслуживание принтера
9	Сигнал от клавиши клавиатуры	1A	Запрос времени и даты
10	Управление дисплеем	1D	Адрес таблицы параметров дисплея
11	Запрос списка оборудования		
12	Запрос режима ОП	1E	Адрес таблицы параметров НГМД
13	Управление НГМД	1F	Адрес таблицы символов с кодами 128–255

Таблица 9.7

Прерывание	Выполняемые действия
20H	Нормальное завершение программы
21H	Обращение к функциям ДОС
22H	Адрес подпрограммы обработки завершения задачи
23H	Адрес подпрограммы реакции на клавиши УПР-С
24H	Адрес подпрограммы обработки неустранимой ошибки
25H	Абсолютное чтение с заданных секторов диска
26H	Абсолютная запись на заданные секторы диска
27H	Завершение программы, оставляющей ее резидентной

Таблица 9.8

Номер	Ситуация	Номер	Ситуация
0	Завершение программы	19	Определение текущего диска
1	Ввод символа с клавиатуры на дисплей	1A	Установка адреса буфера (DTA) для обмена с диском
2	Вывод символа на дисплей	1B	Получение адреса таблицы размещения файлов (FAT)
3	Ввод символа из канала	1C	Получение FAT для устройства
4	Вывод символа в канал	1D–20	Внутренние функции ОС
5	Вывод символа на печать	21	Чтение с диска с прямым доступом
6	Обмен символами с терминалом	22	Запись на диск с прямым доступом
7	Ввод символа с клавиатуры без эха	23	Выдача длины файла
8	Ввод символа с клавиатуры с эхом	24	Задание номера записи
9	Вывод строки символов на экран	25	Установка вектора прерывания
A	Ввод с клавиатуры с буферизацией	26	Создание программного сегмента
B	Проверка ввода с клавиатуры	27	Чтение блока с прямым доступом
C	Очистка буфера ввода с клавиатуры	28	Запись блока с прямым доступом
D	Сброс диска	29	Преобразование имени файла во внутренние параметры
E	Установка текущего диска	2A	Выдача даты
F	Открытие файла с FCB	2B	Установка даты
10	Закрытие файла с FCB	2C	Выдача времени
11	Поиск имени файла по шаблону	2D	Установка времени
12	Продолжение поиска, начатого функцией 11	2E	Установка (отмена) верификации записи на диск
13	Удаление файлов с диска	2F	Выдача адреса области DTA
14	Последовательное чтение из файла	30	Выдача номера версии ДОС
15	Последовательная запись	31	Завершение программы
16	Создание файла в файл	32	Внутренняя операция ДОС
17	Переименование файла	33	Проверка или отмена нажатия клавиш УПР+КЛЮЧ
18	Внутренняя операция ДОС	34	Внутренняя операция ДОС
		35	Выдача вектора прерывания
		36	Выдача свободного пространства на диске

Окончание табл. 9.8

Номер	Ситуация	Номер	Ситуация
37	Внутренняя операция ДОС	56	Переименование файла
38	Выдача форматов, даты, времени	57	Выдача даты и времени модификации файла
39	Создание подкаталога	49	Освобождение блока памяти
3A	Удаление подкаталога	4A	Изменение длины блока памяти
3B	Установка имени подкаталога	4B	Загрузка (выполнение) подзадачи
3C	Создание файла без FCB	4C	Завершение программы с возвратом управления
3	Открытие файла без FCB	4D	Выдача кода завершения
3E	Закрытие файла без FCB	4E	Поиск первого элемента каталога по шаблону
3F	Чтение из файла (с устройства)	4F	Продолжение поиска файлов каталога по шаблону
40	Запись на файл (в устройство)	50–53	Внутренние функции ДОС
41	Удаление файла из каталога	59	Выдача кода ошибки
42	Установка позиции для последовательного доступа	5A	Создание временного файла
43	Установка атрибутов файла	5B	Создание нового файла
44	Обращение к файлу		
45	Дублирование номера файла		
46	Объединение номеров файла		
47	Выдача текущего каталога		
48	Выделение блока памяти		

DEVICE = TOSH.SYS включается в систему драйвер принтера для печати русскими буквами.

Еще одной функцией модуля расширения БСВВ является завершение загрузки ДОС в ОП. Сначала управление передается модулю обработки прерываний, в котором работает программа инициализации. Последняя устанавливает внутренние таблицы, инициализирует векторы прерываний 32–39 и подготавливает загрузку командного интерпретатора. После этого прерывание возвращается в модуль расширения БСВВ, который производит загрузку командного процессора и передает ему управление.

Модуль обработки прерываний ДОС. Он образует верхний уровень системы, с которым взаимодействуют основные прикладные программы. Компонентами данного модуля являются подпрограммы, обеспечивающие работу файловой системы, устройств ввода-вывода, обработку специальных ситуаций. В табл. 9.7 приведен перечень прерываний ДОС. Одно из этих прерываний 21H является основным, поскольку за ним скрывается множество функций ДОС по обслуживанию ВУ и файловой системы. Перечень этих функций приведен в табл. 9.8.

Деление функций ДОС на два уровня обусловлено соображениями модульности и развития системы. Прерывания, кроме приведенных в табл.9.6, зарезервированы (всего прерываний верхнего уровня 32 от 20H до 31H). С другой стороны, список функций ДОС, вызываемых через прерывание 21H, большой. Многие из этих функций связаны, как и рассмотренные выше в прерываниях БСВВ для работы с принтером, дисплеем, клавиатурой, НГМД.

При обращении к функциям ДОС из прикладных программ в регистр AH заносится их код, в регистр DX – адрес (если он нужен), затем выполняется

вызов прерывания 21H. В регистре AL помещается результат выполнения функции. Использование функций ДОС при работе с файлом из программы на языке ассемблера будет рассмотрено в §9.5.

Все функции ДОС разбиты на группы в соответствии с характером предоставляемого ими сервиса. Функции 0–С обеспечивают посимвольный обмен со стандартными внешними устройствами. Ряд функций обеспечивает работу с файловой системой на основе блока управления файлом (FCB). Начиная с версии 2.0 в ДОС введены новые функции для работы с файлами, в которых имена файлов и передаваемые блоки информации задаются в прикладной программе. Эти группы функций дополняются функциями работы с каталогами иерархической файловой системы (номера 11–12, 39–3B, 45–47, 4E–4F, 56–57). Для разработки больших прикладных систем, состоящих из набора взаимосвязанных программ, введены функции 31, 33, 48–4D, позволяющие выделять и освобождать области памяти, а также загружать в ОП и запускать подзадачи.

Имеются три прерывания с номерами 34–36, которые могут обслуживаться прикладной задачей. При решении прикладных задач ДОС возможны три варианта работы с файлами или обмена с терминалом: 1) при работе на уровне языков высокого уровня (ПАСКАЛЬ, БЕЙСИК) можно воспользоваться стандартными функциями; 2) обращение к подпрограммам ДОС, доступным через прерывания 32–63; 3) обращение к прерываниям нижнего уровня (0–31), обслуживаемых БСВВ, однако в таком случае ПП сильно зависит от модели ПЭВМ.

Командный интерпретатор. Он располагается в третьем файле ДОС в произвольном месте системного диска и состоит из резидентной и нерезидентной составляющих. Первая содержит подпрограммы обработки прерываний с номерами 34–36, программу загрузки нерезидентной части в ОП и программу инициализации обработки файла AUTOEXEC.BAT.

Нерезидентная часть включает программу обработки команд ДОС, поступающих с терминала или из командных файлов, а также загрузчик файлов. Задачей последнего является чтение очередной программы с диска в ОП, настройка адресов и передача ей управления.

Поскольку нерезидентная часть интерпретатора располагается в старших адресах ОП (рис. 9.4), любая другая часть может ее стереть. По окончании прикладной программы (ПП) управление всегда передается резидентной части, которая восстанавливает стертую нерезидентную часть.

Основная функция командного интерпретатора заключается в приеме, анализе и выполнении команд ДОС, служащих главным средством общения пользователя с системой до вызова ПП. По окончании работы ПП вновь действует командный интерпретатор.

Команды ДОС позволяют готовить диски для работы, обеспечивают все манипуляции с файлами, запускают любую ПП. В роли последней могут выступать утилиты ДОС – программы, выполняющие сервисные функции. Команду ДОС в общем виде можно записать так:

$$\text{nnn } a_1, ..., a_n [f_1, ..., f_n],$$

где nnn – имя команды (обязательно); $a_1, ..., a_n$ – аргументы (требуются не во всякой команде); $f_1, ..., f_n$ – признаки режима (необязательные парамет-

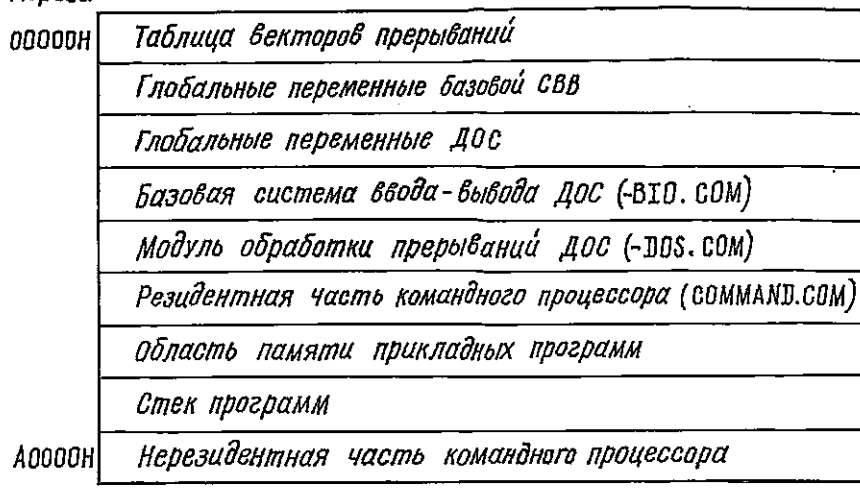


Рис. 9.4

Таблица 9.9

ADOS (MS-DOS)	Описание
	Управление командным файлом
ВЫХОД (EXIT)	Выход из командного процессора ДОС. Используется для возврата в прикладную программу
ДЛЯ ... НЗ (FOR ... IN) ... ВЫП (DO)	Повторяющаяся операция над списком аргументов
ЕСЛИ (IF)	Проверка условия
КОММ (REM)	Строка с комментариями
НА (GOTO)	Переход на метку в командном файле
ПАУЗА (PAUSE)	Приостановка обработки пакетного файла
СДВИГ (SHIFT)	Сдвиг фактических параметров относительно X, выдача сообщений на экран
ЕХО (ECHO)	Отключение "эхо"
	Формирование операционной среды
ВЕРИФ (VERIFY)	Установка (отключение) верификации при записи на диск
ВРЕМЯ (TIME)	Выдача и установка текущего времени
ДАТА (DATA)	Выдача и установка текущей даты
КОНСОЛЬ (CTTY)	Переустановка консоли
МАРШ (PATH)	Задание альтернативных маршрутов для поиска
ПАРАМ (SET)	Задание параметров операционной среды
СТОП (BREAK)	Установка (отключение) прерывания
	Файловая система
КАТ (DIR)	Выдача каталога файлов
КОП (COPY)	Копирование файлов

Окончание табл. 9.9

ADOC (MS-DOS)	Описание
НК (MD)	Создание нового каталога
СК (CD)	Смена текущего каталога
УД (DEL)	Удаление файлов
УК (RD)	Удаление каталога
ВЫВ (TYPE)	Вывод файла на консоль (дисплей)
ИМЯ (REN)	Переименование файлов
	Часто используемые утилиты
ВЕРС (VER)	Выдача номера версии ДОС
ДИСК (CHKDSK)	Выдача общего объема и заполнение диска
ИМДИСК (VOL)	Выдача метки диска
МЕТКА (LABEL)	Задание метки диска
РЕЖИМ (MODE)	Установка режима работы устройств
СИСТ (SYS)	Перенос системных файлов ДОС
ФОРМАТ (FORMAT)	Форматирование (разметка) диска
ЭКР (CLS)	Гашение экрана
	Другие утилиты
АТРИБ (ATRIB)	Установка (снятие) атрибута защиты файлов
СРАВН (COMP)	Сравнение содержимого файлов
ДИСКОП (DISKCOPY)	Копирование диска по дорожкам
ГРАФИКА (GRAPHICS)	Подготовка к печати графической копии
ПЕЧАТЬ (PRINT)	Подготовка файла в очередь на печать
СТРУКТ (TREE)	Выдача иерархической структуры файловой системы
– (EDLINE)	Запуск строчного редактора
– (DEBUG)	Запуск отладчика
– (LINK)	Запуск компоновщика

ры). Аргументы указывают на те объекты, с которыми имеет дело команда; имена дисков, каталогов, файлов, внешних устройств. Аргументы указывают различные режимы в исполнении команды. Приведем примеры команд ДОС:

DIR – выдача каталога;

COPY A: ED.COM B: – копирование файла с диска A на B.

Важную роль в организации взаимодействия пользователя с ДОС играют командные файлы, которые могут содержать последовательности команд и специальные операторы для организации управления (IF, FOR, DO) (табл.9.9).

Выполнение командного файла начинается в тот момент, когда в качестве команды ДОС дается его имя. Командный процессор начинает читать и интерпретировать строки командного файла. Строка может содержать команду, метку или комментарий. Если в строке стоит команда, то осуществляется вызов соответствующей программы, приостанавливается выполнение файла, и начинается реализация вызванной программы. После ее завершения управление передается командному файлу.

Командный процессор различает три вида исполняемых файлов: 1) файлы типа .COM (содержат программы, не требующие настройки адресов после загрузки в память); 2) файлы типа .EXE (должны содержать программы, которые при загрузке с диска в память требуют задать адреса сегментов в заголов-

ке программы); 3) файлы типа .BAT (считаются командными и обрабатываются интерпретатором).

Для удобства автоматической настройки ПЭВМ на конкретного пользователя или задачу в командный процессор входит программа инициализации, которая обрабатывает особый командный файл с именем AUTOEXEC.BAT. В этом файле можно описать режимы работы и запуск определенных программ, что позволяет создать пользователю определенную операционную среду.

Утилиты ДОС. В стандартный комплект ДОС входит ряд сервисных программ, которые находятся на дисках. При практической работе наиболее часто используются такие, как программы начальной разметки дисков (FORMAT. COM), проверки состояния дисков (CHKDSK.COM) и т. д. К утилитам относится текстовый редактор и интерпретатор языка БЕЙСИК (их правильнее относить к категориям системных или прикладных программ).

Перспективы развития ОС ПЭВМ. В настоящее время разрабатываются ПЭВМ на базе 32-разрядных МП I80386. В частности, для таких ПЭВМ может использоваться система OS/2, архитектура которой включает три уровня. Основной уровень — ядро и системный сервис. Второй и третий уровни образуют монитор окон и монитор локальных и общих сетей.

Ядро OS/2 обеспечивает мультизадачную обработку, при этом пользователи имеют возможность параллельно выполнять прикладные и сервисные программы. Для программирования используются макроассемблер, оптимизирующий компилятор языка СИ, а также компоновщики, многооконные редакторы и отладчики.

Выполнение задач под управлением системы OS/2 происходит в режиме разделения времени, причем величина кванта времени базируется на приоритете задачи. Система работает в двух режимах: защиты и реальном. В первом выполняются задачи параллельно с защитой друг от друга, разделяя ресурсы ПЭВМ. В реальном режиме выполняются прикладные программы, разработанные в среде MS-DOS , при этом все ресурсы могут быть захвачены одной задачей.

Программист загружает и выполняет прикладные программы в режиме группового экрана, который представляет собой логическое разделение пользователей при мультизадачной обработке. Групповой экран может включать работу с файловой системой, базой данных, графикой или выполнять команды ОС. При этом переход между окнами организуется по ключу. Процесс пользователя имеет буфера для каждого экрана из группы. При переключении экрана на другую группу происходит отображение результатов предыдущей обработки.

Структурно система OS/2 включает следующие компоненты: создание и выполнение процессов, менеджер памяти, обслуживание устройств, систему управления файлами, коммуникатор процессами, сервисные средства. Для обработки процессов используются первичные средства, которые могут требовать создание новых средств при обращении к ядру. Менеджер памяти реализует виртуальный режим, используя средства МП.

Файлы в ОС М86. Система М86 обеспечивает возможность создания, хранения, поиска и редактирования файлов. В файлах могут храниться программы на языках программирования, в машинных кодах либо данные в виде текстов, чисел, графических образов и т. д.

В соответствии с характером хранимой информации файлу приписывают тип, который содержит три символа и отделяется от имени точкой. Например, A1.КМД – файл машинных команд, A1.АСМ – программа на ассемблере. Имя файла может содержать до восьми символов.

На диске файлы объединяются в логические группы, образующие каталоги. Под последним понимается оглавление файла. Каждый файл может располагаться на конкретном диске. Имя диска, полное имя файла образуют его спецификацию.

Обработка файлов выполняется посредством команд ОС, в которых указывается спецификация файла. Например, по команде A >УД C1.АСМ с диска A удаляется файл C1, содержащий программу на ассемблере. По команде A > >СП:B выдается каталог программ пользователей на диске B.

При обозначении в команде сразу нескольких файлов используется шаблон. Для этого применяются два символа * и ?. По первому можно обозначить любое количество знаков как в имени, так и в расширении файла, по второму – одиночный символ. Приведем примеры шаблонов:

A*.* – все файлы, начинающиеся с символа A;

A > КОП *.АСМ B: – скопировать с диска A на диск B все файлы типа .АСМ;

A > УД ?.* – удалить с диска A все файлы, имя которых содержит один символ.

В ОС М86 предусмотрены средства для защиты файлов от случайных обращений. Для этого файлу присваивается номер пользователя и атрибуты. При создании файлу присваивается текущий номер пользователя, который после загрузки равен 0. Для его изменения используется команда НОМ в диапазоне от 0 до 16. Таким образом, в ОС М86 может быть задано до 16 групп файлов.

Атрибуты файла определяют способ обращения к нему. Они присваиваются по команде СОСТ и могут быть пользовательскими (П), системными (С), только для чтения (ЧТ) и для чтения и записи (ЧЗ). Файлы ОС имеют атрибуты ЧТ/С. Для получения каталога системных файлов используется команда A >СС. Для изменения атрибутов ЧТ на ЧЗ всем файлам каталога на диске B выдается команда B >СОСТ *.ЧТ.

Основной единицей при обращении к диску является запись. По умолчанию длина записи равна 128 байт. Система М86 группирует записи в блоки по 2 К байт. Запись информации блоками сокращает количество операций ввода-вывода и позволяет динамически распределять внешнюю память на диске. При расширении файлу выделяется дополнительный участок диска.

Характеристики файла (полное имя, номер пользователя, атрибуты, расположение на диске) помещаются в область, называемую оглавлением и образующую БУФ. Последний может описывать до 16 блоков. При удалении файла из оглавления стирается его БУФ. После этого элементы оглавления стано-

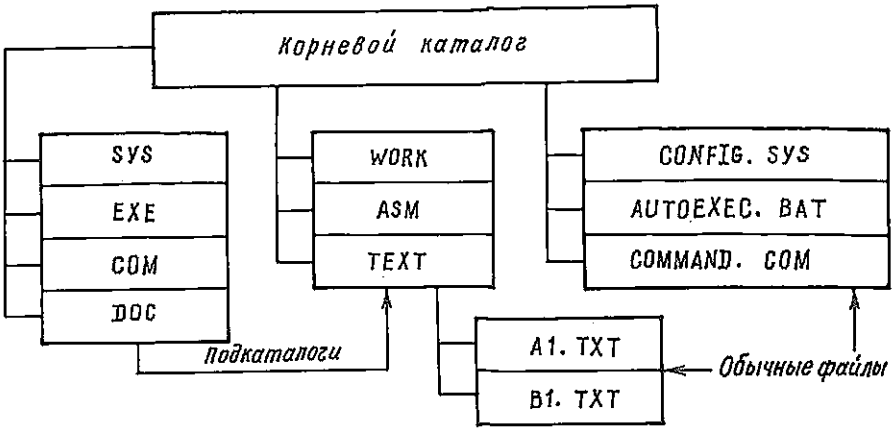


Рис. 9.5

вятся доступными для каталогизации нового файла, а соответствующая файлу память на диске освобождается.

В целом для файлов системы в ОС М86 характерны линейность, ограниченные возможности команд системы для работы с файлами, стандартная длина записей, размер файла, не превышающий 32 К байт.

Файловая система АДОС. Рассмотрим организацию более мощной файловой системы. Обозначение файла включает до 8 символов собственного имени и 3 символа расширения. Например, COMMAND. COM, СТ.ПАК, Ф1.КМД. К стандартным расширениям относятся .ASM, .PAS (для исходных программных файлов); .OBJ , .LST (для объектных файлов и листингов); .LIB (для библиотек объектных модулей); .КМД, .ИСП, .COM, .EXE (для загрузочных файлов); .ПАК, .BAT (для пакетных (командных) файлов).

В АДОС используется в отличие от ОС М86 иерархическая файловая система (рис. 9.5). Она включает главный (корневой) каталог, который в свою очередь может содержать как файлы, так и каталоги первого уровня. Последние могут также включать файлы и каталоги второго уровня и т.д. Таким образом организуется древовидная структура; вверху – корневой каталог, внизу – подчиненные файлы или каталоги. Корневой каталог обозначается (\ ), содержит до 112 файлов (каталогов). Разнесение информации по различным каталогам позволяет логически группировать однородные файлы (например, все программы общего пользования, одного пользователя, все файлы текстового редактора, транслятора и т.д.).

При использовании на ПЭВМ только НГМД иерархическая файловая система не дает особого преимущества, но при наличии винчестерского НМД деление на каталоги очень удобно.

При древовидной структуре файлов надо знать не только их имя, но и указывать маршрут. Под последним понимается цепочка подчиненных каталогов на диске (он отделяется от собственного имени файла разделителем \ ). Таким образом, в ОС АДОС файл полностью задается элементами: именем накопителя, маршрутом и собственно именем файла. Например:

A: P1.PA – файл P1.PA находится на текущем каталоге накопителя A;
A: \ E1 T1.COM – файл T1.COM располагается в каталоге E1 первого уровня текущего накопителя;
A: \ C1 H1.TXT – файл H1.TXT находится в подкаталоге C1 текущего каталога накопителя A.

Так же, как и в ОС М86, в АДОС используются шаблоны. Например:

??.BAS – все файлы типа BA с одно- или двухбуквенными именами;
*.EXE – все файлы типа EXE;
. – все файлы текущего каталога.

Для работы с файлами в состав АДОС входят команды по выдаче каталога, отображения, переименования, удаления, копирования файлов (см. табл. 9.8). Кроме того, ДОС содержит команды, позволяющие работать с каталогами. Прежде всего для просмотра содержимого любого подкаталога используется команда КАТ (DIR):

C: \ ДОК \ ABC > КАТ \ P1 – выдается каталог P1, хотя текущим является каталог ABC второго уровня.

Для создания нового каталога используется команда НК (MD):

A: > \ РАБ \ НК B1 – создается каталог B1 в текущем каталоге РАБ накопителя A.

При создании новый каталог на диске занимает 4 К байт памяти. Пока в нем имеется хотя бы один файл, каталог удалить нельзя.

Для удаления пустого каталога используется команда УД (RD):

B: \ B1 > УД A1 – удаляется подкаталог A1 из текущего каталога B1.

Для смены текущего каталога используется команда СК (CD). Для перехода от текущего каталога к каталогу ABC первого уровня выдается команда A: \ >СК ABC.

Для смены текущего каталога используется команда СК (CD):

B: \ > СК ДОК \ AB – переход в каталог AB из каталога ДОК.

Для перехода на один уровень вверх выдается команда СК ..., для возврата на самый верхний уровень – команда СК \ .

Работа с файлами. Рассмотрим основные команды для работы с файлами в ОС АДОС для копирования (КОП), переименования (ИМЯ), удаления (УД) и вывода (ВЫВ) файлов.

Команда копирования имеет три основных формата: с сохранением имени, с переменой имени, для объединения файлов. Например:

A : \ > КОП *. * B : копирование всех файлов из текущего каталога диска A в текущий каталог диска B;

B: \ > КОП A1 A2 – файл с именем A1 дублируется в том же каталоге под именем A2;

A : \ > КОП A1+A2 A3 – файл A3 образуется путем объединения файлов A1 и A2.

Для переименования файла должны задаваться старое и новое имя (или шаблоны). Например:

A : \ РАБ \ ИМЯ 1: *. txt *. ДОК – происходит изменение типов файлов, находящихся на диске A : , вместо типа txt устанавливается тип ДОК;
B : \ РАБ \ ИМЯ Т ?? . * Р ?? .* – происходит изменение буквенных имен, начинающихся с Т (вместо Т подставляется Р).

При удалении файла из текущего или указанного каталога аргумент команды УД может содержать маршрут, имя или шаблон имен. Например:

B : \ > УД \ РАБ *. EXE – удаление из каталога РАБ всех файлов, имеющих тип EXE;
A : \ > РАБ \ УД – удаление всех файлов из текущего каталога.

При удалении файла происходит вычеркивание из каталога ссылки на него, содержимое файла не уничтожается. Для вывода содержимого текстового файла на экран по команде ВЫВ аргументом может быть только полное имя файла. При этом предполагается, что в файле содержится информация (буквы, цифры или знак, представляемый на экране). Например:

2 : \ > ВЫВ PPOG 1.PAS (вывод файла PPOG1).

На экран не могут быть выведены файлы типов ИСП, КНД, COM, EXE, OBJ . Тексты с файлами программ на ПАСКАЛе, ФОРТРАНе, СИ обычно снабжаются соответствующими типами PAS, FOR, C. Рабочие файлы также непригодны для вывода на экран.

Файлы в машинном коде и командные. Машинный код команды хранится в файлах типа .COM, .EXE. Первые создаются отдельными трансляторами, вторые – редакторами связей. Для этих файлов существуют различные способы загрузки в оперативную память и настройки на фактические адреса. Для запуска программы в машинном коде достаточно указать имя файла без расширения. Некоторые программы могут требовать при загрузке входные параметры.

Командные файлы являются исполняемыми и имеют расширение .BAT. Запускаются они с указанием имени файла без расширения, с требуемыми параметрами и содержат целую группу команд ДОС (обращения к прикладным программам, команды для выдачи информации на экран, метки и команды для организации ветвлений и циклов). Пусть имеем файл C.BAT:

```
ЭХО Нет                          echo off
:М1                              :m1
ЭХО Вывод на принтер файла %1    echo...
ЭХО Для остановки нажмите УПР-С  echo...
КОП %1 пч                        copy %1/prn
ПАУЗА                            pause
НА М1                            goto m1
```

Слева приведен файл с использованием команд в русской нотации, справа – в английской. Вторая строка содержит метку, а последняя – команду перехода на эту метку. Четыре команды внутри повторяемого участка выдают на экран сообщения копирования на принтер файла с именем %1 и О приостановлении работы для возможности прервать процесс печати. Для запуска команды выполнения данного файла выдается команда A >C B1.ДОК. По ней на принтер будут поступать копии файла B1.ДОК до тех пор, пока пользователь не прервет процесс, нажав клавиши УПР-С.

Команда ЕСЛИ (IF) позволяет проверить условие и выполнить команду в зависимости от результата проверки. Например, if errorlevel 4 goto M.1. В этом случае проверяется код завершения программы, если он равен 4, переход на программу M1, иначе следующая команда.

Команда ДЛЯ ... ИЗ ... ВЫП (FOR...IN...DO) обеспечивает циклическое выполнение команд АДОС. Пусть необходимо систематически копировать файлы P1.ASM, P1.OBJ, P1.EXE из рабочего каталога на виртуальный диск D. Для этого в командном файле необходимо записать команду

```
for %%A in (ASM OBJ EXE) do copy PROG %%A d;
```

Здесь формальный параметр %%A сопоставляется со списком фактических параметров в круглых скобках и используется в команде COPY, которая выполняется три раза подряд. Вместо d можно задать также фиктивный параметр %1, тогда назначение копирования можно вводить с клавиатуры.

Для конфигурирования системы и начальной настройки используется два специальных файла CONFIG.SYS и AUTOEXEC.BAT. Эти файлы обрабатываются при включении или перезапуске машины автоматически.

С помощью файла конфигурации можно расширить ОС и изменять некоторые параметры (число буферов, одновременно открытых файлов). Например:

```
break=on
files=12
buffers=8
devise=c:\sys\vdisk.sys 20
```

В первой строке файла устанавливается режим, при котором можно прервать любую работающую программу, нажав клавиши УПР-С в момент операции ввода-вывода (печать, обмен с диском). Второй командой файла устанавливается 12 одновременно открытых файлов, что требуется при работе, например, с базой данных. Третьей командой устанавливается четыре буфера для обмена с диском (по умолчанию 2). Наконец, по четвертой команде к ОС подключается драйвер виртуального диска, под который отводится часть оперативной памяти.

Файл AUTOEXEC является командным, но обрабатывается ОС при инициализации вслед за файлом конфигурации. При этом в него удобно занести команды,которые автоматически создадут для пользователя привычную операционную среду. В файле автозагрузки применяются команды PATH, PROMP, SET, VEP и др. Рассмотрим пример:

```
echo off
path c:\;c:\exe
prompt  $p$q
set AB=c:\ab
```

По команде PATH устанавливаются альтернативные маршруты для поиска исполняемых файлов – подкаталог EXE. По команде PROMP задается формат приглашения ОС ($p – выдача имени текущего каталога, $q – выдача символа >). По команде SET вводится имя AB с параметром C : \ AB, которое является указанием текстовому процессору о дополнительных файлах.

Блок управления файлом. Этот блок обеспечивает связь пользовательской программы с функциями ДОС. При выполнении любой файловой операции к

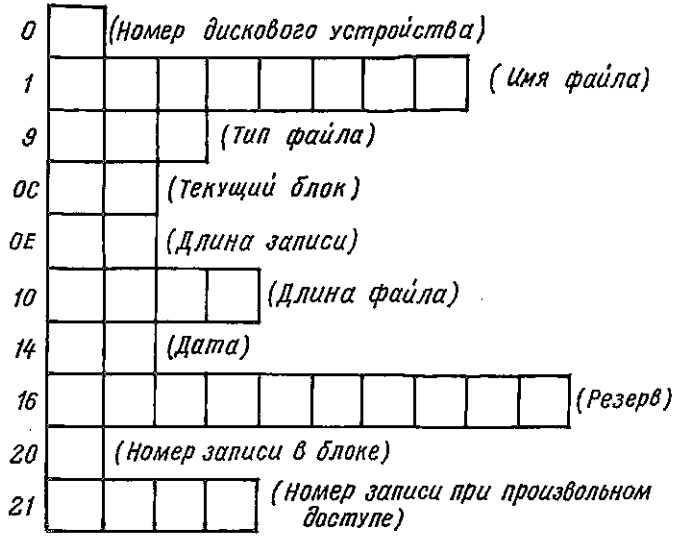


Рис. 9.6

нему производится обращение. На рис. 9.6 приведена структура блока, включающая описание всех атрибутов файла. Номер дисковода, имя и тип файла образуют идентификатор; размер файла и дата являются его атрибутами. Остальные поля определяют место внутри файла при чтении-записи. Существуют последовательный и произвольный способы доступа к записям. В первом случае ДОС наращивает на единицу номер записи, во втором номер записи определяется в программе и передается в FCB .

Рассмотрим пример программы, использующей блок FCB и функции ДОС:

```
   code      segment
             org 05Ch
fcb   label byte;метка поля fcb
fcb_dr       db  ?номер устройства
fcb_name     db8dup(?);имя файла
fcb_ext      db 3dup(?);тип файла
fcb_bl       dw   ?;номер тек.блока
fcb_rec_size dw   ?;размер записи
fcb_fil_size dd ? ;размер файла
fcb_date     dw ? ;дата изменения.
fcb_reserv   dd 10 dup(?);резерв дос
fcb_car_rec  db ?;номер тек. записи
fcb_rfn_rec  dd   ?;номер записи
                    ; прямого доступа
                  org  90h
disk_tran_adr label byte;буфер данных
                  org 100h
       assume cs:code,ds:code,es:code
                  jmp    progr_start
rec_size    equ  32;размер записи
file_err_mes db'файл существ.',10,13,'$'
bad_open_mes db 'ош.откр.файла',10,13,'$'
bad_wr_mes db 'ош. зап.в файл',10,13,'$'
bad_rd_mes db 'ош. чт.файла',10,13,'$'
```

```
bad_cl_mes db 'ош. закр. файла',10,13,'$'
progr_start:mov ah,1ah;установка буфера
                lea dx,disk_tran_adr
                int 21h
      ;-----поиск файла
           mov   ah,11h   ;поиск файла с
           lea   dx,fcb   ;заданным именем
           int   21h
           or    al,al
           jnz    no_file
           lea   dx,file_err_mes
 err_exit  mov ah,9h ;вывод сообщения
           int 21h  ;на экран и выход
           int 20h  ; из программы
       ;создание нового файла
 no_file   mov   16h       ;создание
           lea   dx,fcb    ;файла
           int    21h
           or    al,al
           jz    create
           lea dx,bad_open_mes;сооб.ошибки
           jmp err_exit;при создании файла
        ;установка параметров fcb
 create:     mov fcb_car_rec,0;инициализация
             mov word ptr fcb_ran_rec,0
                              ;номера записи и
       mov word ptr fcb_ran_rec+2,0;размера
   mov word ptr fcb-rec_size,rec_size; записи

        ;запись в файл
           mov     al,'0';в цикле вводятся
  char_l   lea     dx,disk_trans_adr;цифры
           mov     cx,rec_size
           rep     stosb ;заполнение
           push    ax    ;буфера
           lea     dx,fcb;последовательный
           mov     ah,15h;вывод
           int     21h   ;блока символов
           or      al,al
           pop     ax
           jz      write_f;
           lea     dx,bad_write_mes;ошибка
           jmp     err_exit        ;записи
  write_l  inc     al      ;след.цифра
           cmp     al,'9'+1;окончание
           jne     char_l  ;вывода
         ;чтение записи
           mov     al,'5';номер записи
           mov     ah,0
           sub     al,'0';преобразование
           mov     word  ptr fcb_ran_rec,ah
           lea     dx,fcb
           mov     ah,21h;прямое чтение
           int     21h
           or      al,al ;проверка ошибки
           jnz     ran_rec
           lea     dx,bad_rec_mes;сообщение ош.
           jmp     err_exit
   ;конец программы
   prog_exit:mov ah,10h ;закрытие файла
             lea dx,fcb
             int 21h
```

```
            or  al,al  ;проверка ошибки
            jz  close_ok
            lea dx,bad_close_mes;сообщение
            jmp err_exit
close_ok    int 20h    ;выход в дос
code        ends
            end
```

листинг 9.1

Приведенная программа сначала создает файл из 10 записей по 32 символа в каждый. Запись включает набор цифр (первая – "00...0", вторая – "11...1" и т. д.). Файл формируется последовательно. Во второй части программы он рассматривается как файл с произвольным доступом. В ответ на введенный пользователем с клавиатуры запрос программа считывает и выводит на экран одну из 10 записей. Выполнение программы завершается при вводе символа $. Для завершения программы используется прерывание 20H.

Для файла типа .COM, в котором находится программа, ее начало в сегменте имеет смещение 100H. Смещение блока FCB в программе равно 5CH, запись в FCB информации осуществляется командным интерпретатором. Запуск данной программы выполняется по команде A >F1 T1.FIL, где F1 – имя программы, а T1 – имя создаваемого файла. Интерпретатор выбирает имя T1.FIL и помещает его в поле блока FCB со смещением 5CH. Далее текст программы соответствует структуре блока, причем каждое поле имеет имя для обращения к нему. С адреса 90H записывается адрес области связи с диском (буфер), длина которой 128 байт, имя DTA.

Первая команда примера выполняет переход на начало программы. Далее следуют описания используемых в программе данных, сообщений об ошибках.

Сначала производится поиск файла с указанным именем. Так как наш файл создается, то при наличии введенного имени файла выдается ошибка.

Создание файла выполняет участок программы, помеченный NEW_FIL. Если операция не выполнена (отсутствие места на диске или в каталоге), то выдается сообщение об ошибке. При любом обращении ДОС к файлу происходит его открытие, в процессе которого устанавливается связь между ОС и ПП и заполняется поле FCB длины файла. В нашей программе открытие файла происходит при его создании. Для существующего файла эта операция выполняется по 0FH (функция ДОС).

В части программы CHAP_L в файл записывается 10 записей, каждая из которых передается в буфер DTA оператором REP STOSB. На диск записи пересылаются с помощью функции последовательной записи 0AH, при этом проверяется наличие ошибок.

Далее следует чтение произвольной записи, для чего пользователь вводит начальный символ (от 0 до 9). Программа находит соответствующую запись, пересылает ее в буфер, а затем выводит на экран ее содержимое. Закрытие файла производится по функции ДОС 10H.

Приведенный пример иллюстрирует основные способы обращения к файлу с помощью функций ДОС. Необходимо проверять ошибки после выполнения каждой функции ДОС. Для этого анализируется код возврата в регистре AL, если он не равен 0, происходит вывод сообщения об соответствующей ошибке на экран, при этом используется функция ДОС 9H.

10. МНОГОПОЛЬЗОВАТЕЛЬСКИЕ И МУЛЬТИПРОЦЕССОРНЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ

10.1. ОСНОВНЫЕ ПОЛОЖЕНИЯ

Многопользовательская ОС. *Многопользовательской* называется система, которая может обслуживать запросы двух и более пользователей. При этом каждому потенциальному пользователю выделяется виртуальная машина, обладающая "собственной" памятью, файловой системой и независимыми устройствами ввода-вывода. Однако при наличии аппаратной части (одного процессора и неделимого пространства оперативной памяти) каждому потенциальному конкретному пользователю выделяется раздел оперативной памяти и временной промежуток, в течение которого ресурс центрального процессора принадлежит только ему.

Разделение ресурсов системы между пользователями осуществляется специальными аппаратно-программными средствами, обеспечивающими также защиту разделов памяти от несанкционированного доступа. Если в системе не предусмотрены средства связи пользователя, то ни один из них не знает, что имеются другие пользователи. В современных многопользовательских системах есть специальные средства, позволяющие пересылать сообщения от одного пользователя к другому. Таким образом, если два пользователя в системе не обращаются к общим разделам внешней памяти и "не знают" о существовании друг друга, то один для другого представляет фоновую задачу, которая выполняется независимо от текущей явной для пользователя.

Принципы работы мультипрограммной и многопользовательской систем аналогичны, но в первой все программы выполняются под контролем одного пользователя (непосредственно пользователь может управлять только одной программой, остальные скрыты от него и реализуются в режиме с более низким приоритетом, чем текущая).

В многопользовательских и мультипрограммных системах, кроме функций управления вводом-выводом, интерфейса с пользователем, реализуется распределение разделяемых ресурсов системы (центральный процессор, память и периферийные устройства). Эти задачи решаются с помощью специальных аппаратно-программных средств для распределения системных ресурсов. Эти средства формируются при инициализации и доступны только системному программисту (обычному пользователю средства распределения ресурсов системы в явном виде не доступны). Распределение ресурса ЦП системы осуществляет таймер, который через равные промежутки времени вырабатывает сигнал системного прерывания, предписывающего процессору переключиться с одной задачи на другую.

Обслуживание устройств ввода-вывода реализуется следующим образом: каждой задаче, существующей в системе, выделяется виртуальное устройство, которое отображает свойства физических ВУ. При планировании процедур за-

дачи распределения ЦП системы осуществляются только в том случае, если все запросы на ввод-вывод удовлетворены, в противном случае ресурс центрального процессора системы не выделяется. Состояния процессов в системе аналогичны описанным в § 7.1.

Отличие многопользовательских и мультипрограммных систем от однопользовательских заключается в том, что каждая задача в последней системе может быть представлена как фрагмент некоторой системной программы. Системная программа – это процесс, который распределяет ресурсы системы между задачами пользователей или параллельными задачами в мультипрограммных системах. Очень приближенно многопользовательскую или мультипрограммную систему можно представить как процедуру, последовательно вызывающую другие процедуры. Сложность в реализации мультипрограммных и многопользовательских задач заключается в реализации такого алгоритма системной задачи, который позволял бы распределять системные ресурсы так, чтобы минимально загружать их реализацией распределения.

Если в однопользовательской и однопрограммной системах распределение ресурсов происходит статически в момент описания системы и ее создания, то в многопользовательских и мультипрограммных– по алгоритму распределения ресурсов системы на весь период ее развития. Однако как в многопользовательских, так и в мультипрограммных системах основным виртуальным элементом является процесс, который обладает собственной процедурой над его данными.

Распределение памяти. Если в однопользовательских и однопрограммных системах распределение памяти происходит статически с помощью указателей границ памяти, то в многопользовательских системах (в связи с изменяющимся состоянием процессов) динамически, что требует постоянного контроля за состоянием свободных и занятых областей памяти. В однопользовательских системах ISIS-II,CP/M-80 регистрация границы системной памяти осуществляется с помощью специализированных ячеек памяти, причем в системе ISIS-II нижняя граница задается размером ядра операционной системы, а верхняя устанавливается в момент инициализации системы, что позволяет применять различные размеры системной памяти для работы одного и того же ядра. Аналогичная логика определения границ памяти используется в системе CP/M с тем отличием, что верхней границей памяти считается точка входа в ДОС, а нижняя жестко зафиксирована. Общим у этих двух систем является то, что границы системной памяти определяются в момент инициализации системы и в процессе ее развития не меняются.

В многопользовательских ВС системная память значительно большего объема и делится на фиксированные блоки, которые записываются специальными регистраторами. При инициализации системы программы ядра ОС определяют граничные адреса, которые информируют о том, каким объемом памяти располагает система. Вся зарегистрированная в системе область памяти разделена на блоки, которые выделяются процессу при его инициализации. При занятии блока памяти его состояние отмечается в регистраторе свободных блоков, чтобы исключить повторное занятие блока другим процессом. Взаимодействие процесса с регистраторами памяти системы аналогично взаимодействию процесса с устройствами внешней памяти, что отражается командами виртуально-

го процессора ОС. Механизмы распределения блоков памяти были рассмотрены в § 7.4.

Многопроцессорные ОС. По мере удешевления БИС микропроцессоров появилась возможность обеспечить каждого пользователя собственным процессором в системе. Однако расширение областей применения вычислительной техники привело к необходимости введения в системы дополнительных функций, реализация которых ранее на ВС не возлагалась. Если прежде пользователю в системе предоставлялся виртуальный компьютер, то теперь появилась возможность предложить ему несколько виртуальных компьютеров. Для мультипроцессорных вычислительных систем по способу функционирования ОС разделяются на три типа: централизованные, децентрализованные, смешанные.

При организации ОС *централизованного* типа супервизорные функции возлагаются, как правило, на один из процессоров системы, реализующий функции управления. ОС централизованного типа несложна в реализации и максимально совместима с ОС однопроцессорных систем. Однако необходимо заметить, что если супервизорные функции возлагаются на один процессор, то от его надежности зависит надежность всей системы. Несомненным достоинством ОС централизованного типа является простота всей системы в целом.

В *децентрализованной* ОС каждый процессор, входящий в систему, имеет собственную копию ОС (или ее части), что позволяет работать ему как автономному устройству. Супервизорные функции ОС реализуются каждым процессором. Очевидно, что при отказе какого-либо из процессоров работоспособность системы сохраняется. Известные децентрализованные системы организуются из процессоров, имеющих определенную функциональную направленность, что ставит перед разработчиком и пользователем задачи эффективного использования всех ресурсов системы, т. е. задачи равномерной загрузки каждого входящего в систему процессора.

Как правило, ОС *смешанного* типа реализуется в том случае, если система состоит из множества однотипных процессоров, разделяющих общую память и средства ввода-вывода. Каждый из процессоров может выполнять как супервизорные, так и исполнительные функции. Вместе с тем в таких системах многие функции обработки прерываний и ввода-вывода реализуются одновременно.

Наличие в ВС более чем одного процессора предполагает одновременное выполнение различных задач. Однако и в однопроцессорной ВС, работающей в многопользовательском или мультипрограммном режиме, происходит квазипараллельное выполнение задач различных пользователей. Поскольку реальная одновременность вычисления различных процессов не реализуется, то часто в многопользовательских системах можно наблюдать эффект "зависания" системы при ее большой загрузке.

Для повышения эффективности работы ЦП вычислительной системы, построенного на основании модели фон Неймана, уже в первых ЭВМ была введена возможность одновременного выполнения операций счета и выборки данных из устройств памяти и ввода-вывода. Подобный подход построения параллельной работы получил свое развитие во многих современных вычислительных системах. В частности, в микропроцессорном наборе К1810ВМ86, кроме ЦП, обеспечивающего реализацию операций обработки данных и имеюще-

го достаточно обширную библиотеку команд работы с устройствами памяти и ввода-вывода, введены микросхемы вспомогательных микропроцессоров, специализированных на выполнение операций арифметики и обработки устройств ввода-вывода. Процессоры в ВС, ориентированные на какой-нибудь фиксированный набор функции, называются сопроцессорами для системного процессора. Реализация сопроцессоров в мультипроцессорной системе позволяет существенно увеличить ее быстродействие и эффективность.

Принцип параллелизма. Следует сразу условиться, что понимается под параллелизмом. Считаем, что любые процессы, не связанные между собой, являются одновременными или сопроцессами в отличие от процессов, взаимодействующих друг с другом. Процессы, реализация которых осуществляется в определенном порядке (каждый последующий фрагмент связан с предыдущим), являются последовательными. Таким образом, системный процесс, распределяющий ресурсы ВС, будет для каждого реализуемого процесса сопроцессом. Их взаимодействие происходит только на системном уровне, и реализация алгоритма планирования и распределения ресурсов не зависит от текущего состояния задачи. Вместе с тем процессы, реализующие вычисление определенных фрагментов арифметического выражения, являются параллельными, поскольку начинаются они в одной точке и продолжают свое развитие независимо друг от друга. Основное различие параллельных процессов от сопроцессов заключается в том, что последние никогда не взаимодействуют друг с другом. Если разделить процесс на ряд подзадач, то фрагменты одного и того же процесса могут выполняться последовательно и параллельно в зависимости от уровня иерархии системы, на котором они реализуются в настоящий момент.

Объектно-ориентированные системы. Дальнейшим развитием принципов построения мультипрограммных и многопользовательских ОС, которые являются процедурно-ориентированными, будет организация объектно-ориентированных ВС. Реализация процедурно-ориентированных систем требует достаточно большого количества вспомогательных операций, связанных с контролем за развитием процесса в системе. Поэтому в настоящее время получили развитие методы организации вычислительных систем, призванные снизить необходимое количество операций контроля за развитием процессов и упростить представление системы при работе с ней и ее проектировании. Такими системами являются объектно-ориентированные. В настоящее время они реализуются на том же аппаратном базисе, что и процедурно-ориентированные. Если в процедурно-ориентированных системах базовым элементом является процесс, т. е. развитие понятия процедуры на системном уровне, то базовым элементом в объектно-ориентированной системе будет объект.

Отличие объекта от процедуры на логическом уровне заключается в том, что он представляет собой завершенную подсистему с заложенной в него логикой развития в момент инициализации. Объект как элемент ВС должен иметь собственную совокупность процедур, его реализующих в среде автомата фон Неймана, собственные устройства ввода-вывода, собственную систему внешней памяти. Развитие всех элементов, составляющих объект, происходит независимо от логики развития других объектов, входящих в систему. Взаимодействие объектов на системном уровне осуществляется физическими и логическими устройствами ввода-вывода, имеющимися в распоряжении объекта.

Таким образом, при построении объектно-ориентированной системы в среде автомата фон Неймана описание объекта в отличие от описания процесса (см. § 9.1) должно указывать на совокупность устройств ввода-вывода и фрагмента файловой системы (внешней памяти), что в процедурно-ориентированной системе регистрируется на уровне ядра ОС.

По аналогии с процедурно-ориентированными системами для сохранения информации об объектах вводится таблица управления, каждый элемент которой включает информацию (имя объекта, текущее состояние процедуры объекта, ссылку на область стека объекта, ссылку на блок устройств ввода-вывода, ссылку на фрагмент внешней памяти) в виде:

Имя объекта | Состояние процедуры | Ссылка на стек | Ссылка на УВВ | Ссылка на внешнюю память | Ссылка на зависимые объекты |

Под зависимыми объектами в таблице понимаются объекты, порожденные данным объектом или его породившие.

В настоящее время известны две основные ВС, которым присущи черты объектно-ориентированной системы: 1) операционная система iPMX-86, предназначенная для организации вычислительных систем на базе одноплатных компьютеров фирмы Intel; 2) аппаратно-программная система на базе микропроцессора iAPX 432 той же фирмы, в которой организация объектов поддерживается на аппаратном уровне.

В дальнейшем будем использовать логику построения объектно-ориентированной системы на многопроцессорной модели, в основе которой лежит модель автомата фон Неймана.

10.2 ОРГАНИЗАЦИЯ ЯДРА ОПЕРАЦИОННОЙ СИСТЕМЫ МНОГОПРОЦЕССОРНОЙ МИКРОЭВМ

Функции ОС. Операционная система многопроцессорной вычислительной системы выполняет по сравнению с однопроцессорной ряд дополнительных важных функций, обусловленных как назначением системы, так и конфигурацией ее аппаратно-программных средств. К таким функциям относятся: распределение и управление ресурсами; синхронизация процессов; управление процессорами, памятью; защита данных от несанкционированного доступа; разрешение конфликтных ситуаций.

Как указывалось (см. § 10.1), принципы построения многопользовательских, мультипроцессорных и объектно-ориентированных систем содержат много общего. Будем рассматривать далее в качестве модели мультипроцессорную ВС с децентрализованным управлением.

Конфигурация модели. Физическая конфигурация модели включает аппаратную часть. Допустим, что модель мультипроцессорной системы состоит из двух или более эквивалентных высокопроизводительных микропроцессоров и общего блока памяти, равнодоступного обоим процессорам. При этом каждый процессор имеет собственную локальную память с копией операционной системы или ее части, а система обладает достаточным набором устройств ввода-вывода.

Логическая конфигурация системы. В качестве логической конфигурации системы примем модель, которая известна как справочная модель для связи открытых систем (ISO) (рис. 10.1).

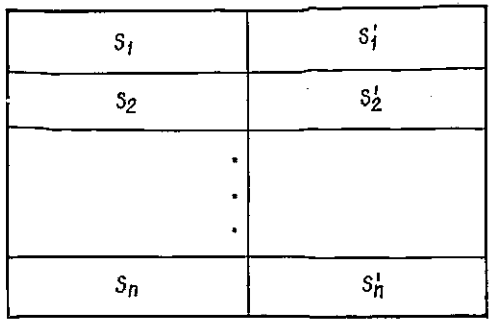


Рис. 10.1

Очевидно, что в мультипроцессорной системе могут быть реализованы как параллельные процессы, так и сопроцессы. Однако, для того чтобы на каждом процессоре существовал "свой" приписанный только ему процесс, необходимо иметь систему с очень большим количеством процессоров. Такая реализация представляется достаточно дорогой и нерентабельной. Поэтому будем считать, что в системе в общем случае процессов больше, чем процессоров. Таким образом, параллелизм или одновременность реализации процессов в системе достигается ее логической организацией. В соответствии с приведенной логической моделью системы реально параллельными могут быть два процесса, выполненные на различных процессорах. Состояние процессов, отображающих объекты в системе, аналогично возможным состояниям процессов, приведенных в § 7.1. Таким образом, в системе необходима реализация двух типов распределения ресурсов – распределение ресурсов процессоров между процессами и распределение ресурсов системной памяти и устройств ввода-вывода. В дальнейшем будем считать, что в системе существуют две надсистемные процедуры распределения ресурсов системы – планировщик процессов и диспетчер устройств ввода-вывода. По аналогии с получившими в настоящее время распространение ОС будем представлять ядро ОС двумя взаимосвязанными, но функционально различными частями: базовой основной (файловой) операционной системой BDOS; базовой системой ввода-вывода BIOS.

Функционально ОС состоит из нескольких иерархически связанных программных модулей: планировщика системы, диспетчера системы, распределения ресурсов памяти, примитивов файловой системы, процессов, отображающих объекты пользователя.

При прохождении задачи по иерархии системы каждый уровень ядра регистрируется в планировщике как отдельный процесс. При переходе процесса от одного уровня иерархии к другому ресурс процессора перераспределяется следующему по очереди процессу из находящихся в состоянии готовности. Этим достигается выравнивание прохождения процессов в ядре ОС и исключение состояний блокировки одного процесса другим, так как несколько процессов находиться на одном уровне иерархии не могут. Процесс продвигается по уровням ядра ОС только в том случае, если соседний необходимый уровень иерархии свободен. Все процессы, находящиеся в обработке в ядре ОС, параллельны.

Рис. 10.2

Базовая операционная система. Небольшой по объему, но важный по значению модуль системного интерфейса служит для обеспечения взаимодействия между процессами, развивающимися вне ядра ОС, и ядерными процессами. Этот модуль регистрирует параметры, передаваемые процессом ядру, ОС и планирует прохождение процесса по уровням иерархии ядра ОС. Все системные вызовы первоначально попадают в модуль системного интерфейса. По таблице функций ОС необходимые параметры процесса передаются на следующий уровень ядра ОС. При реализации команды УНИЧТОЖИТЬ ОБЪЕКТ управление передается не на следующий уровень иерархии, а непосредственно системному планировщику, который исключает процесс объекта из списка процессов и регистрирует освободившуюся область памяти.

Планировщик системы. Все без исключения процессы в системе находятся под управлением планировщика системы. В его функции входит распределение ресурса процессоров процессам во времени и защита распределенных процессам областей памяти от несанкционированного доступа.

Планировщик запоминает текущее состояние процесса и, восстановив состояние следующего по очереди процесса, передает ему управление. Для осуществления этих операций каждый объект в системе ассоциируется с несколькими специализированными областями памяти: блоком управления процесса (БУП); областью памяти процесса (ОПП); дескриптором процесса (ДП).

Необходимо заметить, что область памяти процесса обязательно содержит область стека. В структуре ОПП планировщиком сохраняются текущие состояния процесса и оперативные данные, необходимые ему во время работы.

Структура ДП доступна только планировщику. Она образуется при создании объекта и уничтожается при исключении его из системы. ДП содержит все необходимые ссылки на БУП и сведения о приоритете объекта в системе. Структуры БУП и ДП приведены на рис. 10.2, *а, б* соответственно.

Синхронизация процессов. Синхронизация процессов, отображающих развитие объектов в системе, происходит двумя способами:

1) по абсолютной шкале времени, т. е. для выполнения какого-либо действия процессу отводится фиксированный временной интервал (обычно его размерность находится в пределах около двух миллисекунд). Необходимо за-

метить, что размерность кванта времени для реализации части процесса на одном процессоре также равна приведенному интервалу, т.е. можно задержать реализацию процесса на фиксированный интервал времени, либо разрешить ему обращение к какому-либо ресурсу системы в строго определенный момент времени;

2) по нечеткой шкале времени с помощью сообщений между процессами. Передача сообщений между процессами осуществляется через специально выделенные логические устройства, которые в дальнейшем будем называть каналами.

Каналы представляют собой структуру данных, работа с которыми проходит аналогично работе с файлами. Как и файл, канал может быть создан, открыт (зарегистрирован для работы), уничтожен, из него может быть прочитана информация и в него она может быть записана.

Управление памятью. Управление памятью системы явно осуществляется только для разделяемой области памяти, т.е. области памяти, доступной всем процессорам в системе. Локальная память каждого процессора используется фрагментом ОС для собственных нужд и распределению и перераспределению не подлежит. По аналогии с ОС CP/M-86 и MS-DOS многопроцессорная система сопровождается моделью распределения памяти фиксированными блоками. Минимальный сегмент памяти, отводимый процессу, равен 32 К байт. При создании процесса по умолчанию ему отводится один сегмент памяти. Пользуясь системными вызовами, процесс может запросить необходимое количество сегментов памяти. Выбор свободного сегмента памяти из области свободной памяти происходит по принципу: первый найденный свободный сегмент является самым лучшим. По мере развития объекта в ОС отображающие его процессы могут занимать и освобождать сегменты внутренней памяти. Развитие объекта, не получившего необходимого объема памяти, приостанавливается до тех пор, пока не освободится достаточный объем системной памяти.

10.3. БАЗОВАЯ СИСТЕМА ВВОДА-ВЫВОДА

Виртуальные устройства. Базовая система ввода-вывода предназначена для организации обмена информацией между виртуальными процессорами и устройствами ввода-вывода. Система ввода-вывода реализует как физическое управление аппаратными устройствами, так и формирование протоколов обмена между виртуальными процессорами и физическими драйверами. Эта система, в отличие от блока BIOS распространенных операционных систем, характеризуется следующим образом. Если в ОС CP/M-86 и MS-DOS система ввода-вывода осуществляет реализацию физических и логических драйверов устройств, то в объектно-ориентированной системе BIOS реализует виртуальные устройства для имеющихся в системе объектов и распределяет ресурсы устройств ввода-вывода. Поэтому BIOS объектно-ориентированной системы представляет собой элемент ОС, обладающий собственными средствами распределения ресурсов как физических, так и виртуальных устройств ввода-вывода (в отличие от BIOS систем CP/M-86 и MS-DOS). Средства распределения устройств ввода-вывода будем называть диспетчером системы. На самом нижнем уровне – на уровне взаимодействия с устройствами ввода-вывода, являющимися резидентными для ВС, – реализация как логических, так и фи-

а

XXXX	XXXX	Флажки	Имя
канала...			Длина сообщений
Количество сообщений	XXXX	XXXX	XXXX
XXXX	Начальный адрес буфера		

б

XXXX	XXXX	Флажки	Имя
драйвера . . .			Длина сообщения
Ссылка на программу		XXXX	XXXX
XXXX	Начальный адрес буфера		

Рис. 10.3

зических драйверов устройств не отличается от описанных в § 7.5.

Наличие собственных средств распределения ресурсов устройств ввода-вывода позволяет регистрировать их в системе как резидентные. Подобное решение позволяет упростить реконфигурацию устройств ввода-вывода в ОС в случае необходимости.

Еще одно немаловажное отличие распространенных ОС CP/M и MS-DOS от BIOS заключается в том, что система ввода-вывода обеспечивает взаимодействие не только с физическими устройствами, но и с виртуальными, такими, как виртуальные устройства передачи данных – каналы. Каналы наравне с устройствами ввода-вывода регистрируются диспетчером системы и обслуживаются им. Для эффективного взаимодействия процессов объектов как с физическими, так и логическими устройствами ввода-вывода последние разделяются на три функциональных типа: 1) устройство ввода-вывода предназначено для обеспечения взаимодействия виртуальных процессоров с физическими устройствами ввода-вывода; 2) устройство канала связи предназначено для обеспечения передачи информации от одного виртуального процессора к другому; 3) окно связи предназначено для передачи информации от виртуального процессора к устройству вывода на дисплей консоли оператора.

Логика работы этих типов устройств приблизительно одинакова. Отличие заключается в том, что окно связи имеет фиксированную процедуру, обеспечивающую одностороннюю передачу. Кроме того, оно содержит специализи-

рованную процедуру, регистрирующую свободную область на дисплее консоли оператора. Процедуры, определяющие передачу информации драйверов и каналов связи, формируются пользователем в момент их инициализации в системе. Резидентные драйверы устройств ввода-вывода имеют фиксированные процедуры, отображающие их свойства, поэтому при обращении к ним указание процедуры не требуется. Формат дескрипторов устройств ввода-вывода приведен на рис. 10.3, *а, б* (*а* – канала связи, *б* – канала драйвера). Здесь флажки канала служат для указания его текущего состояния; имя канала – для идентификации канала в системе; длина сообщения – количество байтов в каждом логическом сообщении; количество сообщений – максимально возможное количество сообщений в буфере; ссылка на программу – точка входа в драйвер; начальный адрес буфера – адрес буфера канала. Размерность буфера определяется как произведение длины сообщения на их максимально возможное количество в буфере канала; XXXX – поля, зарезервированные для ядра ОС.

Диспетчер. Распределение ресурсов ввода-вывода выполняется следующим образом: во время создания драйверов ввода-вывода формируется вектор прерывания, в который записывается точка входа в программу диспетчера и адрес ссылки на описатель драйвера. Последний содержит две ссылки на две функционально различные процедуры – процедуру обработки принимаемых данных и процедуру обработки передаваемых данных. Если драйвер закрывается, в его описатель в соответствующее место записываются все нули, что воспринимается диспетчером как отсутствие канала приема-передачи.

На рис. 10.4 показана диаграмма Найси–Шнайдера для алгоритма логического драйвера вывода на консоль оператора.

Файловая система объектно-ориентированной вычислительной системы. Организация файлов и библиотек в достаточной степени хорошо разработана в ОС MS-DOS и на низком уровне может использоваться в объектно-ориентированной системе. Однако в последней существуют некоторые отличия в работе с файлами, которые необходимо учесть при организации файловой системы. Они заключаются в самом понятии файла, которое для процедурно-ориентированной системы представляет собой списочную структуру (см. гл. 9). При такой организации процедура, использующая файл как область временного хранения собственных данных, требует от ОС постоянной регистрации развития файла по его размеру, что и осуществляется в CP/M-86 и MS-DOS. В объектно-ориентированной системе объект организует не один файл, как в процедурно-ориентированной системе, а их совокупность. Объект, обращаясь к файловой системе, ассоциирует открываемый файл с целым разделом файлов, позволяющих регистрировать различные фрагменты развития объекта в системе. Таким образом, основное отличие файловой системы объектно-ориентированной ОС заключается в том, что создание файла в системе предполагает построение целой совокупности файлов, отображающих тот или иной фрагмент объекта. Подобная организация файловой системы в какой-то мере аналогична работе с библиотеками в системе MS-DOS или разделов пользователя в системе CP/M-86.

Кроме того, по логике работы в объектно-ориентированной системе объект, не имеющий возможности развития в настоящее время, выталкивается из разделов системной памяти, им занимаемой, в раздел внешней памяти. Там он находится до тех пор, пока в системе не будут затребованы необходимые дан-

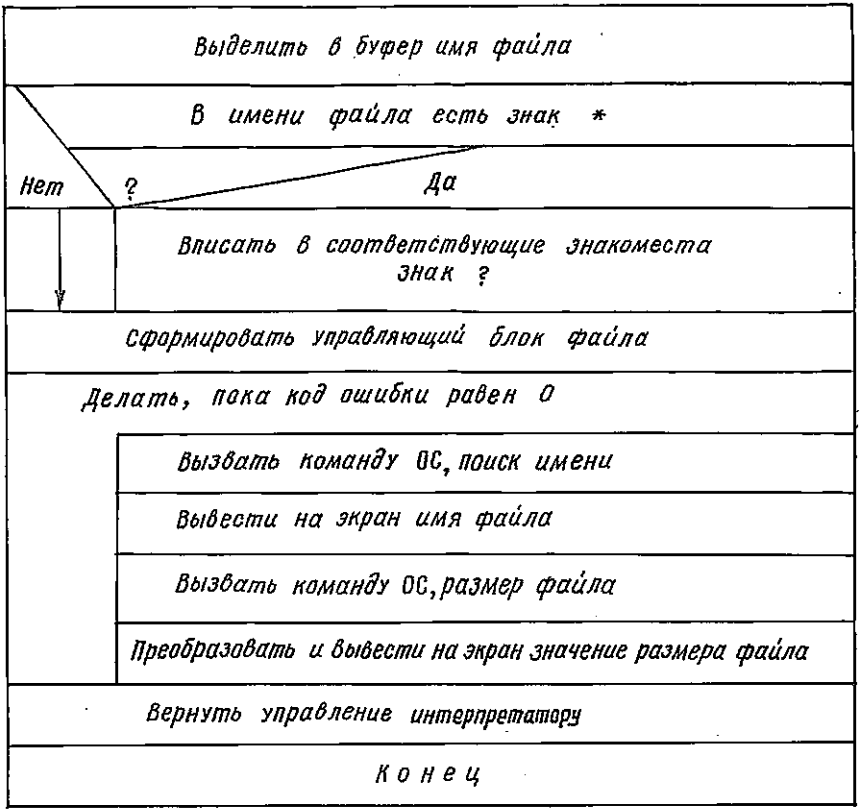


Рис. 10.4

ные, реализуемые этим объектом. Поэтому при инициализации любого объекта в ОС в файловой системе создается раздел, ассоциируемый с этим объектом. По мере развития и уничтожения объектов в ОС происходит постоянная модификация файлов. В связи с указанными особенностями логической организации файловой системы объектно-ориентированной ОС в ее командах существуют некоторые отличия, которые рассмотрим в § 10.5.

10.4. РАЗВИТИЕ ОБЪЕКТОВ В СИСТЕМЕ И УПРАВЛЕНИЕ ИМИ

Несмотря на то что объект для пользователя — это некоторый завершенный компонент с логикой развития, в самой ОС приняты некоторые соглашения о разделении функций объекта. В частности, любой объект, развивающийся в системе, представлен двумя составными частями, каждая из которых характеризует определенные его свойства. Одна часть объекта, определяющая совокупность его элементов, доступных для взаимодействующих объектов, представляет собой систему записей в основной и внешней памяти ПЭВМ, отображающих состояние элементов. В памяти ПЭВМ подобное отображение определяется совокупностью занимаемых элементов ячеек памяти и способом их

выбора из нее. Если провести аналогию с процедурно-ориентированными системами, то в простейшем случае элементы строки символов, отображенные в памяти, и число с плавающей точкой, представленное в определенном формате, могут занимать одно и то же количество байтов, но при обращении к этим областям данных процедура, реализующая определенные функции, "знает", какого типа данные находятся в этой области.

Так, совокупность данных с определенными правилами доступа (выбора) к ним будем называть объектом-носителем. Совершенно очевидно, что при реализации объекта в среде автоматов фон Неймана без дополнительных аппаратных затрат невозможно реализовать определенные правила доступа к элементу или их группе в основной памяти. Причем достаточно трудно определить, какой тип обращения может понадобиться пользователю для решения его задач. Поэтому под объектом-носителем *n*-го уровня абстракции будем понимать структуру данных, не имеющую процедур этого уровня. При реализации правил доступа к элементу объекта-носителя последний может определяться как совокупность процедур, позволяющих задать этот элемент в основной памяти ПЭВМ. Подобное представление объекта-носителя позволяет определять его элементы рекурсивными процедурами вплоть до самого низкого уровня абстракции системной памяти.

Вместе с тем взаимодействие объектов-носителей осуществляется через объекты, не имеющие собственных данных на этом уровне абстракции, что позволяет обеспечить преобразование передаваемых данных от одного объекта к другому. Объект, не имеющий собственных данных на определенном уровне абстракции, будем называть объектом-коммуникатором. Определение такого объекта в системе позволяет обеспечить формирование управляющих структур объекта независимо от состояния объектов-носителей, тем более что свойства объектов по отношению друг к другу определяются или проявляются только в момент их взаимодействия.

Таким образом, в системе на любом уровне абстракции существуют два типа объектов: объекты-носители и объекты-коммуникаторы, полностью определяющие ее поведение на этом уровне. Очевидно, что объект-носитель данного уровня абстракции может иметь теоретически бесконечное число объектов-коммуникаторов для организации связи с объектами-носителями как данного уровня, так и соседних.

Для равномерного развития объекта в системе необходима одновременная реализация обоих типов объектов, причем не квазипараллельная, а действительно параллельная на всех уровнях иерархии.

Поскольку в ОС существуют два типа объектов, система команд виртуальных процессоров требует включения видов для каждого из них.

Команды управления объектами. По аналогии с системами MS-DOS и CP/M86 управление объектами-носителями аналогично управлению процессами.с точки зрения команд ОС, при этом данная ОС создает паспорт объекта-носителя, в котором регистрируется информация о его размещении на ресурсах системы. Команды управления объектами-коммуникаторами аналогичны командам управления элементами внешней памяти системы и имеют три типа дескрипторов (см. § 10.3).

Регистрация объектов в системе. Информация об областях кодов данных и области стека генерируется системным загрузчиком, который на основании

описателей на диске выполняет запрос к регистратору памяти системы и формирует собственно объект в системе (если есть свободная и достаточная по объему область памяти).

Для этого первые 256 байт памяти, отведенной объекту, заполняются информацией, необходимой регистратору памяти системы и планировщику процессов, а также служебными данными об организации объектов-коммуникаторов. В этой области (256 байт) содержится следующая информация: размер области кодов процедур объекта; адрес сегмента кодов; длина данных (3 байта); адрес сегмента данных (2 байта); длина области EXTRA (3 байта); адрес сегмента EXTPA (2 байта); длина области стека (3 байта); адрес сегмента стека (2 байта); служебная информация планировщика, содержащая также и номер дисковода, который данный объект использует по умолчанию (90 байт); область описателей управляющих блоков (38 байт); буфер данных для обменов в объектах-коммуникаторах (128 байт).

Таким образом, описатель объекта полностью определяет среду, в которой реализуется объект.

10.5. ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ И ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ

Командный объект. Элементарный объект в системе для пользователя представляется состоящим из двух частей – объекта-носителя и объекта-коммуникатора, который в свою очередь состоит также из двух составных частей (командного и коммуникационного объектов).

Основное требование к командному объекту – обеспечение интерфейса между пользователем и ОС. Основные функции, которые выполняются командным модулем объекта, следующие: прочесть команду, проверить ее, идентифицировать, выполнить. Командный модуль объекта находится под управлением ОС как часть командного интерпретатора системы. При необходимости установления связи с пользователем командный интерпретатор генерирует подсказку для пользователя, что объект готов к коммуникации. После переключения пользователя на командный уровень объекта пользователь вводит строку команд для объекта. Командный интерпретатор затем выполняет перечисленные действия.

Чтение команды. Чтение команды интерпретатором ОС переводит сообщение, введенное в терминах командного интерпретатора, в коды сообщений объекта. Результатом работы командного интерпретатора при чтении команды является физический адрес строки символов, введенных пользователем. Проверка команды заключается в "сжатии" кодов введенной строки, т. е. в замене кодов пробела принятым в программе интерпретатора разделителем.

При идентификации команды используется информационная строка, "сжатая" модулем проверки команды для ее распознавания. Команда идентифицируется в следующих случаях: 1) комбинация литер соответствует имени команды в списке резидентных команд интерпретатора команды; 2) комбинация литер представляет собой имя файла.

Если в списке резидентных команд ОС найдена соответствующая комбинация, то комбинация в строке команд заменяется соответствующим псевдокодом для следующего модуля интерпретатора. Иначе формируется описатель

возможного процесса с соответствующими запросами в блок управления памятью системы и системный планировщик. Это необходимо для того, чтобы не сканировать одно и то же имя дважды. Если имя не идентифицировано ни в списке резидентных команд интерпретатора, ни в файловой системе, то пользователю выдается сообщение о допущенной ошибке ввода в той части, которая не была распознана.

Выполнение команды. Модуль выполнения команды использует информационную строку, полученную после предыдущего модуля, содержащую необходимые описатели и ссылки на предполагаемые программы. В первую очередь для любого вновь создаваемого объекта проверяется его контекст и среда, в которой он будет существовать. По умолчанию все вновь создаваемые объекты (процессы) считаются существующими в режиме PAR (параллельно). Таким образом, если вновь реализуемому процессу не предшествовало никаких описателей, то модуль выполнения команды приписывает ему состояние параллельной обработки.

Если во введенной команде существует описатель процесса, отличный от оператора PAR, SEQ или ALT , т. е. последовательная или чередующаяся обработка, то проверяется необходимость связи объекта с другими, уже существующими в системе и создаются необходимые объекты связи (каналы).

Необходимо помнить, что параллельный процесс, являющийся сопроцессом по отношению ко всем другим процессам в системе, регистрируется интерпретатором как процесс (объект), имеющий два канала связи – канал инициализации процесса и канал исключения процесса из системы. Эти каналы аналогичны семафорам и записываются в планировщике процессов системы. Явно управлять пользователь этими каналами не имеет возможности.

При создании процессов, отличных от параллельного процесса с описателями типа SEQ и ALT, системой создаются два канала доступных интерпретатору команд системы (через ОС), также являющихся семафорами, но позволяющих передать сообщение о начале либо окончании процесса другому процессу (процессам).

При необходимости передачи сообщения от одного процесса другому пользователь может явно указать канал, по которому коммутируют процессы (CHAN). При этом описателем CHAN пользователь может создать постоянно действующий канал, через который проходят коммутации многих процессов. Межобъектный канал связи существует в системе постоянно на правах процесса (объекта) связи, если объявлен пользователем. Такой механизм позволяет организовать обмен между процессами, не существующими одновременно в системе.

Итак, для организации обмена между объектами в системе используются четыре команды, указывающие очередность реализации процессов и установление связи с другими объектами в системе: PAR, SEQ, ALT, CHAN . Необходимо помнить, что любой процесс в системе может организовать собственный (дочерний) процесс и осуществить с ним связь.

Для организации синхронизации процессов по четкой шкале времени в интерпретаторе команд предусмотрены команды, позволяющие задержать процесс на определенное время либо до наступления заданного события (AFTER, ALT, PRI, ALLOCATF, FLACE). Используя эти команды, пользователь может описать среду объекта и точки взаимодействия.

Необходимо заметить, что пользователь управляет объектом не непосредственно, а через подмодуль объекта – объект связи. Последний может быть создан пользователем либо явно (во время организации пакета), либо неявно (во время организации процесса (программы)). Объект связи осуществляет все коммуникации, необходимые объекту (как межобъектные, так и с устройствами ввода-вывода). Отличие ОС от существующих в настоящее время заключается в следующем: если обычно процесс организует все коммутации сам по заложенному алгоритму, запрашивая только ресурс ввода-вывода, то здесь каждому процессу ставятся в соответствие каналы обменов, которые могут динамически меняться по мере развития процесса пользователем.

Логическая структура объекта коммуникации может быть представлена следующим образом:

Команда обмена (тип) | Идентификатор процесса | Флаги состояния | Содержание сообщения | Ссылка на следующий объект коммуникации.

Для того чтобы обеспечить пользователю возможность контролировать текущее состояние системы, необходимо предусмотреть команды, позволяющие отображать коммуникации и объекты, существующие в системе. Команды интерпретатора по управлению процессами и объектами коммуникаций аналогичны командам работы с файлами. Команды DIRP и DIRC (отобразить процесс и отобразить канал) имеют те же свойства, что и соответствующие команды для отображения содержимого библиотеки на диске – DIR. Аналогично файлам процессы и каналы процессов в ОС могут быть удалены (исключены из системы) командами REMOVEP и REMOVEC. Создание каналов и процессов осуществляется объявлением их имен в командном интерпретаторе системы.

Если имя процесса пользователь специально не оговорил, то оно идентифицируется с именем файла, содержащего исходную командную информацию процесса. Однако пользователь имеет возможность объединить несколько процессов под одним именем с помощью команды CHAIN. Эта же команда по отношению к каналам коммуникаций позволяет тиражировать информацию из одного канала в несколько.

Управление каналами. В рассматриваемой системе концепция канала несколько отличается от концепции канала, принятой в других системах. В нашем случае канал представляет собой сложный объект, осуществляющий не только передачу, но и преобразование передаваемых данных (согласование форматов сообщений). Таким образом, канал – это процедура преобразования данных без собственных данных. Как в любой системе приема-передачи информации, в канале должны быть предусмотрены порты ввода-вывода. Каждый канал теоретически может иметь бесконечное количество портов, которые могут ассоциироваться с любым процессом в системе. Задание порта осуществляется описанием типа передаваемой информации и размерностью буфера для асинхронной передачи. По умолчанию в системе принимается: тип передаваемой информации – байта, размерность буфера – одна единица информации. Кроме того, пользователь должен иметь возможность указать условие ликвидации порта, т. е. значение передаваемой информации или состояние специально оговоренного порта синхронизации.

Формат описания порта:

ИМЯ (PORT) = тип информации, размерность буфера, условие ликвидации

При инициализации системы автоматически инициируются порты консоли оператора, печатающего устройства, клавиатуры и т. п.

Описание порта консоли оператора,принтера имеет вид

CHAN (PORT) = *byte* , 1
LPT (PORT) = *byte*, 1

Задание канала осуществляется на основании объявленных ранее портов приемника и источника информации, а также ссылкой на процедуру преобразования информации. Если в объявлении канала процедура опущена, то передача информации происходит без преобразования.

Формат описания канала:

ИМЯ (CHAN) = ПРИЕМНИК (PORT), ИСТОЧНИК (PORT),ПРОЦЕДУРА (PROC)

Если при функционировании канала ликвидируются все порты со стороны передачи либо приема, то канал также исключается из системы. Если при описании канала вместо любого из параметров указать знак ? , то интерпретатор команд пользователя дополнит определение параметра при инициализации канала. В качестве примера рассмотрим описание канала отображения символьной информации на экране дисплея:

FILE (PORT) = ? ,, 1AH (^Z)
TYPE (CHAN) = CAN, FILE

Эта запись показывает, что передача происходит из порта FILE , который определяется при обращении к каналу при условии ликвидации канала (код 1AH),в порт консоли оператора.

При организации канала копирования файлов формат канала будет иметь вид

COPY (CHAN)=FILE, (FILE, ..., FILE) ,

где запись , ..., означает, что перечислением можно задать несколько файлов в качестве исходных.

Управление процессами. Под процессом в объектно-ориентированной системе по аналогии с другими системами понимается структура данных с процедурой над ней. Процесс может создавать любое количество необходимых ему каналов, однако ни один канал не может создать процесс.

Параллельные (независимые) процессы задаются в системе по следующим правилам: 1) имя процесса без описателей либо оканчивающееся точкой понимается как имя параллельного процесса; 2) перечисление имен процессов, разделенных знаком точки, понимается как совокупность независимых процессов. Приведем примеры:

ПРОЦЕСС – параллельный процесс;
ПРОЦЕСС. – аналогичен предыдущей записи;
ПРОЦЕСС1.ПРОЦЕСС2.ПРОЦЕСС3 – совокупность параллельных процессов.

Последовательные процессы задаются перечислением через запятую. Круг-

лые скобки указывают группу процессов, следующих друг за другом. В этом случае интерпретатором создается совокупность флажков для инициализации процессов в определенной последовательности. Если запятая стоит перед или после имени обособленно, то понимается, что соответственно предыдущий или последующий процесс является последовательным к предыдущим. Например:

ПРОЦЕСС или ПРОЦЕСС

указывает на то, что введенное имя "сцепляется" с ранее введенным или будет "сцепляться" с последующим.

Необходимо заметить, что описатели PAR и SEQ указывают на отношения между процессами, следующими за этими описателями и заключенными в круглые скобки. Описатель ALT может быть применен к процессам только вместе с указанием канала, разделяемого этими процессами. Приведем пример:

```
ALT КАНАЛ (CHAN)
(ПРОЦЕСС1, ПРОЦЕСС2).(ПРОЦЕСС3, ПРОЦЕСС4)
```

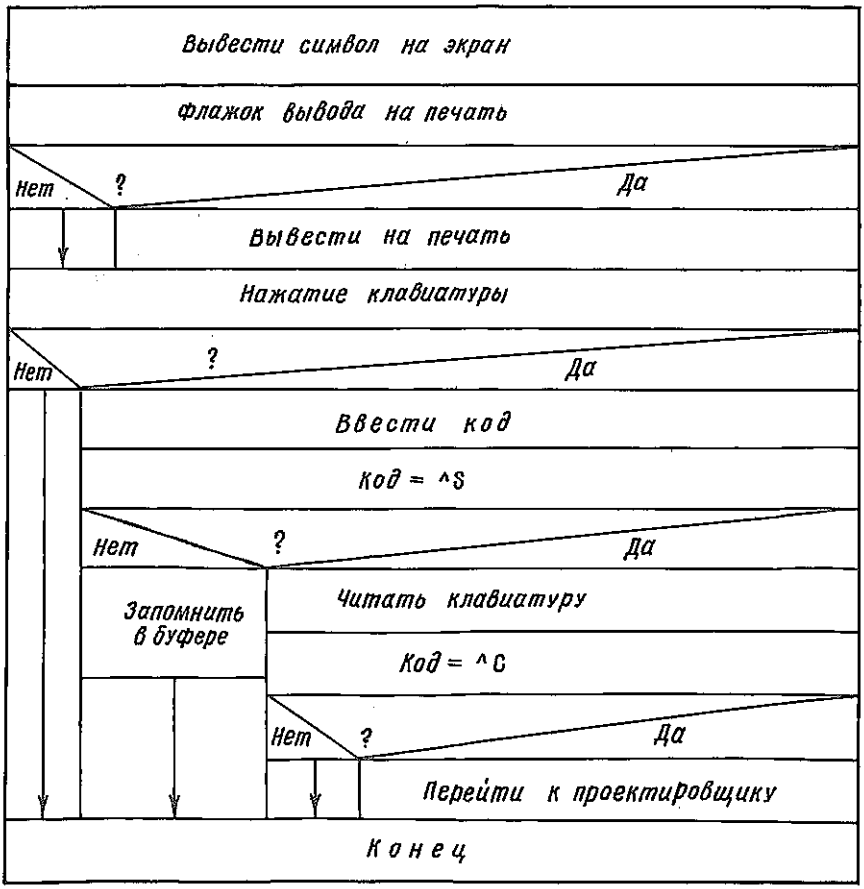


Рис. 10.5

Взаимодействие интерпретатора команд с операционной системой. При инициализации системы в интерпретаторе формируется ряд команд, позволяющих пользователю контролировать и управлять развитием системы. При необходимости пользователь может дополнить перечень встроенных в интерпретатор команд.

Рассмотрим пример работы интерпретатора системы при получении команды DIR (распечатать содержимое библиотеки диска). При этой команде интерпретатор проводит синтаксический и семантический анализ введенной строки. Если в имени файла вместо кодов литер введен знак *, то соответствующие знакоместа заполняются знаком ? , который интерпретируется на уровне команды ОС. После этого формируется управляющий блок файла и производится обращение к ОС.

Например, после введения пользователем команды

DIR FILE.*

интерпретатор команд формирует следующую строку знаков в управляющем блоке файла

FILE ␣␣␣␣???.

После обращения к ОС в буфере канала файловой системы возвращается фрагмент текущей библиотеки с паспортом файла, в имя которого входят литеры, указанные пользователем, а на место со знаками ??? возвращаются любые литеры. Это позволяет искать в библиотеке файлы, у которых имя или расширение точно пользователю не известны. После нахождения в библиотеке имени файла оно (через канал консоли оператора) выводится на дисплей. Затем вызывается команда ОС, возвращающая размер файла. Величина размерности файла преобразуется интерпретатором в десятичный код и выводится на консоль. Поиск производится до получения кода ошибки, указывающей на то, что больше похожих файлов в текущей библиотеке нет. Диаграмма Найси–Шнайдера для упрощенного алгоритма обработки команды DIR интерпретатором команд приведена на рис. 10.5.

Рекомендуемая литература: 2, 12 .

11. ЭЛЕМЕНТЫ ИНТЕЛЛЕКТУАЛИЗАЦИИ МИКРОЭВМ

11.1. ОСНОВНЫЕ КОНЦЕПЦИИ И ПОНЯТИЯ

Проблема программирования. Вопрос об увеличении количества квалифицированных программистов становится все более острым в связи с появлением большого числа ПЭВМ. Решению этой проблемы могли бы помочь стандартизованные программные пакеты, работающие под управлением стандартных ОС, но и в этом случае необходимы также программисты. Идеальным вариантом является возложение задач программирования на ПЭВМ, но для этого необходимо создание пакетов автоматизации проектирования программ, что требует разработки новой технологии решения задач, новых языковых средств для описания задач, хранения и обработки элементов знаний о задачах, программах, понятиях и т. д.

Подобные пакеты могут решать задачи пользователей одним из двух способов: либо генерировать всю программу по исходному описанию задачи сразу на машинном языке, либо составлять резидентный набор заранее отлаженных модулей на время выполнения прикладной задачи. В обоих случаях требуются специальные средства описания исходной задачи, желательно в терминах и понятиях прикладной области. К ним относятся программные спецификации.

Спецификации программ. Под спецификацией понимают описание задачи, которую решает программа ("специфицировать" — значит "описывать"). Но сама программа микроЭВМ также является описанием задачи на языке программирования. По мнению специалистов, спецификация, во-первых, отличается от программы тем, что говорит, что надо сделать, а не как это сделать, т.е. имеет непроцедурный характер. Во-вторых, черта спецификации — однозначность, понятность. В-третьих, спецификация характеризуется полнотой описания задачи (ничто важное не должно быть опущено).

Таким образом, *спецификацией* называется достаточно полное описание задачи, которое пользователю написать, прочесть и понять легче, чем программу решения этой задачи на доступном ему языке программирования. *Средства спецификаций* — это любые средства построения этих описаний, а *язык спецификаций* — синтаксически оформленный язык этих средств. Фактически язык спецификаций является языком более высокого уровня, чем язык программирования, поскольку в нем важна эффективность выразительных средств. На языки спецификаций можно смотреть, как на будущее языков программирования.

В спецификации программы выделяются две части. Первая описывает объекты, задачи, отражает разделение ее на модули, определяет входные и выходные данные, связи между ними, функции решаемой задачи, т.е. является функциональной. Вторая часть описывает требования к ресурсам, надежности программ и называется эксплуатационной.

Кратко охарактеризуем следующие средства спецификаций: таблицы, равенства, логические средства, графы и сети, процедурные средства и математические структуры. Языки спецификаций делятся на универсальные и специализированные. Специализация языка может быть ориентирована на предметную (проблемную) область для использования в конкретной отрасли, основываться на определенном средстве (таблицы, равенства) для использования в различных областях. Для каждого из указанных классов средств спецификации можно дать специализированный язык, основанный на средствах этого класса.

Хотя классов проблемно-ориентированных языков может быть так много как проблемных областей, среди них выделяются языки, ориентированные на управление, и языки, ориентированные на структуры данных. К первым относятся языки описания асинхронных и параллельных систем, ко вторым – языки описания задач обработки данных сложной структуры.

Средства спецификации. Рассмотрим основные средства описания задач. Под *таблицей* понимается объект, состоящий из элементов, расположенных по строкам и столбцам. Таблицы применяются в следующих видах: конечных функций, конечных отношений, табличных структур данных. В программировании применяют таблицы решения, представляющие собой описания конечных функций, аргументы которых являются условиями, а функции – действиями. В ряде языков (ЛИСП, СЕТЛ, РЯОД) определены операции над конечными функциями. Конечные отношения играют основную роль в языках описания и манипулирования реляционных баз данных. Табличные структуры данных, не интерпретируемые явно как функции или отношения, задаются в форме двухмерных массивов во многих языках программирования. Одним из первых языков спецификаций табличного типа является РПГ, в основе которого лежит описание задач генерации отчетов. Матрицы и операции над ними используются при спецификации пакетов программ для задач линейной алгебры, а также в языке АПЛ, где над матрицами выполняется ряд операций (+, * и т.д.).

Под *равенством* понимается выражение вида $P_1 = P_2$, где P_1, P_2 – термы, построенные из функций, переменных и констант. С помощью равенств определяются функции (например, в языках РЕФАЛ, ЛИСП, других языках функционального программирования), а также отношения, но в отличие от таблиц не конечные, а бесконечные (например, в языке УТОПИСТ). Система подстановок, используемая в равенствах, применяется для описания не просто функций, а процессов обработки информации. Подстановки вида $P_1 \to P_2$, называемые продукциями, где P_1 и P_2 – не обязательно выражения, а цепочки символов, применялись в виде грамматик языков программирования, а в последнее время – для представления элементов знаний. Продукция $P \to Q$ применяется к состоянию базы данных, где состояние – это набор кортежей, таблиц или других структур данных. Левая часть продукции P выражает условие применимости продукции к состоянию, правая часть Q указывает, как изменить состояние, для которого применяется условие P. Например, если $x=2 \to r= x+y$ в простейшем случае.

Равенства представляют частный случай *логических* (атомарных) отношений, имеющих вид $P\ (E_1, ..., E_n)$, где P – предикат, а E – термы. Особый интерес представляют формулы вида $\Phi_1 \vee \Phi_2 \vee ... \vee \Phi_n \to \Psi_m$, называемые дизъ-

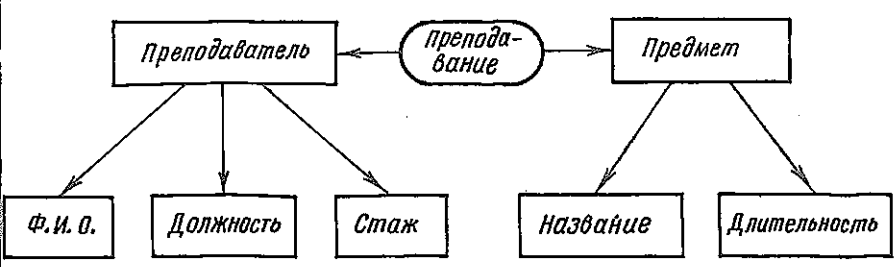


Рис. 11.1

юнктами, где Φ_n и Ψ_m – атомарные формулы. При $0 \leqslant m \leqslant 1$ дизъюнкт называется хорновским. Такие дизъюнкты лежат в основе языка логического программирования ПРОЛОГ. Они оказались удобными для функций и отношений в форме предикатов. При этом исходная программа описывается в виде набора дизъюнктов, а сам механизм вычислений полностью скрыт от пользователя и базируется на мощном средстве логического вывода – принципе резолюций.

Графы и сети являются удобным графическим средством для описания задачи, включающей объекты и связи между ними. С графовой структурой, представляющей задачу, связывается дополнительная информация, которую называют интерпретацией. Последней может быть язык, множество путей, поведение системы, а графовая структура рассматривается вместе с этой надстройкой. В основном известны два класса интерпретаций при спецификации задач. В первом основную роль играет понятие потока информации, во втором графовая структура интерпретируется как описания сложного объекта, имеющего связи.

Примером графовой интерпретации является диаграмма объектов-связей. На рис. 11.1 объекты изображены прямоугольниками, а связи – овалами, причем овал "преподавание" связывает преподавателя с читаемым им предметом. Интерпретация диаграммы заключается в следующем. С узлами–прямоугольниками связывается множество объектов, а с узлами–связями – множество отношений, т.е. диаграмма задает набор отношений или класс моделей. Каждая модель рассматривается как состояние базы данных, а диаграмма используется при ее проектировании.

Процедурные средства. Используются для "крупномасштабного" программирования, т.е. описания и решения задач с помощью крупных объектов, что было впервые предложено Л.В.Канторовичем. В дальнейшем появился язык АПЛ, в котором ведущую роль играют операции над массивами и функциями. Рассмотрим классы структурированных объектов и основные операции над ними.

Главным понятием является последовательность $A = a_1 a_2 ... a_k$, называемая также вектором, кортежем, списком, файлом в зависимости от контекста. Построение последовательности A можно рассматривать как конструирование объекта из символов $a_1, a_2, ..., a_k$. Частным случаем является понятие списка, используемого как базовый элемент языка ЛИСП.

Над последовательностями выполняются следующие операции: сцепление последовательностей A и B; упорядочение элементов последовательности (сортировка); выборка элементов последовательности, удовлетворяющей заданному условию; сопоставление с образцом (входит ли x в цепочку y).

Для спецификации важным является понятие функции, основной операцией над которыми будет композиция $f \circ g$, такая, что $f \circ g(x) = f(g(x))$. Ряд операций над функциями имеется в языке ЛИСП и его диалектах.

При описывании задач важно объединять объекты в математические *структуры*, рассматривая их в дальнейшем как единые объекты. В области программирования таким объектом является понятие абстрактного типа данных и модуля (языки ПАСКАЛЬ, АДА), а в области БД – схема, в области искусственного интеллекта (ИИ) – фрейм.

Простые математические структуры (векторы, последовательности) являются типами данных, а сложные, включающие множества и операции отношений между их элементами, образуют понятие абстрактного типа данных. Понятие схемы в базе данных включает типы объектов и типы связей (например, рассмотренная диаграмма объектов-связей).

Введенное американским исследователем М.Минским понятие фрейма для систем ИИ представляет собой структуру данных, предназначенную для описания стереотипной ситуации. Фрейм описывается в виде некоторой сети с незаполненными узлами (слотами). Фрейм, как и абстрактный тип данных, может включать в себя процедуры.

Таким образом, в трех областях – языках программирования, БД и ИИ – выработались понятия структур, позволяющие типизировать и систематизировать объекты, связи и действия, встречающиеся в решаемых задачах.

11.2. ПОСТРОЕНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ

Понятия аналога. Важным аспектом интеллектуализации микроЭВМ является построение интеллектуального диалога, особенно для начинающего пользователя. Основные типы диалога, нашедшие распространение для микроЭВМ, следующие: меню, командный, заполнение экранных форм, выполнение по шаблону, с использованием светового пера или манипулятора типа "мышь", на основе функциональной клавиатуры на ограниченном естественном языке. Рассмотрим кратко эти типы диалога, для чего введем некоторые понятия.

Обменом назовем вывод информации диалоговой системой (ДС) и возможный ответ пользователя. *Диалоговым сценарием* является последовательность обменов, направленных на достижение определенной цели (справка, обучение, решение задачи, игра, проведение экспертизы).

Диалог типа *меню* включает обмены, состоящие из предлагаемых альтернативных действий диалоговой системы с их выбором пользователем. Частным случаем такого диалога является альтернативный "Да–Нет".

Командный диалог включает выдачу пользователем сообщений, содержащих код команды, параметры. Система выполняет требуемые действия, вступает в поддиалог для уточнения параметров или выдает ошибки.

Заполнение экранных форм заключается в задании значений полей, например, пользователь вводит значения A = ..., B = ... и т. д. *Выполнение по шаблону* заключается в том, что заполнение полей производится по образцу или с подсказкой по аналогии. *Световое перо*, или *манипулятор "мышь"*, применяется для выполнения различных действий (редакторских, указания, графических) на экране. Использование *функциональной клавиатуры* заключается в упрощении командного диалога за счет закрепления за клавишами часто вы-

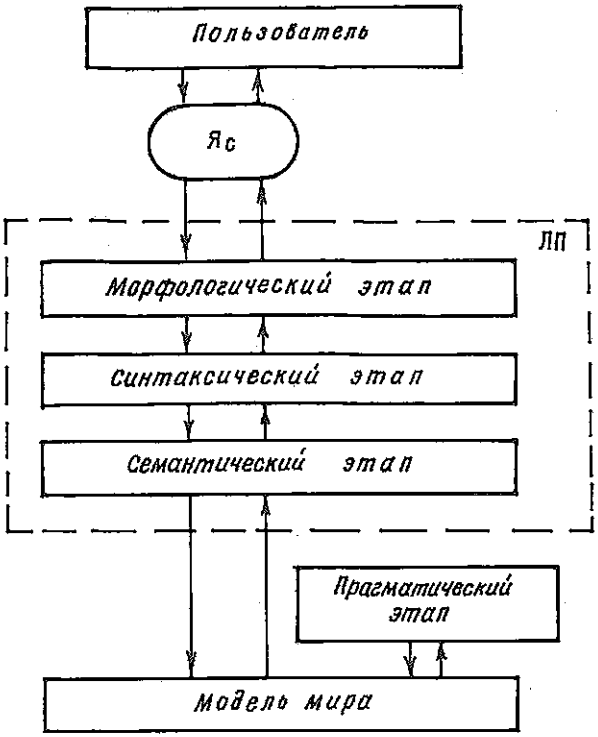


Рис. 11.2

полняемых действий, как правило редакторских, вычислительных процедур и т.д.

Особое место занимает диалог с микроЭВМ на *ограниченном естественном языке,* который находится в стадии разработки. Рассмотрим основные принципы его построения. При переводе с одного языка на другой применяется язык модели мира $Я_с$, на котором описывается информация об окружающей среде. При этом используется принцип обратимости, заключающийся в том, что все процедуры, обеспечивающие перевод от фраз естественного языка к $Я_с$, являются обратными, т. е. их можно использовать при переходе $Я_с$ к естественному языку. С учетом этого схема перевода имеет следующий вид (рис. 11.2).

Этап морфологической обработки включает процедуры определения (формирования) таких характеристик слов, как число, падеж, род, т. е. морфологического анализа (синтеза). Этап синтаксической обработки состоит из процедур определения (формирования) синтаксических конструкций предложений, т. е. анализ (синтез). Этап семантической обработки включает процедуры определения (обработки) смысла логики предложения, т. е. выполнение семантического анализа (синтеза). Прагматический этап отражает внутренние действия системы, связанные с необходимостью проведения дополнительных процедур (вычислений) , нужных для формирования ответа пользователю. Процедуры, выполняющие морфологический, синтансический и семантический этапы, объединяются в так называемый лингвистический процессор (ЛП).

Рассмотрим функционирование лингвистического процессора. На морфологическом этапе к каждому слову предложения добавляется его грамматическая характеристика, взятая из словаря. Для этого слово расчленяется на составляющие. Например, для слова "получаться" это расчленение следующее: по – приставка, луч – корень, а, ть – суффиксы, ся – постфикс. Возможность такого расчленения обеспечивается с помощью словаря основ для используемых слов.

На синтаксическом этапе строится дерево грамматического разбора фразы, поступившей на вход ЛП, так как это делается в трансляторах, однако с большим числом связей по смыслу (30–50 для научно-технических текстов).

Наконец, на семантическом этапе происходит преобразование дерева в модельное представление. Отметим, что синтаксис языка $Я_c$ отличается от синтаксиса естественного языка. Это очевидно, поскольку информация о внешнем мире в памяти человека не хранится в виде текста, об одном событии различные люди рассказывают по-разному.

В настоящее время диалоговые системы, построенные по схеме, показанной на рис. 11.2, очень малочисленны из-за сложностей теоретического характера и практической реализации. Рассмотрим направления создания диалоговых систем.

В диалоговых системах, в которых отсутствует ЛП, общение происходит на языке модели мира и на нем выдаются ответы. Для этого необходимо иметь стандартизованное представление информации во вводимых фразах, согласованное с $Я_c$. Спецификационные списки являются таким способом стандартизации. Их построение рассмотрим на примере.

Пусть в Р хранится информация о интегральных схемах памяти, которая организована в виде следующих записей:

$$(< n_1 >< v_1 >;\ < n_2 >< v_2 >, \ldots, < n_k >< v_k >),$$

где n_i – имена признаков; v_i – их значения. Пусть n_1 – технология изготовления; n_2 – объем; n_3 – организация слов; n_4 – величина задержки.

Следующая запись

< Технология > < n-МОП >; < объем > < 16 К >; < организация > <16КХ1>
< задержка > < 370 нс >;

отражает конкретный спецификационный список. На любом месте v_i может быть поставлен символ ? , который означает вопрос по данному признаку. Например, при v_1 = ? запрос аналогичен фразе "назвать все микросхемы памяти с различными технологиями, имеющие организацию 16 К х 1 и задержку 370 нс . Иногда вместо v_i разрешается ставить значение предиката (например, задержка < 400 нс), т. е. назвать микросхемы, имеющие задержку чтения меньше 400 нс. Однако данные ДС являются слишком "жесткими".

Более гибкими являются ДС, в которых ЛП используют идею дескрипторов. В качестве дескрипторов выбраны основы словоформ, список которых задан. На этапе морфологического анализа производится поиск вхождения дескрипторов в запрос. Например, пусть в словаре дескрипторов содержатся основы: ПЕРЕЧ, ТИП СХ, ТЕХНОЛ, ОБЪЕМ, ОРГАНИЗАЦ, ЗАДЕРЖ. На вход такой ДС поступил запрос: перечислить все схемы с объемом 4 К бит. Этот запрос преобразуется к виду: ПЕРЕЧ ТИП СХ ОБЪЕМ 4 К бит, которому соот-

ветствует спецификационный список

< технология > < ? > < объем > < 4К > < организация > < ? >
< задержка > < ? > .

Следующим усложнением ДС является введение отношений между дескрипторами в виде графа связи, на верхнем уровне которого находится вершина, общая для предметной области (например, вершина – "интегральная память", которая делится на статическую и динамическую и т. д.). С каждым уровнем могут быть связаны определители. Анализируя словоформы, поступившие на вход ДС, морфологический блок выделяет дескрипторы и определители, а синтаксический блок строит подграф, на основе которого в модели предметной области выделяется подобласть запроса. Такие запросы применяются в информационно-поисковых системах, но неполный ЛП ограничивает их возможности. Эти возможности дают лишь ЛП с развитым этапом семантического анализа и языками представления знаний, которые для микроЭВМ находятся в стадии разработки и исследования.

Рассмотрим построение ДС с представлением сценариев диалога в виде фреймов. На основе этого подхода строятся справочные обучающие, интегрирующие подсистемы. Для разработки любой из таких подсистем необходимо иметь инструментальные средства создания сценариев и их интерпретации. Под сценарием будем понимать связанный набор фреймов, определяющий действия системы в любой точке диалогового процесса. Такими действиями могут быть получение информации в виде набора сообщений, выдача вопросов и введение ответов, за которые выставляется оценка, запуск прикладной либо системной программы, просмотр результатов выполнения программы и т.д. Сценарий разрабатывается проектировщиком, описывается на ограниченном естественном языке и помещается в базу сценария.

В общем случае фрейм может включать как заполненные, так и незаполненные части (слоты). Последние заполняются в процессе диалога в соответствии с определенными условиями. В общем случае фреймом является структура вида

$$n(v_1, q_1, p_1), (v_2, q_2, p_2), ..., (v_k, q_k, p_k),$$

где n – имя фрейма; v_i – тип слота; q_i – значение слота; p_i – процедура, которая может и отсутствовать.

Определим следующие виды фреймов для описания диалоговых сценариев: Меню, Сообщение, Таблица, Программа, Процедура, Анализ. Под Меню будем понимать фрейм с несколькими фиксированными слотами (сообщениями, ответами на вопрос) и с одним вводимым значением слота (ответом, реализующим выбор одной из возможных альтернатив). Сообщение – фрейм, не требующий определения значения слота и отображающий ситуацию (результат, действие и т. д.); Таблица – фрейм, требующий определения (диалогового ввода) нескольких значений слотов (системных переменных); Программа – фрейм, реализующий выполнение прикладной программы, изменяющий дальнейший ход диалогового процесса. Процедура – фрейм, реализующий арифметические и логические операции, производящий анализ значений предметных переменных, определяющий следующий фрейм, а также выполняющий команды базового монитора. Анализ – фрейм, осуществляющий просмотр наборов

данных, предназначенных для вывода на консоль и полученных в результате выполнения программ.

Описания фреймов диалоговых сценариев могут включать следующие типы слотов: предметных переменных, процедурных, экрана, характеристик программ и данных. В слоте *предметных переменных* каждой переменной присваивается имя и определяются ее характеристики: тип, длина, смысл на естественном языке, область применения. Этот слот присутствует в фреймах типа Меню, Таблица, Программа. *Процедурный* слот используется для выполнения различных действий, а также почти во всех типах фреймов и представляет отдельный фрейм Процедуру. В слоте *экрана* записывается постоянная информация, которая отображается на экране в ходе диалога и может входить в описание любого фрейма. Фрейм типа Сообщение содержит только слоты экрана. Для фрейма типа Программа введены дополнительные слоты *характеристик программ и данных*. Первый включает имя программы, ее расположение на внешней памяти, максимальное время выполнения, способ вызова и передачи параметров (через область данных, по соглашениям ОС). В слоте данных записываются данные, необходимые программе или процедуре.

Рассмотренные виды фреймов для представления диалогового сценария позволяют на одной основе подходить к разработке инструментальных диалоговых систем справочного, обучающего, контролирующего, интегрирующего и простейшего экспертного характера. Этот принцип заключается в следующем. Сначала определяются типы фреймов, образующих данный сценарий, и типы слотов, входящих в эти фреймы. Затем разрабатываются элементы диалогового языка в терминах проблемной области, проектируется база диалоговых сценариев, блок их создания, блок интерпретации и необходимые сервисные блоки. В режиме создания проектировщик диалога задает описание требуемых слотов (часть из которых может доопределяться в процессе диалога) для построения фреймов сценария и записывает их в базу диалога. В режиме интерпретации из базы выбираются фреймы сценария, для их отдельных слотов пользователем определяются затребованные значения (происходит заполнение фрейма) и реализуется вариант диалога.

Информационно-справочная подсистема. Рассмотрим проектирование такой подсистемы, используя описанный принцип построения фрейм-сценария. Сначала определим функции разрабатываемой подсистемы. К ним относятся: выдача справочной информации, организованной по иерархическому принципу; возможность создавать любые проблемные справки на ограниченном языке; просмотр информации сверху вниз и снизу вверх; получение помощи при использовании подсистемы.

При построении справочного сценария достаточно фреймов двух типов: Меню и Сообщение. Сам фрейм-сценарий имеет форму дерева, в корне которого находится фрейм-описание работы подсистемы, затем следует главное Меню, из которого строятся переходы вниз.

Фрейм Меню содержит информационные сообщения (слоты) и команды перехода вниз (по номеру или значению сообщения) и вверх (по числу вершин). Например, выдается три сообщения в данном фрейме. Значит, имеется возможность перехода по трем направлениям вниз. Вверх можно перейти на 1, 2, ..., n вершин до начальной. Пользователю предоставляются следующие виды команд: H — переход в начало сценария, N — переход вниз по номеру сооб-

щения; *–n* – возврат на *n* шагов вверх по дереву; К – завершить просмотр и выйти из подсистемы.

При создании сценария проектировщик вводит конкретные тексты сообщений, указывает номера фреймов для перехода (фрейм Меню). Например:

Фрейм 1. Какие типы микропроцессоров желаете просмотреть?
1. Однокристальные; 2. Секционные; переходы 2, 3.
Фрейм 2: Какие однокристальные микропроцессоры желаете просмотреть?
1. К580ИК80; 2. К1810ВМ86; переходы 4, 5.
Фрейм 3: Какие секционные микропроцессоры необходимы?
1. К1804, ...; 5. К589; переходы 6–10 и т.д.

Обучающе-контролирующая подсистема. Общая идея построения таких систем заключается в том, что сначала составляется сценарий обучения-контроля, включающий набор фреймов, а затем он интерпретируется. Для построения такого сценария необходимы следующие фреймы: Стартовый (включает служебный слот и слот экрана); Меню (содержит служебный слот, слоты интервалов, ответов и экрана); Таблица (включает характеристики предметных переменных); Программный (содержит служебный слот с описанием инициируемой программы); Оценки (включает служебный слот и слот интервалов). В общем фрейм может иметь следующую структуру: служебный слот, слоты ответов, интервалов, экрана (рис. 11.3).

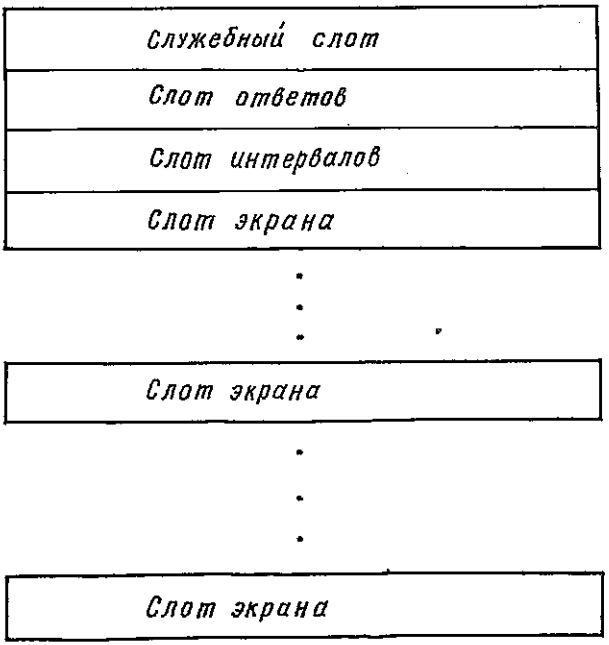


Рис. 11.3

Служебный слот содержит:

#1 = 1 – стартовый раздел; 2 – последний раздел; 3 – любой другой раздел
[, V = количество ответов в слоте ответов]
[, I = количество интервалов в слоте интервалов]
[, K = команда перехода] [, B = метка перехода]
[, P = имя подпрограммы пользователя]
[, T = время существования транзитного сообщения]
[, O = время на ответ пользователя]

Слот ответов имеет следующую структуру:

#2 – начало слота < возможный ответ →команда > < другой ответ →команда

Слот интервалов включает:

#3 – начало слота $< A_1 – B_1$ →команда ... $A_k – B_k$ →команда & признак конца слота и начало слота экрана.

Команда: = +число | –число | = число [, команда перехода] [реплика]

Левая часть отражает изменение балла за ответ, средняя – переход по сце нарию обучения, правая – реплику, появляющуюся после ответа.

Команда перехода:

=CALL: имя1 [имя2] | RETURN | BACK | имя фрейма,

где имя1 – фрейм возврата в основной диалог; имя2 – начальная вершин основного поддиалога; RETURN – переход в вызывающую вершину; BACK - шаг назад по сценарию диалога; имя фрейма – переход к фрейму.

Слот экрана содержит постоянную информацию, которую необходимо по казать на экране в процессе диалога.

11.3. ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЕ ЯЗЫКИ

Виды языков. С общей точки зрения языки программирования разделяют ся на декларативные и императивные. Первые получили распространение в по следние годы. Среди них выделяются языки функционального (ЛИСП) и ло гического (ПРОЛОГ) программирования. ЛИСП признается эффективным языком искусственного интеллекта. Основные его элементы – атомы и спи ки, над которыми могут выполняться как встроенные функции, так и функ ции, определенные пользователем, Значительное распространение в последни годы получил язык ПРОЛОГ благодаря использованию в неявном виде проце дурного механизма логических выводов – принципа резолюций.

Среди императивных языков различают универсальные (ПЛ/М, БЕЙСИК и ориентированные на различные области применения (ФОРТРАН, КОБО и т.д.).

Отдельное место занимают языки, ориентированные на решение задач использованием представления и обработки объектов и называемые объектно ориентированными. Наиболее мощным из них является УТОПИСТ (универ сальное текстовое описание терминов), являющийся основой для построени других проблемно-ориентированных языков.

Элементы языка УТОПИСТ. Основное назначение языка УТОПИСТ – опи сание понятий и задач, хотя в языке предусмотрены и процедурные средства

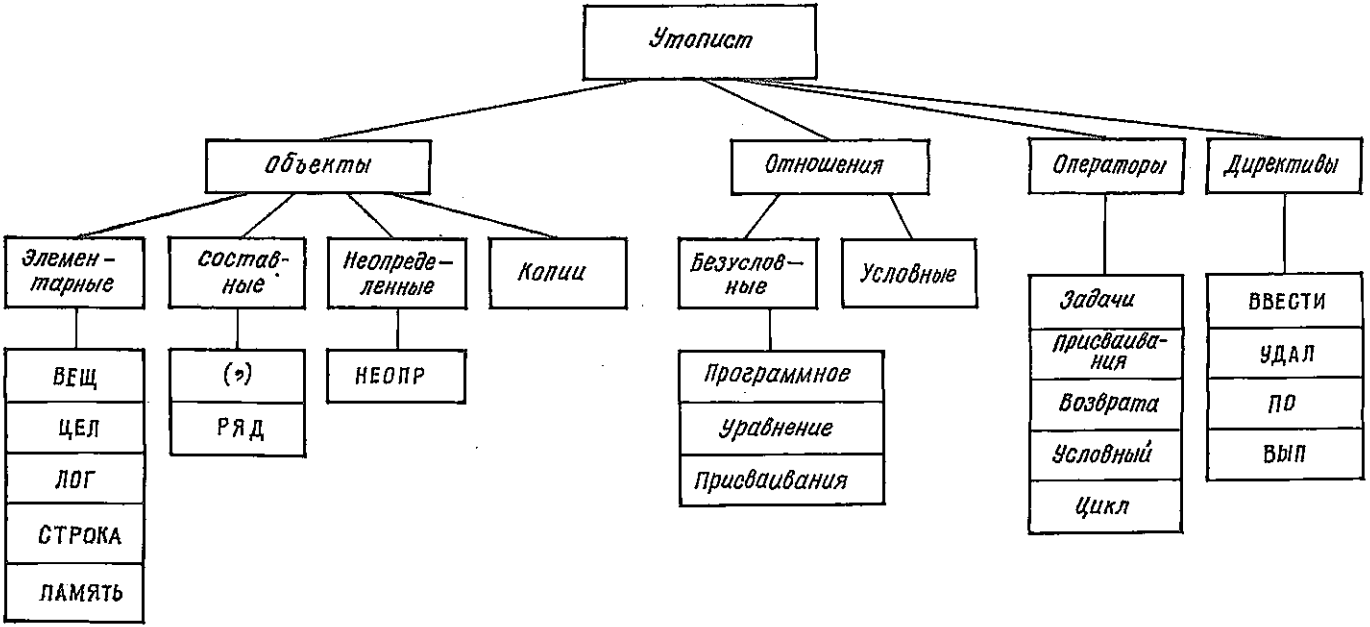

Утопист
Объекты
Отношения
Операторы
Директивы
Элемен-тарные
Состав-ные
Неопреде-ленные
Копии
Безуслов-ные
Условные
ВЕЩ
ЦЕЛ
ЛОГ
СТРОКА
ПАМЯТЬ
(?)
РЯД
НЕОПР
Программное
Уравнение
Присваивания
Задачи
Присваива-ния
Возврата
Условный
Цикл
ВВЕСТИ
УДАЛ
ПО
ВЫП

Рис. 11.4

Одним из элементов описания является объект, который разрабатывает пользователем на основе встроенных средств — понятий, постоянно хран щихся в памяти ЭВМ.

Программа, написанная на языке УТОПИСТ, состоит из начала, тела и ко ца. Тело программы содержит описание данных и действий. Данные задаются виде одного или нескольких описаний, в которых объекты связаны между с бой отношениями. Результатом трансляции описания данных является моде задачи. Действия — это последовательность операторов, записанная либо пос слова Действия либо после слова Подпрограмма. В первом случае резуль том действия после трансляции является управляющая программа, во вт ром — подпрограмма. В одной программе не разрешается использовать оба в рианта.

Описание данных и действий может чередоваться. Кроме того, в текст пр граммы вставляются директивы и комментарии. Директивами являют предложения, требующие немедленного выполнения в ходе трансляции, ко ментарии начинаются с символа * .

Основные понятия языка приведены на рис. 11.4. Это объекты, отнош ния, операторы и директивы. Объекты в свою очередь могут быть элемента ные, составные, неопределенные и копии. В языке определены следующие эл ментарные типы объектов: вещественный (ВЕЩ), целый (ЦЕЛЫЙ), логич ский (ЛОГ) и текстовой (СТРОКА). Примеры описаний объектов с элеме тарными типами:

```
В:ВЕЩ ; К: ЛОГ ; ИМЯ: СТРОКА 6;
```

Составные описатели объектов включают описание его компонентов и отношений, причем компоненты в свою очередь могут быть составным В этом случае в модели описываемого объекта образуются структурные отн шения, связывающие описываемый объект и его компоненты. Для приме приведем описание составного объекта КРУГ, включающего компонент РАДИУС, ПЛОЩАДЬ, ЦЕНТР. Последний объект является также составны

```
КРУГ:
(РАДИУС, ПЛОЩАДЬ: ВЕЩ; ЦЕНТР: (X, Y :ВЕЩ));
```

Существуют и такие составные объекты , которые включают однотипны компоненты. Для их описания используются номера в введенном понятии ря Например, А: МНОЖ (1, ..., 10) :ВЕЩ;

Для описания объектов, которые к данному моменту не определены и в различных задачах разные, введен тип описателя неопределенный (НЕОПР Его определение уточняется указанием имени описанного объекта.

Для расширения языка существует возможность использовать в качест описателя имя ранее описанного объекта. Например, если описан объект типа х, то X можно взять в качестве нового описателя. Пусть точка описа так:

```
ТОЧКА: X, Y: ВЕЩ;
```

Тогда описание круга можно определить на уже имеющемся понят ТОЧКА:

```
КРУГ: (РАД, ПЛ: ВЕЩ; ЦЕНТР:ТОЧКА);
```

Программными являются *отношения*, которые задаются на основании готовых модулей. При этом описание отношения включает все сведения для вызова модуля (имя, вид оформления и параметры), а также дополнительную информацию. Примером описания отношения, заданного модулем ПРОГ, является

МОДУЛЬ ПРОГ X ВЫХ Y ;

При этом объекты, которые используются в отношении, являются входными, а объекты, значения которых вычисляются, – выходными. Но не всегда возможно определение входных и выходных объектов, поэтому в язык включено средство описания слабосвязанных параметров, которые могут быть как входными, так и выходными в зависимости от условий. Например, при описании модуля, задающего зависимости между током, напряжением и сопротивлением, используется отношение

МОДУЛЬ 01 СЛ I , U , R ;

Данный модуль включает три ветви для вычисления соответственно I , U , R по значениям двух других.

Для вычисления арифметических или логических выражений с присвоением их значения объекту введена конструкция ПРИСВ, которая имеет вид

ПРИСВ имя := выражение значение.

Например, конструкция ПРИСВ Z:=X+Y означает, что добавляется новое отношение.

Для задания отношений между объектами вещественного типа используется уравнение следующего вида:

УРАВ арифметическое выражение = арифметическое выражение.

Например, УРАВ I = I + P выполнимо, если определены I и P .

Некоторое отношение выполнимо лишь при определенных условиях и реализуется с помощью условных отношений вида: ЕСЛИ лог-выраж ТО ... Например, ЕСЛИ X = 2 ТО Z = X + Y .

В языке допускаются арифметические и логические выражения, которые применяются для описания вычислений. Для арифметических выражений допускается вычисление с помощью операций: + , – , * , / , ** , а также различных функций. Например, EXP X1, Y** 2. Логические выражения могут включать операции НЕ, И, ИЛИ, СР (сравнение). Например, А ИЛИ В И С. Логические и арифметические выражения вычисляются в соответствии с приоритетами операций, аналогичных приоритетам других языков программирования.

В языке УТОПИСТ предусмотрена возможность описывать задачу, не задавая явно алгоритма ее решения. Для этого служит оператор задачи, который имеет форму:

НА М ВЫЧИСЛИТЬ Y1, Y2, ..., YK ПО X1, X2 , ..., XM,

где X1, ... , XM , Y1 , ... , YK – имена объектов, описанных моделью задачи M, причем X – входные, Y – выходные параметры. Модель, на которой описывается задача оператором ВЫЧ, может являться моделью любого объекта или понятия из библиотеки моделей. Для решения задачи система ПРИЗ

синтезирует алгоритм и оформляет его в виде модуля решения задачи на языке ассемблер. После выполнения модуля вычисленные параметры и управление передаются вызывающей программе.

Кроме оператора задачи в языке УТОПИСТ присутствуют другие процедурные операторы, совпадающие с аналогичными операторами в языках программирования. К ним относятся: оператор присваивания (имя:=выражение), условный оператор для управления вычислениями (ЕСЛИ лог-выраж ТО безусловный оператор ИНАЧЕ оператор), операторы цикла для повторения некоторого участка программы (ПОКА лог-выраж ЦИКЛ оператор) и обращения к подпрограмме (ВЫЗОВ идент (факт-параметр,...)).

Приведем примеры перечисленных операторов в порядке их описания:

```
присваивания: B:=1; A:=B+2xC;
условный: ЕСЛИ Y1 БР Y2 ТО I = Y1 – Y2/P ИНАЧЕ I:=0;
(БР – больше, равно),
цикла: X:=1; K:=2; ПОКА K МР 10 ЦИКЛ X = X * K; K:=K+1;
(МР – меньше, равно),
обращения = ВЫЗОВ ПРОГ1 (A1, A2, T, A);
```

Для работы с библиотекой моделей предназначены следующие директивы: ВВЕСТИ, УДАЛИТЬ, ПО. Первая меняет состояние библиотеки, по ней (ВВЕСТИ' имя) в библиотеку записывается объект с указанным именем. Вторая директива исключает из библиотеки понятие с данным именем (УДАЛИТЬ' имя). Например, по директиве УДАЛИТЬ' ЦЕПИ – освобождается в библиотеке целый раздел.

По директиве ПО' сокращаются имена библиотечных понятий, применяемых при описании данных и действий. Например,

```
ПУСТЬ' ФИГУРА: КР:ГЕОМ. КРУГ; заменится ПО' ГЕОМ;
ПУСТЬ' ФИГУРА: КР:КРУГ;
```

Кроме директив работы с библиотекой, используется директива ВЫПОЛНИТЬ, которая дает возможность во время трансляции выполнить заданные программы.

Например, директива ВЫПОЛНИТЬ' A1+++ выполняет программу A1.

Если создана модель предметной области, то она помещается в библиотеку моделей для использования при описании объектов из этой предметной области. Поясним изложенное на примере. Пусть создана семантическая модель геометрии ГЕОМ, которая находится в библиотеке моделей и включает понятия круг, квадрат, треугольник и т.д. Если программируя на независимом входном языке системы необходимо указать, что объект К является кругом, то в описании данных с использованием директивы ПО запишем

```
ПО' ГЕОМ; ПУСТЬ K:КРУГ;
```

По этому описанию создается модель, содержащая подробное описание объекта К и его компонентов.

Рассмотрим программу вычисления площади круга, вписанного в квадрат:

```
ПРОГРАММА' ПРИМЕР 1; ПО' ГЕОМ; ПУСТЬ K1:КВАДРАТ ДИАГОН=6;
```

```
                    К2:КРУГ ДИАМЕТР К1.СТОРОНА
              ДЕЙСТВИЯ  НА ПРИМЕР  1 ВЫЧ К2.ПЛОЩАДЬ;
КОНЕЦ +++
```

Элементы языка СИРЗ. Рассмотрим вариант языка для представления знаний и описания постановки задачи в системе интеллектуального решения задач (СИРЗ). Запрос состоит из ряда операторов для описания: опций, объектов, предметной области, рабочего объекта, фактов, продукционных правил, уравнений, отношений, исходных данных и результатов. Рассмотрим описание каждого оператора, используя нотацию БНФ.

Опции задают режим работы системы и являются необязательными:

< Опции > ::= < тип опции > < значение1 > < значение2 >

Тип опции задает режим работы СИРЗ (отладочный с выводом промежуточных результатов и рабочий без вывода), а также определяет по значению 2 вывод результата (на экран, печать, НГМД):

< Объекты > ::= < тип > < имя1 > ... < имя *n* > ...

Тип – это имя раздела библиотеки, где хранится описание объекта данного типа, а имя – имя конкретного объекта, используемого при решении задачи:

< Предметная область > ::= < имя области > [< тип > , < тип >]...

Имя области – это имя библиотеки, в которой хранятся описания типов, указанных в скобках. Данная секция является обязательной.

Оператор рабочего объекта описывает те объекты, которые отсутствуют в базе знаний

< Рабочий объект > ::= < уровень > < имя > < тип >

Описание фактов является необязательным и имеет вид

< Факт> ::= < имя предиката > (< список переменных >)

Продукционные правила включают теоремы, относящиеся к рабочему объекту, или те, которых нет в базе данных и знаний

< Правило > ::= < предикат > [& < предикат >] ... → < заключение >
< Предикат > ::= < имя предиката > < список переменных >

Далее следует описание уравнений и отношений в виде

< Отношение > ::= (< список идентификаторов > → < переменная >)
< текст >
< Уравнение > ::= < выражение > = < выражение >
< Выражение > ::= < операнд > < оператор> <операнд>...

В операторе "Дано" описываются значения объектов, которые являются исходными и задаются в следующем виде:

< Исходные данные > ::= < имя > = < константа > ...
< Константа > ::= < цифровая константа > < символьная константа >

Последним записывается оператор "Найти", который содержит имена компонентов. Эти компоненты требуется вычислить или определить

< Выходные результаты > ::= < имя > [< имя > ... < имя >]

Последние два оператора являются обязательными, их отсутствие вызывает прекращение работы и выдачу сообщения об ошибке.

11.4. ОРГАНИЗАЦИЯ ДАННЫХ И ЗНАНИЙ

Понятия знаний. Термин "знание" в интеллектуализации микроЭВМ так же важен, как и данные в программировании. Сразу оговоримся, что между понятием данных и знаний не существует строгой границы. Просто переход от данных к знаниям — логическое следствие усложнения информационных структур, обрабатываемых современными ЭВМ и микроЭВМ. Рассмотрим качественные отличия понятий данных и знаний.

Интерпретируемость — данные в памяти ЭВМ могут интерпретироваться только программой и в отрыве от нее не имеют содержательной информации. Знания всегда могут быть интерпретированы без программы.

При хранении данных не обеспечивается возможности компактного описания всех связей между различными типами данных. Например, как для элемента множества, так и для множества в целом требуется описывать многократно одни и те же сведения. При переходе к знаниям между их отдельными единицами можно установить такие классифицирующие отношения, как элемент-множество, тип-подтип. Эта информация автоматически передается любому элементу множества, что называется "наследованием" информации.

Ситуативные связи определяют ситуативную совместимость отдельных событий или фактов, а также отношения одновременности, расположения в пространстве и т.д. Ситуативные связи позволяют реализовывать процедуры анализа знаний на совместимость, противоречивость.

Появление знаний как информационных объектов потребовало их структурной и физической организации в базы знаний, которые являются развитием баз данных. Система управления базой знаний (СУБЗ) отличается от СУБД более мощными обслуживающими процедурами. Например, пользователь может не только применять имеющиеся структуры, но и создавать свои, а для них разрабатывать новые процедуры обслуживания и т.д.

Важным вопросом при создании базы знаний является способ описания знаний при взаимодействии с пользователем. Модели представления знаний и процедуры их обработки образуют систему представления знаний.

Формы представления знаний разделяются на декларативные (описательные) и процедуральные (вычислительные). *Декларативные* знания — это множество утверждений, независимых от предметной области. Моделирование предметной области в такой форме требует ее полного описания. Выводы поиска решений скрыты и опираются на процедуры поиска в пространстве состояний. *Процедуральные* знания содержат в явном виде описание отдельных процедур, которые обслуживают область базы данных. Это позволяет отказаться от хранения всех возможных состояний, требующихся для решения, ограничиться только описанием начального состояния и процедуры, вычисляющей требуемые описания из начального.

Однако разделение знаний на декларативные и процедуральные носит условный характер, поскольку модели могут использовать обе формы представления знаний. Наибольшее распространение получили две модели представления знаний – логическая и сетевая.

Основой *логических* моделей является формальная система, задаваемая четверкой M = (T, P, A, Ф), где T – множество базовых элементов; P – множество синтаксических правил для построения из T правильных выражений; A – множество истинных выражений (аксиом); Ф – семантические правила для построения аксиом из других выражений. Наиболее широкое распространение получили логические модели, основанные на исчислении предикатов (особенно после создания процедур логического вывода, основанного на принципе резолюций).

Сетевые модели в отличие от логических позволяют дать более сложные структуры знаний за счет включения в явной форме всех отношений, образующих информационную структуру с описанием их семантики. Основой модели является сеть, вершины которой отождествляются с понятиями, а дуги – с отношениями между понятиями. При этом вершины могут иметь свою внутреннюю структуру. Сетевые модели бывают в основном двух типов – семантические сети и фреймы.

Для организации интеллектуального диалога используются лингвистические знания, включающие модели описания естественного языка, алгоритмы обработки текстов и средства реализации этих моделей и алгоритмов. Совокупность перечисленных компонентов определяет систему представления лингвистических знаний, которая делится на модельно-алгоритмическую и программно-аппаратную подсистемы. Основой первой подсистемы является теория формального языка. Программно-аппаратная подсистема реализуется лингвистическим процессором. В последнем выделяют декларативную (описание словарей) и процедуральную (алгоритмы анализа и синтеза текстов) части.

Семантические сети. Основой формализации семантических знаний является направленный граф с вершинами (объектами) и дугами (семантическими отношениями между ними). В семантических сетях используются три основных типа объектов: понятия, события и свойства. *Понятия* представляют собой сведения об абстрактных или физических объектах предметной области. *События* – это действия, которые могут внести изменения в предметную область. *Свойства* используются для уточнения понятий, событий или других свойств.

Рассмотрим реализацию системы семантических моделей для представления знаний при решении задач (система микроПРИЗ). Модели, на которых описываются задачи, хранятся в семантической памяти, имеющей структуру дерева. В вершинах дерева расположены понятия, в конечных вершинах – элементарные понятия. Примером фрагмента семантической модели является приведенное описание понятия ГЕОМЕТРИЯ (рис. 11.5). Каждое понятие в семантической памяти имеет описание – семантическую модель, куда входят все его компоненты и отношения между понятиями. Для работы с семантической памятью используются директивы языка УТОПИСТ, которые позволяют вводить новые понятия, удалять старые, выделять часть памяти и т.д. Для конкретного понятия указывается имя, которое может быть составным. На

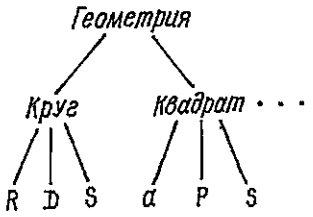


Рис. 11.5

рис. 11.5 такими именами являются: ГЕОМЕТРИЯ, КРУГ, КВАДРАТ. Семантическая модель имеет то же имя, что и понятие.

Информация о содержании задачи находится в семантической модели, имя которой задается в операторе задачи языка УТОПИСТ. Семантическая модель содержит переменные и отношения. Переменным семантической модели соответствуют содержательные понятия предметной области, на которой решается задача. Отношения выражают связь между значениями переменных и могут быть заданы модулем, уравнением, описанием структуры данных, моделью или макроопределением. Приведем пример.

Пример. Пусть имеются переменные I1, I2, I3, R1, R2, R3, U1, U2, U3 (вещественные). Два модуля A и B описывают соотношения между этими переменными в виде A–1) I:=U/R; 2) U:=I*R; 3) R:=U/I; и B – 1) I1:= –12–13; 2) I2:=–I1–I3; 3) I3:=–I1–I2. Пользуясь модулем A, опишем три отношения Q1, Q2, Q3, которые соответственно связывают отношения между I, U, R. Установим между параметрами модулей A и B и переменными отношений следующие соответствия:

Q1:	U...U1	Q2:	U...U2	Q3:	U...U3	Q4:	I1...I1
	I ...I1		I1...I2		I...I3		I2...I2
	R...R1		R...R2		R...R3		I3...I3

Отношения Q1–Q4 задают потенциальную возможность вычисления значений переменных. На вычислительной модели, содержащей N переменных, может быть описано 2^{2N} различных задач, у которых в качестве входных и выходных переменных выступают переменные на данной модели. Некоторые из этих задач являются разрешимыми. Например, задача

НА Z ВЫЧИСЛИТЬ I3 по R1, R2, R3, U1, U2

Рассмотрим модель представления знаний в СИРЗ, реализующей идеи концептуального программирования и отличающейся от системы микроПРИЗ использованием продукционной системы и непроцедурным языком представления знаний. Формализация знаний в СИРЗ базируется на следующих понятиях: объект, тип, отношение вычислимости, факт, правило. Под объектом понимаем любую структуру данных. Тип объектов характеризуется внутренней структурой, отношениями вычислимости, заданными на этой структуре, правилами, справедливыми для этого типа. Отношение вычислимости представляет собой A → B (если известно A, то можно найти B). Правило имеет вид A_1 & , A_2 & ... &A_n → B, где A_i , B – атомарные формулы вида P $(x_1, ..., x_n)$, а x_i – переменные либо константы. Правило без левой части является фактом.

Опишем синтаксис отношений вычислимости и правил на метаязыке с использованием нотаций Бэкуса–Наура:

< Отношение > ::= < список идентификаторов1 > < список идентификаторов2 > < ком. >
< Список идентификаторов > := < идентиф > ... < идентиф >
< Ком. > ::= < текст >
< Правило > ::= < предикат > & [< предикат >] ... →< заключение >
< Предикат > ::= < имя предиката > < список переменных и констант >
< Заключение > ::= отношение | уравнение | предикат

Опишем тип объекта "треугольник" в базе знаний "геометрия":

1 треугольник; 2 АВ:отрезок; 2 ВС:отрезок; 2 АС:отрезок,
1 отрезок; 2 начало:точка, 2 конец: точка, 2 длина:ВЕЩ
Правило: треугольник (☒) & прямой угол (☒катет 1, ☒катет 2, ☒гипотенуза) → уравнение: ☒гипотенуза^2 = ☒катет1^2 + ☒катет2^2

На основании базы знаний к СИРЗ могут выдаваться запросы вида:

< Запрос> = < Предметная область > < Объекты > < Дано > < Найти >
< Дано > := < уравнение | отношение | факт > ...
< Найти > : = < список идентификаторов >

11.5. ПОДХОДЫ К КОНСТРУИРОВАНИЮ ЗАДАЧ

Общие положения. На основе анализа современного состояния средств интеграции программ можно выделить следующие подходы к конструированию задач:

1) использование встроенных в язык программирования средств интеграции задачи из модулей (общие области, передача параметров, вызовы требуемых подпрограмм). Этот уровень является традиционным и требует от пользователя написания управляющей программы, детального знания структуры комплексируемых программ, знания программирования;

2) включение средств в универсальные диалоговые системы, которые вызывают в требуемые моменты времени нужные прикладные программы, без знания особенностей их вызова. Такие системы содержат аппарат поддержки паспортов (перечисление всех характеристик) модулей. Используя информацию из таких паспортов, диалоговая система выполняет функции по загрузке и сборке модулей, передаче параметров и управлению вызовами программ;

3) встраивание в современные ОС программных средств: механизмов почты, поддержку межмодульного интерфейса независимо от языка программирования (например, "конвейер" в ОС UNIX , язык управления процессами в многопроцессорной ОС, рассмотренной в главе 10);

4) применение понятийного (концептуального) программирования. Концептуальное программирование – это способ решения задачи на ЭВМ, заключающийся в описании понятий предметной области, достаточных для выражения смысла задачи. В терминах этих понятий описываются задачи, алгоритмы решения которых синтезируются автоматически методом структурного синтеза. По описанной схеме работает на микроЭВМ система микроПРИЗ, входным языком которой является язык УТОПИСТ.

Для концептуального программирования свойственны следующие особенности: программирование в терминах предметной области, использование ЭВМ

на этапе постановки задачи, автоматический синтез программ решения и использование семантической памяти для хранения и накопления знаний о решаемых задачах. Рассмотрим подробнее этот подход на конкретной системе.

Работа системы микроПРИЗ. Путь от спецификации задачи к ее решению выглядит следующим образом:

спецификация → теорема → доказательство → алгоритм → результат задачи

По описанной схеме работает система микроПРИЗ. Она позволяет в тексте программы на входном языке описывать задачи, не задавая явно алгоритма решения. При этом постановка задачи представлена в виде оператора, который является семантически правильным, если задача разрешима. Для семантически правильной задачи в системе синтезируется алгоритм на ассемблере в виде модуля решения. При выполнении программы, исходный текст которой содержит оператор задачи, происходит обращение к автоматически составленному модулю решения. Последний в свою очередь содержит обращения к другим, автоматически вызываемым модулям, в качестве которых используются подпрограммы либо макроопределения.

Система микроПРИЗ содержит четыре компонента, которые работают последовательно: транслятор переводит понятия в аксиомы, а решаемые задачи – в теоремы; синтезатор строит доказательства теорем; генератор извлекает из доказательства алгоритм; интерпретатор вычисляет результат.

Ограничением данного подхода является относительная сложность как самого вычислительного процесса, так и структуры системы, его поддерживающей, что ведет к снижению производительности микроЭВМ. Альтернативным подходом является реализация основных идей концептуального программирования в виде системы расширения (надстройки) над широко используемыми языками программирования (БЕЙСИК).

Организация СИРЗ. Для расширения языка в него требуется включить оперирование со знаниями трех уровней: 1) абстрактными общими (о предметной области); 2) абстрактными специальными (о задачах в рамках предметной области) и 3) конкретными (о конкретных объектах в рамках решаемой задачи).

Знаниям первого уровня соответствуют пакеты, содержащие классы и правила, второго уровня – описания конкретной обстановки, третьего уровня – конкретные объекты. Классы определяют понятия предметной области, объекты являются представителями классов. Правила выражают законы общего вида, применяемые к классам и объектам. В базе знаний находятся пакеты и рабочие контексты, внутри которых содержатся конкретные объекты.

Класс определяют структуру объектов и операции, допустимые над ними. Операции описываются отношениями. Встроенными классами являются: bool, numeric, text, prog и any. В простейшем случае явного описания класса задаются его компоненты (например, point:(x; numeric; y: numeric)–определяет класс объектов точка с числовыми компонентами x и y).

Отношения могут быть двух видов: в виде уравнений и программные (как в языке УТОПИСТ). Например, при описании комплексных чисел имеем

```
comp (re:numeric; im:numeric; mod:numeric;
rl:mod**2:=re**2+im**2);
```

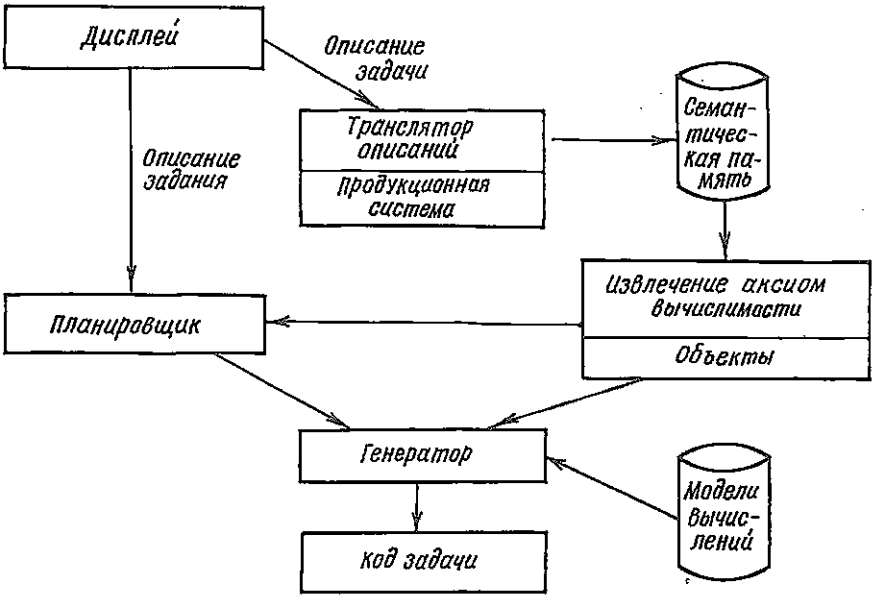


Рис. 11.6

Все вычисления организуются на уровне объектов. При этом вычисление представляет собой: 1) порождение или уничтожение объектов; 2) изменение или запрос их значений. Значение объекта может быть простым или составным. Элемент составного объекта — значение другого объекта, т. е. составной объект может иметь произвольную иерархическую структуру. Если отсутствует инициализация объекта при его порождении, то начальные значения составного объекта равны nil (0). Существуют предопределенные объекты (константы), которыми являются числа, текстовые константы и булевы значения.

Программы — объекты специального типа, которые должны задаваться явно. Программа состоит из спецификации (входные и выходные переменные) и тела (текст на языке программирования).

Правила имеют форму импликаций и выражают факты и законы, по которым можно выводить новые факты (строить описания, создавать объекты). Правила описывают связи между классами или объектами.

СИРЗ реализуется на основе языка программирования микроЭВМ (рис. 11.6). Ее назначение — планирование решения задач с использованием функциональных модулей. Задача на базе СИРЗ решается следующим образом. Описывается вычислительная обстановка (перечень определенных ранее абстрактных объектов), записываются конкретные условия задачи через компоненты абстрактных объектов и цель.

Абстрактный объект характеризуется своей структурой отношения между его компонентами, отношения выражаются программно или через уравнения. Если задание в описании условия правильно, а задача поставлена конкретно, то СИРЗ генерирует результирующую программу, в которую войдут необходимые модули, используемые в описании вычислительной модели. На персональ-

ной ЭВМ с развитой ОС, множеством языков программирования, понятием загрузочного модуля и аппаратом передачи параметров СИРЗ выступает как средство планирования решения задачи и интеграции программных модулей.

Описание задачи в СИРЗ выполняется в стиле языка УТОПИСТ в виде

ОПИСАНИЕ::= { описание структуры объекта | описание отношения }

Описание структуры объекта::= 1 ИМЯ
2 ИМЯ { базовый тип, | неопределенный тип, | копия }
2
.
.
.

Описание отношений::= { уравнение | программа }

Уравнение::=два уравнения { арифметическое выражение=арифметическое выражение | имя логического выражения | логическое выражение | имя=имя }

Программное отношение::= модуль ИМЯ { входы | выходы }

Описания задачи запоминаются в семантической памяти. С последней выполняются следующие операции: включение нового описания, удаление описания, модификация описания. Описания могут быть заданы непосредственно в условиях задачи или копироваться из семантической памяти, реализованной на основе иерархической файловой системы.

Оператор постановки задачи имеет следующий вид:

НА имя абстрактного объекта ПО условия задачи ВЫЧИСЛИТЬ имя объекта

Суть метода структурного синтеза заключается в интерпретации реальных уравнений и модулей как предложений вычислимости $\Gamma \in X \xrightarrow{f} Y$, где Γ – условие применения предложения либо истина (аксиома); X – множество входных объектов; Y – множество выходных объектов; f – соответствует некоторой реализуемой функции. Рассмотрим пример.

Пример. Пусть имеем три объекта: скорость, время и расстояние. Опишем эту модель в виде

1. ДВИЖЕНИЕ
   2. ПУТЬ: вещ;
   2. СКОРОСТЬ: вещ;
   2. ВРЕМЯ: вещ;

Уравнение ПУТЬ:= СКОРОСТЬ*ВРЕМЯ
СКОРОСТЬ:=ПУТЬ/ВРЕМЯ
ВРЕМЯ:=ПУТЬ/СКОРОСТЬ

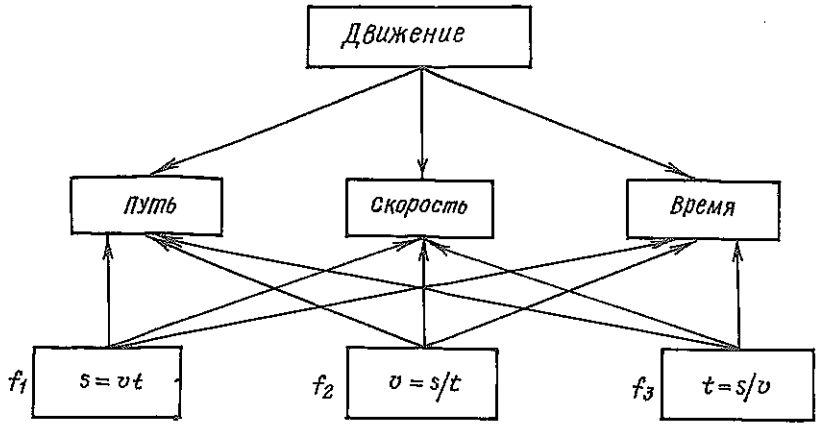


Рис. 11.7

Семантическая сеть для данной вычислительной модели представлена на рис. 11.7. Задача может быть описана следующим оператором постановки:

НА ДВИЖЕНИЕ ПО ВРЕМЯ=..., ПУТЬ=... ВЫЧИСЛИТЬ СКОРОСТЬ

Извлекаются из модели аксиомы: 1) Скорость , время $\xrightarrow{f_1}$ путь; 2) Путь, время $\xrightarrow{f_2}$ скорость; 3) Путь, скорость $\xrightarrow{f_3}$ время. Поскольку в данной ситуации известны значения пути и времени, применима только вторая аксиома, по которой вычисляется скорость. Поскольку скорость получена, то дальнейшее применение аксиом вычислимости не требуется. Таким образом, доказана теорема вычислимости скорости. Из доказательства может быть извлечена, например, программа на языке БЕЙСИК:

```
10 DIM S,T,V
20 S:=10; T:=1; V:=T/S;
30 PRINT V"км/час"
40 END
```

Если в примере аксиому 2 описать не уравнением, а ссылкой на функцию, вычисляемую подпрограммой, то ход доказательства теоремы вычислимости остается без изменения, но в результирующей программе v = s/t будет заменено на обращение к программе, вычисляющей f_2 .

Рассмотрим обобщенный алгоритм построения плана решения задачи в СИРЗ (рис. 11.8). На основании запроса, пользуясь содержимым базы знаний, выполняются следующие действия: 1) строятся таблицы данных, объектов, типов, системных фактов; 2) включаются в рабочий контекст все уравнения и отношения, которые связаны с используемыми типами; 3) если с используемыми типами были связаны теоремы, то продукционная система на основании их системных фактов и фактов секции "Дано" расширит контекст задачи (введутся дополнительные уравнения и отношения); 4) разрешаются уравнения относительно всех переменных, формируются соответствующие отношения вычислимости; 5) выполняется поиск всех возможных путей решения,

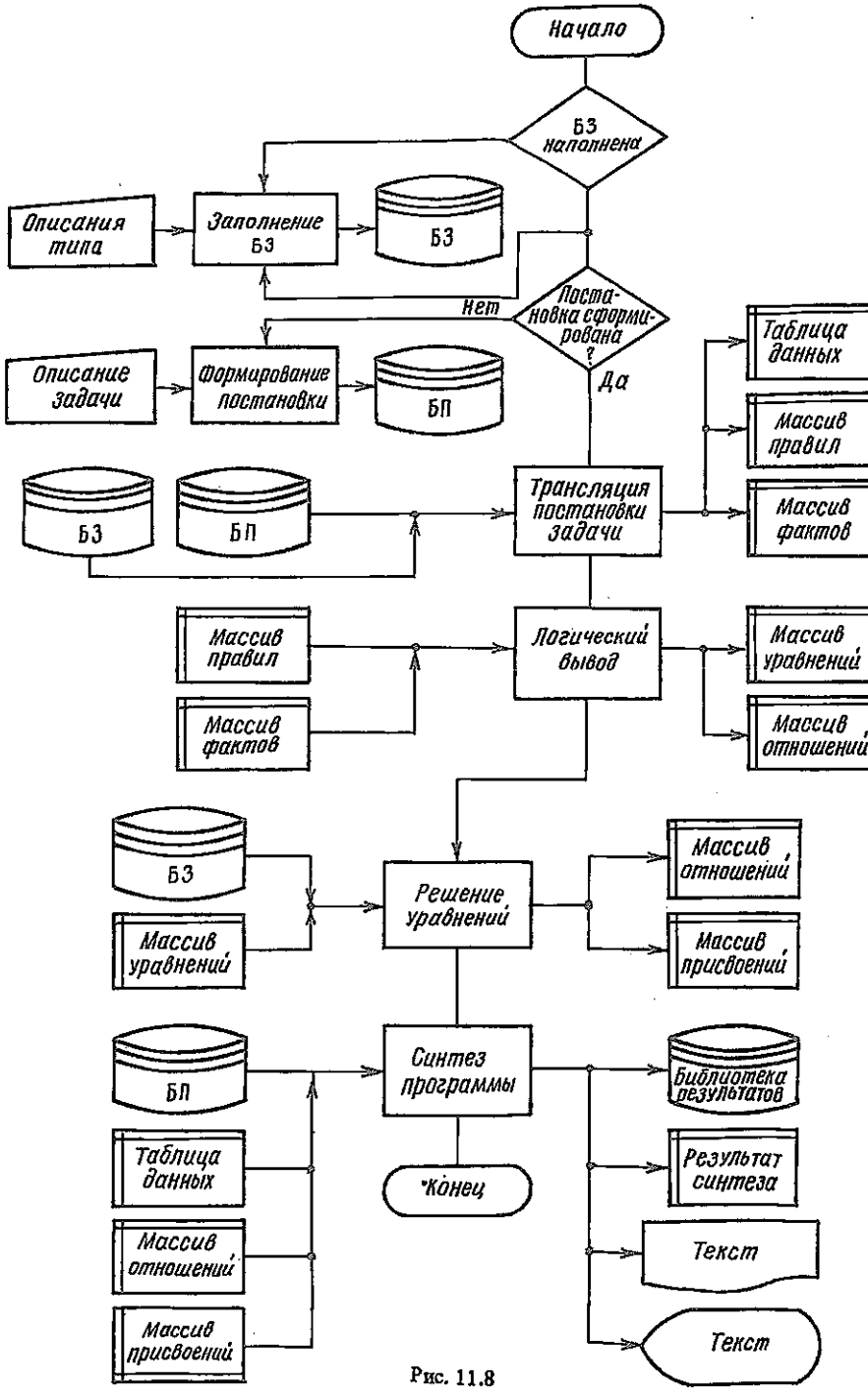

Начало
БЗ наполнена
Описания типа
Заполнение БЗ
БЗ
Нет
Постановка сформирована ?
Да
Описание задачи
Формирование постановки
БП
Таблица данных
Массив правил
Массив фактов
БЗ
БП
Трансляция постановки задачи
Массив правил
Массив фактов
Логический вывод
Массив уравнений
Массив отношений
БЗ
Массив уравнений
Решение уравнений
Массив отношений
Массив присвоений
БП
Таблица данных
Массив отношений
Массив присвоений
Синтез программы
Конец
Библиотека результатов
Результат синтеза
Текст
Текст

Рис. 11.8

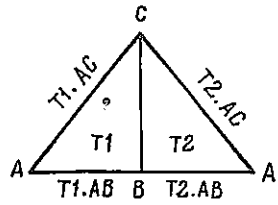


Рис. 11.9

число которых минимизируется; 6) на основании найденного плана решения генерируется ответ на выбранном языке (программирования или естественном).

Рассмотрим генерацию решения для конкретной ситуации.

Задача. У двух треугольников T1, T2 смежный катет BC. У T1 заданы гипотенуза и катет AB, у T2 – гипотенуза. Найти оба катета T2 (рис. 11.9). Структура запроса будет следующая. Группа понятий: геометрия; объекты T1, T2: треугольник

Д а н о: уравн. T1.AC длина=8; уравн: T1.AB длина=5; уравн: T2.AC длина=15; уравн: T1.BC=T2.BC: Н а й т и: T2.BC, T2.AB

В результате формируются уравнения для T1 и T2, на основании которых генерируются отношения вычислимости: ОТН:T1.AB.длина. T1.BC.длина → T1.AC.длина=SQR (T1.AB.дл^2 + T1.BC.дл^2)

ОТН:T2.AC.длина.T2.BC.длина→T2.AB.длина=SQR(T2.AC.дл^2+T2.BC.дл^2)

ОТН:T1.BC.длина=T2.BC.длина

На основании первого отношения вычисляется T1.BC≅6.31. С помощью остальных отношений определяется T2.AB≅13,6.

11.6. ПОНЯТИЕ О СИСТЕМЕ ЛОГИЧЕСКИХ ВЫВОДОВ

Нередко встречаются задачи в следующей постановке: имеется некоторое количество исходных объектов и правил построения новых объектов из уже существующих; построить все возможные объекты или отдельные из них. Такие системы называются дедуктивными или исчислениями. Классическим примером исчислений является исчисление высказываний, более известное как алгебра логики. Однако для описания внешнего мира и поиска решений в ИИ широко используется язык и аппарат исчисления предикатов, представляющих собой развитие исчисления высказываний.

Предикатом называется функция от любого числа аргументов, принимающих значение 1 или 0. Аргументы принимают значения из произвольного множества, называемого предметной областью. Предикат от n переменных является n-местным. Например, $F(x)$, $G(x, y)$ – предикаты, причем F, G – предикатные буквы, а x, y – предметные переменные. Вместо предметных переменных в предикат могут быть подставлены константы и функции $f(y_i)$. Предикат $F(x)$, заданный на предметной области M, интерпретируется как высказывание "x обладает свойством F".

Основным объектом исследований в логике предикатов является формула. При ее определении используется понятие терма, который задается следующим образом: всякая предметная переменная (константа) есть терм; если f – n-местная функция, а f_1, ..., f_n – термы, то $f(f_1, ..., f_2)$ – терм. Никакие

другие выражения не являются термами. Формула определяется следующим образом: всякая элементарная формула есть формула, а комбинации $(\forall x P)$, $(\exists x P)$, $\bar{P}$, $(P \vee R)$, $(P \wedge R)$, $(P \to R)$ – также формулы, если P и R являются формулами. Причем элементарной формулой будет $F(t_1, ..., t_n)$, если $f_1, ..., f_n$ – термы, а P_n – местный предикат.

Формула имеет определенный смысл, если существует какая-либо ее интерпретация, т. е. конкретизация предметной области и установка соответствий между символами (предметной константы, функциональной и предикатной букв), входящими в формулу, с одной стороны, и элементами, функциями и отношениями – с другой.

Исчисление предикатов широко используется для решения задач. Метод решения при этом основан на представлении задачи в виде теоремы: формула F логически следует из формулы (множества) Φ_0. Доказательство этой теоремы сводится к показанию невыполнимости формулы $\Phi_1 = \Phi_0 \overline{UF}$. При доказательстве теоремы все формулы в Φ_1 представляются в виде дизъюнкции литералов. Литерал – элементарная формула или ее отрицание.

Основным методом доказательства является метод резолюций, который заключается в следующем. Пусть имеются два дизъюнкта $R(x) \vee P(x)$ и $\overline{R}(x) \vee \overline{T}(x)$. Из них выводится дизъюнкт $P(x) \vee T(x)$, который логически следует из исходных. Вновь полученный дизъюнкт называется резольвентой, а процесс его получения – резолюцией. Доказано, что если какая-то интерпретация удовлетворяет исходным предложениям, то она удовлетворяет и резольвенте.

Доказательство теоремы (решение задачи) с помощью принципа резолюций заключается в следующем. Доказательство теоремы эквивалентно доказательству невыполнимости объединения $\Phi_0 \overline{UF}$. Для доказательства невыполнимости последнего достаточно показать, что в случае применения принципа резолюций к множеству Ф и его резольвентам получается пустое предложение. Принцип резолюции является полным в смысле следующей теоремы, доказанной Д.Робинсоном: если конечное множество предложений невыполнимо, то опровержение может быть получено за конечное число шагов применений принципа резолюций.

Непосредственное применение принципа резолюций ведет к полному перебору вариантов решения. Поэтому на практике используют различные стратегии: упрощения, очищения и упорядочивания. При стратегии упрощения путем исключения отдельных предложений снижается число роста новых резольвент. В стратегии очищения применяют базовые предложения, исключаются аксиомы (факты). В стратегии упорядочивания в резолюционный процесс включаются определенные предложения и резолюции (одночлены). Рассмотрим пример.

Пример. Пусть триггер является частью регистра, а регистр – частью микропроцессора. Необходимо показать, что триггер является частью микропроцессора. Для цели автоматического доказательства используем аксиому транзитивности, определяющую отношение P – быть частью. Запишем эту аксиому в виде

Если $P(x, y)$ и $P(y, z)$, то $P(x, z)$ для любых x, y, z.

Преобразуем ее по правилу $A \leftarrow \overline{B \& C} = A \vee \overline{B} \vee \overline{C}$ и получим $P(x, y) \vee \overline{P(y,z)} \vee \overline{P(x,z)}$.

Окончательно условие задачи запишется в виде

1) $P(x, z)$ $\overline{P(x, y)}$ $\overline{P(y, z)}$; 2) $P(t, p)$; 3) $P(p, m)$; 4) $P(t, m)$ – инвертированный запрос

Здесь символы t , p , m – константы; x , y , z – переменные. Необходимо найти в различных предложениях два литерала такие, чтобы после допустимых подстановок один становился отрицанием другого до тех пор, пока не получим пустое предложение. Это будет свидетельствовать о доказательстве запроса. Сделаем подстановку в первое предложение $x = t$, $y = p$. В результате получим преобразованное первое предложение P(t, z) P(t, p) P(p, z) , которое со вторым дает резольвенту P(t, z) P(p, z). Делаем для этой резольвенты подстановку z=m и получаем P(t, m) P(p, m). Третье и шестое предложения снова вступают в резолюционный процесс и получаем шестое предложение – резольвенту P(t, m), которая, вступая в резолюционный процесс с запросом 4, дает пустой дизъюнкт. Это указывает на противоречие запроса с отрицанием, а следовательно, на доказательство нашей задачи.

Принцип резолюций и решение задач на его основе легли в основу построения языков логического программирования – различных вариантов ПРОЛОГа. Рассмотрим элементы основных конструкций языка и их семантику без учета встроенных процедур и средств управления вычислительным процессом, которые относятся к "чистому" ПРОЛОГу.

ПРОЛОГ-программа – это множество правил, каждое из которых имеет вид B_0 B_1, ..., B_m , где B_i –атомарные формулы; B_0 – левая часть правила, а B_1–B_m – тело правила. При $m = 0$ такие правила называются фактами. Атом имеет вид $P(t_1, ..., t_n)$, где P–n-й предикатный символ, а t_1, ..., t_n – термы.

Запрос имеет вид C_1, ..., C_r, где C_i – атомы, $i = 1,..., r$. Каждая программа фиксирует множество употребленных в ней предикатных символов, которые могут использоваться в запросах к ней, а также множество переменных и функциональных символов. Тем самым определено множество атомов.

Для задания семантики вводится понятие подстановки, которая применяется к произвольному выражению и заменяет в нем каждое вхождение переменной x_i на терм t_i. Формально подстановка – это произвольная функция θ:X → T, которая применяется к множеству выражений – термов, атомов, правил и запросов. Унификатором атомов A_1 и A_2 является подстановка θ, такая, что $\theta A_1 = \theta A_2$. Подстановки были использованы при организации вычислительного процесса доказательства.

Пусть запрос Q имеет вид C_1 , ..., C_r и в нем выделен атом C_i , а B_0 B_i , ..., B_m – вариант некоторого правила программы, такой, что все его переменные отличны от переменных запроса Q . Если θ – самый общий унификатор для атомов C_i и B_0 , то считается, что запрос $Q' = (C_1, ..., C_i, B_0, ..., B_m, C_1, ..., C_r)$ выводим из запроса Q . Получение Q' из Q – это элементарный шаг вычислений в ПРОЛОГе. Шаг может кончиться неудачей, поскольку не всегда существует общий унификатор для C_i и B_0 . Производительность интерпретаторов ПРОЛОГа измеряют количеством таких шагов в секунду (LIPS – логических выводов в секунду).

Одной из широко распространенных версий языка ПРОЛОГ является его реализация на ПЭВМ – микроПРОЛОГ. Этот язык характерен тем, что программа на нем включает описания функций, тело которых задает обращение к другим функциям языка или пользователя.

Основные элементы языка – идентификаторы, переменные, числа и списки. Идентификаторы определяются как последовательность символов, начинающаяся с буквы. В качестве переменных используются имена, начинающиеся с букв x , y , z , X , Y , Z. Числа в языке записываются в форме целых, веще-

ственных (46, 1.24, 7.Е-1). Список имеет формат (T_1, ..., T_n), где T_i – идентификатор, число или переменная.

Программа представляет собой описание правил и фактов, на основании которых можно получить новые факты. Факты (простые предложения) записываются в виде < A >< O > или < A1 >< O >< A2 >, где A_i – аргумент, O – имя отношения. Правило имеет вид < ПП >if < $ПП_1$ >& < $ПП_n$ >, где $ПП_1$ – простое предложение. Например, дед (X,Z) if отец (X, Y)& отец (X,Z).

Основные операторы языка позволяют:

записывать предложение – add < предложение >;

добавлять группу – accept < имя отношения >;

выводить программу на экран – list;

уничтожать предложение в БД – delete или отношения kill < имя отношения >;

редактировать предложения edit < имя отношения >

Команды запросов к программе дают возможность проверить выполнение факта

is (предложение 1 &, ..., & предложение N) true or false

или получить аргумент, имя которого в запросе заменено на переменную

wich (образец ответа : предложение1 ... предложение N)

Рассмотрим простую программу на языке микроПРОЛОГ, занеся ее в БД:

```
&.add(Пётр отец Ивана)        Посмотрим содержимое БД
&.add(Иван отец Ольги)           &.list all
&.accept(женат)                  Пётр отец Ивана
  женат.(Пётр)                   Иван отец Ольги
  женат.(Иван)                   Пётр женат
                                 Иван женат

      Зададим вопросы
    is (Пётр отец Ивана) YES;
    is (Пётр отец Ольги) NO;
    wich(y: Пётр отец x&x отец y)
      Ольга
```

Дальнейшим развитием идей концептуального программирования является введение продукционных правил. Эти правила выражают законы, применимые к предикатам и объектам. Правила имеют форму импликаций и выражают факты и законы, по которым можно выводить новые факты. В основном правила используются для автоматического расширения описания задачи с учетом данного контекста: $P_1 \& P_2 \& , ..., \& P_n \to Q$, $n \geq 0$; при $n = 0$ имеем факт $\to Q$, где P, Q – атомарные формулы, включающие имя предиката и список параметров.

Имя предиката в продукционных правилах может иметь самую различную природу: абстрактный символ, имя программы (вычисляемый предикат), имена абстрактных объектов, описания. Параметры являются либо свободными переменными либо предметными (имена объектов и их компоненты). Для построения новых фактов по данным правилам используют следующий алгоритм (СИРЗ), предназначенный для автоматического расширения описания задачи на основании условий и правил:

1) для каждого атома посылочной части правила представляется список фактов на место этого атома;

2) из этого списка исключаются все использованные в этом правиле комбинации, а также те комбинации, о которых ранее было установлено, что их применять нельзя. Для остальных комбинаций выполняются п. 3, 4;

3) устанавливается соответствие между свободными переменными и объектами подставленных фактов, называемое означиванием переменных;

4) на основании правила при данной комбинации выводится следствие, если оно дает новый факт или новое описание и если означивание выполнено корректно, т. е. одной свободной переменной во всех предикатах соответствует один и тот же объект.

Вывод следствия заключается в добавлении к списку фактов полученного следствия. Работа алгоритма заканчивается, если ни одно правило не дает новых фактов или описаний. Рассмотрим пример.

Пример. Имеем факты:

→человек (Иван)→человек (Петр)→человек (Сергей)→отец (Иван,Петр)→отец (Сергей, Иван); и правило: человек (ωу) & отец (ωх, ωу) & отец (ωу, ωz)→дед (ωх, ωу)

На основании фактов и правила получаем новый факт → дед (Сергей, Петр).

Пространством вычислений для программы S является множество всех запросов. Вычислением запроса Q в пространстве вычислений называется последовательность, возможно бесконечная, запросов Q_1 , ..., Q_i , такая, что Q_{i+1} выводим из Q_i . Если из Q_n не выводим ни один запрос, то вычисление конечно.

Описанное пространство содержит пути всех вариантов вычислений. Каждая ПРОЛОГ-машина реализует ту или иную стратегию обхода пространства вычислений. Таких стратегий много (как последовательных, так и параллельных). Наиболее распространенной является последовательная, которая реализуется существующими архитектурами микроЭВМ. В этой стратегии дерево вычислений упорядочивается в двух направлениях, слева – направо на атомах каждого запроса и сверху – вниз на правилах программы.

На каждом шаге вычислений выделенным считается левый атом запроса и к нему применяется первое правило программы (с данным предикатом). При этом в стеке запоминаются возможные разветвления (следующее правило с тем же предикатом). Если вычисление заходит в тупик, происходит возврат к месту последнего разветвления. Такой же возврат осуществляется и при успешном вычислении для получения других ответов. Интерпретатор кончает работу, когда стек разветвлений пуст. Основное достоинство данной стратегии – простота, недостаток – большой перебор вариантов для отдельных программ и запросов (возможно попадание в бесконечную ветвь). Поэтому применяются и другие стратегии (например, сначала по ширине дерева).

Важным достоинством языка логического программирования является возможность распараллеливания вычислений, что требует других, не "фоннеймановских" машин. Обычно различают два типа параллельности: ИЛИ-параллельность (параллельное применение к выделенному атому запроса всех правил программы с данным предикатом) и И-параллельность (параллельная обработка всех или нескольких атомов запроса). Возможна параллельность и при выполнении шага вычисления (например, при унификации).

Рекомендуемая литература: 2, 16.

12. ПЕРСПЕКТИВЫ РАЗВИТИЯ ЭВМ

12.1. ПРОГРАММЫ ЭВМ ПЯТОГО ПОКОЛЕНИЯ

Предполагается, что в 90-е гг. – время ЭВМ пятого поколения, системы обработки информации станут важнейшим фактором жизни общества в области экономики, искусства, науки, управления, международных отношений, образования и т.д. С точки зрения пользователя ЭВМ пятого поколения должны отличаться следующими пятью особенностями: 1) простотой пользования функциональными возможностями (от пользователя не требуется профессиональная подготовка); 2) моделированием "человеческих" функциональных возможностей, таких, как построение доказательств и принятие решений; 3) гибкостью конфигурации, обеспечивающей приспосабливаемость ЭВМ к условиям выполнения широкого диапазона работ; 4) наличием функций, облегчающих программирование; 5) критериями, определяющими эффективность и надежность ВС.

В целом базовый компьютер пятого поколения, видимо, будет обладать тремя основными функциональными возможностями: накапливать информацию, решать задачи и общаться с людьми. Каждая из этих функций должна выполняться с помощью специализированных аппаратных и программных систем (модулей), показанных на рис. 12.1. В результате пользователь получит возможность сформировать конфигурацию речевой машины с интересующей его функцией или робота с развитым зрительным восприятием и т.д.

Однако успех в создании таких систем, т.е. успех проекта машин пятого поколения, зависит от качества информационной обработки знаний.

Аппаратное решение, обеспечивающее основу такого рода деятельности, не будет соответствовать классической архитектуре машины фон Неймана. С развитием технологии СБИС системы параллельной обработки (типа магистральных и матричных) получат широкое распространение, так как скорость выполнения операций, которой можно достичь, применяя перспективную элементную базу, ограничивается скоростью света. С перспективными современными подходами к параллельной обработке данных на базе систолических матриц и транспьютеров мы познакомимся при изучении транспьютеров и систолических процессоров.

В настоящее время хорошо известным языком обеспечения процедуры решения проблем является язык ПРОЛОГ. Предполагается, что быстродействие параллельной машины вывода (векторная обработка около 1000 процессоров) будет составлять от 100 до 1000 млн логических выводов (ЛВС) в секунду. По описанию К.Фучи структура машины пятого поколения должна иметь семь уровней. Самые верхние уровни, на которых реализованы прикладные системы и человеко-машинные интерфейсы с использованием естественного языка, предполагается выполнить в виде средств обработки знаний на виртуальной машине. Первый уровень виртуальной машины сможет обращаться к

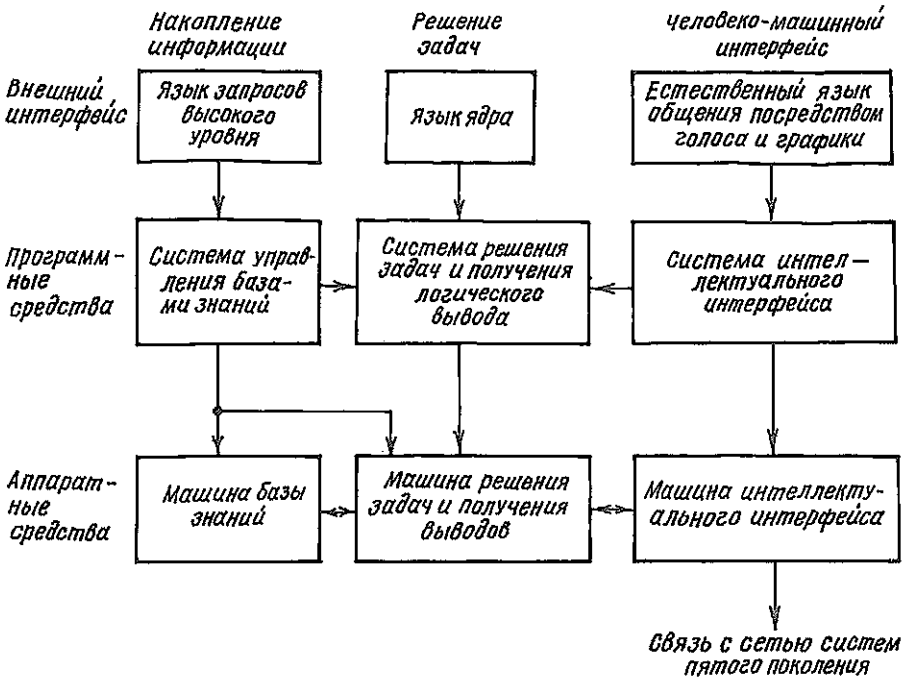


Рис. 12.1

следующему уровню, образованному в действительности тремя другими виртуальными машинами: процессором, базой данных и машиной сети передачи данных. Ряд процессоров специального назначения составит еще один уровень виртуализации; сюда войдут матричный процессор, процессор глобальных связей, процессор локальных связей, машина на базе данных, процессор операционной системы и т.д. Предполагается, что эти виртуальные машины будут выполнены на реальных машинах (построенных на СБИС) и между ними распределятся перечисленные функции.

Выделяются три фазы проекта создания машин пятого поколения (рис. 12.2): 1) создание технологии для аппаратных и программных средств; 2) разработка подсистем логического вывода и баз знаний; 3) интегрирование этих подсистем для получения прототипа системы.

Приведем характерные особенности ЭВМ пятого поколения: 1) значительно расширятся выполняемые функции, типы и уровни ЭВМ — от машин со сверхвысокой производительностью до процессоров, имеющих специализированное назначение, персональных и встроенных ЭВМ; 2) ослабнет существовавшая тенденция к универсализации и возрастет роль специализации ЭВМ; 3) большее внимание будет уделено архитектурным решениям, отличным от архитектуры ЭВМ фон-Неймана; 4) возрастет значение новых микроархитектур, системной архитектуры; широкое применение найдут системы, образованные комбинацией разного числа процессоров и других устройств, программного обеспечения и модулей ПЗУ с "защитными" программами.

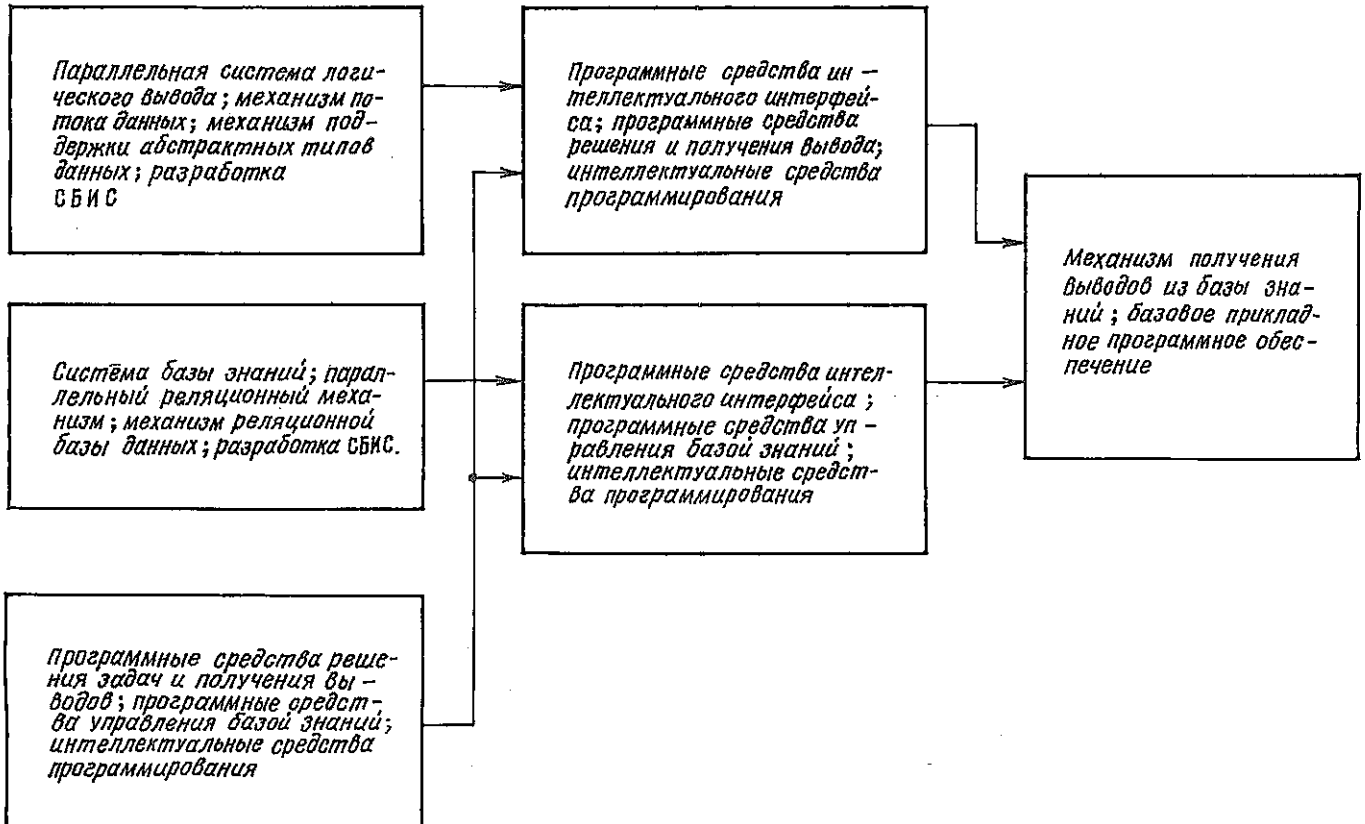

Параллельная система логического вывода; механизм потока данных; механизм поддержки абстрактных типов данных; разработка СБИС
Система базы знаний; параллельный реляционный механизм; механизм реляционной базы данных; разработка СБИС.
Программные средства решения задач и получения выводов; программные средства управления базой знаний; интеллектуальные средства программирования
Программные средства интеллектуального интерфейса; программные средства решения и получения вывода; интеллектуальные средства программирования
Программные средства интеллектуального интерфейса; программные средства управления базой знаний; интеллектуальные средства программирования
Механизм получения выводов из базы знаний; базовое прикладное программное обеспечение

Рис. 12.2

Следует отметить, что необходимость создания ЭВМ нового поколения появилась, прежде чем определились такие аспекты проблемы, как аппаратные и архитектурные решения, элементная база, программное обеспечение, языки, обработка знаний и масштаб планов проводимых исследований и разработок.

12.2. СИСТЕМЫ НА ОСНОВЕ ТРАНСПЬЮТЕРОВ И ЯЗЫК ОССАМ ДЛЯ НИХ

При использовании традиционных микропроцессорных средств не всегда удается удовлетворить требованиям мультипроцессорных систем, сохраняя умеренную стоимость последних. Это привело к необходимости создания нового вычислительного модуля для построения сверхвысокопроизводительных систем с большим числом процессорных элементов (ПЭ) (порядка 10^2–10^3) БИС. Таким элементом стал транспьютер (transputer), разработанный фирмой Inmos (Великобритания).

Транспьютер предназначен для реализации программ, написанных на специально разработанном языке параллельного программирования ОССАМ. С помощью этого языка функционирование любой программной или аппаратной системы описывается как выполнение некоторой совокупности параллельных процессов, взаимодействующих с помощью обмена сообщениями.

Модель взаимодействия процессов путем обмена сообщениями по логическим каналам, связывающим эти процессы, оказала решающее влияние на архитектуру мультитранспьютерных систем. Эти системы строятся не на основе традиционной шинной структуры, а используют для связи между ПЭ двухточечные последовательные каналы, которые по существу являются аппаратной реализацией логических каналов, определяемых в языке ОССАМ. Этим обеспечиваются большие возможности для наращивания числа ПЭ в системе с целью увеличения ее производительности. В шинных структурах существует предел такого наращивания вследствие повышения интенсивности обменов данными при увеличении числа ПЭ и ограниченной пропускной способности шинной структуры. Таким образом, транспьютер, язык ОССАМ и система разработки программ на этом языке составляют единый комплекс средств для построения мультитранспьютерных систем.

Принципы построения транспьютеров определили создание семейства программируемых СБИС, которые могут быть названы транспьютерными системами. Типичный представитель этого семейства – СБИС транспьютера, содержащая процессор, запоминающие устройства и устройства интерфейса каналов связи, которые обеспечивают возможность попарных соединений между транспьютерами.

Семейство транспьютерных СБИС, выпускаемых фирмой Inmos, включает ряд СБИС транспьютеров, а также внешних ЗУ, предназначенных для подключения к транспьютерам. Основными типами транспьютеров являются: транспьютер со статическим ОЗУ (IMS T424), транспьютер – связной адаптер, транспьютер – контроллер графических устройств, транспьютер – контроллер массовой памяти. Каждый из этих типов реализуется в виде ряда СБИС, различающихся процессором, видом и конфигурацией внутреннего ЗУ, быстродействием и т.д.

Транспьютер IMS T424 представляет собой СБИС, которая содержит 32-

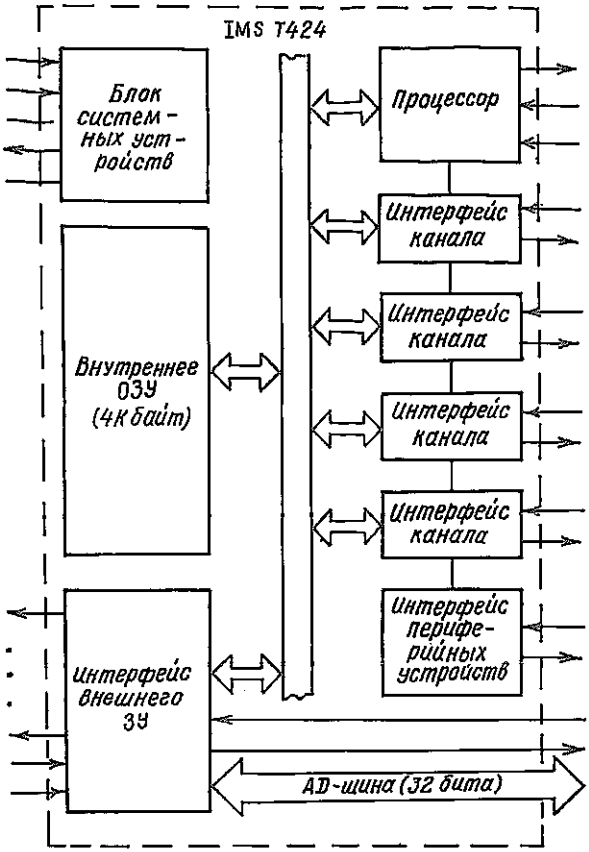


Рис. 12.3

разрядный процессор, внутреннее ОЗУ емкостью 4 К байт, блок системных устройств, четыре блока интерфейса каналов связи, блок интерфейса внешнего ЗУ и блок интерфейса периферийных устройств (рис. 12.3).

Процессор обеспечивает выполнение программ, написанных на алгоритмических языках высокого уровня, и высокопроизводительную связь между параллельными процессами. В основу его построения положены принципы архитектуры ЭВМ с сокращенным набором команд (т. е. архитектуры типа RISC–Reduced Instruction Set Computer). Операторы языка высокого уровня обрабатываются в "стеке оценки", состоящем из 32-разрядных регистров. Эти регистры задаются командами неявно, чем обеспечивается компактное кодирование команд. Корректная и оптимальная последовательность таких команд легко генерируется компилятором. Все команды, имеющие длину 1 байт, просты для декодирования, что гарантирует высокую производительность процессора (рис. 12.4).

Процессор осуществляет непосредственную реализацию модели параллельных вычислений и связи между процессами, принятой в языке ОССАМ. В про-

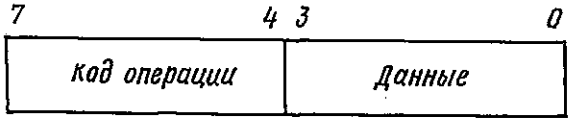


Рис. 12.4

цессоре имеется планировщик процессов, который позволяет выполнять любое число параллельных процессов в режиме разделения времени. Связь процессов реализуется с помощью операций пересылки данных между блоками внутреннего ОЗУ. Малое время коммутации процессов возможно при использовании внутреннего ОЗУ совместно с небольшим числом регистров, необходимых для формирования контекста процесса.

Процессор обеспечивает два уровня приоритетов. Процессы наивысшего приоритета используются для маршрутизации сообщений и быстрой реакции на внешние события. В процессоре реализована также возможность альтернативного ввода (соответствующая концепции языка OCCAM), что позволяет программировать на языке высокого уровня аппаратные и программные прерывания. Процессор содержит таймер, и выполняемый процесс может либо следить за временем таймера, либо ждать достижения определенного момента времени. Процессор рассматривает весь объем памяти транспьютера как линейное пространство адресов емкостью 2^{32} байта (4 Г бит) и не различает внутреннее и внешнее ЗУ. Внутреннее статическое ОЗУ емкостью 4 К байт представляет собой массив, содержащий 1 К 32-разрядных слов.

Как правило, наиболее эффективно размещение во внешнем ЗУ не данных, а программ. Это объясняется тем, что цикл внешней памяти может быть использован либо для обращения к одному слову данных, либо для выборки четырех команд. Поэтому в основном программы для доступа к данным требуют большего числа обращений к памяти, чем для выборки команд. Если программа и локальные переменные могут быть размещены во внутреннем ОЗУ так, что большинство обращений к памяти составляют обращения к внутреннему ОЗУ, то производительность процессора близка к производительности, достигаемой при размещении во внутреннем ОЗУ всех программ и данных.

Загрузка процессора выполняется им самим либо по программе, находящейся во внешнем ПЗУ, либо с использованием любого из четырех интерфейсов каналов связи транспьютера.

С помощью блоков интерфейса каналов реализуются последовательные каналы связи, которые обеспечивают высокоскоростную связь между транспьютерами и позволяют строить разнообразные структуры систем на основе транспьютеров. Эти каналы обеспечивают стандартную скорость передачи данных (10 М бит/с) и строятся в соответствии со стандартами фирмы Inmos (они называются Inmos-каналами). Интерфейсы каналов связи и процессор работают параллельно. Интерфейс каждого канала является автономным и имеет вход и выход, связанные с внешними выводами СБИС транспьютера и используемые для передачи как данных, так и сигналов, необходимых для реализации протокола связи (т. е. управляющей информации). Сообщения между транспьютерами передаются в виде последовательностей пакетов, каждый из которых содержит 1 байт данных. После передачи пакета данных транспьютер-отправитель ожидает получения подтверждения от транспьютера-получателя о том, что последний готов принять следующий пакет данных.

Цикл процессора составляет 50 нс. Передача данных осуществляется асинхронно.

Интерфейс внешнего ЗУ обеспечивает для процессора быстрый доступ к памяти с помощью 32-разрядной шины адресов и данных (AD-шины). Возможно использование разнообразных внешних ЗУ, так как данный интерфейс реализует различные временные режимы работы, входящие в фиксированный для транспьютера набор. Этот набор включает как режимы, необходимые для работы с быстродействующими статическими и динамическими ОЗУ с временем обращения, равным трем циклам процессора, так и медленные универсальные, обеспечивающие время обращения, равное девяти циклам процессора. Требуемый режим, соответствующий определенной конфигурации интерфейса, выбирается путем соединения двух выводов схемы интерфейса с соответствующими выводами AD-шины.

Транспьютер спроектирован таким образом, чтобы обеспечить эффективную реализацию специально разработанного для него языка параллельного программирования OCCAM. Этот язык позволяет в явном виде программировать функционирование параллельных систем. С одной стороны, он, как и ассемблер, обеспечивает эффективное использование памяти и высокую производительность при выполнении программ, а с другой стороны, как языки высокого уровня – большую эффективность при разработке программного обеспечения.

Язык OCCAM является основой методологии проектирования параллельных систем на базе транспьютеров аналогично тому, как булева алгебра служит основой методологии проектирования современных электронных систем на базе логических элементов.

Задача проектировщика транспьютерных систем облегчается благодаря архитектурному соответствию языка OCCAM и транспьютера. Программа, выполняемая одним транспьютером, формально эквивалентна одному процессу языка OCCAM. Следовательно, систему, состоящую из некоторой совокупности транспьютеров, можно непосредственно описать с помощью программы на этом языке. Основное понятие, используемое в языке OCCAM, – это понятие процесса. Каждый процесс рассматривается как некоторый объект ("черный ящик"), характеризующийся своим внутренним состоянием. Процесс может его изменять и взаимодействовать с другими процессами посредством обмена сообщениями,используя для этого двухточечные каналы связи (point-to-point communication channels). Под каналом понимается языковая конструкция, обеспечивающая возможность организации связи между процессами, а не физический канал (link), используемый для связи между процессорными модулями.

Все процессы состоят из следующих примитивных (основных) процессов: присвоения, ввода, вывода, пропуска, останова. Процесс присвоения определяет значение выражения и присваивает его некоторой переменной. Процессы ввода и вывода используются для организации взаимодействия между более сложными процессами.

Два параллельных процесса могут осуществлять связь друг с другом по однонаправленному каналу, связывающему эти два процесса. При выполнении такой связи один процесс посылает (выводит) сообщение в канал, а другой принимает (вводит) это сообщение из канала. Ключевая концепция языка

OCCAM состоит в том, что эта связь является синхронной и осуществляется только в том случае, когда первый процесс готов выполнить вывод, а второй – ввод.

Когда программы, написанные на языке OCCAM, реализуются отдельным транспьютером, он работает в режиме разделения времени между параллельными процессами, а каналы связи между процессами реализуются посредством пересылок данных внутри памяти транспьютера. В языке OCCAM для написания программ, реализуемых в мультитранспьютерной системе, каждый транспьютер выполняет некоторое множество процессов, связь между которыми осуществляется непосредственно с помощью канальных интерфейсов и каналов связи между транспьютерами.

Язык OCCAM имеет механизм разделения и содержит три примитива: присвоение (:=), ввод (?), вывод (!). Примитив присвоения изменяет величину переменной; "вход" получает величину из канала (C ? *x* имеет следующий смысл: выполняется ввод значения некоторой величины из канала C, присвоение этого значения переменной *x* и завершение); "выход" посылает величину в канал (C ! *l* – вывод значения *l* в канал C).

Во всех реализациях языка OCCAM имеются следующие примитивные типы: CHAN, BOOL, BYTE, INT. Переменные VAR и каналы CHAN объявляются программистом в начале программы. В формировании процессов участвуют три конструкции. Конструкция-"последовательность" SEQ вызывает последовательное исполнение компонентов процесса и заканчивается на последнем компоненте. Например,

```
SEQ
C1 ? x
x:=x+1
C2 ! x
```

Эта конструкция выполняет ввод значения некоторой величины из канала C1, прибавление к ней единицы и вывод результата в канал C2.

Конструкция "параллель" PAR , позволяющая одновременно использовать компоненты процесса, прекращает функционирование только после того, как все компоненты исчерпаны. Так, конструкция

```
PAR
C1 ? x
C2 ! y
```

выполняет одновременно ввод из канала C1 в x и вывод y в канал C2.

Конструкция "условие" IF имеет следующий вид:

```
IF
условие 1
P1
условие 2
P2
...
```

Процесс P1 выполняется, если справедливо условие 1, в противном случае проверяется условие 2; если оно справедливо, то выполняется процесс P2 и т. д.

Конструкция "альтернатива" ALT означает ожидание готовности одной из операций ввода input1, input2, ... к выполнению. В случае готовности первой реализуется операция input$i$, а затем – процесс p_i ($i = 1, 2, 3, ...$).
Эта конструкция производит либо ввод из канала count и увеличение переменной counter на 1, либо ввод из канала total, а затем осуществляет последовательность процессов: вывод текущего значения переменной counter в канал out и присвоение этой переменной значения 0.

Конструкция "повторение процессов"

```
WHILE условие
  P
```

задает повторное выполнение процесса P до тех пор, пока условие остается справедливым.

Конструкция "размножение процессов" предполагает многократное выполнение одного и того же процесса, являющегося ее компонентом, причем конструкция позволяет задать требуемое число выполнений этого процесса. Например,

```
SEQ i = 0 FOR n
  P
```

позволяет описать обычный цикл. Здесь процесс P должен быть выполнен n раз. Выражение

```
PAR i=0 FOR n
  Pi
```

задает массив, состоящий из n аналогичных параллельных процессов P0, P1, ..., P_{n-1}.

В табл. 12.1 даны сводка всех основных операторов языка OCCAM и время выполнения их на транспьютере.

Таблица 12.1

Наименование	Обозначение	Количество байт	Время выполнения, нс
Арифметический	+, –	1	50
	* (умножение)	1	950
	/ (деление)	2	1950
Сравнения	>, =, <>, <, <=, >=	2	100
Логический	AND, OR	1	50
	∧, ∨, >< (XOR)	2	100
Сдвига	<< [n], >> [n]	2	50n+50
Конструкции	SEQ [n]	0	0
	PAR [n]	9n–7	450n – 200
	ALT [n]	8n+7	600n + 600
	IF [n]	3n	150n
	WHILE	4	200
Примитивы	! (вывод), ? (ввод)	4	625
	! [n], ? [n] (вектор)	4	50n+625
	:=	0	0
	:= [n] (вектор)	4	100n+300

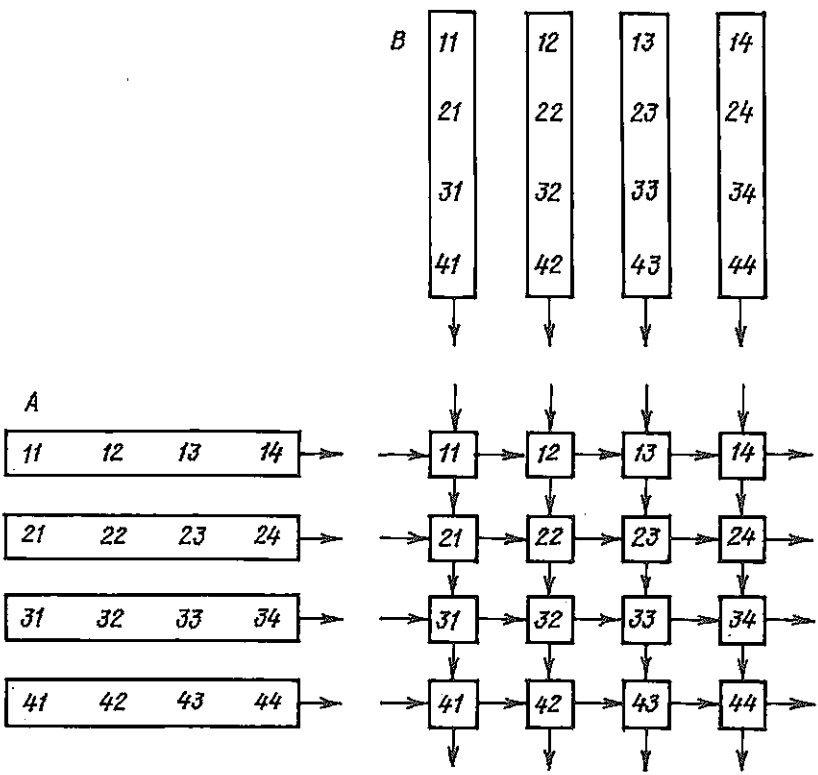


Рис. 12.5

Можно комбинировать различные примитивные конструкции (параллельные наборы последовательно выполняемых операторов или процессов), причем, для того чтобы компилятор мог определять контекст какого-либо оператора, в последнем используется отступ, как при введении абзаца, а не дополнительные ключевые слова.

Рассмотрим простейший пример умножения двух матриц A и B на языке OCCAM. Структура данных и матрица транспьютеров для данной задачи показаны на рис. 12.5. Процедура mult устанавливает три последовательных шага для каждого ПЭ матрицы. Сначала по входным каналам up и left вводятся данные в ПЭ и осуществляется вычисление компонента результирующей матрицы: $r:=r+(a*b)$. Наконец, входные данные передаются на выходные каналы, называемые down и right:

```
PROC mult(CHAN up,down,left,night)
 VAR r,a,b
 SEQ
   r:=0
   SEQ i FOR n
     SEQ
       PAR
```

```
    up ?a
    left ?b
    r:=r+(a*b)
  PAR
    down !a
    right !b
```

Данная программа описывает работу одного ПЭ. Работа же всей матрицы транспьютеров задается следующей программой:

```
CHAN vert [n*(n+1)];
CHAN horz [n*(n+1)];
PAR
  PAR i [0 FOR n]
    PAR j [0 FOR n]
  mult (vert [(n*i)+j]
       vert [(n*i)+j+1]
       horz [(n*i)+j]
       horz [(n*(i+1)+j]
```

Далее идут процессы ввода и вывода.

Транспьютер, обладающий достаточно большой вычислительной мощностью, используется для построения систем, содержащих всего один такой элемент. При этом можно применять стандартные алгоритмические языки высокого уровня. Однако в основном он применяется как ПЭ при построении мультитранспьютерных систем. При этом система, представляющая собой небольшой массив транспьютеров, может выступать как дополнительный блок персональной ЭВМ, повышающий ее производительность, в частности, за счет ускорения выполнения таких процедур, как компиляция, форматизация заданий, инженерное моделирование, цифровая обработка сигналов в реальном масштабе времени.

На рис. 12.6 показана система из четырех транспьютеров, используемая в качестве дополнительного блока персональной ЭВМ. Каждый транспьютер соединен с четырьмя ОЗУ с длиной слова 1 байт каждое. Кроме того, в систему входят тактовый генератор, канальный адаптер, адаптер периферийного интерфейса и интерфейсное устройство на основе программируемой логической матрицы (ПЛМ). Система не содержит никаких других устройств на основе ТТЛ-схем, а также ПЗУ. Первичная загрузка массива транспьютеров выполняется с помощью канального адаптера, работающего под управлением персональной ЭВМ. После загрузки программы этот адаптер используется для передачи данных между персональной ЭВМ и массивом. Интерфейсное устройство ПЛМ выполняет все логические операции, требуемые для сброса транспьютеров и связанные с анализом ошибочных ситуаций. Массив транспьютеров представляет собой конвейерную систему, в которой транспьютеры соединены друг с другом с помощью Inmos-каналов.

В настоящее время создаются мультипроцессорные системы, содержащие большое число транспьютеров. Так, фирма Inmos разработала типовые блоки, из которых могут строиться мультитранспьютерные системы различной конфигурации, обеспечивающие производительность от 160 млн до 2,5 млрд оп/с. Система любой из этих конфигураций размещается в шкафу размером 0,6×0,64×1 м.

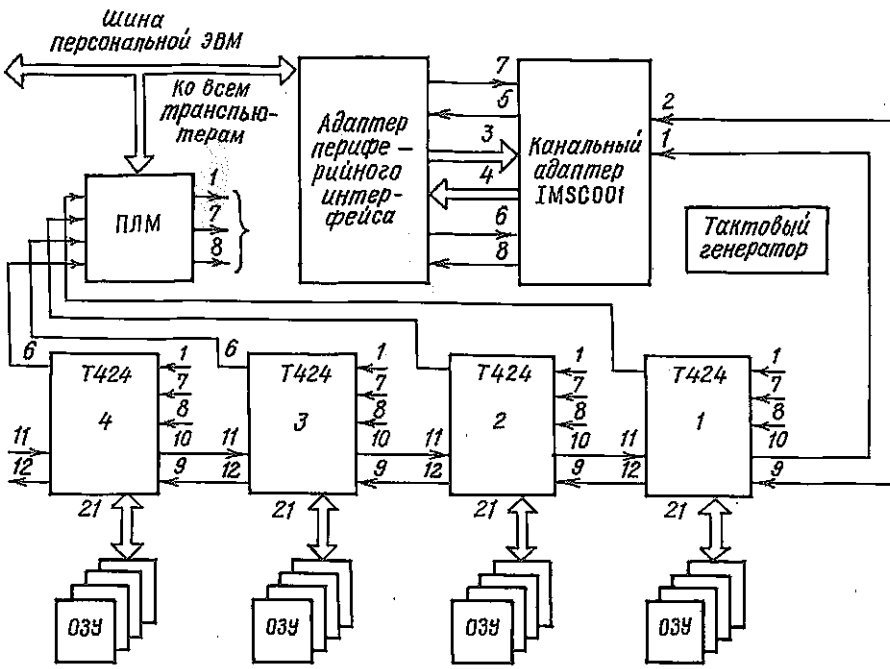


Рис. 12.6

За рубежом появились транспьютеры, содержащие умножитель чисел с ПЗ и специальный процессор для управления каналом передачи данных. Совершенствование транспьютера не влечет за собой необходимость изменения уже разработанных систем и их программного обеспечения. Это важное свойство связано с тем, что система команд транспьютера скрыта от пользователя и единственным средством общения пользователя с транспьютерной системой является язык высокого уровня – ОССАМ.

12.3. СИСТОЛИЧЕСКИЕ ПРОЦЕССОРЫ

Необходимость обработки больших массивов информации в реальном масштабе времени (управление сложными технологическими процессами, создание экспертных систем, решение задач математической физики, метеорологии, обработки сигналов и изображений) связана со значительным расширением существующих вычислительных возможностей как по объему обрабатываемой информации, так и по быстродействию вычислительных средств.

Параллельные ЭВМ в соответствии с их структурными особенностями можно разделить на три класса: векторные процессоры, многопроцессорные системы и матричные процессоры. К первым двум классам относятся универсальные ЭВМ, а к третьему классу – специализированные вычислительные машины, для создания которых необходимо знание взаимосвязей алгоритмов параллельных вычислений с оптимально реализующими их аппаратными и программными средствами.

Наиболее полно преимущества матричных однородных структур используются в современной технологии, когда вся матричная структура, содержащая сотни ПЭ, строится на одной большой базовой пластине кремния без разрезания ее на отдельные части. Такой подход позволяет избежать соединений между ПЭ вне СБИС и связанных с ними задержек в передаче сигналов.

В конце 70-х гг. С.Кунгом был предложен новый принцип обработки информации в однородных вычислительных системах, основанный на циркуляции потоков данных через вычислительные ячейки массива подобно циркуляции крови в системе кровообращения. Каждая ячейка, т.е. ПЭ, выполняет перед каждой передачей данных другой ячейки некоторые относительно простые и короткие вычисления. Эти вычисления могут быть либо одинаковыми для всех ПЭ, либо различными для некоторых из них.

Основные принципы систолического подхода заключаются в том, что обработка информации осуществляется в матрице, состоящей из множества ПЭ с простыми локальными и регулярными связями. Потоки данных и промежуточных результатов перемещаются по матрице ПЭ и взаимодействуют друг с другом в ПЭ в соответствии с алгоритмом решаемой задачи, а конечные результаты выводятся во внешнюю память в темпе вычислений. При этом каждая величина извлекается из внешней памяти один раз и используется в матрице ПЭ многократно (столько раз, сколько необходимо в соответствии с алгоритмом).

Алгоритмы, предназначенные для реализации в системе с такими свойствами, называются систолическими, а аппаратные средства для их реализации – систолическими процессорами.

Концепция систолической обработки данных содержит ряд направлений: 1) конструирование систолических алгоритмов, удовлетворяющих требованиям оптимальной параллельной обработки в матрице ПЭ; 2) разработка методологии отображения этих алгоритмов на систолическую структуру; 3) конструирование СБИС процессорных элементов для систолической структуры; 4) создание языковых и программных средств, обеспечивающих эффективное выполнение параллельных процессов в систолической структуре.

Анализ многообразия различных типов систолических структур и тенденций их развития позволяет классифицировать эти структуры по нескольким признакам. По степени гибкости систолические структуры делятся на специализированные, алгоритмически ориентированные и программируемые. По типу обрабатываемой информации в ПЭ различают одноразрядные и многоразрядные систолические структуры. По характеру локально-пространственных связей систолические структуры классифицируются на одно-, двух- и трехмерные в зависимости от вида обрабатываемой информации (вектора, матрицы или множества другого типа).

Алгоритмически ориентированные структуры обладают возможностью программирования конфигурации локально-пространственных связей между ПЭ или самих ПЭ. В программируемых систолических структурах существует возможность программирования как конфигурации межпроцессорных связей, так и самих ПЭ. При этом ПЭ обладают локальной памятью программ и, хотя все они имеют одну и ту же организацию, в один и тот же момент времени могут выполнять различные операции из некоторого набора.

Существуют два основных типа ПЭ (рис. 12.7): 1) ПЭ с топологией (см.

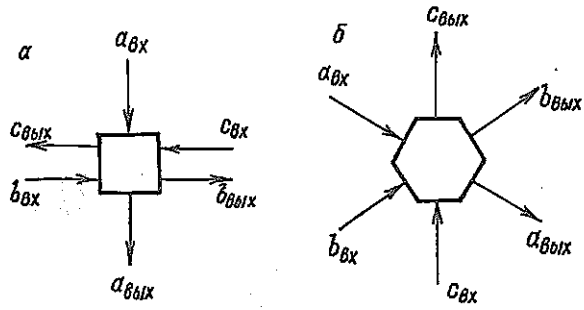


Рис. 12.7

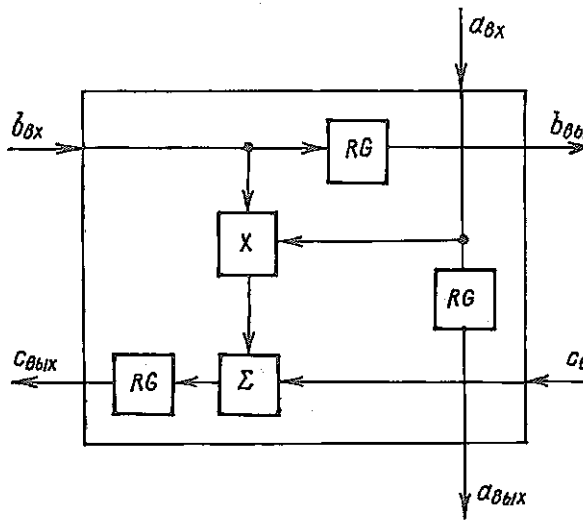


Рис. 12.8

рис. 12.7, *а*) служат основой для построения линейных и ортогональных систолических множеств, которые используются для выполнения таких алгоритмов, как дискретное преобразование Фурье, свертка, рекурсивная и нерекурсивная цифровая фильтрация, решение треугольных систем уравнений, умножение вектора на матрицу и др.; 2) ПЭ с топологией (см. рис. 12.7, *б*) применяется при построении гексагональных систолических процессоров для выполнения матричных операций (разложение, декомпозиция, обращения матриц, работа с матрицами специального вида (Теплица, Хенкеля)).

Так, для такого широкого класса задач, как цифровая обработка изображений и сигналов, базовой операцией является умножение с накоплением. При этом все ПЭ независимо от топологии выполняют следующие операции:

$$c_{вых}(n) = c_{вх}(n-1) + a_{вх}(n-1)b_{вх}(n-1);$$

$$a_{вых}(n) = a_{вх}(n-1);\ b_{вых}(n) = b_{вх}(n-1).$$

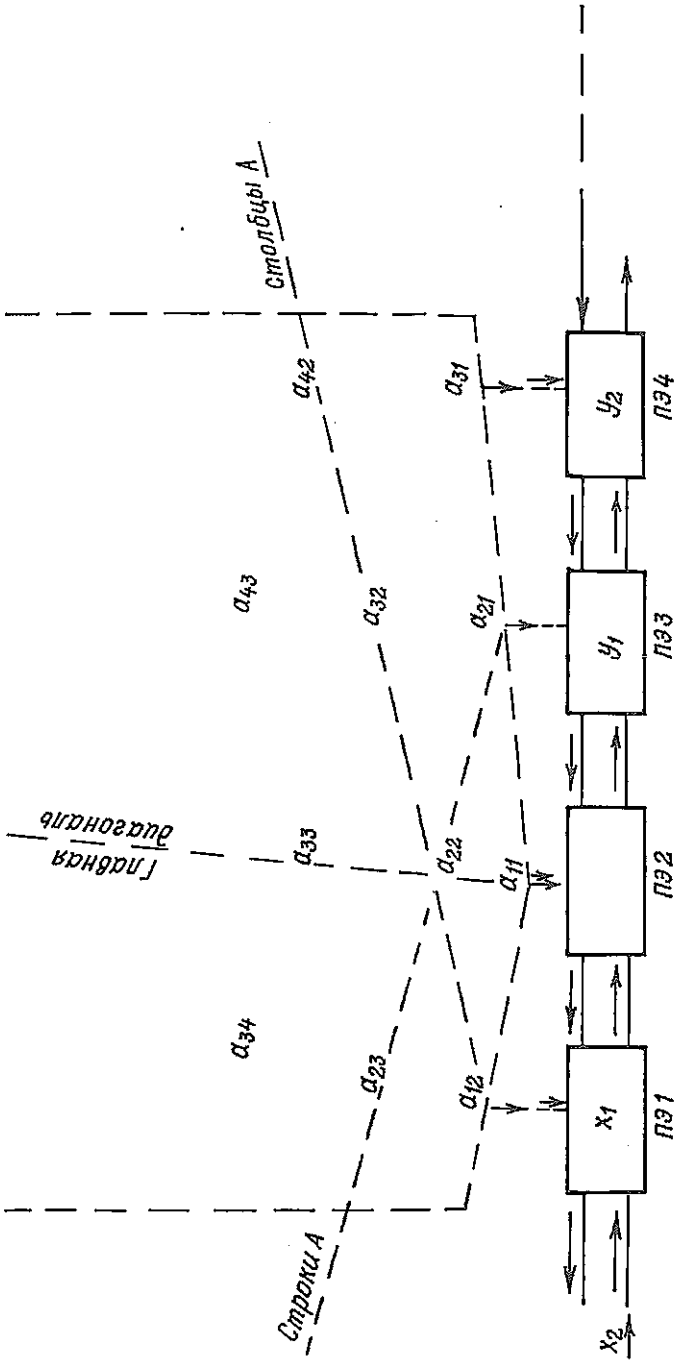

Столбцы A
Главная диагональ
Строки A
a_{42}
a_{31}
a_{43}
a_{32}
a_{21}
a_{33}
a_{22}
a_{11}
a_{34}
a_{23}
a_{12}
y_2
ПЭ4
y_1
ПЭ3
ПЭ2
x_1
ПЭ1
x_2

Рис. 12.9

В состав такого ПЭ (рис. 12.8) входят умножитель, сумматор и три регистра, два из которых отведены для хранения операндов, поступающих из соседних ПЭ. Третий регистр предназначен для хранения получаемого в ПЭ результата операции.

Организацию вычислительного процесса в систолических массивах с ПЭ (см. рис. 12.7, *а*) рассмотрим на примере умножения матрицы $A = \{a_{ij}\}$ на вектор $X = \{x_1, ..., x_n\}$. Элементы вектора произведения могут быть получены следующим образом:

$$y_i(1) = 0;$$
$$y_i(k+1) = y_i(k) + a_{ik}x_k;$$
$$y_i = y_i(n+1),$$

где k – номер шага вычислений.

Пусть A – матрица размером $n \times n$ с шириной полосы ненулевых элементов:

$$A = \begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ a_{31} & a_{32} & a_{33} & a_{34} & & \\ & a_{42} & a_{43} & a_{44} & a_{45} & \\ & & \ddots & \ddots & \ddots & \end{bmatrix}.$$

Тогда определенная последовательность операций для вычисления компонентов вектора Y может быть получена при конвейерной обработке x_i и y_i на $p + q - 1$ последовательно соединенных ПЭ (рис. 12.9). Для данной структуры первые семь шагов алгоритма умножения представлены в табл. 12.2.

Заметим, что при рассмотренной организации вычислительного процесса для каждого ПЭ такты выполнения операции чередуются с тактами простоя. Таким образом, в каждый момент времени активно работают только $(p + q - 1)/2$ ПЭ. Следовательно, каждый выходной результат формируется за два такта. Для вычисления всех n элементов выходного вектора Y необходимо $2n + p + q - 1$ временных тактов.

Аналогичным образом организуется процесс вычислений в гексагональной структуре. Так, для схемы умножения матриц с шириной полосы ненулевых элементов l_A и l_B

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & 0 \\ c_{21} & a_{22} & c_{23} & c_{24} & \\ c_{31} & c_{32} & c_{33} & c_{34} & \\ c_{41} & c_{42} & \ddots & & \\ 0 & & & & \end{bmatrix} = \begin{bmatrix} \overbrace{a_{11}\ a_{12}}^{l_A} & 0 & & \\ a_{21} & a_{22} & a_{23} & \\ a_{31} & a_{32} & a_{33} & a_{34} \\ & a_{42} & \ddots & \ddots \\ 0 & & & \end{bmatrix} \times \begin{bmatrix} \overbrace{b_{11}\ b_{12}\ b_{13}}^{l_B} & 0 & & \\ b_{21} & b_{22} & b_{23} & b_{24} \\ & b_{32} & b_{33} & b_{34} \\ & & b_{43} & \ddots \\ 0 & & & \end{bmatrix}$$

Организация вычислений в гексагональной систолической структуре показана на рис. 12.10.

Таблица 12.2

Шаг	Состояние				Комментарий
	ПЭ1	ПЭ2	ПЭ3	ПЭ4	
0					Элемент y_1 поступил в ПЭ4, в начале $y_1 = 0$
1	x_1		y_1		Элемент x_1 поступил в ПЭ1, элемент y_1 движется влево
2		y_1 a_{11} x_1		y_2	Элемент a_{11} поступил в ПЭ2, $y_1 = y_1 + a_{11}x_1$, т. е. $y_1 = a_{11}x_1$
3	y_1 a_{12} x_2,		y_2 a_{21} x_1		Элемент a_{12} поступил в ПЭ1, $a_{21} \rightarrow$ в ПЭ3, $y_1 = a_{11}x_1 + a_{12}x_2$, $y_2 = a_{21}x_1$
4		y_2 a_{22} x_2		y_3 a_{31} x_1	Элемент y_1 вышел из ПЭ1, $y_2 = a_{21}x_1 + a_{22}x_2$, $y_3 = a_{31}x_1$
5	y_2 a_{23} x_3		y_3 a_{32} x_2		$y_2 = a_{21}x_1 + a_{22}x_2 + a_{23}x_3$, $y_3 = a_{31}x_1 + a_{32}x_2$
6		y_3 a_{33} x_3		y_4 a_{42} x_2	Элемент y_2 вышел из ПЭ1, $y_4 = a_{42}x_2$, $y_3 = a_{31}x_1 + a_{32}x_2 + a_{33}x_3$

Элементы матриц A, B, C передвигаются через структуру синхронно в трех разных направлениях. Элементы c_{ij}, первоначально равные нулю, по мере прохождения ПЭ накапливают свои результаты в соответствии с соотношениями

$$c_{ij}^{(1)} = 0;$$

$$c_{ij}^{(k+1)} = c_{ij}^{(k)} + a_{ik}\, b_{kj}, \quad k = 1, 2, \ldots, n;$$

$$c_{ij} = c_{ij}^{(n+1)}$$

В каждый момент времени в любой строке и любом столбце гексагональной структуры операцию выполняет только один ПЭ.

Класс алгоритмов, который можно реализовать на систолических структурах, составляют алгоритмы, обладающие двумя особенностями. Первая особенность связана с соотношением между числом вычислительных операций, которые необходимо выполнить для реализации данного алгоритма, и числом операций, связанных с вводом-выводом данных. Если число вычислительных операций превышает число операций, связанных с вводом-выводом, то такие

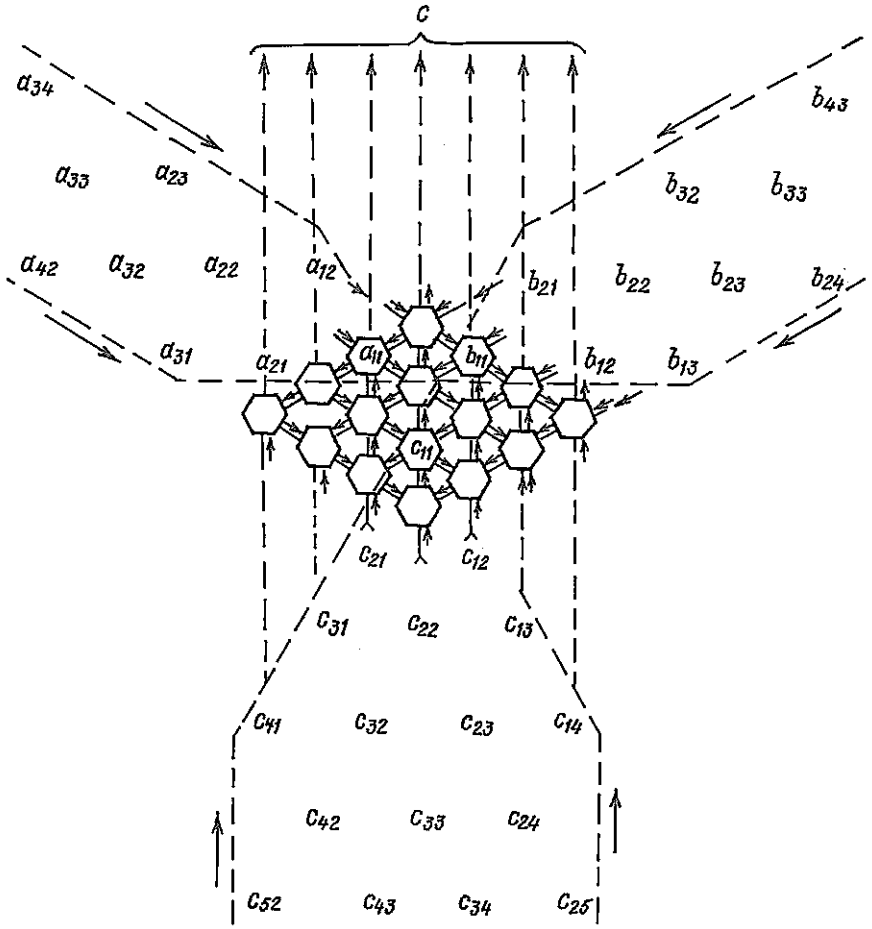


Рис. 12.10

алгоритмы называются вычислительно-ограниченными, в противном случае — ограниченными по вводу-выводу. Число операций по вводу-выводу связано, как видно из примеров, с числом исходных операндов и количеством получаемых скалярных результатов. Вторая важная особенность рассматриваемого класса алгоритмов заключается в их локальной рекурсивности. Свойство рекурсивности является следствием однородности вычислительной среды, а локальность определяется организацией связей между ПЭ в систолическом массиве.

В настоящее время систолические процессоры используются как сопроцессоры (периферийные процессоры) к универсальным ЭВМ. На рис. 12.11 представлено программное обеспечение такого процессора. В будущем они найдут применение как элементы вычислительных систем пятого поколения, потому что уже к концу 1991 г. ожидается производительность ЭВМ порядка 1000 млрд оп/с.

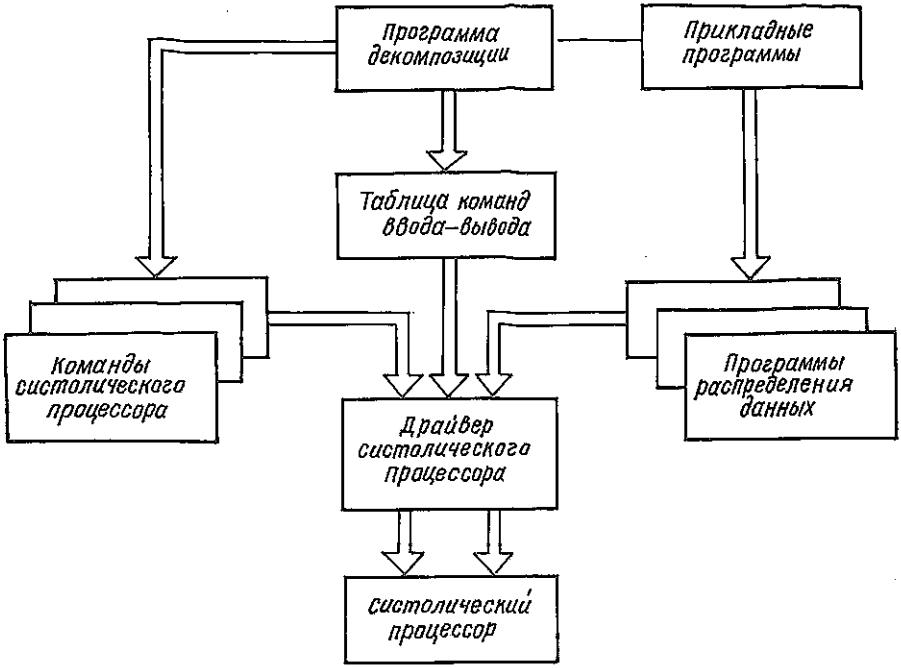


Рис. 12.11

Таким образом, систолическая обработка представляет собой определенный подход к созданию высокопроизводительных специализированных вычислителей. При этом требуется некоторое переосмысливание как самих алгоритмов обработки, так и способов программирования этих алгоритмов для каждой конкретной структуры, т.е. рациональное взаимное отображение алгоритма и структуры матрицы.

Рекомендуемая литература: 14.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Балашов Е.П., Григорьев В.Л., Петров А.Г. Мини- и микроЭВМ. – Л.: Энергоатомиздат, 1986. – 376 с.

2. Бряброн В.М. Программное обеспечение персональных ЭВМ. – М.: Наука, 1988. – 272 с.

3. Вигдорчик Г.В., Воробьев А.Ю., Праченко В.Д. Основы программирования на ассемблере для СМ ЭВМ. – М.: Финансы и статистика, 1987. – 240 с.

4. Григорьев В.Л. Программирование для микроЭВМ. – М.: Энергоатомзидат, 1983. – 283 с.

5. Григорьев В.Л. Программирование однокристальных микропроцессоров. – М.: Энергоатомиздат, 1987. – 287 с.

6. Дамке К. ОС микроЭВМ. – М.: Финансы и статистика, 1985. – 150 с.

7. Дейтел Г. Введение в операционные системы. – М.: Мир, 1987. – Т. 2. – 398 с.

8. Кейслер С. Проектирование ОС для малых ЭВМ. – М.: Мир, 1986. – 679 с.

9. Кокорин В.С., Попов А.А., Шишкевич А.А. МикроЭВМ. Персональные ЭВМ. – М.: Высш. шк., 1988. – Кн. 2. – 159 с.

10. Котов В.Е. Сети Петри. – М.: Наука, 1984. – 188 с.

11. МикроЭВМ. Персональные профессиональные ЭВМ / Г.П. Лопато, М.Е. Неменман, В.Я. Пыхтин, В.Н. Тикменов. – М.: Высш. шк., 1988. – Кн. 5. – 143 с.

12. Назаретов В.М., Ким Д.П. Техническая имитация интеллекта. – М.; Высш. шк., 1986. – 143 с.

13. Персональные компьютеры единой системы ЭВМ / А.П. Запольский, В.Я. Пыхтин, А.Н. Чистяков, В.Б. Шкляр. – М.: Финансы и статистика, 1988. – 143 с.

14. Прангишвили И.В. Микропроцессоры и локальные сети микроЭВМ в распределенных системах управления. – М.: Энергоатомиздат, 1984. – 272 с.

15. Пул Л. Работа на персональном компьютере. – М.: Мир, 1986. – 393 с.

16. Талов И.Л., Соловьев А.Н., Борисенков В.Д. МикроЭВМ. Семейство ЭВМ "Электроника 60". – М.: Высш. шк., 1988. – Кн. 1. – 172 с.

17. Уокерли Дж. Архитектура и программирование микроЭВМ. –М.: Мир, 1984. – Т. 1. – 496 с.; Т.2. – 359 с.

18. Френк Т.С. PDP-11: Архитектура и программирование. – М.: Радио и связь, 1986. – 372 с.

19. Шаньгин В.Ф., Костин А.Е. Организация вычислительных процессов на микроЭВМ. – М.: Высш. шк. – 1984. – 119 с.

20. Экхауз Р., Моррис Л. Мини-ЭВМ: Организация и программирование. – М.: Финансы и статистика, 1983. – 359 с.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Учебное издание

Вишняков Владимир Анатольевич
Петровский Александр Александрович

СИСТЕМНОЕ ОБЕСПЕЧЕНИЕ микроЭВМ

Заведующий редакцией *А.Ф.Зиновьев.* Редактор *Н.М.Латышева.* Младшие редакторы *А.М.Апель, А.П.Берлина, Т.И.Крючкова.* Художественный редактор *В.Н.Валентович.* Технический редактор *Л.И.Счисленок.* Корректоры *Н.В.Кудрейко, Л.А.Шлыкович, Т.М.Рутковская.* Оператор *А.И. Маль.*

ИБ № 3054

Подписано в печать с оригинала-макета 24.05.90 г. АТ 03638.Формат 60 х 90/16. Бумага офсетная. Гарнитура Пресс Роман. Печать офсетная. Усл.-печ. л. 19. Усл. кр.-отт. 19 . Уч.-изд. л. 21,82. Тираж 12200 экз. Заказ 5404. Цена 1 р. 20 к.

Издательство "Вышэйшая школа" Государственного комитета Белорусской ССР по печати. 220048. Минск, проспект Машерова, 11.
Типография "Победа". 222310, Молодечно, ул. Тавлая, 11.