

БИБЛИОТЕЧКА
ПРОГРАММИСТА

С.Н. БУШЕВ, М.С. БЕСФАМИЛЬНЫЙ

Программно- аппаратные методы управления данными



С. Н. БУШЕВ, М. С. БЕСФАМИЛЬНЫЙ

ПРОГРАММНО- АППАРАТНЫЕ МЕТОДЫ УПРАВЛЕНИЯ ДАННЫМИ

Под редакцией С. В. ЕМЕЛЬЯНОВА



МОСКВА «НАУКА»

ГЛАВНАЯ РЕДАКЦИЯ

ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ

1982

22.18

Б 90

УДК 519.6

Программно-аппаратные методы управления данными. С. Н. Бушев, М. С. Бесфамильный. — М.: Наука. Главная редакция физико-математической литературы, 1982. — 240 с.

Книга посвящена рассмотрению основных архитектурных решений моделей ЕС ЭВМ и их программного обеспечения. Наибольшее внимание уделено вопросам организации ввода-вывода в ЕС ЭВМ.

В книге рассмотрены, наряду с аппаратными, программные средства управления данными в ОС ЕС ЭВМ. Освещены вопросы организации систем управления данными, приведены приемы использования основных методов доступа при написании прикладных программ. Изложение систем управления данными не обременено излишними деталями. Вместе с тем в книге приведены минимально необходимые сведения, достаточные для программирования операций ввода-вывода в ЕС ЭВМ.

Книга может служить введением перед освоением эксплуатационной документации по ЕС ЭВМ.

Бушев Станислав Николаевич, Бесфамильный Михаил Серафимович

ПРОГРАММНО-АППАРАТНЫЕ МЕТОДЫ УПРАВЛЕНИЯ ДАННЫМИ

Редактор: *Н. Н. Васина*

Технический редактор: *С. Я. Шкляр*. Корректор: *О. А. Бутусова*

ИБ № 12002

Сдано в набор 12.10.81. Подписано к печати 16.06.82. Т-13505. Формат 84 × 108^{1/32}. Бумага тип. № 1. Гарнитура таймс. Высокая печать. Условн. печ. л. 12,6. Уч.-изд. л. 13,41. Тираж 25000 экз. Заказ № 139. Цена 85 коп.

Издательство «Наука»

Главная редакция физико-математической литературы
117071, Москва, В-71, Ленинский проспект, 15

Ордена Октябрьской Революции, ордена Трудового Красного Знамени Ленинградское производственно-техническое объединение «Печатный Двор» имени А. М. Горького Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли.
197136, Ленинград, П-136, Чкаловский пр., 15

© Издательство «Наука».
Главная редакция
физико-математической
литературы, 1982

Б 1502000000-084 50-82
053(02)-82

ОГЛАВЛЕНИЕ

Предисловие	5
Введение	7
Глава 1. Общесистемные вопросы построения ЕС ЭВМ	11
1.1. Общая структура ЕС ЭВМ	11
1.2. Структура технических средств ЕС ЭВМ	12
1.3. Структура системы программного обеспечения ЕС ЭВМ	22
Глава 2. Сведения по логической структуре моделей ЕС ЭВМ	33
2.1. Общая структура моделей	33
2.2. Память ЭВМ Единой Системы	34
2.3. Структура центрального процессора	37
2.4. Каналы и аппаратная система ввода-вывода	60
2.5. Программирование операций ввода-вывода без использования средств операционной системы	78
Глава 3. Управляющая программа — основа операционной системы	92
3.1. Мультипрограммирование и его реализация	92
3.2. Ввод-вывод программ в операционных системах	95
3.3. Специфика пропуска задач в операционных системах	98
3.4. Структура управляющей программы операционной системы	101
3.5. Выполнение операций ввода-вывода с помощью компонент управляющей программы	102
Глава 4. Принципы построения системы управления данными	105
4.1. Основные объекты системы управления данными	105
4.2. Уровни управления данными и средства их реализации	108
4.3. Физическая система управления данными	110
4.4. Базисно-логическая система управления данными	113
4.5. Общая схема организации выполнения запросов в системе управления данными	119
4.6. Особенности системы управления данными	121
Глава 5. Внешние запоминающие устройства и организация размещения данных	129
5.1. Особенности внешних запоминающих устройств ЕС ЭВМ	129
5.2. Записи наборов данных и их форматы	137
5.3. Организация наборов данных	139
5.4. Методы доступа к данным	146
5.5. Хранение и поиск файлов	149

Глава 6. Программирование операций ввода-вывода	164
6.1. Организация системы программирования операций ввода-вывода	164
6.2. Основные этапы программирования операций ввода-вывода	168
6.3. Связи макрокоманд ввода-вывода с программами системы управления данными	172
6.4. Описание и обеспечение доступа к наборам данных	178
Глава 7. Режимы управления данными	200
7.1. Управление последовательно-организованными файлами	200
7.2. Управление индексно-последовательными файлами	212
7.3. Управление библиотечными файлами	223
7.4. Управление файлами с прямой организацией	228
Литература	240

ПРЕДИСЛОВИЕ

В последнее время отечественная научно-техническая литература пополнилась значительным количеством изданий о современных ЭВМ. Особенно много публикаций появилось в связи с широким использованием наиболее мощной Единой Системы — ЕС ЭВМ. Вышло много книг о технических средствах, о системе программного обеспечения ЕС ЭВМ, появились сведения о прикладном программном обеспечении.

Однако следует отметить, что в публикуемых изданиях мало внимания уделено вопросам интегрирования программных и аппаратных средств, а также нет достаточных сведений об основных принципах устройства современных операционных систем, особенно касающихся системы управления данными.

В предлагаемой книге предпринята попытка восполнить указанные пробелы и осветить современные вычислительные системы одновременно с позиций аппаратных и программных средств. За основу такого рассмотрения принята интегральная система управления данными, берущая свое начало в архитектуре ЭВМ и постепенно переходящая в программную систему, являющуюся стержневой компонентой операционной системы ЭВМ. В совокупности представленный материал посвящен аппаратно-программной системе управления данными в ЕС ЭВМ. Материал книги распределен следующим образом.

Первые две главы книги посвящены общим системным решениям, принятым в ЕС ЭВМ. Наибольшее внимание в этой части уделено принципиальным вопросам логической структуры моделей ЕС ЭВМ, общим для всех машин, а также определяющим наиболее важные свойства системы программного обеспечения.

В главах 3—7 книги рассматриваются основные программные средства управления данными в операционной

системе ЕС ЭВМ. Анализируются вопросы внутреннего устройства систем управления данными, а также вопросы применения основных методов доступа при написании прикладных программ.

Кроме того, авторы ставили еще одну задачу при написании книги, суть которой состоит в том, чтобы вооружить специалистов знаниями основных принципов современных средств управления данными, не обременяя излишними деталями практической реализации. Успешное решение этой задачи позволит специалистам в процессе самостоятельной работы более успешно овладеть материалом, излагаемым в эксплуатационной документации по операционным системам.

Книга рассчитана на широкие круги научных и инженерных работников, специализирующихся в области применения средств ЕС ЭВМ в различных отраслях науки и техники. Представленный материал может быть использован аспирантами и студентами вузов соответствующих специальностей.

Авторы выражают свою благодарность профессору А. Г. Дьячко, канд. техн. наук Ю. Ю. Прокопчуку и А. И. Широкову, сделавшим ряд замечаний при подготовке текста книги.

Авторы признательны чл.-корр. АН УССР А. А. Стогнию и канд. техн. наук И. А. Базилевичу, взявшим на себя труд по рецензированию текста книги и сделавшим ряд полезных замечаний, способствовавших ее улучшению.

ВВЕДЕНИЕ

Вычислительные машины третьего поколения находят все более широкое применение, завоевывая новые сферы в отраслях народного хозяйства. Важнейшим качественным отличием ЭВМ третьего поколения является создание систем ЭВМ, являющихся единым универсальным инструментом решения чрезвычайно широкого круга задач обработки информации. Такая система ЭВМ включает в себя ряд моделей различной производительности и, соответственно, различной стоимости, оснащенных единой системой программного обеспечения.

Одной из важнейших характеристик вычислительных машин третьего поколения является наличие мощной операционной системы. Создание операционных систем, которые могут рассматриваться в качестве составной части машин, преследует цели:

- 1) повышения производительности машин;
- 2) повышения производительности труда программистов;
- 3) упрощения управлением ЭВМ;
- 4) обеспечения универсальных средств для построения более сложных программных систем.

Повышение производительности ЭВМ достигается путем автоматизации большинства процессов обработки, включая постановку задачи на машину, процессы обмена информацией между периферийными и центральными устройствами, а также путем использования принципов мультипрограммирования.

В процессе программирования задач можно использовать символические языки программирования различных уровней. Производительность труда программистов растет благодаря широкому применению языков высокого уровня. На уровне машинно-ориентированного языка рост производительности обеспечивается благодаря использованию большого количества стандартных макрокоманд.

Управление работой ЭВМ осуществляется с помощью специального языка, удобного для использования и состоящего из весьма ограниченного количества директивных

операторов. Связь оператора с ЭВМ осуществляется в разговорном режиме по принципу «запрос — ответ». А программист для общения с ЭВМ имеет в своем распоряжении специальный язык описания процесса решения и системные сообщения о ходе выполнения его задачи.

Использование возможностей, предоставляемых пользователю техническими средствами ЕС ЭВМ и системой программного обеспечения, требует широкой и разносторонней подготовки специалистов как по аппаратным, так и по программным средствам. Следует заметить, что программист, составляющий программы на языках высокого уровня, вполне может обойтись без глубокого знания характеристик аппаратуры. Для тех же пользователей, которые конструируют программы на машинно-ориентированных языках и стремятся к достижению высокой эффективности, знание технических средств и их возможностей является необходимым условием.

Применение современных ЭВМ немыслимо без развитых систем управления данными, оснащенных мощными средствами программирования операций ввода-вывода. Система управления данными представляет собой совокупность программных и аппаратных средств, предназначенных для программирования и выполнения операций обмена информацией между оперативной памятью и периферийными устройствами машины. Доля программных средств управления данными по отношению к аппаратным в машинах третьего поколения стала доминирующей.

Аппаратура центральной части ЭВМ в процессе операций обмена реализует только минимально необходимые действия. Из системы команд центральных процессоров исчезли инструкции, предназначенные для программирования операций ввода-вывода. Разработаны специализированные процессоры (каналы), предназначенные для ввода и вывода информации. Каналы осуществляют обмен данными так же, как центральный процессор — обработку, т.е. с помощью средств программного управления. Роль центрального процессора при выполнении ввода или вывода информации ограничивается функциями управления каналами при запуске, окончании или необычном протекании операций обмена.

Основная цель книги состоит в том, чтобы дать представление об аппаратно-программных методах управления данными и принципах реализации методов, дать минимально необходимые сведения о средствах программирования операций ввода-вывода в современных ЭВМ.

За основу приняты средства управления данными в наиболее развитой операционной системе — ОС ЕС ЭВМ. Вызвано это, во-первых, тем, что ЕС ЭВМ — наиболее распространенная у нас в стране система машин третьего поколения, и, во-вторых, — ОС ЕС включает в себя большинство практических достижений программирования в области управления данными.

При чтении книги следует иметь в виду, что система управления данными ОС ЕС рассматривается в сокращенном варианте. В ней освещены программные средства управления данными только для устройств ввода-вывода и внешних запоминающих устройств. Не затронуты средства управления данными такими внешними устройствами, как дисплеи и абонентские пункты, входящие в состав системы телеобработки данных.

При анализе средств управления данными затрагиваются только главные элементы, которые образуют «скелет» системы. К примеру, использование методов доступа демонстрируется путем рассмотрения основных режимов с помощью простейших программ. Аппарат меток, используемый при обработке данных, размещаемых на внешних запоминающих устройствах, приведен только для основного режима — режима применения стандартных меток.

Система управления данными ОС ЕС освещается с позиции прикладного программиста — разработчика программ.

Методика выбора программных средств управления данными, возникающая перед разработчиками программ, не затрагивается. Однако для того, чтобы пользователь мог правильно ориентироваться при решении этой задачи, в книге анализируются основные принципы программной реализации системы управления данными. Эти принципы, иллюстрирующие основной механизм выполнения операций ввода-вывода, могут помочь определить подходы к оценке эффективности составленных программ обработки данных.

Главная особенность средств программирования операций ввода-вывода в современных операционных системах ЕС ЭВМ состоит в том, что все они реализованы программным путем. Это в какой-то мере объясняет сложность изучения и использования систем управления данными, поскольку их средства доступны программистам только на уровне машинно-ориентированного языка ассемблера. А в некоторых случаях знание средств ввода-вывода ассемблера требуется также при использовании языков высокого уровня, что вызвано специфическими особенностями программной реализации системы управления данными.

Для успешного изучения системы управления данными ОС ЕС требуется знание принципов работы, архитектуры моделей ЕС ЭВМ, а также основ базового языка ассемблера и его макросредств. Некоторые сведения по управляющей программе операционной системы приводятся по мере изложения материала книги.

Настоящая книга состоит из семи глав. Первая содержит общие сведения о ЕС ЭВМ, в ней изложено описание структуры Единой Системы в целом, состава и архитектуры технических средств, основных принципов системы программного обеспечения.

Вторая глава содержит материал по логической структуре моделей ЕС ЭВМ, включающий общее описание ЭВМ, описание архитектурных особенностей машины и принципов функционирования наиболее важных устройств. Материал излагается под углом зрения программиста и содержит минимальный набор сведений, необходимых для понимания логики функционирования моделей. Особое внимание уделено системе ввода-вывода ЕС ЭВМ на аппаратном уровне.

В третьей главе приведены упрощенные алгоритмы реализации главных частей управляющей программы операционной системы, помогающие представить «физический смысл» их функционирования.

Четвертая глава содержит предварительные сведения о различных элементах (накопители, буферы и т. п.), принимающих участие в выполнении операций обмена и являющихся составными частями этого процесса.

Пятая глава посвящена рассмотрению особенностей внешних запоминающих устройств, вопросов организации размещения наборов данных на этих носителях, методов доступа, используемых при поиске наборов данных.

Шестая глава посвящена вопросам общей комплексной организации системы программирования, разработанной для использования системой управления данными.

В последней, седьмой, главе рассмотрены основные режимы управления данными, которые подкреплены практическими примерами.

Все практические примеры, приведенные в седьмой главе, написаны так, что их можно переносить на перфокарты и вводить в ЭВМ. Использование сравнительно большого числа примеров в законченном виде вызвано тем, что ни в документации, ни в литературе подобных примеров с использованием средств ввода-вывода системы управления данными ОС ЕС, к сожалению, не приведено.

ОБЩЕСИСТЕМНЫЕ ВОПРОСЫ ПОСТРОЕНИЯ ЕС ЭВМ

1.1. Общая структура ЕС ЭВМ

Единая Система электронных вычислительных машин является примером комплексного, системного подхода к проблеме создания и внедрения ЭВМ.

Система состоит из двух компонент: технические средства (ТС) и системы программного обеспечения (СПО).

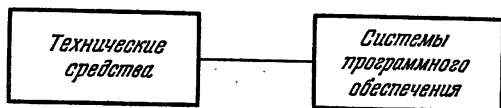


Рис. 1.1.

Общая укрупненная схема ЕС ЭВМ представлена на рис. 1.1.

Технические средства ЕС ЭВМ составляют центральные устройства (оперативная память, процессоры, каналы) и совокупность внешних устройств.

Системы программного обеспечения представляют собой комплекс программ, обеспечивающих эффективное использование технических средств ЕС ЭВМ. Эта компонента имеет в своем составе: набор управляющих программ, систем программирования и прикладных программ.

На рис. 1.2, кроме перечисленных выше элементов ТС и СПО, в структурной схеме ЕС ЭВМ определены еще две компоненты: модели и операционные системы. На рисунке показано также, что модели ЕС ЭВМ комплектуются из множества устройств, входящих в состав технических средств, а операционные системы — из множества программ системы программного обеспечения.

Модель Единой Системы формируется на базе процессора и других центральных устройств вместе с набором внешних устройств, но одновременно с этим допускает подключение внешних устройств в самых разнообразных вариантах.

Конкретная операционная система предназначена для обеспечения использования модели определенной конфигурации в некоторой области применения. Состав и возможности операционной системы определяются типом управляющей программы, конфигурацией модели, используемыми

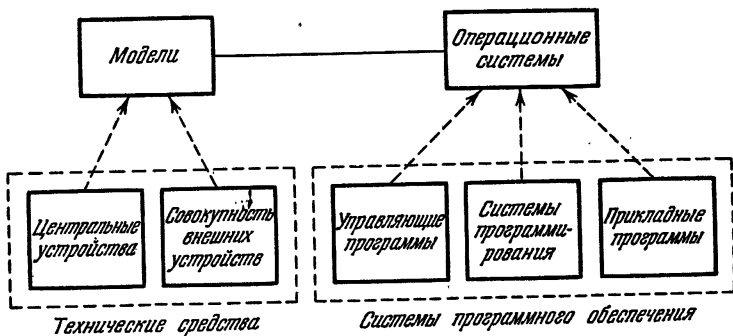


Рис. 1.2.

системами программирования и применяемым набором прикладных программ.

Перейдем к рассмотрению отдельных компонент Единой Системы ЭВМ.

1.2. Структура технических средств ЕС ЭВМ

Главный, центральный элемент в общей структуре средств ЕС ЭВМ (рис. 1.3) — это семейство моделей ЭВМ. Номенклатура моделей этого семейства не является неизменной. Единая Система представляет собой развивающуюся систему, в которой непрерывно улучшаются характеристики отдельных ее составных частей, совершенствуется номенклатура центральных и внешних устройств, растут технико-экономические показатели.

На рисунке приведено семейство моделей ЕС ЭВМ I и II очередей. Семейство моделей I очереди (Ряд 1) состоит из 10 ЭВМ: ЕС-1010, ЕС-1012, ЕС-1020, ЕС-1021, ЕС-1022, ЕС-1030, ЕС-1032, ЕС-1033, ЕС-1040, ЕС-1050. Семейство моделей II очереди (Ряд 2) состоит из 7 ЭВМ: ЕС-1015, ЕС-1025, ЕС-1035, ЕС-1045, ЕС-1055, ЕС-1060, ЕС-1065. Приведенное семейство ЭВМ отражает текущее состояние системы.

Из схемы видно, что модели ЕС ЭВМ составляются из устройств, принадлежащих множеству центральных и семейству внешних устройств.

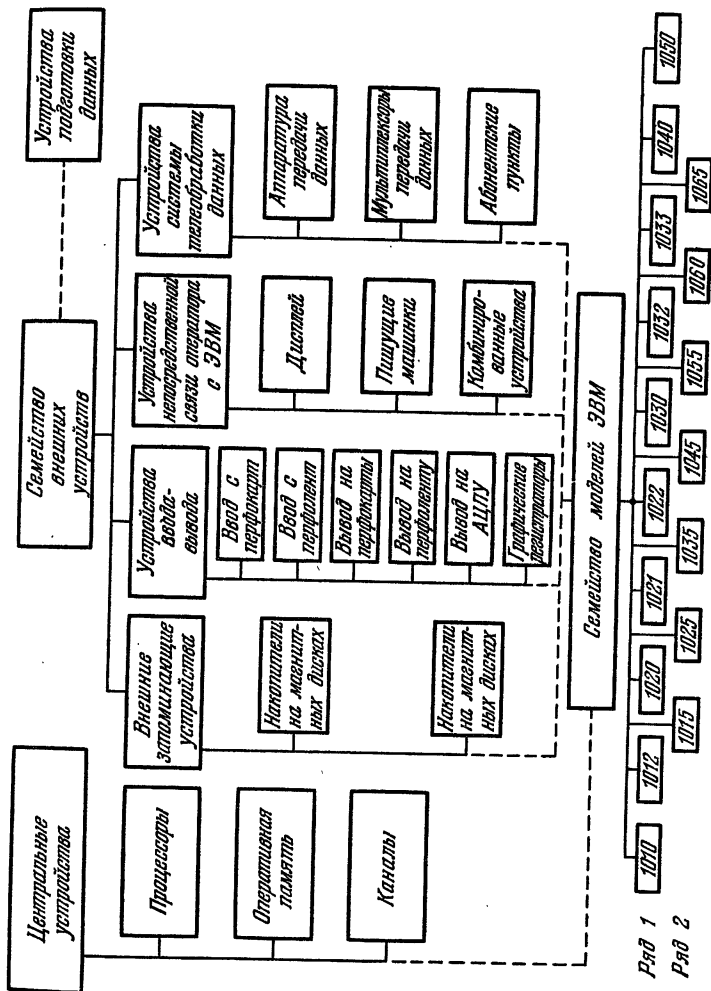


Рис. 1.3.

Состав любой модели не является строго определенным и неизменяемым. Например, понятие модели ЕС-1060 определяет конкретную ЭВМ, составленную из центрального процессора ЕС-2060 и укомплектованную определенным набором внешних устройств. Процесс составления и комплектации любой модели начинается с некоторого типового (минимального) состава, включающего в себя, как правило, один процессор, несколько каналов, минимальный объем оперативной памяти и конкретный набор внешних устройств.

Поскольку в минимальном составе возможности модели минимальны, то «наращивать» возможности конкретной модели можно путём увеличения состава как центральных, так и внешних устройств. Естественно, «наращивание» технических средств модели может происходить в определенных пределах, в соответствии с аппаратными возможностями. Некоторые предельные характеристики отечественных моделей ЕС ЭВМ I очереди приведены в табл. 1.1. Из нее видно, что наращивание возможностей центральных устройств, в основном, может осуществляться путем увеличения количества каналов и объема оперативной памяти.

Таблица 1.1

Некоторые предельные характеристики моделей ЕС ЭВМ

Характеристики	Модели				
	ЕС-1020	ЕС-1022	ЕС-1030	ЕС-1033	ЕС-1050
Количество процессоров	1	1	1	1	1
Количество каналов:					
мультиплексных	1	1	1	1	1
селекторных:					
минимум	2	2	2	2	2
максимум	2	2	3	3	6
Объем оперативной памяти (в килобайтах):					
минимум	64	128	128	256	512
максимум	256	512	512	1024	1024
Максимальное количество внешних устройств, подключаемых к мультиплексному каналу	128	128	128	256	256
Максимальное количество устройств управления, подключаемых к каналам:					
мультиплексному	8	16	8	32	40
селекторному	8	10	8	10	10

Количество и номенклатура внешних устройств моделей определяются параметрами каналов.

Рассмотрим подробнее центральные устройства. В семейство центральных устройств входят устройства, образующие собственно вычислительные машины без периферийного оборудования. Рассматриваемое семейство образуют три группы устройств, довольно тесно связанных между собой:

- 1) центральные процессоры;
- 2) оперативная память;
- 3) каналы.

Центральный процессор выполняет команды и обеспечивает выполнение программы в целом. Оперативная память, в ЕС ЭВМ ее называют основной, служит для хранения данных и программ. Каналы представляют собой специализированные процессоры, работающие с оперативной памятью и выполняющие только операции ввода-вывода по заданной программе обмена.

Структура центрального устройства модели ЭВМ Единой Системы представлена на рис. 1.4, где дан фрагмент

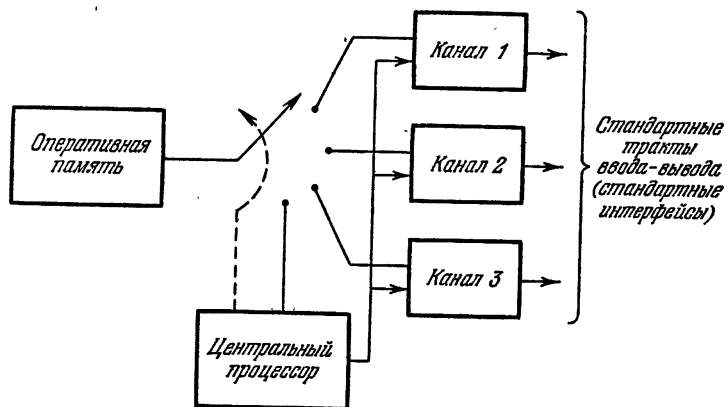


Рис. 1.4.

ЭВМ, имеющий в своем составе три канала для выполнения операций ввода-вывода. В составе ЭВМ имеется оперативная память с одним входом. С этой памятью взаимодействуют четыре процессора: центральный и три специализированных (каналы). Канал не может самостоятельно начать функционировать, а начинает свою работу только по инициативе центрального процессора.

Эта схема в какой-то мере объясняет, почему часто ЭВМ третьего поколения называют системой. Так как в состав современной ЭВМ входит несколько процессоров и каждый из них может автономно работать по своей собственной программе, то такую ЭВМ естественно называть системой. В системе можно выделить два типа процессоров. Один процессор выполняет большое количество операций, в том числе операции по управлению другими процессорами. Последние в свою очередь могут выполнять только операции ввода-вывода, т.е. передачи информации из оперативной памяти на внешние устройства или с внешних устройств в оперативную память через стандартное сопряжение, называемое стандартным интерфейсом ввода-вывода.

Рис. 1.4 поясняет также принцип взаимодействия процессоров ЭВМ с оперативной памятью. Применение последовательного способа обслуживания процессора и каналов связано с тем, что оперативная память обычно имеет один вход, а в процессе работы ЭВМ, как правило, имеется достаточный резерв времени для ее взаимодействия с несколькими процессорами.

Вернемся к вопросу о взаимосвязи семейства центральных устройств. Каждая ЭВМ Единой Системы разрабатывается как единая машина, и все ее центральные устройства обычно проектируются и изготавливаются применительно к этой конкретной ЭВМ. Поэтому устройства, входящие в группу оперативной памяти и каналов, как правило, однозначно привязаны к конкретным центральным процессорам, которые, в свою очередь, являются основой для комплектации модели.

Возможность построения с помощью центральных устройств целого семейства, а не единственной машины, объясняется необходимостью иметь ЭВМ различной производительности как с точки зрения выполняемых на ЭВМ объемов работ, так и с экономических позиций.

Основной характеристикой центральных устройств является их производительность. В группе процессоров для оценки производительности может быть использовано время выполнения отдельных операций, в группе оперативной памяти — время обращения, отнесенное к одному байту, в группе каналов — пропускная способность. ЭВМ Единой Системы перекрывают диапазон производительности от 3 тыс. до 2 млн. операций в секунду. Характеристики центральных устройств отечественных ЕС ЭВМ I очереди даны в табл. 1.2.

Характеристики центральных устройств ЕС ЭВМ

Характеристики	Модели				
	ЕС-1020	ЕС-1022	ЕС-1030	ЕС-1033	ЕС-1050
Производительность процессора (тыс. опер./с)	10—20	80—90	60	200	500
Пропускная способность каналов (кбайт/с)					
мультиплексных	16	80	40	70	110
селекторных	300	500	800	800	1300
Оперативная память время обращения (мкс)	2	2	1,15	1,2	1
Число одновременно выбираемых байтов	2	4	4	4	8

Рассмотрим подробнее состав семейства внешних устройств ЕС ЭВМ. Из этого семейства в настоящее время выделено несколько самостоятельных групп. Как и вся система ЕС ЭВМ, каждая группа устройств со временем будет изменяться и пополняться, вероятно, появятся и новые группы внешних устройств, которые органически будут вписываться в общую структуру.

В составе внешних устройств можно выделить четыре основные группы:

- 1) устройства ввода-вывода;
- 2) внешние запоминающие устройства;
- 3) устройства непосредственной связи оператора с ЭВМ;
- 4) устройства систем телеобработки данных.

Пятая группа является вспомогательной. Она представляет собой группу устройств подготовки данных.

Рассмотрим каждую из основных групп более подробно.

Устройства ввода-вывода. Группа устройств ввода-вывода (см. рис. 1.3) включает в себя все традиционные устройства ввода-вывода, применяемые в вычислительной технике:

- 1) устройства ввода с перфокарт;
- 2) устройства ввода с перфолент;
- 3) устройства вывода на перфокарты;
- 4) устройства вывода на перфоленту;
- 5) устройства вывода на АЦПУ;
- 6) графические регистрирующие устройства.

Рассматриваемая группа устройств ввода-вывода в основном скомпонована по признаку носителя информации, применяемого в них. Во всех устройствах этой группы обычно

используется бумажный носитель (перфолента, перфокарта, бумажная лента).

Назначение каждой функциональной подгруппы ясно из названия. Общая схема подключения внешних устройств к некоторой гипотетической ЭВМ приведена на рис. 1.5.

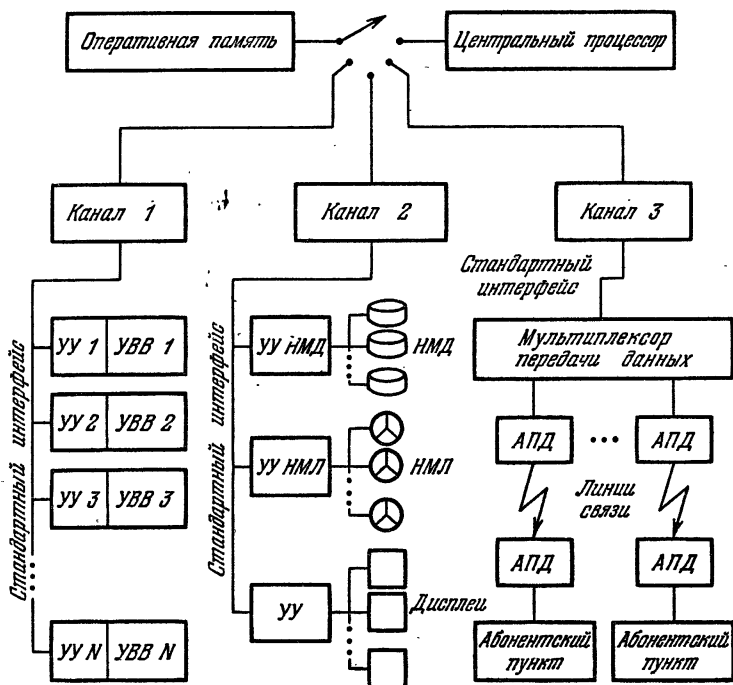


Рис. 1.5.

Все внешние устройства подключаются к каналам через стандартный интерфейс, оформленный в виде унифицированного кабеля. Здесь же показана группа устройств ввода-вывода, присоединенных к каналу 1. Отличительной особенностью этих устройств является наличие отдельного устройства управления для каждого устройства ввода-вывода. Многие устройства группы ввода-вывода конструктивно выполнены в виде специального стола, в который встроено устройство управления и собственно устройство ввода-вывода.

Отметим также, что все устройства ввода-вывода (УВВ) относятся к классу среднескоростных, значения скоростей которых лежат в пределах от 15 до 2000 байт/с. Поэтому такие типы устройств подключаются, как правило, к

мультиплексным каналам, обладающим невысокой пропускной способностью.

Внешние запоминающие устройства. Рассмотрим группу внешних запоминающих устройств. Из них (см. рис. 1.3) выделены две функциональные подгруппы:

- 1) накопители на магнитной ленте (НМЛ);
- 2) накопители на магнитном диске (НМД).

Особенностью большинства внешних запоминающих устройств ЕС ЭВМ является то, что они подключаются к каналу через групповое устройство управления (см. на рис. 1.5 УУ НМД и УУ НМЛ).

Несколько накопителей одного типа соединены и работают под управлением общего устройства управления (УУ), причем соединение осуществляется с помощью так называемого «малого» интерфейса. Конструктивно и УУ, и собственно внешние запоминающие устройства оформлены в виде отдельных стоек. Внешние запоминающие устройства принадлежат к классу высокоскоростных и обладают пропускной способностью от 64 000 до 156 000 байт/с и выше, поэтому они подключаются только к каналам с высокой пропускной способностью.

С точки зрения применений наиболее интересным устройством является накопитель на магнитном диске. Современные накопители на магнитных дисках по емкости не уступают магнитным лентам, а по скорости доступа к данным значительно их превосходят. Большинство накопителей являются устройствами со сменными пакетами дисков, что способствует увеличению общей емкости внешней памяти ЭВМ. Организация хранения данных на магнитном диске в ЕС ЭВМ обеспечивает малое время поиска данных и облегчает их поиск. Накопитель на магнитном диске является запоминающим устройством ассоциативного типа, т.е. устройством, где хранение, поиск и запись данных осуществляются по специальным служебным признакам, причем в качестве таких признаков могут использоваться некоторые составные части данных, такие как порядковые номера, шифры, наименования, ключи и т.д.

Устройства непосредственной связи оператора с ЭВМ. Эта группа устройств отражает новые тенденции, появившиеся в вычислительной технике третьего поколения. Как известно, на ЭВМ первого поколения человек, использующий машину (программист), работал непосредственно за пультом управления и сам следил за ходом вычислений. Учитывая, что такой режим не выгоден с точки зрения использования оборудования, был осуществлен переход к работе на

ЭВМ в операторном режиме, т.е. пользователь не сам работал с ЭВМ, а через посредника — оператора, который управлял процессами на машине с помощью некоторой программной системы. Такой режим работы обладает рядом преимуществ: эффективнее используется дорогостоящее оборудование и увеличивается пропускная способность ЭВМ, программисту нет необходимости обучаться общению с машиной, он избавляется от необходимости выполнять несвойственные ему функции (работа за пультом, подготовка носителей и устройств и т. п.).

Недостаток такого режима — невозможность оперативного вмешательства в процесс выполнения программы со стороны ее составителя.

Поэтому появились пульта, позволяющие оперативно взаимодействовать с ЭВМ не только операторам, но и пользователям. Эта тенденция в вычислительной технике привела к тому, что в составе ЕС ЭВМ появились три типа устройств, позволяющие человеку непосредственно взаимодействовать с ЭВМ:

- 1) дисплеи;
- 2) пишущие машинки;
- 3) комбинированные устройства.

Дисплей представляет собой внешнее устройство, соединенное с ЭВМ и позволяющее вводить и выводить информацию на экран электронно-лучевой трубки. Существуют два типа дисплеев:

- 1) алфавитно-цифровые;
- 2) графические.

Алфавитно-цифровые дисплеи позволяют вводить в ЭВМ и выводить на экран только текстовую информацию, а графические — текстовую информацию и графическое изображение.

Схема подключения алфавитно-цифровых дисплеев к ЭВМ приведена на рис. 1.5. Устройство управления оформлено в виде специальной стойки, а дисплей, внешне напоминающий обычный телевизор, через кабель длиной до 0,5 километра соединяется со стойкой управления. Графический дисплей представляет собой, по существу, малую вычислительную машину и оформляется в виде отдельного устройства с одним переносным пультом.

Дисплеи являются наиболее перспективными внешними устройствами современных ЭВМ, максимально приближающими человека к ЭВМ. Пишущие же машинки — традиционные устройства, и рассматривать здесь их функции нет необходимости.

Комбинированные устройства представляют собой так называемую «станцию», оснащенную одновременно несколькими устройствами, которые в совокупности позволяют осуществлять ввод информации в ЭВМ, вывод и управление работой самой станции. В ЕС ЭВМ к таким устройствам относится, например, перфоленточная станция (ЕС-7902). Устройства этой группы обычно присоединяются к медленным каналам. Однако, поскольку некоторые устройства (дисплеи), благодаря наличию буферной памяти, позволяют вести обмен с ЭВМ с высокими скоростями, то возможно их подключение и к быстрым каналам.

Рассмотренная группа устройств позволяет человеку работать с ЭВМ только на небольшом расстоянии от вычислительного центра (ВЦ). Проблему работы с ЭВМ на больших расстояниях в ЕС ЭВМ решает еще одна группа внешних устройств.

Устройства систем телеобработки данных. Основная роль в системах коллективного пользования принадлежит средствам телеобработки. Понятие «телеобработка» включает в себя такой способ обработки данных, при котором пользователь получает доступ к ЭВМ, находясь на значительном расстоянии от нее и используя при этом оборудование средств связи. Использование телеобработки в рамках ЕС ЭВМ является стандартным и одним из наиболее перспективных направлений применения средств Единой Системы. Технические средства телеобработки делятся на три основные группы:

- 1) аппаратура передачи данных (АПД);
- 2) мультиплексоры передачи данных (МПД);
- 3) абонентские пункты (АП).

Аппаратура передачи данных позволяет использовать стандартным образом различные типы каналов связи: телеграфные, телефонные, узкополосные и широкополосные физические линии. К аппаратуре передачи данных относятся модемы, устройства преобразования сигналов, автономные устройства защиты от ошибок. Модемы служат для преобразования дискретных сигналов ЭВМ или абонентских пунктов в такой вид сигнала, который используется оборудованием связи в узкополосных, телефонных, широкополосных каналах связи, и для обратного преобразования. Устройства преобразования сигналов предназначены для работы с телеграфными каналами. Устройства защиты от ошибок используются для увеличения надежности передачи данных по линиям связи путем применения помехоустойчивых кодов.

Мультиплексоры передачи данных обеспечивают сопряжение линий связи со стандартным каналом ЭВМ (по стандартному интерфейсу). В составе ЕС ЭВМ потребитель может найти несколько типов мультиплексоров данных, обслуживающих разное количество линий связи с разными скоростными характеристиками.

Абонентские пункты представляют собой периферийное оборудование, предназначенное для сбора и передачи данных, ввода и вывода сообщений.

В номенклатуру ЕС ЭВМ входит несколько типов абонентских пунктов, которые оснащены таким оборудованием, как телетайп, пишущие машинки, устройства печати, устройства ввода/вывода на перфоленду и перфокарты и, наконец, дисплей.

Схема подключения средств телеобработки к ЭВМ приведена на рис. 1.5, где к каналу 3 через мультиплексор передачи данных, аппаратуру передачи данных и линии связи подключены два абонентских пункта (терминала). Расстояние, на котором установлены терминалы, определяется, по существу, только типом линии связи.

1.3. Структура системы программного обеспечения ЕС ЭВМ

Общая структура. В предыдущем разделе кратко была рассмотрена структура и состав технических средств ЕС ЭВМ. Как показывает опыт применения ЭВМ в ряде зарубежных стран и в нашей стране, главным и определяющим фактором широкого использования средств вычислительной техники в различных сферах применения являются не только технические средства, но и программное обеспечение, к рассмотрению которого и перейдем.

Система программного обеспечения ЕС ЭВМ состоит из совокупности пяти операционных систем:

- 1) операционная система для обеспечения работы ЕС-1010 (ОС-10 ЕС);
- 2) операционная система для обеспечения работы ЕС-1021 (МОС ЕС);
- 3) операционная система для тестовых и диагностических программ;
- 4) дисковая операционная система (ДОС ЕС);
- 5) основная операционная система (ОС ЕС).

Две последние являются основными и наиболее развитыми операционными системами. Перейдем к рассмотрению операционной системы ОС ЕС.

Операционная система ОС ЕС (рис. 1.6) состоит из трех крупных составляющих частей: системы программирования, прикладных средств, управляющей системы.

Следует отметить, что первые две части появились вместе с появлением вычислительных машин. Под системой

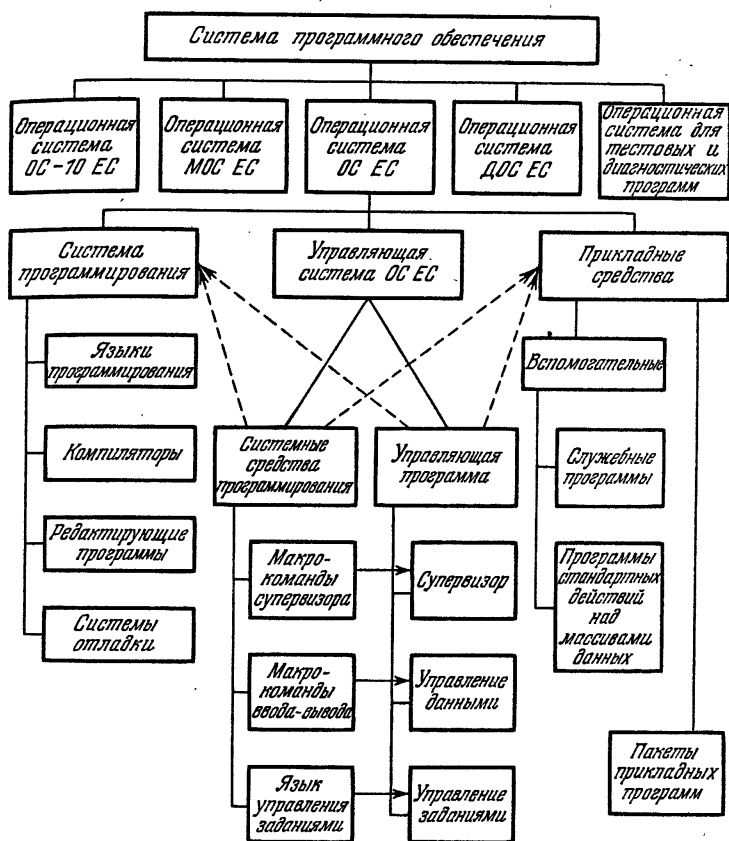


Рис. 1.6.

программирования первоначально понималась совокупность реализуемых аппаратно команд. Позднее в период становления ЭВМ второго поколения появились алгоритмические языки и трансляторы. Прикладные средства или, просто, программы для решения конкретных практических задач, естественно, появились вместе с появлением самих вычислительных машин. Третья часть, управляющая система, поя-

вилась позднее и наиболее значительное развитие получила на ЭВМ третьего поколения.

Управляющая часть операционной системы является основным связующим звеном всего программного хозяйства на ЭВМ. Она осуществляет связь как с двумя другими компонентами ОС, так и непосредственно с техническими средствами ЭВМ.

Каждая из составляющих частей операционной системы представляет собой множество программ, обеспечивающих выполнение определенных функций. В это множество включаются как программы, поставляемые потребителю вместе с ЭВМ, так и программы, разрабатываемые пользователем. Главным образом это относится к прикладным средствам и в значительно меньшей степени к системам программирования и управляющей части.

Пользователь вместе с ЭВМ получает некоторый типовый стандартный комплект программ, состоящий из вышеупомянутых трех частей. Далее пользователь из этого комплекта выбирает, применительно к своим нуждам, только те программы, которые необходимы для решения поставленных перед ним задач. Выбранные программы образуют некоторый конкретный вариант операционной системы данной модели.

Кроме типового комплекта программ, поставляемого с ЭВМ, имеется множество прикладных программ, находящихся в централизованном архиве и поставляемых пользователю только по его специальному заказу. И, наконец, пользователь сам разрабатывает прикладные программы.

Таким образом, на конкретной модели функционирует некоторый вариант операционной системы, включающий в себя программы из стандартного комплекта, из централизованного архива и программы, составленные самим пользователем. Название операционной системы определяется, главным образом, типом управляющей части стандартного комплекта (ДОС ЕС, ОС ЕС и т. д.).

Перейдем к рассмотрению отдельных составляющих частей операционной системы ЕС ЭВМ.

Система программирования. В состав системы программирования (рис. 1.6) входят: языки программирования, компиляторы, редактирующие программы, системы отладки.

Первой важнейшей характеристикой системы программирования следует считать степень ее универсальности, которая определяется количеством языков программирования, типами этих языков и накоплением приемов и средств составления программ на уровне машинно-ориентированного языка.

Второй важной характеристикой системы программирования является расширяемость системы. В ЕС ЭВМ такими свойствами обладают машинно-ориентированные языки и некоторые языки высокого уровня.

Третьей характеристикой системы программирования является степень отражения в языке программирования конкретных свойств аппаратуры, которая, в зависимости от обстоятельств, может рассматриваться и как достоинство, и как недостаток.

Для машинно-ориентированного языка это качество относится к достоинствам, а для языков высокого уровня — к недостаткам.

Четвертой характеристикой является обеспечение модульности программ, конструируемых с помощью системы программирования. Это свойство обеспечивается редактирующими программами.

Рассмотрим организацию системы программирования (рис. 1.7) более подробно. Рабочие программы по этой схеме получают не сразу на выходе компилятора, а сначала на выходе редактора связей — специальной программы, с помощью которой можно получить не только рабочую программу, написанную на каком-либо одном алгоритмическом языке, но и рабочую программу, отдельные части которой написаны на различных алгоритмических языках. В этом смысле редактор связей выполняет функции объединения различных языков программирования.

Для того чтобы редактор мог выполнить предписанные ему функции, необходимо принять единое стандартное представление выходной информации компиляторов. Здесь отметим, что в составе системы программирования, как правило, имеется только одна программа, которая выполняет функции редактора связей.

Последней составной частью системы программирования является система отладки. Это один из важных компонентов, так как известно, что процесс отладки занимает значительное место в практике программирования. По некоторым оценкам на отладку программ уходит от 60 до 80 % общего времени, затрачиваемого на разработку программы. Поэтому даже небольшое сокращение времени, затрачиваемого на отладку, дает значительный выигрыш в общем ресурсе времени.

Для сокращения времени отладки разрабатываются специальные системы отладки. Подобные системы учитывают специфику языков программирования и привязаны к конкретным языкам, они представляют собой

совокупность средств, позволяющих облегчить отдельные операции отладки.

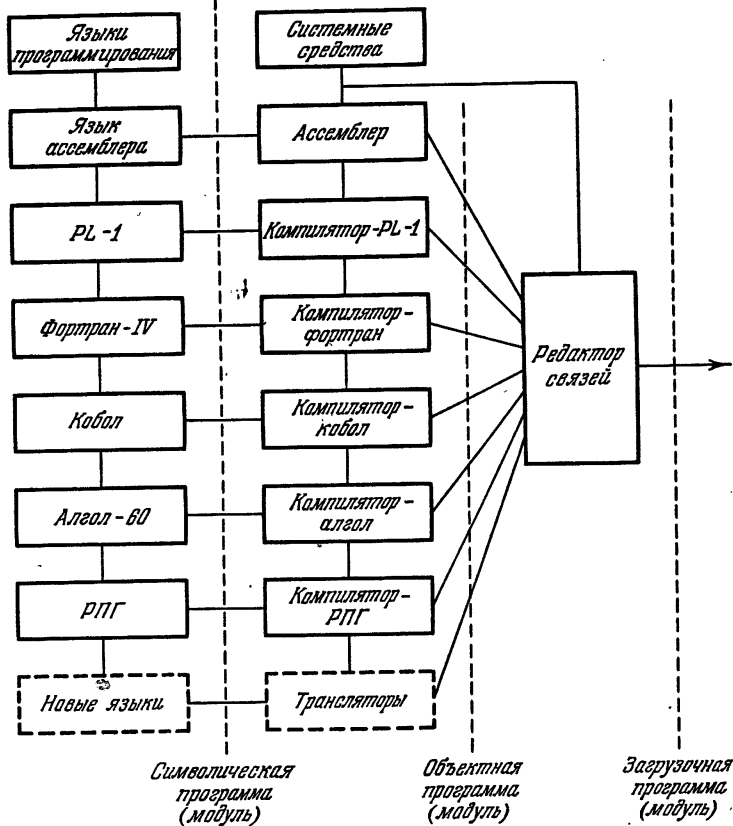


Рис. 1.7.

Прикладные средства. Другой, наиболее важной для использования частью операционной системы являются прикладные средства или прикладные программы.

По объему прикладные средства превышают все остальные части операционной системы, вместе взятые, и, кроме того, находятся в состоянии постоянного и непрерывного расширения. Детальная классификация программ прикладного характера выходит за рамки тематики этой книги. Ограничимся лишь некоторыми общими соображениями о классификации прикладных средств. Важное место среди них занимают вспомогательные программы.

На рис. 1.6 они представлены двумя группами:

- 1) служебные программы;
- 2) программы стандартных действий над массивами данных.

К служебным относятся программы, с помощью которых операционная система записывается на физический носитель, готовится к работе и обслуживается. На практике для обслуживания такой специфичной системы, как операционная, существуют программы, позволяющие следить за состоянием системы, вносить в нее изменения, копировать ее и производить много других операций.

Программы стандартных действий над массивами данных позволяют выполнять ряд часто используемых на практике типовых операций с массивами данных. К подобным операциям относятся операции сортировки массивов по самым различным признакам, операции перезаписи данных, стандартные процедуры ввода-вывода и многие другие.

Поскольку рассмотренные группы программ чрезвычайно часто используются при эксплуатации средств ЕС ЭВМ и, более того, без некоторых из них работа с операционной системой просто невозможна, то они включаются в состав операционной системы и всегда поставляются вместе с ней.

Из прикладных средств на рис. 1.6 выделена еще одна группа — группа так называемых пакетов прикладных программ. Для любой операционной системы характерно наличие пакетов прикладных программ. Эти программы образуют централизованный архив, из которого они поставляются пользователю только по его желанию. Класс пакетов прикладных программ не входит в типовой стандартный комплект операционной системы, поскольку, во-первых, он просто громаден по объему, во-вторых, из всего объема каждому пользователю нужны только некоторые программы, и, наконец, в момент поставки системы нужных конкретному потребителю программ в составе пакетов может просто не оказаться. Здесь следует отметить одно из наиболее важных свойств ЭВМ и СПО Единой Системы — написанные в некоторой операционной системе программы, в принципе, могут работать на всех стандартных моделях ЕС ЭВМ. Это свойство и позволяет создавать единый фонд алгоритмов для всех пользователей.

Множество пакетов прикладных программ постоянно расширяется по мере включения новых разработанных программ.

Управляющая система.

Общие положения. Наиболее важной компонентой операционной системы является управляющая система, являющаяся базой для построения двух других компонент — системы программирования и прикладных средств. Отметим, что управляющая система представляет собой как бы продолжение технических средств, увеличивающее их возможности. Эти возможности реализуются управляющей программой операционной системы (см. рис. 1.6). Кроме управляющей программы, в состав управляющей системы входит еще одна компонента — системные средства программирования.

По существу, управляющую систему можно рассматривать как совокупность дополнительных команд ЭВМ (системные средства программирования), реализованных программным путем (управляющая программа). Компоненты операционной системы используют средства программирования управляющей системы, что отмечено на рис. 1.6 пунктирными связями.

Управляющая программа. Рассмотрим более подробно управляющую программу. Как видно из схемы (см. рис. 1.6), она представлена в виде трех элементов: супервизор, управление данными, управление заданиями. Каждый из этих элементов управляющей программы привязан к конкретным составным частям структуры ЭВМ.

Супервизор (его иногда называют «управление задачами») выполняет функции по реализации режима мультипрограммирования на ЭВМ Единой Системы. В существующих ЭВМ Единой Системы собственно технические средства не позволяют непосредственно реализовать режим мультипрограммирования. Для этого необходимо иметь некоторую совокупность дополнительных программных средств, которые предоставляет супервизор. К дополнительным средствам следует отнести переключение с одной задачи на другую, защиту каждой задачи от влияния других, распределение памяти между задачами, распределение каналов между задачами и т. д. Супервизор взаимодействует с центральным процессором ЭВМ и представляет собой совокупность так называемых SVC-программ, обращение к которым производится с помощью команды процессора — вызов супервизора (SVC), предназначенной для расширения системы команд процессора.

На рис. 1.8 условно изображена система команд ЕС ЭВМ, состоящая из 144 команд (Ряд 1). Среди них находится

команда SVC (вызов супервизора), с помощью которой можно расширить систему команд ЭВМ еще на 256 команд, реализуемых программным путем.

Таким образом, супервизор расширяет возможности центрального процессора и обеспечивает управление

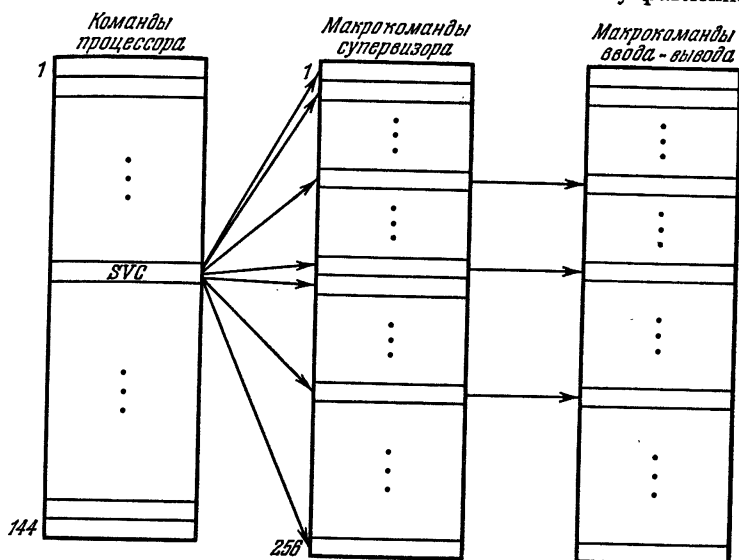


Рис. 1.8.

процессами, происходящими на уровне двух устройств ЭВМ — оперативной памяти и процессора.

Кроме центрального процессора, в составе ЭВМ (см. рис. 1.5) имеется несколько процессоров ввода-вывода — каналов, которые обеспечивают возможность одновременного ввода и вывода информации из ЭВМ сразу по нескольким трактам, что создает условия для совмещения операций ввода-вывода с работой центрального процессора и одновременного осуществления нескольких операций ввода-вывода как для одной задачи, так и для нескольких.

Управление этими процессами осуществляет система управления данными (см. рис. 1.6).

Система управления данными обеспечивает связь внешних устройств с основной (оперативной) памятью ЭВМ. Через каналы ввода-вывода программы, входящие в состав этой компоненты, обеспечивают выполнение всех операций ввода-вывода в ЭВМ.

Для программистов супервизор и система управления данными представляет собой множество макрокоманд супервизора и ввода-вывода.

Макрокоманды супервизора обеспечивают обращение к программам супервизора с помощью команды SVC, а макрокоманды ввода-вывода обеспечивают обращение к программам системы управления данными с помощью средств обращения к подпрограммам. Системные средства программирования условно представлены на рис. 1.8 в виде трех наборов команд, состоящих из системы команд процессора, команд супервизора и команд ввода-вывода. Часто команды ввода-вывода используют команды супервизора, что на рис. 1.8 отмечено: связями между соответствующими наборами команд. Таким образом, супервизор и управление данными обеспечивают функционирование всех устройств в составе ЭВМ.

На рис. 1.6 управляющая программа содержит еще одну составную часть — управление заданиями. Управление заданиями обеспечивает связь человека с ЭВМ. Его можно рассматривать как некоторое «устройство», которое управляет использованием всей ЭВМ в целом. На ЭВМ первого и второго поколений медленные действия человека (по сравнению с устройствами) в процессе постановки и запуска задач на ЭВМ, в момент снятия и окончания задач на ЭВМ, а также часто и в процессе решения снижали общую производительность системы.

На ЭВМ, управляемой супервизором и системой управления данными, просто невысказано, чтобы указанные операции выполнялись вручную. Поэтому эти функции возложены на специальную часть управляющей программы — управление заданиями. Основная его задача — ликвидировать, по мере возможности, разрыв между скоростными характеристиками ЭВМ и человека на вышеупомянутых стадиях реализации программ.

Компонента управления заданиями автоматически принимает задачи, запускает их, оканчивает, управляет некоторыми операциями, необходимыми в процессе выполнения задач. На долю человека остаются лишь некоторые функции управления: ответ на сообщения системы и выполнение чисто физических действий, таких как установка носителей информации, включение и выключение устройств и некоторые другие.

Такой подход к использованию ЭВМ предполагает, что программисты со своими программами не выходят непосредственно на машину. Однако для того, чтобы их

программы можно было выполнить на ЭВМ без их присутствия, необходимо, чтобы организация процесса прохождения программы пользователя была формально описана.

Управляющая программа должна «знать», какие ресурсы системы нужны для выполнения каждой программы и какие действия следует предпринять после завершения программы или в процессе ее выполнения при возникновении определенных ситуаций. При работе с операционной системой программист каждую свою программу или сразу несколько программ оформляет в виде отдельного задания, используя для этого специальный язык управления заданиями.

Системные средства программирования. Как уже отмечалось, супервизор расширяет систему команд ЭВМ (см. рис. 1.8). По существу, предлагается набор команд супервизора, которые могут быть использованы любым программистом, работающим на машинно-ориентированном языке ассемблер. С помощью команд супервизора можно программировать многие стандартные, часто встречающиеся операции, такие, например, как поиск программ, вызов их в оперативную память и передачу им управления, резервирование памяти для рабочих областей, не заботясь о их конкретном местонахождении в программе на этапе ее составления. Супервизор в своем составе имеет много команд, оказывающихся полезными при составлении сложных программ, например, аппарат синхронизации событий, использующийся при синхронизации операций ввода-вывода, аппарат управления ресурсами, облегчающий программирование операций обработки общих массивов информации и т. д.

С помощью команд ввода-вывода можно программировать операции ввода-вывода практически для всех внешних устройств ЕС ЭВМ. Реализацию этих операций осуществляют программы управления данными. Команды ввода-вывода вместе с программами, их реализующими, образуют систему управления данными.

Система управления данными любой операционной системы дает возможность пользователям работать с данными различной организации и структуры. Для каждого способа организации массивов данных имеются свои команды ввода-вывода, позволяющие программистам работать на некотором абстрактном, логическом уровне. Таким образом, программирование операций ввода-вывода состоит в том, что программист вставляет в свою программу определенную

команду (макрокоманду), представляющую собой запрос на выполнение соответствующей операции, а реализуется данный запрос только в момент выполнения программы средствами управления данными.

Нужно отметить, что на этапе программирования пользователь может не учитывать тип внешнего устройства, поскольку тип устройства достаточно указать в задании, используя средства языка управления заданиями. Это обстоятельство оказывается важным для достижения совместимости различных ЭВМ Единой Системы. Отлаженная на ЭВМ определенной конфигурации программа может быть поставлена на ЭВМ, которая оснащена другими внешними устройствами, отличающимися от исходных внешних устройств не только количеством, но и типом. Для такого переноса достаточно изменить формулировку задания с помощью средств языка управления заданиями.

Команды супервизора и ввода-вывода в совокупности представляют собой системные команды, образующие системные средства программирования. Нужно сказать, что системные средства программирования были и могут быть использованы для создания компонент операционной системы. На этой основе созданы все компиляторы, большинство вспомогательных прикладных программ, пакеты прикладных программ и т. д. В настоящее время данная система является основной для многих пользователей.

Особое место среди средств программирования управляющей системы занимает язык управления заданиями. Это тоже средство программирования, но программирования заданий. Одна из важных особенностей ЭВМ Единой Системы, работающей под управлением операционной системы, состоит в обязательности использования этого языка. Использование обязательно, независимо от того, на каком уровне производится работа: на уровне машинно-ориентированного языка, на уровне языков высокого уровня или даже на уровне готовых прикладных программ. Это означает, что запустить какую-либо программу на ЕС ЭВМ, работающей под управлением операционной системы, можно только, оформив и описав ее как задание на языке управления заданием.

СВЕДЕНИЯ ПО ЛОГИЧЕСКОЙ СТРУКТУРЕ МОДЕЛЕЙ ЕС ЭВМ

В данной главе рассмотрим логическую структуру, или, как ее называют иначе, архитектуру моделей ЕС ЭВМ. Главный акцент сделан на архитектуру центральной части машины, состоящей из центрального процессора, оперативной памяти и каналов. Периферийного оборудования только коснемся, анализируя лишь концептуальные аспекты аппаратной системы ввода-вывода.

Детально логическую структуру ряда важнейших внешних устройств — запоминающих устройств на магнитной ленте и магнитных дисках — рассмотрим позднее в процессе анализа основных принципов системы управления данными. (гл. 4).

В основу рассмотрения положен принцип анализа главных аппаратных средств, обеспечивающих достижение программной совместимости современных вычислительных машин. Указанные средства, кроме этого, определяют важнейшие функции и структурные особенности системного программного обеспечения — операционных систем.

Необходимыми и достаточными условиями достижения программной совместимости ЭВМ являются идентичность реализации и представления информации, системы команд и средств мультипрограммной работы. Руководствуясь этой схемой, продолжим изложение.

2.1. Общая структура моделей

Любая модель ЕС ЭВМ (см. рис. 1.5) состоит из обязательного набора четырех типов устройств:

- 1) центральный процессор (ЦП), который содержит схемы для выполнения различных операций и схемы управления;
- 2) оперативная (основная) память (ОП);
- 3) канал — специализированный процессор, обеспечивающий операции обмена (число каналов колеблется в зависимости от модели от трех до семи);
- 4) набор внешних устройств (ВУ) с устройствами управления (УУ).

Количество подключаемых к каналу УУ колеблется от одного до десяти, к каждому из них обычно может быть подключено от одного до восьми внешних устройств.

Устройства управления внешними устройствами подключаются к каналу через стандартный интерфейс, который представляет собой специальный (34-проводный) кабель с унифицированными параметрами. Канал самостоятельно осуществляет коммутацию между оперативной памятью и внешними устройствами, но начинает операции обмена по инициативе центрального процессора. Основное назначение канала — передача информации из ОП во внешнее устройство и от внешнего устройства в ОП. Никаких преобразований информации в канале не производится.

Различают два типа каналов: 1) мультиплексный, 2) селекторный. К некоторым моделям ЕС может быть подсоединено до семи каналов (шесть селекторных и один мультиплексный).

Внешние устройства отличаются друг от друга по скорости передачи и поиска информации. Высокоскоростные внешние устройства — накопители на магнитных лентах и дисках — подключаются только к селекторным каналам. Устройства ввода с перфокарт, устройства вывода на перфокарты, пишущие машинки, т. е. низкоскоростные устройства, обслуживает, как правило, мультиплексный канал.

Подключение внешних устройств к моделям ЕС ЭВМ через автономно работающие каналы позволяет увеличить производительность ЭВМ, определяемую во многих случаях процессами ввода и вывода информации. Поскольку информацию можно вводить одновременно с нескольких устройств, увеличивается интегральная производительность ЭВМ, эффективнее используется основная память. Заметим, что мультиплексный канал одновременно может обслуживать несколько внешних устройств, а селекторный — только одно.

2.2. Память ЭВМ Единой Системы

Основная память. Одним из главных устройств машины является основная (оперативная) память. Если в I и II поколениях ЭВМ логическая структура ОП однозначно соответствовала физической, то в ЕС ЭВМ логическая структура ОП значительно усложнилась.

Физически оперативная память моделей ЕС ЭВМ состоит из набора адресуемых регистров. Каждый регистр содержит

определенное число двоичных разрядов. На машинах I и II поколений данные (и команды) могли располагаться в ячейках только одного типа, совпадающих по размерам с регистрами физической ОП. В моделях Единой Системы логически различают ячейки нескольких типов, которые

Структура памяти ЭВМ II поколения



Структура памяти ЭВМ III поколения

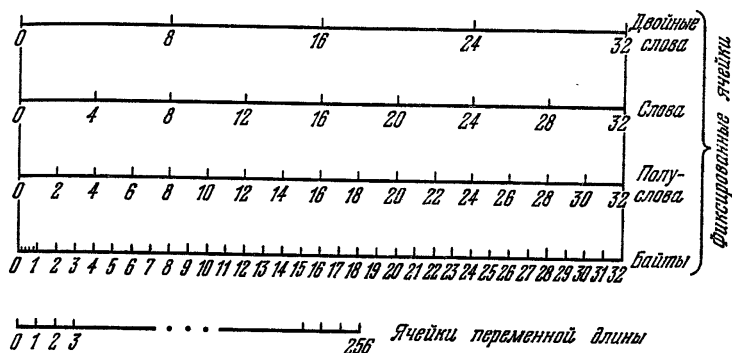


Рис. 2.1.

позволяют вести обработку данных практически любой структуры (рис. 2.1). В ЕС ЭВМ ячейкой считают некоторое поле конечной длины, к которому производится обращение. Рассмотрим имеющиеся типы ячеек.

Минимальной адресуемой единицей памяти является ячейка, называемая байтом. Байт — 8-разрядное или 8-битовое машинное слово.

Двухбайтовые ячейки — полуслова (16-разрядные), адресация таких ячеек производится только с помощью четных чисел, т. е. адрес должен быть кратен двум.

Четырехбайтовые ячейки — слова (32-разрядные). Такие ячейки используются в основном для представления чисел с фиксированной запятой, адрес должен быть кратен четырем.

Восьмибайтовые ячейки — двойные слова (64-разрядные) — используются обычно для работы с числами, представленными в форме с плавающей запятой, а также для

работы с числами с фиксированной запятой повышенной точности, адрес двойного слова должен быть кратен восьми.

И, наконец, ячейки переменной длины — поля переменной длины — используются для обработки десятичных чисел и информации, представленной в виде набора символов. Поля переменной длины могут занимать от 1 до 256 последовательно расположенных байтов.

Адрес такого типа ячеек состоит из двух составных частей: адреса нулевого (левого) байта поля и указателя длины поля, задаваемого целым числом байтов.

Объем ОП принято оценивать в байтах. В ЕС ЭВМ максимальный объем оперативной памяти может быть $(2^{24} - 1)$ байтов.

Вопрос о выравнивании границ. Наличие нескольких типов ячеек в ОП III поколения порождает возникновение вопроса о выравнивании границ. Как уже отмечалось, физически память представляет собой упорядоченный набор регистров определенной длины (за одно обращение можно прочитать или записать информацию только в один регистр), каждый из которых имеет определенный физический адрес. Будем называть такой регистр физической ячейкой. Размер физической ячейки (число двоичных разрядов) больше или кратен 8. А логический выбор информации из памяти при выполнении команд производится, как было отмечено, побайтно, полусловами и т. д. Получается, что логическая структура памяти (с точки зрения программиста) и физическая структура (с точки зрения инженера) не соответствуют друг другу.

Согласование логического и физического формата осуществляется центральным процессором аппаратным способом. Однако для того чтобы не произошло резкого снижения производительности центрального процессора из-за неоправданного увеличения числа обращений в ОП со стороны центрального процессора, на расположение данных длиной в 2, 4, 8 байтов накладываются ограничения, которые сводятся к тому, что адрес таких данных должен быть кратен соответственно 2, 4, 8. Это означает, что программист должен сам следить за кратностью адресов в своей программе.

При несоблюдении кратности в ЭВМ происходит прерывание выполнения программы (что равносильно аварийному окончанию).

Проиллюстрируем это на примере. Пусть память состоит из 16-разрядных физических ячеек (из полуслов), а центральный процессор выполняет команду сложения

содержимого некоторого регистра с числом с фиксированной запятой в коротком формате (2 байта). Если адрес операнда (полуслово) — четное число, то для выполнения операции требуется одно обращение к ОП, если же адрес — число нечетное, то нужно два обращения к ОП (одна половина операнда в одной ячейке, а вторая — в следующей). Реализация второго случая аппаратным способом привела бы к резкому снижению производительности ЭВМ.

Для того чтобы исключить второй случай (когда адрес не кратен 2), процессор команду не выполняет, а выдает сигнал о прерывании выполнения программы. При программировании это обстоятельство во многих случаях должен учитывать сам программист. Практика показывает, что в подобных случаях допускается много ошибок. Средства машинно-ориентированного языка содержат некоторые возможности автоматического выравнивания границ, однако полного решения этого вопроса не дают.

2.3. Структура центрального процессора

Центральный процессор состоит из двух устройств (рис. 2.2):

- 1) центральное устройство управления (ЦУУ);
- 2) арифметико-логический блок (АЛБ), который взаимодействует с основной (оперативной) памятью (ОП) модели.

Всеми процессами в машине управляет ЦУУ. При помощи специального регистра (счетчик команд), содержащего адрес команд, из основной памяти выбираются команды. Данные выбираются и записываются в основную память с помощью регистра данных в АЛБ. Счетчик команд и дополнительная служебная информация о состоянии ЭВМ и выполняемой программы, находящаяся в регистрах ЦУУ, объединены в слово состояния программы (ССП). В ССП находятся, например, признак результата, код длины команды, ключ защиты памяти, признак состояния супервизора или состояния задачи, а в момент прерывания в ССП заносится информация о причине прерывания.

Формат большинства команд машины двухадресный, однако есть ряд трехадресных и одноадресных команд. Большинство операций производится над числами, находящимися по первому и второму адресам, результат исполнения операции помещается обычно по адресу первого операнда, второй при этом остается без изменения.

Набор команд в моделях ЕС ЭВМ Ряда 1 составляет 144 команды, а Ряда 2 — 167.

Арифметико-логический блок реализует выполнение большинства команд центрального процессора. К ним относятся арифметические, логические и ряд специальных

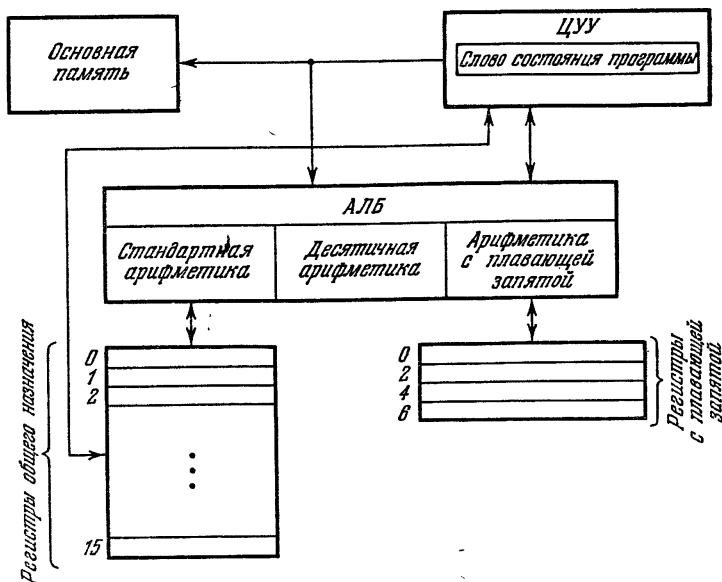


Рис. 2.2.

операций. В структуре АЛБ можно выделить три относительно независимые составные части, аппаратно реализующие:

- 1) стандартную арифметику;
- 2) десятичную арифметику;
- 3) арифметику с плавающей запятой.

Схемы стандартной арифметики, в дальнейшем называемые стандартной арифметикой, управляют выполнением команд с целочисленными операндами (числа с фиксированной запятой), логическими и рядом специальных операций. Отметим, что реализация команд стандартной арифметики осуществляется с использованием одного или нескольких регистров общего назначения (см. рис. 2.2), общее число которых равно 16. Длина каждого регистра — 32 двоичных разряда. Такой набор регистров несколько условно можно рассматривать как сверхоперативную память ЭВМ, которая имеет небольшой объем и является составной частью центрального процессора.

Десятичная арифметика позволяет работать на ЭВМ с числами, представленными в двоично-десятичной системе счисления. Арифметика с плавающей запятой реализуется аналогично стандартной арифметике на четырех специальных регистрах с плавающей запятой, длина каждого из которых равна 64 двоичным разрядам. Таким образом, центральный процессор может работать:

- 1) с числами с фиксированной запятой;
- 2) с числами с плавающей запятой;
- 3) с десятичными числами;
- 4) с логическими данными.

Кроме арифметических и логических операций, центральный процессор реализует операции управления, а также содержит целый ряд средств, позволяющих организовать мультипрограммный режим.

Представление информации в моделях ЕС ЭВМ.

Коды, используемые в ЭВМ Единой Системы. Информационная совместимость, обеспечиваемая аппаратурой моделей ЕС, базируется на едином способе представления и кодирования информации, участвующей в процессе обработки.

В машинах Единой Системы в основном используется побайтный способ представления информации. Программист, разрабатывающий программу, для записи алгоритмов, исходных, промежуточных и выходных данных и т. п. использует некоторый регламентированный набор символов и знаков, применяющихся для ЭВМ данной системы.

В процессе обработки информация, заданная программистом с помощью символов и знаков, внутри ЭВМ представляется в закодированном виде, при котором каждый символ или знак записываются в однобайтовой ячейке в некотором принятом двоичном представлении.

Таким образом, программисту, работающему с ЭВМ, необходимо, как минимум, знать две вещи: во-первых, какими символами он может пользоваться при составлении программы и, во-вторых, какое двоичное представление соответствует каждому символу внутри ЭВМ.

Для обмена информацией в ЕС ЭВМ используется так называемый «двоичный код для обмена и обработки информации» (ДКОИ), символы которого представлены в табл. 2.1.

В ДКОИ дается только внутреннее двоичное представление регламентированных символов и знаков. В то же время программисту на практике часто приходится иметь дело с промежуточным представлением информации на

Таблица 2.1

Символы двоичного кода для обмена и обработки информации

Зональная группа									
0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	1	1	1	0	0
2	0	0	1	0	0	1	1	0	0
3	0	1	0	1	0	1	1	0	1

Цифровая группа									
0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ПУС	АР1	Д00	Д16	ПР	&	- /	ц	й	я	ь	{	}	дз	0
1	НЗ	СУ1	Д01	Д17				а	й	-	ы	А	К	с	1
2	НТ	СУ2	Д02	Д18				б	й	s	з	В	Л	т	2
3	КТ	СУ3	Д03	Д19				с	й	t	ш	С	М	u	3
4	Д28	Д29	Д04	Д20				d	й	и	э	Е	N	v	4
5	ГТ	Д05	ПС	Д21				e	й	ц	щ	С	O	w	5
6	Д06	ВШ	КБ	Д22			ю	f	й	х	ч	С	P	x	6
7	ЗБ	Д07	АР2	КП			б	g	й	у	ь	С	Q	y	7
8	Д23	АН	Д08	Д24			,	h	й	з	ю	С	R	z	8
9	Д13	КН	Д09	Д25			:	i	й	т	Б	С			9
A	Д14	Д18	Д10	Д26			#	д	й	ж	Ц	И	П	я	З
B	ВТ	Д15	Д11	Д27			@	ф	й	в	Д	И	я		Э
C	ПФ	РФ	Д12	СТП			%	г	й		Ф	И			Щ
D	ВК	РГ	КТМ	НЕТ			-	и	й	т	Г	И			Ч
E	РУС	РЗ	ДА	Д30			>		й	ж	Ф	Л			Б
F	ЛАТ	РЭ	ЗВ	ЗМ			? *		й	в	Г				ЗБ

*) Допускается логическое «не».

**) Допускается прописная буква «б».

перфокартах. В ЕС ЭВМ в большинстве случаев применяются стандартные 12-позиционные и 80-колонные перфокарты.

Информация программиста чаще всего наносится на перфокарты, с которых она вводится в ЭВМ в коде ДКОИ. Каждый символ исходной информации перфорируется в отдельной колонке перфокарт. Для представления символов ДКОИ на двенадцатипозиционной перфокарте разработан специальный код перфокарт (КПК-12). Преобразование символов из кода перфокарт в код ДКОИ осуществляется аппаратно соответствующим устройством в процессе ввода.

При рассмотрении логической структуры оперативной памяти были выделены 5 типов ячеек: байт, полуслово, слово, двойное слово и поле переменной длины. Первые четыре типа предназначены для обработки данных фиксированного формата, к которым относятся числа с фиксированной и плавающей запятой, команды процессора и каналов и т. д. Поля переменной длины используются для обработки десятичных чисел, текстовой информации и т. д. Если текстовую информацию в моделях ЕС ЭВМ можно записывать, используя символы ДКОИ, то числовая информация, команды и ряд специальных управляющих слов имеют свои способы представления в ЭВМ. Рассмотрим эти способы представления отдельно.

Кодирование чисел. Способы представления чисел в моделях ЕС показаны на рис. 2.3.

Числа с фиксированной запятой. Число с фиксированной запятой занимает ячейку в два, а чаще всего в 4 байта (рис. 2.3). Самый левый (нулевой) бит отводится под знак числа: 0 означает плюс, а 1 — минус. Запятая фиксируется справа; отрицательные числа представляются в дополнительном коде.

Числа с плавающей запятой. Число с плавающей запятой занимает ячейку типа слово или двойное слово (рис. 2.3). Самый левый (нулевой) бит отводится под знак числа: 0 соответствует плюсу, а 1 — минусу. Следующие 7 битов (с 1 по 7) содержат характеристику. Оставшиеся биты содержат мантиссу. Мантисса в ячейке типа слова задается 6 шестнадцатеричными числами (обычная точность), а в ячейке двойного слова 14 шестнадцатеричными числами (удвоенная точность). Значение числа с плавающей запятой получается умножением мантиссы, у которой запятая фиксируется слева, на $16^X - 64$, где X — характеристика. При таком способе

Двоичные числа с фиксированной запятой

Полуслово

±	целое	
0	1	15

Полное слово

±	целое	
0	1	31

Двоичные числа с плавающей запятой

Короткое

±	характеристика	мантисса
0	1	31

Длинное

±	характеристика	мантисса
0	1	63

Десятичные данные

Формат с зоной

зона	цифра	зона	цифра
0	3	7	15

зона	цифра	знак	цифра
0	3	7	15

Упакованный формат

цифра	цифра	цифра	цифра
0	3	7	15

цифра	цифра	цифра	знак
0	3	7	15

Логические данные

Байт

07

Полуслово

0	15

Полное слово

031

Поле переменной длины

символ	символ		символ
0	7	15	31

Рис. 2.3.

представления можно записывать числа, которые по абсолютной величине больше 10^{-78} и меньше 10^{+78} .

Десятичные числа. В моделях ЕС ЭВМ имеется возможность работать непосредственно с десятичными числами. Десятичные данные представляются в двух форматах (см. рис. 2.3): формат с зоной, упакованный формат. Знак числа представляется в виде шестнадцатеричного кода: С означает плюс, а D — минус.

Команды десятичной арифметики обрабатывают числа только в упакованном формате. Рассмотрим пример представления десятичного числа — 4123:

F4F1F2D3 — зонный формат,
04123D — упакованный.

Обратим внимание на расположение кода знака числа в зонном формате. Помещать знак на свое место в число должен сам программист, что на первых порах является источником многочисленных ошибок.

Число десятичных цифр в числах, используемых для обработки в зонном формате, лежит в диапазоне от 1 до 16, а в упакованном формате от 1 до 31 и всегда нечетно, так как число должно занимать целое число байтов, а самая правая позиция отводится под код знака.

Кроме того, в моделях ЕС ЭВМ имеется возможность работать с логическими данными. Логические данные могут быть размещены в полном слове, в полуслове, в одном байте, а могут занимать поле переменной длины. Форматы представления логических данных приведены на рис. 2.3.

Система команд центрального процессора. Следуя принятой в начале главы схеме анализа программной совместимости, рассмотрим основные вопросы построения системы команд. При этом не будем касаться содержательной части команд и занимать внимание их перечислением. Указанные сведения можно получить из соответствующих справочников. Главное внимание уделим существенным особенностям, определяющим как большие функциональные возможности, так и основу программной совместимости.

Кодирование команд. Схематично команды моделей ЕС ЭВМ можно представить в виде

КОП : операнды
где КОП — код операции.

Все команды в первом байте содержат код операции, а в последующих байтах — операнды или их адреса.

Команды имеют разную длину и бывают:

- 1) двухбайтовые;
- 2) четырехбайтовые;
- 3) шестибайтовые.

Переменная длина команды объясняется наличием двух уровней оперативной памяти: сверхоперативной (регистры) и основной. Поэтому в процессоре предусмотрены команды, работающие, во-первых, только с регистрами (двухбайтовые команды), во-вторых, только с основной памятью (шестибайтовые команды) и, в-третьих, с регистрами и с основной памятью одновременно (четыребайтовые команды). Самыми быстрыми командами являются двухбайтовые, так как они работают только с регистрами.

В четырехбайтовых командах, как правило, первый операнд находится в регистре, а второй — в основной памяти. В шестибайтовых командах оба операнда находятся в оперативной памяти.

Рассмотрим принятые способы записи адресов операндов.

Для операндов, находящихся в регистрах сверхоперативной памяти, адрес задается с помощью двоичного числа, определяющего номер соответствующего регистра. Для задания регистра требуется одна шестнадцатеричная цифра (4 бита), так как регистров не больше 16.

Адреса 16 регистров общего назначения лежат в пределах от 0 до 15, а адреса 4 регистров с плавающей запятой имеют значения 0, 2, 4, 6, так как длина каждого из них соответствует длине двух регистров общего назначения.

Адреса основной памяти кодируются специальным образом в пределах полуслова (рис. 2.4), где В — базовый регистр, D — смещение (занимает 12 битов).

Адрес в машине образуется как $(B) + D$ (рис. 2.4), где (В) обозначает содержимое регистра В. Представленный способ адресации имеет ряд преимуществ: во-первых, экономия 8 разрядов на представление адреса (максимальный адрес в основной памяти для ЕС ЭВМ — $2^{24}-1$), и, во-вторых, перемещаемость программы, поскольку программа в таком случае не зависит от содержимого базового регистра и пишется в относительных адресах. На рис. 2.4 можно видеть пример кодирования адреса основной памяти, равного 7048. (Регистр 2 содержит базу, равную 5000.)

Одновременно следует отметить и недостатки данного принципа адресации, которые сводятся к следующему:

1. Некоторое увеличение времени цикла исполнения команды за счет дополнительной операции сложения при вычислении физического адреса операнда.

2. Усложнение процесса конструирования программ, размер которых превышает число 4096 (2^{12}) байтов. При

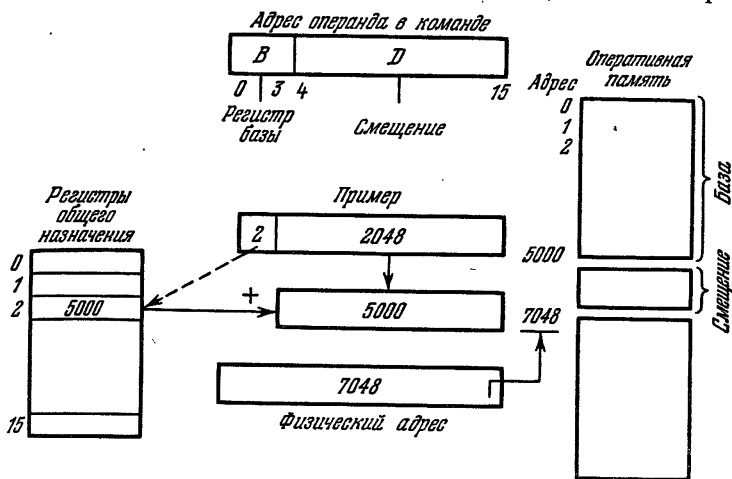


Рис. 2.4.

этом возникают проблемы, известные под названием секционирования программ и формирования базовых регистров.

Последний недостаток оказывает существенное влияние на построение системы программирования машинно-ориентированного типа (ассемблер), а также на другие ее составные части (компиляторы, редакторы и т. д.).

Во многих командах адреса основной памяти задаются с помощью трех составных элементов: X, B, D, где B и D — уже знакомые нам базовый регистр и смещение, а X — адрес одного из общих регистров, играющий роль индексного регистра. При таком способе задания адрес в машине образуется сложением $(X) + (B) + D$, где (X) — содержимое индексного регистра. Для представления адреса с индексным регистром необходимо двадцать двоичных разрядов (два байта и одна шестнадцатеричная цифра).

В качестве индексных и базовых регистров могут использоваться любые 15 общих регистров (1–15). Регистр 0 в качестве базового и индексного использовать нельзя. Нулевое значение полей B или X в адресной части

Формат	Структура команды и обозначение операндов	Коды команд ассемблера
RR	<div> <div>Код операции</div> <div>R₁</div> <div>R₂</div> </div> <p>R₁, R₂</p>	все команды RR, кроме SPM и SVC
	<div> <div>Код операции</div> <div>R₁</div> <div></div> <div></div> <div></div> <div></div> </div> <p>R₁</p>	SPM
	<div> <div>Код операции</div> <div>I</div> </div> <p>I</p>	SVC
RX	<div> <div>Код операции</div> <div>R₁</div> <div>X₂</div> <div>B₂</div> <div>D₂</div> </div> <p>R₁, D₂, (X₂, B₂) R₁, S₂, (X₂)</p>	все команды RX
RS	<div> <div>Код операции</div> <div>R₁</div> <div>R₃</div> <div>B₂</div> <div>D₂</div> </div> <p>R₁, R₃, D₂ (B₂) R₁, R₃, S₂</p>	BXH, BXLE, STM, LM
	<div> <div>Код операции</div> <div>R₁</div> <div></div> <div></div> <div></div> <div>B₂</div> <div>D₂</div> </div> <p>R₁, D₂ (B₂) R₁, S₂</p>	все команды сдвига

Рис. 2.5.

Формат	Структура команды и обозначение операндов	Коды команд ассемблера																				
SI	<table border="1" style="margin-bottom: 10px;"> <tr> <td>Код операции</td> <td>I_2</td> <td>B_1</td> <td>D_1</td> </tr> </table> <p>$D_1 (B_1) I_2$ S_1, I_2</p> <table border="1" style="margin-bottom: 10px;"> <tr> <td>Код операции</td> <td></td> <td>B_1</td> <td>D_1</td> </tr> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table> <p>$D_1 (B_1)$ S_1</p>	Код операции	I_2	B_1	D_1	Код операции		B_1	D_1													<p>NI, CLI, OI, XI, TM, MVI, RDD, WRD</p> <p>SIO, HIO, TIO, TCH, SSM, LPSW, TS</p>
Код операции	I_2	B_1	D_1																			
Код операции		B_1	D_1																			
SS	<table border="1" style="margin-bottom: 10px;"> <tr> <td>Код операции</td> <td>L_1</td> <td>L_2</td> <td>B_1</td> <td>D_1</td> <td>B_2</td> <td>D_2</td> </tr> </table> <p>$D_1 (L_1, B_1), D_2 (L_2, B_2)$ $D_1 (L_1, B_1), S_2 (L_2)$ $S_1 (L_1), D_2 (L_2, B_2)$ $S_1 (L_1), S_2 (L_2)$</p> <table border="1" style="margin-bottom: 10px;"> <tr> <td>Код операции</td> <td>L</td> <td>B_1</td> <td>D_1</td> <td>B_2</td> <td>D_2</td> </tr> </table> <p>$D_1 (L, B_1), D_2 (B_2)$ $D_1 (L, B_1), S_2$ $S_1 (L), D_2 (B_2)$ $S_1 (L), S_2$</p>	Код операции	L_1	L_2	B_1	D_1	B_2	D_2	Код операции	L	B_1	D_1	B_2	D_2	<p>AP, SP, MP, DP, CP, ZAP, MVO, PACK, UNPK</p> <p>NC, CLC, OC, XC, MVC, MVN, MVZ, TR, TRT, EDMK, ED</p>							
Код операции	L_1	L_2	B_1	D_1	B_2	D_2																
Код операции	L	B_1	D_1	B_2	D_2																	

Условные обозначения:

R_1, R_2, R_3 — абсолютные выражения, определяющие регистры;
 D_1, D_2 — абсолютные выражения, не превышающие 4096, определяющие смещения;
 B_1, B_2 — абсолютные выражения, определяющие регистры базы;
 X_2 — абсолютное выражение, определяющее регистр индекса;
 L, L_1, L_2 — абсолютные выражения, определяющие длины полей;
 I, I_2 — абсолютные выражения, представляющие непосредственные данные, от 0 до 255;
 S_1, S_2 — абсолютные или перемещаемые выражения, определяющие адреса памяти.

Рис. 2.5 (продолжение)

означает не содержимое регистра 0, а нулевое значение базы или индекса.

Структура команды в ЕС ЭВМ в основном двухадресная, схема исполнения которой следующая. Операция, указанная в КОП, выполняется с участием двух операндов, заданных в адресной части команды, результат операции засылается на место первого операнда.

Незначительное количество команд имеют структуру, отличную от двухадресной. Еще меньше команд, в которых используется непосредственная адресация.

Форматы команд. Все двухбайтовые команды имеют формат RR. К командам этого формата относятся двухадресные команды,¹ у которых оба операнда хранятся в регистрах, а также некоторые команды (рис. 2.5) со структурой, отличной от двухадресной.

Наиболее употребительными являются команды формата RX, у которых R_1 — адрес регистра, содержащего первый операнд, а X_2 , B_2 , D_2 задают адрес ячейки основной памяти, содержащей второй операнд (рис. 2.5). Заметим, что использование принципа индексации возможно только в командах этого формата.

В моделях ЕС ЭВМ имеется несколько команд, для выполнения которых требуется три адреса — команды формата RS (рис. 2.5). В этих командах поле индексного регистра используется как третий операнд, а в командах сдвига вовсе не используется и может содержать произвольную информацию.

Некоторые типы команд длиной в 4 байта используют в качестве второго операнда содержимое части самой команды (непосредственная адресация), а некоторые используют только один операнд, находящийся в основной памяти. К ним относятся команды формата SI (рис. 2.5). Поле I_2 в командах этого формата определяет один байт данных, участвующий в операции.

Все типы команд, рассмотренные выше, производят действия в основном над данными фиксированного формата. Данные переменного формата (поля переменной длины) обрабатываются командами формата SS (см. рис. 2.5). Команды этого формата выполняют действия над двумя операндами, находящимися в основной памяти машины. Длина команд этого типа равна шести байтам (см. рис. 2.5).

На рис. 2.6 представлены примеры команд всех форматов.

Рассмотренная нами классификация команд осуществлена по признакам длины команды и внутренней структуры

адресной части. Иногда удобно классифицировать команды по типам и форматам данных, используемых в процессе обработки. По этому признаку можно выделить следующие подмножества:

- 1) команды с фиксированной запятой;
- 2) команды с плавающей запятой;
- 3) команды десятичной арифметики;
- 4) логические команды и т. д.

RR

AR	7	9
----	---	---

Сложение содержимого регистров 7 и 9.
Результат засылается в регистр 7.

RX

ST	3	10	14	300
----	---	----	----	-----

Пересылка содержимого 3 регистра в основную память по адресу: $300 + (14) + (10)$

RS

LM	3	4	11	200
----	---	---	----	-----

Загрузка регистров с 3 по 4 данными из основной памяти, находящимися по адресу: $200 + (11)$

SI

MVI	A	12	10
-----	---	----	----

Двоичное представление символа A записывается по адресу: $10 + (12)$

SS

AP	3	3	6	24	6	48
----	---	---	---	----	---	----

Десятичное число длиной в 4 байта по адресу: $24 + (6)$ складывается с числом такой же длины по адресу: $48 + (6)$. Результат засылается по адресу: $24 + (6)$.

Рис. 2.6.

Такая классификация команд более удобна для первоначального ознакомления с системой команд.

Возможны также и другие подходы к классификации множества команд ЭВМ. Упомянем здесь только еще об одной схеме классификации, принятой для центральных процессоров ЕС. Речь идет о разделении системы команд на 2 группы: привилегированные и непривилегированные. Привилегированные операции используются для организации мультипрограммного режима и управления некоторыми аппаратными средствами. К непривилегированным относятся все остальные. Привилегированные команды

применяются управляющими программами (супервизором) и поэтому не могут быть употреблены в прикладных программах.

Средства управления многопрограммным режимом работы. В сложной вычислительной системе, какой является модель ЕС ЭВМ, очень важную роль играет возможность автоматического управления ходом решения задач. Для реализации этой возможности необходимо иметь аппарат, который позволял бы фиксировать все изменения, возникающие в системе, чтобы можно было на них соответствующим образом реагировать. Таким аппаратом является система прерываний.

Система прерываний позволяет решать такие задачи, как управление передачей данных, управление прохождением задачи через ЭВМ, управление режимом мультипрограммирования и т. д.

Система прерываний. Прежде чем отвечать на вопрос о реализации системы прерываний, рассмотрим некоторые аспекты, непосредственно влияющие на принятые решения. Прерывание, как некоторый сигнал об изменении ситуации в системе, может происходить в произвольный момент времени, и желательно, чтобы соответствующие действия системы были выполнены за возможно короткий интервал времени. Поэтому прерывание реализуется аппаратным способом.

В то же время при одновременном возникновении случайным образом множества сигналов прерываний нужны средства для упорядочения их обработки. Для этого все прерывания разбиты на классы, каждый из которых определяет некоторое подмножество прерываний, причем каждому классу присвоен определенный приоритет, и, во-вторых, предусмотрена возможность либо игнорировать некоторые запросы на прерывания, либо отложить их обработку.

Важным является аспект, связанный с необходимостью восстановления ситуации, имевшей место до прерывания, иными словами, с возможностью продолжения прерванной программы от точки прерывания. Для этого нужно сохранить всю необходимую информацию о программе в точке прерывания.

И, наконец, прерывание, как сигнал об изменении состояния системы, должен нести информацию о конкретной причине, вызвавшей прерывание, чтобы можно было правильно отреагировать на него.

Рассмотрим классификацию прерываний, приведенную на рис. 2.7. Первый уровень — деление прерываний на

внешние и внутренние. Внешние прерывания вызываются причинами, лежащими вне ЭВМ, внутренние — причинами, обусловленными процессами, происходящими внутри ЭВМ. Следующие уровни рассматриваются только для внутренних причин. Внутри ЭВМ прерывания могут быть планируемые

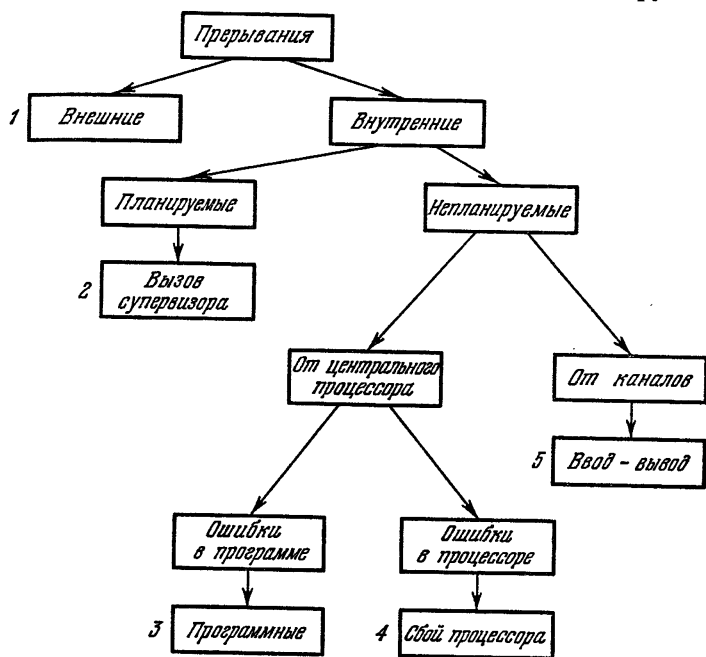


Рис. 2.7.

и непланируемые. К планируемому относится такое прерывание, для которого можно однозначно указать точку в программе, где оно произойдет, в противном случае прерывание непланируемое. Среди непланируемых в ЕС ЭВМ выделяются еще два подмножества прерываний: от каналов ввода-вывода и от центрального процессора.

При работе каналов фиксируются сигналы об окончании работы каналов по реализации операций ввода-вывода и др. Прерывания от центрального процессора, как правило, обусловлены ошибками. Ошибочные ситуации, в свою очередь, распадаются на две: ошибки в программе и ошибки в процессоре.

Приведенная на рис. 2.7 классификация определяет пять классов прерываний:

- 1) прерывания внешние;
- 2) прерывания при обращении к супервизору;
- 3) программные прерывания;
- 4) прерывания от схем контроля машины (сбой процессора);
- 5) прерывания при вводе-выводе.

Перед рассмотрением каждого в отдельности класса прерываний проанализируем процессы обработки прерываний центральным устройством управления ЭВМ.

Схема физической реализации процесса прерывания приведена на рис. 2.8. Как видно из рис. 2.8, часть оперативной памяти выделена специально для использования ее

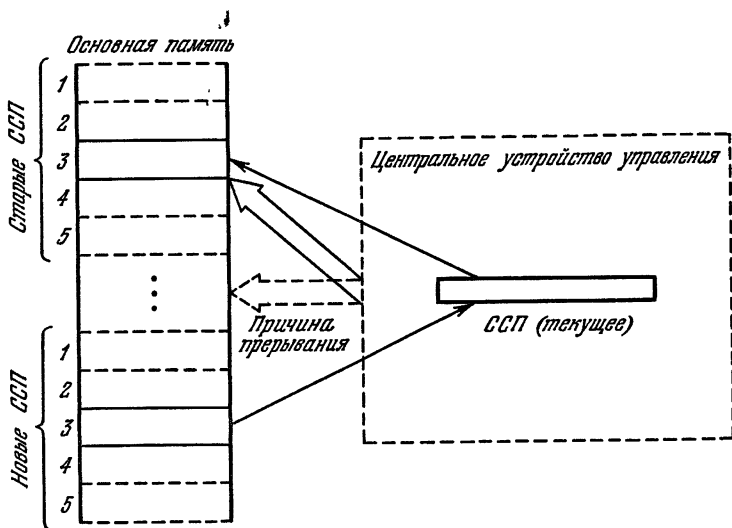


Рис. 2.8.

в системе прерываний. В этой части оперативной памяти, которая называется постоянно-распределенной областью, находятся управляющие слова, используемые при прерываниях.

Структура постоянно-распределенной области памяти приведена в табл. 2.2.

Однако вернемся к рассмотрению схемы обработки прерываний (см. рис. 2.8).

Центральное устройство управления (ЦУУ) содержит текущее слово состояния программы (ССП). При прерывании текущее ССП из ЦУУ автоматически записывается в область старого ССП, а затем замещается на новое

Таблица 2.2

Назначение постоянно-распределенной области памяти

Адрес	Класс прерывания	Назначение
0 8 16		используются в процедуре начального ввода
24 32 40 48 56	внешнее прерывание вызов супервизора программное прерывание сбой процессора ввод-вывод	ячейки старых ССП для соответствующих классов прерываний
64 72		используются в системе ввода-вывода
80		таймер
88 96 104 112 120	внешнее прерывание вызов супервизора программное прерывание сбой процессора ввод-вывод	ячейки новых ССП для соответствующих классов прерываний
128 . . . 300		используются при обработке машинных сбоев

из области нового ССП в зависимости от класса прерывания (всего 5 классов прерываний) (см. рис. 2.7).

Фактически в момент прерывания сохраняется вся информация для продолжения программы (в старом ССП), за исключением содержимого общих регистров, которое должно быть также сохранено для возобновления работы программы. Кроме этого, схемы машины фиксируют в старом ССП причину прерывания. Смена ССП практически означает передачу управления некоторой программе, называемой обработчиком прерываний, начальный адрес которой находится в новом ССП.

Состав ССП. В слове состояния программы условно можно выделить 3 группы данных, используемых в процессе управления ходом решения задач:

- 1) информация о состоянии процессора в момент прерывания;
- 2) информация о причине прерывания;
- 3) информация, необходимая для продолжения прерванной программы.

Рассмотрим подробно каждую из групп.

Информация о состоянии процессора в момент прерывания. Сначала обратимся к рис. 2.9, на котором изображена схема возможных состояний центрального процессора.

Состояния «стоп» и «работа» регулируются с помощью соответствующих кнопок на пульте управления ЭВМ. В состоянии «стоп» процес-

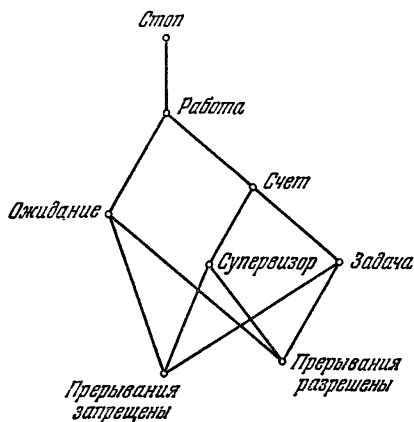


Рис. 2.9.

сор не выполняет никаких действий. Если нажата кнопка «пуск», то процессор может находиться в двух различных состояниях: «счет» и «ожидание».

В состоянии «счет» процессор выполняет команды, может принимать и обрабатывать прерывания, а также ведет отсчет времени с помощью интервального таймера.

В состоянии «ожидание» процессор про-

изводит все действия, характерные для состояния «счет», кроме выполнения команд. Процессор, находясь в состоянии «счет», может выполнять команды либо в состоянии «супервизор», либо в состоянии «задача».

В состоянии «супервизор» могут выполняться все команды, в состоянии «задача» не могут выполняться лишь привилегированные команды. К привилегированным, как отмечалось выше, относятся команды, с помощью которых можно управлять ходом выполнения программы на ЭВМ. Это команды управления вводом-выводом, загрузки ССП, смены масок системы, установки ключей в памяти ключей защиты (ПКЗ) и ряд других.

В условиях мультипрограммной работы управление осуществляется с помощью супервизора, который распределяет имеющиеся ресурсы между программами. Чтобы про-

граммы не имели возможности «захватывать» ресурсы «без ведома» супервизора, им отводится возможность работы только в состоянии «задача», супервизор же работает в таком состоянии, когда можно использовать привилегированные команды, позволяющие «захватывать» ресурсы.

Например, доступ к каналу можно получить только с помощью команд управления вводом-выводом, которые являются привилегированными. Если бы эти команды были непривилегированными, то любая задача могла бы получить доступ к любому каналу в любой момент времени, что сделало бы невозможным централизованное управление процессами ввода-вывода, и, соответственно, невозможным режим мультипрограммирования. Примерно то же самое можно сказать о применении команды установки ключей защиты памяти, которые предназначены для защиты программ от взаимного влияния.

Наконец, процессор, имеющий состояния «ожидание» и «счет» («супервизор» или «задача»), может находиться в таком состоянии, когда некоторые типы прерываний либо разрешены, либо запрещены (замаскированы) (см. рис. 2.9).

Теперь рассмотрим, какие поля в ССП (рис. 2.10) определяют нахождение процессора в конкретном состоянии.

Маски системы (биты 0—7) регулируют прерывания от внешних причин и от ввода-вывода. Биты 0—6 используются для маскировки (запрещения) прерывания от каналов ввода-вывода, а бит 7 — для маскировки внешних прерываний. Если один из битов маски системы равен 0, то соответствующий тип прерываний замаскирован, при этом сигнал прерываний не будет обработан. Этот сигнал вызовет прерывание, как только в качестве текущего будет загружено такое ССП, в котором этот бит будет равным 1.

Изменить содержимое битов системной маски можно с помощью специальной команды («установить маску системы») непосредственно в текущем ССП или путем загрузки в качестве текущего другого ССП.

Маска прерываний от схем контроля машины (бит 13) позволяет маскировать прерывание по классу машинного сбоя. Обычно значение этого бита равно 1 (прерывание разрешено), нулевое значение используется лишь при проведении на ЭВМ обслуживающим персоналом специальных работ.

Бит ожидания (бит 14) фиксирует нахождение процессора в состояниях «ожидание» (равен 1) и «счет» (равен 0). Бит 15 фиксирует нахождение процессора в состояниях

Отсутствие в поле кода прерывания детальной информации о конкретной причине прерывания по классу ввода-вывода и сбоем процессора означает, что схемы машины формируют и помещают указанную информацию в специальные дополнительные области памяти — байты 64—72 для 5 класса и байты 128—300 для 4 класса (см. рис. 2.8 и табл. 2.2).

Информация, необходимая для продолжения прерванной программы. Код длины выполняемой команды (биты 32—33) (см. рис. 2.10) содержит количество полуслов в последней выполнявшейся команде: 01 — для двухбайтовых команд, 10 — для четырехбайтовых и 11 — для шестибайтовых. Код 00 указывает, что длина команды не была определена. Значение этого поля очень часто используется программистом в процессе отладки программ для определения адреса и типа команды, предшествующей точке прерывания.

Биты 34 и 35 (см. рис. 2.10) содержат признак результата выполнения команды, который используется для передачи управления в программах. Признак результата может принимать 4 типичных значения для операций, выполняющих действия над числами:

- 00 — если результат равен 0;
- 01 — если результат меньше 0;
- 10 — если результат больше 0;
- 11 — при переполнении.

Смысл, приписываемый признаку результата зависит от типа выполняемой команды. Например, в операциях ввода-вывода признаки используются для других целей.

В команде перехода используется специальная маска, которая предназначена для управления переходом. Признак результата соответствует следующим состояниям битов маски:

Признак результата	Маска в команде перехода	Соответствие битов маски
00	1000	0-й бит соответствует 00
01	0100	1-й бит соответствует 01
10	0010	2-й бит соответствует 10
11	0001	3-й бит соответствует 11

Например, если нужно выполнить переход по признаку результата, равного 10, то маска в команде перехода

указывается равной 0010. Если маска равна 0111, то переход будет осуществлен лишь в том случае, если признак результата не равен 0.

Биты 36—39 (см. рис. 2.10) содержат маску программы. Они позволяют маскировать некоторые типы программных прерываний. Например, при программной организации операций повышенной точности в арифметике с фиксированной запятой необходимо игнорировать прерывание по переполнению. Изменить содержимое маски программы можно либо с помощью непривилегированной команды «установить маску программы», либо сменой текущего ССП.

Маскируются только следующие типы прерываний:

- 1) переполнение с фиксированной запятой;
- 2) десятичное переполнение;
- 3) переполнение порядка;
- 4) исчезновение значимости.

Биты 40—63 (см. рис. 2.10) содержат адрес следующей выполняемой команды. К этой же группе относится поле ключа защиты памяти (биты 8—11) (см. рис. 2.10), механизм использования которого будет рассмотрен позже.

Классы и приоритет прерываний. Как было рассмотрено ранее, в ЭВМ используется 5 классов прерываний (5 старых и 5 новых ССП). Прерывания от ввода-вывода вызываются каналами. Внешние прерывания вызываются внешними сигналами. К ним относятся прерывания: от интервального таймера, когда его значение становится равным 0, от нажатия кнопки прерывания на пульте и от шести дополнительных внешних сигналов, поступающих по шести линиям, которые могут быть подсоединены к процессору.

Прерывания от схем контроля машины возникают при машинных сбоях.

Программные прерывания фиксируют некоторые возможные ошибки в программе: переполнение, попытку деления на нуль, неверное размещение данных, обращение в недозволённую часть памяти и т. д. Список 15 программных прерываний приведен в табл. 2.3.

Прерывания при обращении к супервизору выполняет специальная команда с кодом SVC (вызов супервизора), производя следующие действия: вызывает прерывание, заносит операнд команды в поле кода прерывания старого ССП.

Прерывания всех классов происходят после обнаружения причины прерывания и завершения выполнения команды. Исключение составляют прерывания от схем контроля маши-

Перечень программных прерываний

Код прерывания	Причина прерывания
1	операция
2	привилегированная команда
3	команда EX
4	защита памяти
5	адресация
6	спецификация
7	ошибка в данных
8	переполнение с фиксированной запятой
9	деление с фиксированной запятой
10	десятичное переполнение
11	десятичное деление
12	переполнение порядка
13	исчезновение порядка
14	ошибка значимости
15	деление с плавающей запятой

ны, которые происходят немедленно после появления ошибки, не дожидаясь завершения выполнения текущей команды.

Вообще говоря, в ЭВМ может иметь место ситуация, когда одновременно приходят запросы по 4 классам прерывания (прерывания «вызов супервизора» и программное одновременно возникнуть не могут). В этом случае должен быть определен порядок приема и обработки прерываний. Если разрешены прерывания по всем классам, то принимаются они в порядке установления приоритетов. Первым исполняется прерывание по сбою процессора, вторым — по классу «вызов супервизора» или программное, третьим — внешнее и, наконец, последним — прерывание по классу ввода-вывода. Причем смена ССП в момент прерываний происходит сразу друг за другом по всем указанным классам, без исполнения каких-либо команд до тех пор, пока не будут приняты все прерывания, запрос на выполнение которых был обнаружен.

Обработка же прерываний, принятых процессором, фактически производится в обратном порядке.

При такой схеме исполнения прерываний возможны случаи потери информации о причинах некоторых прерываний. Например, если по классу прерываний от ввода-вывода придут сразу два сигнала прерывания, то причина одного из них будет потеряна, так как запись текущего ССП происходит в одно и то же место, т. е. старое ССП одного из прерываний будет потеряно. Чтобы подобные

случаи можно было регулировать программным путем, в ССП имеются специальные разряды — маски (см. рис. 2.10), позволяющие управлять, по существу, приоритетом прерываний на уровне специальной программы обработчика.

Но некоторыми прерываниями управлять невозможно. К ним относятся прерывания по обращению к супервизору и 11 немаскируемых типов программных прерываний. Прерывания по остальным классам маскируются.

Средства защиты памяти. В ЭВМ имеется специальный аппарат защиты памяти. Необходимость защиты памяти обуславливается мультипрограммным режимом работы. Весьма велика вероятность, особенно в режиме отладки, что одна программа может испортить другую, в связи с этим и предусмотрена аппаратная защита памяти. Память разбивается на группы по 2К ($K = 1024$ байтов). Каждой группе соответствует свой четырехбитовый ключ памяти, находящийся в специальном запоминающем устройстве — памяти ключей защиты (ПКЗ).

В каждую ячейку ПКЗ, представляющую 4-разрядный ключ памяти, можно записать одно из 16 значений от 0 до 15. Таким образом, вся оперативная память (ОП) может быть разбита на 16 частей, каждой из которых поставлена в соответствие группа ячеек из ПКЗ с одинаковыми значениями ключей.

В ССП центрального процессора находится ключ защиты памяти программы (см. рис. 2.10), который регулирует процессы обращения в оперативную память. Если при обращении к разделу ОП ключ защиты памяти, связанный с этим разделом, не соответствует находящемуся в ССП ключу защиты программы, работающей в данный момент, то происходит программное прерывание по защите памяти. Проверка соответствия ключей защиты памяти производится аппаратно. Программа с нулевым ключом защиты является привилегированной и располагается в области супервизора (из программы с нулевым ключом защиты можно обращаться в любую область памяти). Ключ защиты памяти можно установить с помощью специальных команд, работающих в режиме супервизора.

2.4. Каналы и аппаратная система ввода-вывода

Общие сведения. В вычислительных машинах серии ЕС ЭВМ применяется сложная, но достаточно стройная система ввода-вывода, обеспечивающая широкие возможности. На ЭВМ I и II поколений ввод-вывод был сравни-

тельно прост, и команды ввода-вывода являлись составной частью системы команд ЭВМ. В ЕС ЭВМ ситуация иная: в состав команд процессора входят всего 4 команды, позволяющие управлять такими операциями, как запуск и останов операций вывода-ввода и некоторые другие, но с помощью этих команд запрограммировать операцию ввода-вывода невозможно. Для их программирования предусмотрены другие команды — команды канала, который обеспечивает выполнение операций ввода-вывода по так называемым канальным программам.

Это вызвано потребностями практики. Известно, что в среднем при работе программ 40 % времени занимает собственно обработка и 60 % — ввод-вывод. В экономических задачах ввод-вывод занимает еще большее место. ЭВМ I и II поколений были неудобны потому, что команды ввода-вывода выполнялись самим процессором. Отсюда мала и общая производительность ЭВМ (число решенных задач в единицу времени) в силу невозможности совмещения операций ввода-вывода с вычислениями. Рассмотрим, к чему приводит отсутствие такого совмещения.

Известно, что одна перфокарта вводится в среднем со скоростью 1 мс. Самые быстрые устройства затрачивают 700 мкс на ввод 1 перфокарты, а обработку ее машина производит примерно в 1000 раз быстрее. КПД использования вычислительной машины в этом случае низок. Поэтому, чтобы повысить КПД оборудования ЭВМ, нужно совместить ввод-вывод и счет, т. е. иметь возможность параллельно с вводом-выводом производить вычисления. Кроме этого, естественно, хорошо бы иметь машину, выполняющую несколько операций ввода-вывода одновременно. Идея эта реализована с помощью введения в состав ЭВМ специализированной вычислительной машины — канала.

На малых моделях ЕС ЭВМ обычно имеется три канала: один канал мультиплексный, остальные — селекторные (в старших моделях, как отмечалось, каналов может быть больше). Канал (К) (см. рис. 2.11) имеет самостоятельный доступ в основную память (ОП). Он, как и центральный процессор (ЦП), работает по своей программе, называемой канальной программой, которая находится, как и основная, в оперативной памяти.

Рассмотрим, что может дать такое усовершенствование структуры ЭВМ в аспекте программной совместимости. На каждой модели существует различная конфигурация (набор) внешних устройств, и так как подсоединение их

стандартное, то возможно большое разнообразие наборов. В ЭВМ I и II поколений программа зависела от набора внешних устройств, что не позволяло без переделок программы выполнять ее на ЭВМ того же типа, но с иной конфигурацией внешних устройств. В ЕС ЭВМ решение

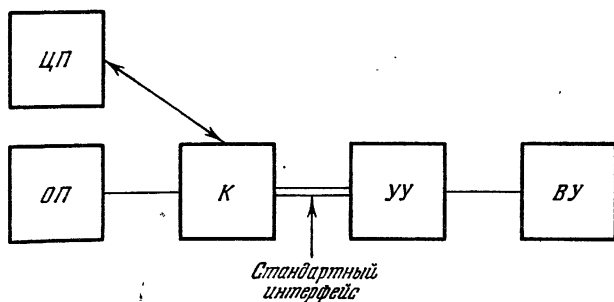


Рис. 2.11.

проблемы независимости программ от конфигурации внешних устройств и даже типов устройств осуществлено благодаря применению специальной аппаратной системы ввода-вывода и применению операционной системы (ОС). Система ввода-вывода представляет собой набор технических и программных средств по реализации операций ввода-вывода на аппаратном уровне.

К техническим средствам относятся каналы (К), стандартный интерфейс, внешние устройства (ВУ) с устройствами управления (УУ) (см. рис. 2.11), к программным — специальные команды процессора, позволяющие управлять каналами, и команды каналов, являющиеся командами ввода-вывода.

Операционная система в общем случае представляет собой комплекс программ, обеспечивающих автоматическое управление всеми процессами прохождения задач через ЭВМ и располагающих средствами для написания программ на языках, мало зависящих от языка ЭВМ.

Обсуждение роли операционной системы в реализации операций ввода-вывода будет приведено позже.

В системе ввода-вывода на машинном уровне наиболее характерной особенностью является та, что команды ввода-вывода в основной программе писать не нужно, а вместо них пишутся команды управления каналом, который работает по своей программе. При таком способе записи операций ввода-вывода в основной программе практически нет необходимости исправлять основную программу при

переходе с одного устройства на другое. В моделях ЕС ЭВМ имеется 4 команды управления каналом:

- 1) начать ввод-вывод (SIO);
- 2) опросить ввод-вывод (TIO);
- 3) остановить ввод-вывод (HIO);
- 4) опросить канал (TCH).

Если работа ведется на ЭВМ без операционной системы, то приходится писать (составлять) программы канала. Это увеличивает трудоемкость составления программ, так как программист пишет 2 программы: основную программу и программу канала. Но это тоже еще не все. Если необходимо работать с различными устройствами, то все каналные программы для всех устройств должны быть составлены заранее и предусмотрены средства, обеспечивающие достаточно простой переход с одного устройства на другое.

Команды канала для работы с различными устройствами унифицированы. Они являются основными (базовыми) для всех внешних устройств. Каждое устройство располагает своим набором команд, с помощью которых осуществляется управление его работой.

Естественно, что составление каналных программ требует от программиста определенного искусства и хорошего знания специфики внешних устройств, с которыми ему приходится работать. Для того чтобы основная программа стала независимой от типа внешних устройств при использовании операций ввода-вывода, необходимо освободить программиста от составления каналных программ, обеспечив их автоматическое подключение.

В рамках ЕС ЭВМ в состав операционной системы входят все необходимые каналные программы. ОС же обеспечивает переход с одного устройства на другое. Например, в ОС ЕС переход с магнитной ленты на магнитный диск достаточно прост и осуществляется на этапе запуска программы. Все остальное делает ОС: использует нужную каналную программу, назначает устройства, а иногда и носитель. Далее будут подробно рассмотрены физические основы ввода-вывода.

Каналы. Известно, что в ЕС ЭВМ имеются мультиплексный и селекторный каналы. Чем они отличаются?

Ранее отмечалось, что к каналу обычно подключается несколько устройств, число которых колеблется от 1 до 255. Рассмотрим два режима передачи данных, используемые каналами ЕС ЭВМ.

На рис. 2.12 схематично изображены канал и подключенные к нему 4 устройства (1, 2, 3, 4).

В монопольном режиме канал обслуживает только одно устройство. Для этого режима характерно то, что скорость передачи данных довольно близка к максимальной скорости канала.

Если же устройства достаточно медленны, то канал может опрашивать каждое внешнее устройство по очереди,

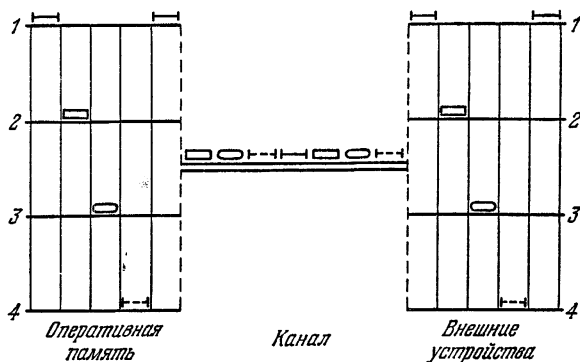


Рис. 2.12,

передавая подготовленные байты и переходя от одного устройства к другому. Причем он будет «пробегать» по всем устройствам настолько быстро, что каждое внешнее устройство будет работать так, как если бы оно одно обслуживалось каналом. Такой режим работы называют мультиплексным (рис. 2.12).

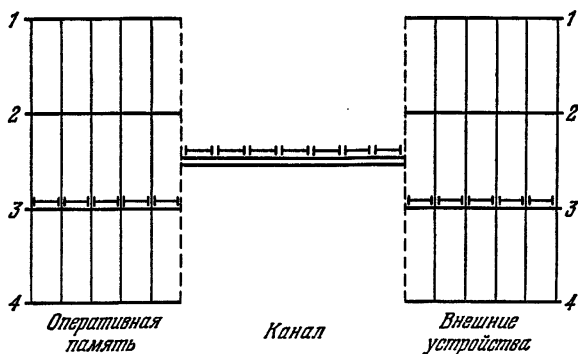


Рис. 2.13.

Селекторный канал (рис. 2.13) одновременно может выполнять операцию ввода-вывода только с одним устройством. В этом случае говорят, что он имеет 1 подканал.

Мультиплексный канал в любой модели может иметь более 100 подканалов, т. е. мультиплексный канал может одновременно выполнять операции ввода-вывода с более чем 100 медленно действующими устройствами.

Операции канала. Канал выполняет функции передачи данных от внешнего устройства в оперативную память (операция чтения) и наоборот (операция записи), не производя при этом никаких функциональных и прочих преобразований передаваемой информации. Поэтому для канала предусмотрена только одна форма представления данных — побайтовая. Передача данных осуществляется определенными последовательностями байтов — блоками.

Рассмотрим более детально основные операции канала как специализированного процессора. Как отмечалось, имеется шесть операций, шестнадцатеричные коды которых приведены ниже:

- 01 — запись (передача из основной памяти);
- 02 — чтение (передача в основную память);
- 03 — управление устройством ввода-вывода;
- 04 — уточнить состояние;
- 08 — переход в канале (безусловная передача управления);
- 0С — чтение в обратном направлении (для МЛ).

Заметим, что код 00 — НЕДОПУСТИМ. Наличие его в программе канала воспринимается как ошибка.

Первые 4 бита кода операции используются обычно для модификации команд применительно к конкретным устройствам. Особенно много модификаций у команды управления устройствами. При получении команды «запись» или «чтение» канал производит передачу данных из памяти или в память. Команда «управление» — это особый случай команды записи. По этой команде управляющая информация посылается во внешнее устройство без записи данных на носитель. С помощью этой информации могут выполняться, например, такие действия, как продвижение бумаги на АЦПУ, перемотка магнитной ленты и т. д. Частный случай команды «управление» (03) интерпретируется как пустая команда (нет операции). В таком виде эта команда используется в цепочках команд для облегчения режима работы канала. Команда «уточнить состояние» является особым случаем команды «чтение», по этой команде вместо данных в память считываются специальные байты уточненного состояния.

Команда «переход в канале» производит выбор команды канала, адрес которой указан в поле адреса текущей команды.

Она является командой безусловного перехода в программе канала. В ряде случаев она играет роль команды условного перехода.

Команда «чтение в обратном направлении» используется только для тех устройств, в которых можно читать данные с конца, т. е. в порядке убывания адресов. Например, считывание блоков с магнитной ленты, движущейся в обратном направлении.

Формат команды канала. Команда канала имеет длину 8 байтов и называется, в отличие от команды ЦП, командным словом канала (КСК). Формат и структура КСК приведены на рис. 2.14.

Команды канала построены по одноадресному принципу. В общем случае команда должна указывать 2 адреса: адрес приемника и адрес передатчика информации. В качестве приемника и передатчика данных выступают попеременно, в зависимости от типа выполняемой операции (запись или чтение), то внешнее устройство, то некоторая область оперативной памяти, называемая буфером. Адрес буфера состоит из двух частей: адреса первого байта и числа, обозначающего общее количество байтов, отведенных для данного буфера. Адрес ВУ, как увидим позже, представляет собой целое число. Учитывая, что в процессе запуска канала, который производится центральным процессором, адрес ВУ должен быть обязательно указан, в формате КСК оставлен только составной адрес буфера.

Рассмотрим формат КСК подробнее.

Код операции (КОП) занимает самый левый (нулевой) байт. Следующие три байта (адрес данных) содержат адрес первого байта данных. Байт 4 содержит признаки. Байт 5 игнорируется и может содержать любую информацию. Последние два байта задают число байтов, предназначенных для обмена. В канале содержимое счетчика уменьшается на 1 после передачи одного байта. Передача данных заканчивается тогда, когда или значение счетчика станет равным 0, или будет обнаружен конец физического блока.

Опишем байт признаков подробнее. В разрядах с 32 по 36 КСК располагаются признаки, которые позволяют управлять дополнительными функциями каналов. Рассмотрим каждый признак отдельно.

Бит 32 используется в специальном режиме, когда информация из одного блока может записываться в различные места основной памяти (цепочка данных). Наличие единицы в бите 32 означает, что следующая команда в канальной программе используется особым образом.

КСК	КОП		адрес данных		признаки		000		-		счетчик
0	7	18	31	32	36		40	47	48	63	

Признаки:

32 — цепочка данных

33 — цепочка команд

34 — блокировка неправильной длины

35 — блокировка записи в память

36 — УПП

Состояние

ССК	Ключ		0000		адрес КСК		устройство		канал		счетчик	
0	3	4	7	8	31	32	39	40	47	48	63	

АСК	Ключ		0000		адрес команды	
0			7		31	

32 — внимание

33 — модификатор
состояния

34 — УУ кончилось

35 — занято

36 — канал кончит

37 — УВВ кончилось

38 — ошибка в УВВ

39 — особый случай
в УВВ

40 — УПП

41 — неправильная длина

42 — ошибка в программе

43 — нарушение защиты

44 — ошибка в данных

45 — ошибка в канале

46 — ошибка в интерфейсе

47 — ошибка в цепочке

Формат команд (SIO, TIO, TCH, HIO)

КОП		B1		D	
0	7	15	19	20	31

Полученный адрес даст

		К		У		В	
0	20	21	24			31	

Байт уточненного состояния

0 — команда отвергнута

1 — требуется вмешательство оператора

2 — ошибка в выходной шине (четность)

3 — ошибка оборудования

4 — ошибка данных

5 — переполнение

Рис. 2.14.

А именно: канал извлекает команду в момент передачи данных, проверку КОП не производит, а использует сразу адрес и счетчик. Используя аппарат цепочки данных, можно, например, считывать содержимое одной перфокарты в разные места основной памяти.

Бит 33 используется в так называемых цепочках команд. В сущности, значение этого бита указывает каналу, когда нужно кончать работу. Если бит 33 равен 1, то выбирается следующая команда, если бит 33 равен 0, то канал кончает работу.

Бит 34 служит для указания признака блокировки ошибки при несовпадении длины блока на носителе и в команде. При запуске каналом операции чтения происходит чтение блока данных. Если признак блокировки ошибки не задан, то при несовпадении длины блока на носителе с длиной, указанной в команде, происходит прерывание с фиксацией ошибки. Но иногда приходится считывать блоки, длина которых заранее не известна. В этом случае необходимо использовать бит 34.

Бит 35 предназначен для указания блокировки записи данных в память. Этот признак можно использовать для фиктивного чтения данных. При этом работает канал, устройство, т. е. весь тракт ввода-вывода, но записи в память не происходит. Таким образом, используя бит 35, можно осуществить, например, проверку записанной информации путем контрольного чтения.

Бит 36 используется для управляемого программного прерывания (УПП) при динамическом изменении канальных программ. При наличии единицы в бите 36 после выборки команды и начала операции происходит прерывание по классу ввода-вывода. Программа, которая получает управление в этом случае, может перестраивать или достраивать канальную программу в момент ее выполнения каналом. Признак используется в ОС ЕС и не используется в ДОС ЕС.

В разрядах 37—39 всегда должны быть нули. В противном случае это рассматривается как ошибка в программе канала.

Пример канальной программы:

0200160000000050₍₁₆₎

Программа состоит из одной команды и означает: считать в память, начиная с адреса 1600 (шестнадцатеричное число), 80 байтов (50 в шестнадцатеричном представлении); все признаки равны нулю.

Если нужно разместить содержимое одной перфокарты в различные участки основной памяти, то это можно сделать, используя следующую программу:

02	002000	8000	0020
02	000100	8000	0010
02	000B00	0000	0020

Первая команда программы считывает 32 байта (20 в шестнадцатеричном виде) в память, начиная с 2000 байта, вторая команда производит считывание 16 байтов в память, начиная с адреса 100, по третьей команде происходит считывание 32 байтов в память, начиная с адреса B00. Цепочка данных задается кодом 80 в пятом байте первой и второй команд. Режим цепочки данных предполагает достаточно высокую скорость канала. При считывании перфокарты, например, канал должен успеть выбрать очередную команду и после этого направить данные в другое место. Что касается перфокарты, то она считывается достаточно медленно, поэтому канал успевает реализовать режим цепочки данных. Но в случае работы с быстрыми устройствами (МЛ, МД и т. д.), где считывание или запись происходит быстро, канал, в принципе, может не успеть сменить адрес и счетчик. Если канал не успел обработать цепочку данных, то происходит прерывание.

В отдельных случаях можно облегчить режим работы канала, используя бит блокировки записи в основную память, но этот прием возможен только в режиме считывания и в том случае, когда нужно читать не целиком блок, а только его часть.

Рассмотрим применение цепочки команд на следующем примере.

02	001800	4000	0100 — чтение
01	002000	0000	0100 — запись

Приведенная программа канала обеспечивает считывание 256 байтов в основную память (адрес 1800), а затем запись 256 байтов на внешнее устройство из основной памяти (адрес 2000). Цепочка команд задается кодом 40 в пятом байте первой команды.

Подводя итоги обсуждения использования цепочки данных (ЦД) и цепочки команд (ЦК), отметим, что основное отличие ЦД от ЦК состоит в том, что в режиме ЦК канал выполняет новую команду (например, считывает новый блок), а в режиме ЦД продолжается выполнение

старой команды (например, продолжается чтение текущего блока).

Режим ЦД дает возможность:

- 1) считать один блок в различные места памяти;
- 2) считать часть блока, устанавливая в некоторых командах блокировку записи;
- 3) записывать на внешнее устройство блоки, размер которых превышает число 64К.

Приведенные примеры по использованию признаков ЦД и ЦК позволяют отметить важные особенности как процесса составления канальных программ, так и процесса их реализации. В отличие от основных программ, в которых команды могут записываться в произвольном порядке и ход выполнения программы не зависит от быстродействия устройств, ее реализующих, программы канала составляются с учетом определенной последовательности записи команд. Кроме этого, ход их выполнения зависит от быстродействия устройств, участвующих в реализации канальных программ. Таким образом, в процессе написания программ для канала указанные моменты необходимо учитывать.

К примеру, цепочка команд, состоящая из двух команд «чтение» и «запись», является некорректной для устройства ввода с перфокарт, поскольку это устройство не может производить операции записи. Практически для любого внешнего устройства можно указать подобные ограничения. С точки зрения времени выполнения для любой канальной программы всегда можно указать минимальную пропускную способность канала, ниже которой выполнение ее становится невозможным.

Таким образом, к важнейшим особенностям программирования канальных программ относятся, во-первых, необходимость учета ограничений, накладываемых на последовательность написания команд канала, и, во-вторых, ограничений на время выполнения команд канальной программы.

Канальные программы, написанные программистом (при работе на физическом уровне), не проверяются аппаратурой перед выполнением. Поэтому все ошибочные ситуации должен обрабатывать сам составитель. Здесь имеются в виду различного рода ошибки как в канальных программах, так и в аппаратуре, выполняющей канальные программы, многие из которых могут быть зафиксированы аппаратными средствами и к которым программист может иметь доступ. Если же программист работает с помощью программ операционной системы, то он освобождается, как уже рассматривалось ранее, от составления канальных программ

и от самостоятельной обработки многих ошибок. Однако это не значит, что он будет полностью освобожден от выяснения причин, по которым запрограммированная им операция ввода-вывода не совершилась.

Слово состояния канала (ССК). Аналогом ССП центрального процессора у канала является ССК, которое используется следующим образом.

После окончания работы канал вызывает прерывание центрального процессора, если оно, конечно, не замаскировано. В противном случае прерывание запоминается в канале (ставится в очередь) и будет обработано немедленно после снятия запрета согласно установленному приоритету. Физически в момент прерывания ввода-вывода, кроме смены ССП, происходит запись в 64-ю ячейку слова состояния канала, в котором находится информация о состоянии операции ввода-вывода в момент прерывания. В частности, в ССК фиксируется информация о нормальном завершении операции и о всех сбоях, фиксируемых аппаратно, т. е. информация о причине прерывания. Структура ССК приведена на рис. 2.14.

Поле ключа содержит ключ защиты выполняемой программы канала. В поле адреса КСК записывается адрес последней выполненной команды канала, увеличенный на 8. В поле счетчика фиксируется разность между содержимым счетчика в последней выполненной команде канала и длиной блока на носителе. С точки зрения обработки прерываний наиболее важное значение имеют два байта в ССК: байт состояния устройства и байт состояния канала. Рассмотрим эти байты подробно.

Байт состояния устройства занимает биты с 32-го по 39-й.

Бит 32 — признак внимания. Прерывание вызвано специальным сигналом «внимание». Например, нажата специальная кнопка на внешнем устройстве непосредственной связи с оператором (пишущая машинка, дисплей и т. д.).

Бит 33 — признак модификатора состояния. Он используется вместе с другими разрядами байта состояния устройства только для групповых устройств управления (когда несколько устройств работают с одним устройством управления).

Бит 34 — признак окончания работы устройства управления. Этот сигнал используется только для групповых УУ.

Бит 35 — признак занятости устройства. Например, идет печать информации из буфера.

Бит 36 — признак окончания работы канала. Канал не нужен устройству, операция выполнена.

Бит 37 — признак окончания работы устройства ввода-вывода. Этот бит устанавливается в единицу, если устройство кончило свою операцию и готово к выполнению следующей.

Для одной операции ввода-вывода устанавливаются два сигнала прерывания: «канал кончил» и «устройство кончило». Для медленных устройств оба сигнала, как правило, приходят в разные моменты времени, для быстрых — одновременно.

Бит 38 — признак ошибки в устройстве. Может прийти одновременно с признаком в 37-м разряде.

Бит 39 — признак особого случая в устройстве. Например, конец файла. Файл на магнитной ленте ограничивается маркерами, и когда производится попытка читать маркер, то информация не передается, а возникает прерывание и 39-й разряд ССК устанавливается в единицу, что фактически означает конец файла. Конец массива перфокарт (т. е. считывание перфокарты из пустого кармана) также фиксируется установкой 39-го разряда в единичное состояние. При считывании с магнитного диска блока нулевой длины 39-й разряд также устанавливается в единицу. Такая ситуация характерна для всех типов устройств ЕС ЭВМ, что означает аппаратную фиксацию конца массива. Дальнейшая обработка этой ситуации производится операционной системой.

Байт состояния канала занимает биты 40—47.

Бит 40 — признак управляемого программно прерывания. Если прерывание произошло, то признак в команде канала установлен в 1.

Бит 41 — признак неправильной длины блока. Счетчик в команде не совпадает с фактической длиной блока и не задан признак блокировки неправильной длины.

Бит 42 — признак ошибки в программе канала. Например, используется недопустимый код операции.

Бит 43 — признак нарушения защиты памяти. Он возникает, когда ключ в адресном слове канала не совпадает с ключом защиты памяти.

Бит 44 — признак ошибки в данных. Например, четность, свертка и т. д.

Бит 45 — признак ошибки в канале. Ошибка фиксируется схемами контроля канала.

Бит 46 — признак ошибки в интерфейсе. Ошибка фиксируется схемами контроля интерфейса.

Бит 47 — признак ошибки в цепочке. Например, использована недопустимая последовательность команд (2 раза подряд следует команда чтения без поиска информации на магнитном диске). Много ошибочных и особых ситуаций, фиксируемых при прерывании, можно исправить программным путем.

Динамика выполнения операций обмена.

Процедура начала операций обмена. Канал самостоятельно начать работу не может, его должен запустить в работу ЦП по специальной команде. Для того чтобы канал начал функционировать самостоятельно, ему необходимо сообщить, где находится канальная программа и с каким устройством ему нужно работать.

Начальный адрес канальной программы в момент запуска любого канала должен находиться в 72-й ячейке постоянно распределенной области памяти. В 72-й ячейке находится адресное слово канала (АСК) (см. рис. 2.14). Адрес, находящийся в АСК, должен быть выставлен на границу двойного слова, поскольку длина команды канала равна 8 байтам. Каналы запускаются по очереди, одновременно запустить несколько каналов нельзя. После запуска каналы могут работать одновременно и совершенно самостоятельно, в конце работы канала выдается сигнал прерывания.

Каждое устройство в машине имеет свой физический адрес, который кодируется тремя шестнадцатеричными цифрами:

- 1) номер канала (К);
- 2) номер устройства управления (У);
- 3) номер устройства ввода-вывода (В).

Номера каналов принимают значения 0 — для мультиплексного канала и 1—6 — для селекторных каналов.

Если номер канала в команде будет больше 6, то возникает прерывание. В качестве номера устройства управления, а также номера устройства ввода-вывода может использоваться любая шестнадцатеричная цифра от 0 до F.

Отметим, что для мультиплексного канала «У» и «В» вместе обозначают номер подканала.

Приведем примеры физических адресов типовых устройств:

190 — магнитный диск,

00A, 00C — устройство ввода с перфокарт,

00F, 00E — принтер (АЦПУ),

00B, 00D — устройство вывода на перфокарты.

Процессором для управления работой канала используются четыре команды формата SI:

SIO — начать ввод-вывод,
NIO — остановить ввод-вывод,
TIO — опросить ввод-вывод,
TCH — опросить канал.

Формат этих команд приведен на рис. 2.14.

После вычисления адреса при выполнении команд управления каналом ($D + (B1)$) (см. рис. 2.14) учитываются только последние 16 битов, в которых указаны номер канала, номер устройства управления и номер устройства. Эти 16 битов передаются в канал, канал обращается в 72-ю ячейку за адресом канальной программы и начинает работу.

Запуск канала осуществляется с помощью команды SIO, которая выполняет наиболее важные функции в системе ввода-вывода. При получении кода операции SIO процессор определяет адрес устройства, передает этот адрес в канал и получает информацию о том, как закончилась попытка запустить операцию ввода-вывода, в виде значения признака результата в ССП. Возможные признаки результата (с двоичным эквивалентом) для команды SIO приведены ниже:

0(00) — операция ввода-вывода началась нормально,
3(11) — устройство не доступно,
2(10) — канал занят,
1(01) — все остальные причины.

Установленный в ССП признак результата можно использовать для анализа окончания процесса запуска операции ввода-вывода. Если признак результата равен 0, то основную программу можно продолжать. Если признак результата не равен 0, то необходимо определить причину, по которой не произошло запуска операции. Если признак результата равен 2, то канал занят и нужно «подождать», пока он освободится, а затем снова попытаться запустить операцию. Если признак результата равен 3, то это означает, что устройство выключено и его необходимо включить. Во всех остальных случаях устанавливается признак результата, равный 1. Информация о конкретных причинах заносится в слово состояния канала (ССК), которое располагается в 64-й ячейке памяти в виде двух байтов, характеризующих состояния ВУ и канала.

По ССК можно определить, что произошло, а иногда точно установить причину окончания операции обмена. ССК играет решающую роль также при обработке прерываний от ввода-вывода. Обсуждение структуры и назначения ССК будет приведено ниже. Остальные команды управления каналом вырабатывают следующие коды условий.

Команда НЮ останавливает операцию ввода-вывода и вырабатывает признаки:

- 3 — устройство недоступно,
- 2 — канал занят,
- 1 — ССК установлено,
- 0 — канал свободен.

Команда ТСН используется редко, применяется для опроса канала и вырабатывает признаки:

- 3 — канал недоступен,
- 2 — канал занят,
- 1 — имеются необработанные прерывания,
- 0 — канал свободен.

Команда ТЮ введена для более быстрой обработки прерываний, она вырабатывает признаки:

- 3 — устройство недоступно,
- 2 — канал занят,
- 1 — имеются необработанные прерывания,
- 0 — необработанных прерываний нет.

Процедура окончания операции обмена. Основным средством автоматического окончания операции ввода-вывода является прерывание по классу ввода-вывода. Завершая операцию, канал формирует сигнал прерывания, посылая его в центральный процессор, и одновременно подготавливает ССК для помещения его в оперативную память. По прерыванию управление должна получить специальная программа, называемая обработчиком прерываний. Основной задачей этой программы является выяснение причины прерывания и выработка соответствующего решения. Информация о причинах прерывания, за небольшим исключением, находится в ССК.

В общем случае прерывания по классу ввода-вывода могут быть вызваны следующими причинами:

- 1) нормальное завершение операции;
- 2) внешние причины (сигнал «внимание»);
- 3) планируемые прерывания (программно-управляемые);
- 4) непланируемые (сбои и некоторые исключительные состояния).

Самыми нежелательными являются прерывания, фиксирующие сбойные ситуации — ошибки в аппаратуре и др. Все остальные причины были затронуты в той или иной мере в процессе детального анализа содержимого ССК. Поэтому основное внимание уделим рассмотрению сбойных ситуаций.

Как отмечалось ранее, факт ошибки при выполнении операции обмена фиксируется 38-м разрядом ССК (ошибка в устройстве).

При обнаружении сигнала «ошибка в устройстве» дополнительно (программным путем) можно получить информацию о состоянии отдельных устройств, которая помещается в основную память по указанному адресу. Эта информация может размещаться в нескольких байтах уточненного состояния, количество которых колеблется от 1 (для устройства ввода с перфокарт) до 6 (для внешних запоминающих устройств с прямым доступом). Для устройств управления внешними устройствами всегда имеется, по крайней мере, 1 байт уточненного состояния устройства.

Бит «ошибка в устройстве» ССК устанавливается в единичное состояние при наличии хотя бы одной единицы в байте уточненного состояния для устройства управления. Заметим, что при этом байты уточненного состояния в оперативную память не передаются. Для получения их в основной памяти нужно в программе канала записать специальную команду «уточнить состояние».

Некоторые ошибочные ситуации, зафиксированные в байтах уточненного состояния, легко исправляются программным путем. Например, при поиске записи на дорожке дискового накопителя может встретиться ситуация, когда нужной записи нет, в этом случае бит «конец дорожки» устанавливается в единичное состояние. Для выхода из этой ситуации можно продолжить поиск на следующей дорожке, программно изменив адрес в канальной программе, и запустить канал повторно.

Байты уточненного состояния устройства. Итак, помимо основного байта состояния устройства, засылаемого в момент прерывания в ССК, каждое УВВ имеет свои байты уточненного состояния, в которых находится информация об ошибках, обнаруживаемых устройством. В байте основного состояния устройства фиксируется лишь факт: «произошла ошибка». Путем анализа байтов уточненного состояния можно установить конкретную причину многих ошибок.

Рассмотрим подробно структуру первого байта уточненного состояния, являющегося для всех устройств унифицированным и обязательным.

Первый байт для всех устройств одинаков (см. рис. 2.14), остальные байты (если они есть) зависят от типа устройства, которое их формирует, и отражают его особенности и специфику. В первом байте используется шесть битов, каждый из которых устанавливается в единицу при следующих условиях:

0 — неправильный код операции (команда отвергнута);

1 — требуется вмешательство оператора (нужно нажать какую-либо кнопку или выполнить некоторые другие действия);

2 — ошибка в выходной шине (контроль четности);

3 — ошибка в оборудовании;

4 — ошибка в данных (например, недопустимый код на перфокарте). Этот бит фиксирует ошибки на внешнем носителе, а не в устройстве;

5 — переполнение (это явление связано с перегрузкой канала, канал «не успел» передать данные). Это означает, что общая скорость операций ввода-вывода превышает пропускную способность канала.

Как видно из рассмотрения, в первом байте фиксируются ошибки в основных узловых элементах ввода-вывода: код операции, вводимая информация, оборудование, тракт передачи, действия оператора.

Подводя итоги, можно сказать следующее. Все операции ввода-вывода выполняются через специализированную вычислительную машину — канал. Канал работает по собственной программе (канальной программе), которая находится в основной памяти вместе с другими программами. Канал может выполнять только шесть операций: «читать», «писать», «управлять устройством», «переход в канале», «читать в обратном направлении», «уточнить состояние». Канальная программа состоит из цепочки команд канала, каждая команда канала фиксирована по длине и структуре.

Важнейшим понятием в системе ввода-вывода является понятие операции ввода-вывода. Под операцией ввода-вывода понимается выполнение каналом одной канальной программы. Запускается операция ввода-вывода из основной программы процессора с помощью команды «начать ввод-вывод». Эта команда входит в состав команд процессора. Важно отметить, что эта команда, как и все остальные, связанные с управлением работой канала, является

привилегированной и выполняется только в состоянии «супервизор».

На машине с операционной системой этими командами пользуется только супервизор, программисту они недоступны.

При запуске канала пользователю сообщается номер устройства (в команде SIO) и начальный адрес канальной программы (в 72-й ячейке). Канал вырабатывает признак результата, сигнализирующий о процессе запуска канала. Затем канал начинает функционировать самостоятельно. По окончании операции ввода-вывода или в случае необычных ситуаций канал прерывает работу процессора и одновременно с этим устанавливает слово состояния канала (ССК) в 64-ю ячейку памяти.

Программа, обрабатывающая прерывание ввода-вывода, анализирует байты слова состояния канала и определяет причину прерывания.

Если в байте основного состояния зафиксирована ошибка в устройстве, то необходимо проанализировать причину ошибки. Причины фиксируются в байтах уточненного состояния, которые к моменту прерывания находятся в устройстве. Байты уточненного состояния считываются в память каналом по специальной команде «уточнить состояние». После соответствующего анализа байтов уточненного состояния можно принять решение по ликвидации ошибочной ситуации. Таковы основные моменты аппаратной системы ввода-вывода.

2.5. Программирование операций ввода-вывода без использования средств операционной системы

Последовательность операций программирования. Методика составления канальных программ рассматриваться здесь не будет, рассмотрим лишь последовательность программирования операции ввода-вывода в основной программе. Сразу оговоримся, что будем рассматривать программирование операции ввода-вывода на машинном уровне, когда в нашем распоряжении имеется ЭВМ и нет операционной системы.

Поэтому этим материалом нельзя пользоваться при программировании в рамках операционной системы. Он предназначен лишь для иллюстрации возможностей аппаратной системы ввода-вывода, с его помощью, на наш взгляд, можно глубже понять вопросы реализации системы ввода-вывода.

Общая схема программирования выглядит следующим образом:

- 1) составить канальную программу;
- 2) в основной программе (программе процессора) установить начальный адрес канальной программы в ячейку 72 (АСК);
- 3) запустить операцию ввода-вывода на заданном устройстве с помощью команды SIO;
- 4) с помощью условных переходов проверить правильность запуска операций, используя код условия, вырабатываемый командой SIO;
- 5) предусмотреть средства для принятия или сброса прерывания в конце операции или при ошибочной ситуации;
- 6) предусмотреть средства для чтения и анализа байтов уточненного состояния при ошибке в устройстве.

В некоторых случаях шестое условие выполнять не нужно, остальные условия обязательны при любых обстоятельствах. Например, шестое условие не выполняется, когда прерывания замаскированы и сбрасываются с помощью команды TIO.

Рассмотрим пример запуска канала и анализ правильности начала операции ввода-вывода. На рис. 2.15 приведен фрагмент программы, который запускает операцию

Имя	Код операции	Операнды
ADACK	LA	7,ADACK
	ST	7,72
	SIO	X'00C'
	BC	7,OSHIB

	DC	X'0200200000000050'

Рис. 2.15.

на устройстве с адресом «00C» по канальной программе, состоящей из одной команды и находящейся по адресу ADACK (команда приведена в шестнадцатеричном виде).

С помощью первых двух команд происходит установка начального адреса программы канала в ячейку 72. Команда SIO заставляет канал начать операцию. В момент запуска канал устанавливает код условия. Причем, как отмечалось ранее, код, равный нулю, соответствует нормальному началу операции. Код, отличный от нуля, сигнала

лизирует о том, что операция по каким-то причинам не началась. Четвертая команда в приведенном фрагменте является командой условной передачи управления, по которой управление передается команде с меткой OSHIB в случае, если код условия в результате выполнения предыдущей команды не равен нулю. Полный перечень наименований инструкций языка ассемблер, использованных в этом и последующих примерах, приведен в [11].

Канальная программа в рассматриваемом примере, как уже упоминалось, состоит из одной команды. Смысл команды: прочитать 80_{10} (50_{16}) байтов перфокарты (устройство 00C — устройство ввода с перфокарт) в основную память, начиная с 2000_{16} адреса. В этом примере присутствуют первые четыре элемента последовательности программирования ввода-вывода, рассмотренных выше.

Использование системы прерывания. Рассмотрим использование пятого из перечисленных выше условий (прерывание по окончанию операции). Пример этого

Имя	Код операции	Операнды
NALO STOP OSHIB IONOW ADACK WAIT WAIT1 WAIT2 INBBD	START	
	BALR	10,0
	USING	*,10
	MVC	120 (8),IONOW
	LA	7,ADACK
	ST	7,72
	SIO	X'00C'
	BC	7,OSHIB
	LPSW	WAIT
	LPSW	WAIT1
	LPSW	WAIT2
	DC	X'0004000000'
	DC	AL3 (STOP)
	CCW	X'02',INBBD,X'00',80
	DC	X'800200000000F00F'
	DC	X'00020000000000FF'
	DC	X'000200000000FFFF'
	DS	CL80

Рис. 2.16.

использования приведен на рис. 2.16. Так же как в примере на рис. 2.15, программа приводится на языке ассемблер [5].

Рассмотрим программу покомандно. В программе имеются 11 собственно команд, 6 констант и 1 рабочая область, которые соответственно расположены в порядке перечисления. Первая команда не имеет эквивалента в системе команд машины, она воспринимается только компилятором — программой, которая переводит программу с языка ассемблер на машинный язык. Вторая команда загружает базовый регистр адресом, значение которого соответствует адресу следующей команды программы. Такая операция необходима, поскольку при адресации используется формула «база + смещение». Смещение в программе подсчитывается автоматически компилятором, а значение базы устанавливается по месту загрузки программы, так как начальный адрес программы заранее не известен и узнать его можно только в момент загрузки. Поэтому для того чтобы не привязываться к процедуре загрузки, базу устанавливают непосредственно в программе, используя прием, рассмотренный выше.

Третья команда объявляет, что регистр 10 будет использоваться в программе в качестве базового. Это указание необходимо для компилятора, поскольку программист работает с символическими адресами (метками), а компилятору при формировании действительного адреса по формуле «база + смещение» необходимо указать, какой регистр будет базовым для данного участка программы.

Четвертая команда заготавливает новое ССП для прерывания по окончании операции ввода-вывода. Заметим, что в рассматриваемом случае побеспокоиться о прерывании необходимо самому программисту, иначе в момент прерывания процессор может уйти из-под управления. Здесь это делается перед запуском операции ввода-вывода. Следующие четыре команды уже были рассмотрены в примере на рис. 2.15.

Последние три команды представляют собой примитивный обработчик прерываний. Рассмотрим его детальнее.

Девятая команда выполняется после того, как канал начал операцию нормально. В результате выполнения команды процессор переходит в состояние «ожидание» с разрешенными прерываниями по мультиплексному каналу (8 в ССП). По сути эта команда синхронизирует программу процессора с окончанием операции. Программа находится в состоянии ожидания, пока канал не кончит ввод. Канал об окончании операции сигнализирует прерыванием. В момент прерывания управление будет передаваться команде, адрес которой находится в новом ССП.

В данной программе это команда по адресу STOP, которая переводит процессор в состояние «останов», поскольку процессор не выполняет команды (состояние «ожидание») и запрещены какие-либо прерывания (все маски в ССП нулевые). Последняя, одиннадцатая, команда выполняется, если канал по каким-либо причинам не начал запрашиваемую операцию ввода-вывода. По этой команде процессор переходит в состояние «ожидание» с запрещенными прерываниями.

В данной программе процессор может оказаться в состоянии «ожидание» в результате выполнения трех последних команд, причем заранее сказать о том, какая команда будет выполнена, нельзя (по индикации на пульте можно судить только о состоянии «ожидание»). Однако с пульта управления можно определить, в каком месте программы находится процессор. Для этого нужно посмотреть адресную часть текущего ССП (точнее, два последних байта). В программе предусмотрены разные коды двух последних байтов, устанавливаемых для различных случаев.

Если процессор находится в состоянии «ожидание», а код в последних двух байтах текущего ССП F00F, то это означает, что процессор ожидает окончания уже начатой операции ввода-вывода. Если код в текущем ССП равен 00FF, то это означает, что операция ввода закончилась успешно. Если же код равен FFFF, то это значит, что операция ввода по каким-то причинам не началась. Собственно, сказанное выше и составляет суть программы, приведенной на рис. 2.16. Поясним только некоторые константы. Первые две константы в совокупности образуют новое ССП, которое используется в программе для приема прерываний по классу ввода-вывода. В частности, во второй используется специальное средство языка, называемое адресной константой. Смысл константы: «сформировать адрес, соответствующий метке STOP, в трех текущих байтах». Третья константа (ADACK) представляет собой канальную программу, состоящую из одной команды. Здесь записана точно такая же команда, как и для примера на рис. 2.15.

Как видно, в таком виде команда записывается нагляднее и проще. Для этого используется также специальное средство языка — команда CCW, четыре операнда которой соответствуют четырем основным частям команды канала: код операции (X «02») — «читать», адрес основной памяти (INBBD) — «куда читать», признаки (X «00») — не используются, счетчик (80) — «сколько байтов читать». Сле-

дующие три константы соответствуют трем ССП для случаев, уже рассмотренных выше.

Последняя команда задает рабочую область (буфер), куда будет считываться перфокарта.

Таким образом, рассмотренная программа считывает только одну перфокарту. Если перфокарта считана (правильно или с ошибкой), то в последних двух байтах текущего ССП будет код 00FF (процессор переходит в состояние «ожидание»). Если же канал почему-либо не начал операцию, то в текущем ССП будет код FFFF.

Обработка прерываний. В рассмотренном примере, конечно, никакой обработки прерывания не производится, но тем не менее оно происходит и воспринимается самой программой. При этом программа, выполняемая процессором, просто останавливается. Однако практически нужно еще проанализировать, как окончилась операция, и продолжить основную программу. Мало того, при работе с некоторыми типами устройств, а в отдельных случаях и со всеми, в результате выполнения одной операции ввода-вывода по классу «ввод-вывод» может прийти не один, а несколько сигналов прерываний, их тоже нужно принять и обработать. Другими словами, ситуация с прерываниями на практике намного сложнее, чем было рассмотрено.

При программировании обработчика прерываний всегда нужно учитывать тип и специфику устройства. Например, если основная программа выводит информацию на АЦПУ, располагающее собственным буфером, то в нормальном режиме по окончании операции выдаются два прерывания:

- 1) канал кончил, т. е. канал заполнил буфер АЦПУ и освобождается для обслуживания других устройств;
- 2) устройство кончило, т. е. АЦПУ закончило распечатку буфера в одной строке.

Временная диаграмма работы тракта ввода-вывода приведена на рис. 2.17.

Для некоторых устройств сигналы «канал закончил» и «устройство кончило» совпадают по времени. Все это справедливо, когда операция ввода-вывода протекает без ошибок. В случае ошибочных ситуаций происходит прерывание, а также, кроме этого, возможны специально запланированные прерывания ввода-вывода. Таким образом, запросы на прерывания по классу ввода-вывода происходят в результате

- 1) окончания операции ввода-вывода;
- 2) обнаружения ошибки в момент выполнения;

- 3) использования программно-управляемого прерывания;
- 4) перехода устройства в состояние готовности («внимание»).

Прерывания от ввода-вывода исчезнуть не могут. Если прерывания запрещены (замаскированы), то они запоминаются и обрабатываются немедленно после снятия

SIO	Основная программа	Обработка прерываний	Основная программа	Обработка прерываний
	Запуск канала	Канал кончил		Устройство кончило
	Канал заполняет буфер АЦПУ		АЦПУ печатает содержимое буфера	
		Прерывание		Прерывание

Рис. 2.17

запрета в порядке установленного приоритета. А это означает, что любое прерывание должно быть либо обработано, либо сброшено, но сброшено программным путем. Обработка прерывания состоит в передаче управления специальной программе. Сбросить прерывание без обработки можно, только используя команду ТЮ (команда процессора).

Рассмотрим применение команды ТЮ. Пусть требуется ввести в основную память массив перфокарт, для простоты предполагаем, что все перфокарты вводятся в одно и то же место. Программа, соответствующая поставленной задаче, приведена на рис. 2.18.

Назначение первых трех команд программы такое же, как для примера, приведенного на рис. 2.16. Следующие три команды также были рассмотрены ранее и выполняют функции загрузки начального адреса канальной программы и запуска операции ввода на устройство с адресом 00С. Седьмая команда приведенной программы выполняет проверку правильности запуска операции. При нормальном запуске управление передается следующей команде программы. Если же запустить операцию невозможно (признак результата не равен нулю), то управление передается участку программы, где производится анализ причины,

показывающей, почему операция не началась. Данный участок программы начинается с команды, имеющей метку CHECK. Сначала рассмотрим действие процессора при нормальном запуске операции. Этот участок программы начинается с команды, символический адрес которой CBP.

Имя	Код операции	Операнды
CHIT	START	
	BALR	10,0
	USING	*,10
	LA	7,ADACK
CBP	ST	7,72
	SIO	X'00C"
	BC	7,CHECK
	TIO	X'00C"
CHECK	BC	7,CBP
	BC	15,CHIT
	TM	68,X'01'
	BC	1,KON
KON	LPSW	OSHIB
	LPSW	NOMA
	CCW	X'02', X'0800; X'00',80
	DC	X'000200000000FFFF'
ADACK	DC	X'00020000000000FF'
OSHIB		
NOMA		

Рис. 2.18.

В данном случае это команда «опросить вывод». В программе она использована, во-первых, для того, чтобы установить состояние тракта канал — устройство путем выработки признака результата, и, во-вторых, для того, чтобы сбросить сигнал прерывания, который замаскирован. В рассматриваемой точке программы возможны две ситуации:

- 1) операция еще не кончилась;
- 2) операция уже кончилась.

В первом случае по команде TIO вырабатывается признак 2 (канал занят). По следующей команде условного перехода в случае ненулевого признака результата управление возвращается снова к команде TIO. Образуется цикл, который программно имитирует операцию ожидания конца ввода.

Итак, пока операция ввода не закончится, процессор будет выполнять всего две команды: «опросить ввод-вывод» и «передача управления» снова на команду TIO. Как уже говорилось, процессор не выйдет из цикла до тех пор, пока операция не кончится.

По окончании операции канал освобождается, но по команде ТЮ будет выработан признак 1 (есть необработанные прерывания, поскольку предполагается работа с замаскированными прерываниями). Команда условной передачи вновь вернет управление команде ТЮ, но в этом случае управление будет передано в последний раз.

Дело в том, что команда ТЮ сбрасывает необработанные прерывания (это одна из основных ее функций), а это означает, что признак результата после выполнения ТЮ будет установлен в 0 (нет необработанных прерываний). В результате управление будет передано команде, символический адрес которой СНИТ, т. е. программа переходит к чтению следующей перфокарты.

Таким образом организуется программный цикл, позволяющий считывать целый массив перфокарт. На первый взгляд может показаться, что здесь имеется бесконечный цикл, т. е. «зацикливание», если говорить языком программистов. Если внимательно посмотреть на все команды «бесконечного» цикла, то видно, что выйти из цикла можно только в одном месте — после выполнения команды условной передачи (BC 7, CHECK), расположенной вслед за SIO. По смыслу эта ситуация означает невозможность запустить операцию по каким-либо причинам.

Если принять во внимание, что ситуация физической поломки здесь просто не рассматривается, то, кажется, что выхода из цикла нет. Однако спасает одна неизбежная ситуация. Это та ситуация, когда в подающем кармане считывающего устройства кончились перфокарты. Программа, конечно, не может «знать», когда перфокарты кончаются, и поэтому операция «читать» запускается и в этом случае. Вот тут-то и нельзя запустить операции, и управление получит участок программы, начинающийся по адресу CHECK.

Первая команда рассматриваемого участка проверяет один из битов байта основного состояния устройства в слове состояния канала, а именно бит — «особый случай в устройстве». При считывании перфокарт этот бит устанавливается в единицу при попытке считать перфокарту из пустого подающего кармана. Таким образом проверяется, «кончился ли массив перфокарт?» Если причина незапуска операции именно в этом, то следующая команда (команда условного перехода) передает управление по метке KON. Команда, выполняющаяся в результате такого перехода, прекращает выполнение программы с кодом в двух последних байтах ССП 00FF.

Если операция не запускается по причине «исключение в устройстве», то программа останавливается с кодом FFFF в двух последних байтах ССП (фактически это может означать, что устройство вышло из строя).

Как видно из рассмотренного выше примера, программирование операций ввода-вывода на машинном уровне не является тривиальной задачей. Это подчеркивается несколькими обстоятельствами:

- рассматривалось наиболее простое устройство — устройство ввода перфокарт;

- рассматривалась самая простая канальная программа, состоящая всего из одной команды;

- рассматривалась программа, вводящая массив перфокарт в одно и то же место памяти без всякой обработки, что практически не имеет смысла, кроме учебной демонстрации;

- обработка прерывания не проводилась;

- фактически ошибки программным путем не обрабатывались.

Отметим, что в условиях использования аппаратной системы ввода-вывода мультипрограммный режим невозможен. Такой вывод можно сделать после рассмотрения примера на рис. 2.18. Очевидно, что если процессор будет выполнять программу такого же типа, как на рис. 2.18, никакая другая программа получить управление не сможет до тех пор, пока данная программа не закончится. Отсюда следует, что для мультипрограммирования необходимы какие-то другие средства, которые реализуются только программным путем и обычно являются составной частью операционной системы, осуществляющей управление мультипрограммным режимом. Для реализации мультипрограммного режима необходимо выполнение, как минимум, двух условий: наличие системы прерываний и наличие операционной системы.

Имея систему прерываний и работая на машинном уровне, мультипрограммирование вряд ли будет возможно, если не будут разработаны какие-то универсальные процедуры для выполнения операций ввода-вывода и реализации принципа совмещения операций в рамках мультипрограммирования. Набор множества таких стандартных процедур представляет операционная система.

Остановимся еще на одном аспекте аппаратной системы ввода-вывода. Этот аспект имеет прямое отношение к вопросу «как ввести и запустить программу в модели ЕС ЭВМ?»

Процедура начальной загрузки. Для первоначального ввода информации в машину автоматическим способом существует специальная процедура начальной загрузки (ПНЗ). Реализована процедура стандартными средствами системы ввода-вывода. Начальная стадия ее выполняется аппаратными средствами, а продолжается с помощью специальных команд, написанных человеком.

Функции ПНЗ заключаются в том, чтобы, во-первых, ввести программу или данные в основную память и, во-вторых, передать управление программе, которая должна выполняться непосредственно после ввода. Для того чтобы осуществить операцию ввода, нужно:

- 1) указать устройство, с которого будет осуществляться ввод;
- 2) указать адрес основной памяти (т. е. куда вводить);
- 3) указать число вводимых байтов (т. е. сколько вводить).

Для того чтобы передать управление программе, нужно знать начальный адрес и поместить его в текущее слово состояния программы (ССП). Все это определяется программистом заранее, в процессе подготовки.

Процесс ввода выглядит следующим образом. Адрес устройства устанавливается на пульте управления машины с помощью специальных переключателей. Основная информация (куда вводить, сколько вводить, куда передать управление) наносится в определенном порядке, чаще всего на перфокарту, которая устанавливается на устройство ввода. Затем устанавливается массив, подлежащий вводу, и устройство приводится в состояние готовности.

После всех подготовительных операций на пульте нажимается специальная кнопка «загрузка». С этого момента машина выполняет процедуру начальной загрузки. Аппаратно запускается операция «читать первые 24 байта в основную память, начиная с нулевого байта, с устройства, адрес которого установлен на пульте». В результате выполнения этой операции заполняются 24 байта постоянно распределенных ячеек оперативной памяти (см. табл. 2.2), которые предназначены для использования в момент начальной загрузки.

Прежде чем проследить дальнейший ход операции, рассмотрим структуру первых 24 байтов постоянно распределенных ячеек памяти. Первые восемь байтов служат для формирования СПП, по которому будет осуществляться передача управления после завершения процедуры начальной загрузки. Следующие восемь байтов представляют

собой командное слово канала (КСК), которое будет выполняться после загрузки 24 байтов, в нем содержится информация, куда и сколько загружать. Последние восемь байтов содержат команду канала, которая может не использоваться. Эта команда используется совместно с предыдущей командой как продолжение «цепочки команд».

Продолжим рассмотрение хода выполнения процедуры начальной загрузки. После загрузки первых 24 байтов канал начинает выполнять команду, расположенную с восьмого байта. Отметим, что эта команда попадает в память в процессе выполнения процедуры начальной загрузки с внешнего носителя информации, и, собственно, только с момента выполнения этой команды начинается запрограммированная операция ввода.

Процедура начальной загрузки заканчивается, когда обрывается цепочка команд, выполняемая каналом, но процессор еще не закончил процедуру начальной загрузки. Процессор по окончании работы канала аппаратно, самостоятельно выбирает ССП из введенных в процессе начальной загрузки 24 байтов, находящихся в первых восьми байтах, и делает его текущим. Напомним, что это ССП заранее подготавливается программистом, осуществляющим ввод. После этого процедура начальной загрузки завершается, процессор начинает выполнять программу, которая, как правило, вводится в основную память с помощью процедуры начальной загрузки.

Подводя итог, отметим, что процедура начальной загрузки предназначена только для ввода программ, а не информации для обработки. Заметим также, что с помощью процедуры начальной загрузки, используя 24 первых байта основной памяти, ввести программу большого объема весьма затруднительно. Большие программы, какой, в частности, является операционная система, загружаются в оперативную память ступенчато: сначала с помощью процедуры начальной загрузки вводится программа небольшого размера, которая, получив управление, затем вводит основную программу.

Таким образом, ввод программы в память машины (даже первоначальный) всегда программируется, и если говорить более конкретно, то почти всегда для большой программы нужно описать, кроме нее самой, еще две вспомогательные: первую — 24 байта процедуры начальной загрузки и вторую — для ввода основной программы.

Знание и понимание процесса начальной загрузки оказывается очень полезным при работе с операционной си-

стеймой как во время ее подготовки к работе, так и во время запуска.

Следует заметить, что здесь много внимания уделено вводу-выводу на аппаратном уровне. При этом преследуются две цели: первая — подчеркнуть, что в большинстве задач, решаемых на ЭВМ, процессы ввода-вывода занимают значительное место и время; вторая — относительная сложность программирования операций ввода-вывода

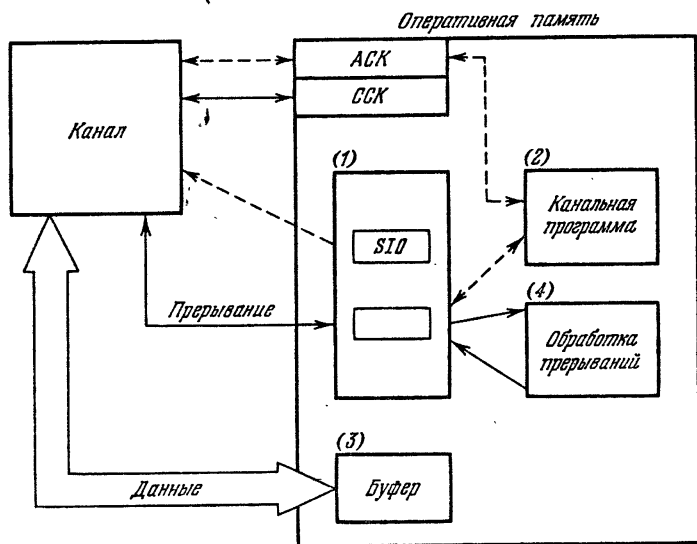


Рис. 2.19.

на аппаратном уровне приводит к необходимости разработки специальных программных средств в операционной системе, которые позволили бы облегчить процесс программирования ввода-вывода. Рассмотрим некоторую обобщенную схему, иллюстрирующую основные моменты системы ввода-вывода на аппаратном уровне.

Обратимся к рис. 2.19, на котором условно изображены поле основной памяти ЭВМ и канал. Рассмотрим взаимодействие представленных на рисунке компонент. Пусть процессор выполняет основную программу (1). В данной программе требуется выполнить некоторую операцию ввода-вывода. Непосредственно перед запуском устанавливается АСК и дается команда SIO. В момент выполнения команды SIO канал может установить ССК, необходимое для анализа некоторой ситуации в канале. В ре-

зультате нормального запуска канал получает доступ к своей программе, находящейся тут же в основной памяти (2).

Начиная с этого момента, канал и процессор функционируют самостоятельно и независимо друг от друга; процессор продолжает выполнять основную программу, а канал реализует канальную программу, пересылая необходимые данные или в некоторую, или из некоторой области основной памяти, называемой буфером (3).

По окончании работы канал прерывает процесс, в результате чего управление передается специальной программе, выполняющей обработку прерывания (4).

Исходные данные для обработки прерывания программа получает в старом ССП и в ССК, которое устанавливается в момент прерывания. После соответствующей обработки прерывания управление передается в ту точку основной программы, в которой произошло прерывание. Процессор продолжает выполнение основной программы.

Такая схема реализации одной операции ввода-вывода позволяет сделать вывод об относительной сложности программирования операций ввода-вывода. Достаточно рассмотреть только две ситуации. Первая — если появляется необходимость в момент совмещения функционирования процессора и канала выполнить еще одну операцию на том же канале, то программирование такой операции в рамки рассмотренной схемы уже не укладывается. Вторая — если в ЭВМ выполняются несколько программ, то программирование даже одной операции ввода-вывода в отдельной программе также не укладывается в рамки рассмотренной схемы.

Кроме этого, следует отметить, что в данной книге не рассмотрены все тонкости системы ввода-вывода, которые вряд ли можно описать в рамках настоящей книги. Все сказанное выше делает программирование на физическом уровне для программистов нецелесообразным.

Фактически выполнение программ, составленных программистом, происходит под управлением специальной управляющей программы, являющейся частью операционной системы и выполняющей за программиста многочисленные стандартные операции.

УПРАВЛЯЮЩАЯ ПРОГРАММА — ОСНОВА ОПЕРАЦИОННОЙ СИСТЕМЫ

3.1. Мультипрограммирование и его реализация

Идея мультипрограммирования положена в основу всех существующих операционных систем.

Появление мультипрограммирования обусловлено стремлением увеличить эффективность использования оборудования за счет имеющихся в вычислительной машине временных и аппаратных ресурсов при решении задачи. При этом любая универсальная ЭВМ состоит из набора устройств, сильно отличающихся друг от друга скоростными характеристиками. Например, центральный процессор способен вести обработку данных со скоростью от десятков тысяч до нескольких миллионов байтов в секунду, а устройство ввода с перфокарт позволяет вводить информацию со скоростью 800—1000 байтов в секунду, т. е. на один или несколько порядков медленнее.

Естественно, что при решении одной задачи на ЭВМ быстрые центральные устройства всегда имеют резерв времени. В современных ЭВМ этот резерв времени становится доступным для использования благодаря совмещению операций ввода-вывода с вычислениями, допускается одновременная реализация операций обработки информации и операций обмена.

Использовать образующийся резерв времени можно двумя способами. Во-первых, не дожидаясь окончания операции ввода-вывода, продолжать решение текущей задачи (однако, как показывает практика, это не всегда возможно). Во-вторых, пока выполняется операция ввода-вывода одной задачи, использовать центральное устройство ЭВМ для решения другой задачи. Однако в этом случае для обеспечения решения другой задачи ЭВМ должна располагать необходимым дополнительным оборудованием, а именно — свободной оперативной памятью и нужным числом внешних устройств.

В ЭВМ, работающих в режиме мультипрограммирования, появляющиеся резервы времени можно использовать обоими способами. Резервы времени, используемые для реализации

мультипрограммирования, образуются, главным образом, в моменты выполнения операций ввода-вывода. Операции ввода-вывода при использовании ЭВМ играют большую роль и относятся к основным факторам, определяющим возможности ЭВМ.

В системе программного обеспечения (СПО) ЕС ЭВМ реализация режима мультипрограммирования базируется

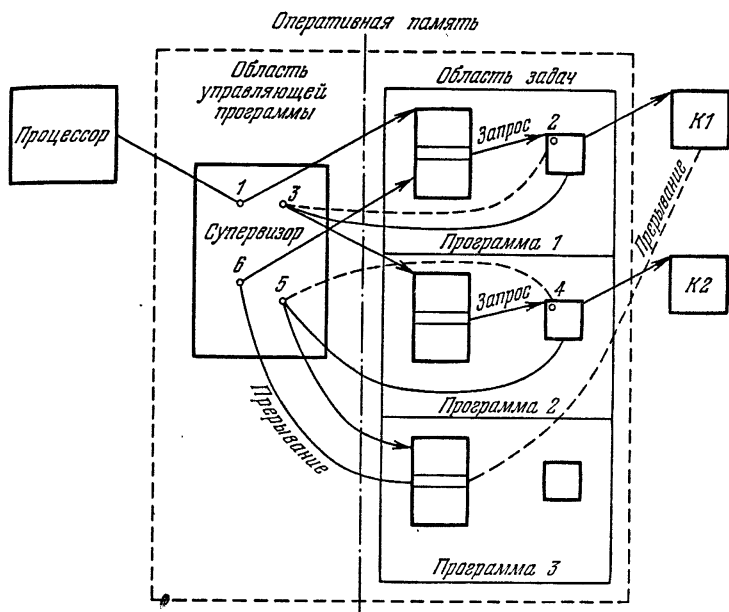


Рис. 3.1.

в основном на использовании принципа совмещения операций ввода-вывода с вычислениями.

Рассмотрим схему, иллюстрирующую принцип реализации мультипрограммирования в моделях ЕС ЭВМ (рис. 3.1). На схеме изображены 4 центральных устройства ЭВМ: процессор, оперативная память и два канала K1, K2. Поле оперативной памяти разбито на две области: область управляющей программы и область задач.

В области задач находятся 3 программы, решающие 3 различные задачи, не связанные друг с другом. Каждой программе выделен в области задач определенный участок оперативной памяти, называемый разделом. Поскольку ЭВМ в рассматриваемом примере располагает всего одним про-

цессором, то для реализации трехпрограммного режима необходима некоторая управляющая программа, находящаяся в специально отведенной для нее области оперативной памяти.

Сразу оговоримся, что в оперативной памяти находится наиболее часто используемая часть всей управляющей программы, обычно называемая ядром. Ядро управляющей программы является, в сущности, некоторым диспетчером, который содержит так называемые резидентные программы супервизора. Супервизор управляет решением всех находящихся в ЭВМ программ. Через супервизор программы получают управление, а также возможность использовать каналы и другие устройства машины.

Рассмотрим на рис. 3.1 последовательность процессов, происходящих в ЭВМ в режиме мультипрограммирования. Супервизор запускает программу 1 в условной точке 1. Это означает, что процессор поступает в распоряжение программы 1. Программа 1 использует процессор до тех пор, пока ей не потребуется выполнить операцию ввода-вывода. В этом случае из программы поступает запрос на запуск операции. Этот запрос обрабатывается системной программой запроса (точка 2), которая, пользуясь услугами супервизора (точка 3), запускает операцию ввода-вывода через канал 1. Так как после запуска операции ввода процессор и канал работают независимо друг от друга, то супервизор в точке 3 передает процессор в распоряжение программы 2. Программа 2 использует процессор также до того момента, пока ей не понадобится выполнить операцию ввода-вывода. Запрос на ввод-вывод от программы 2, в свою очередь, обрабатывается аналогичной системной программой (точка 4), которая, как и для программы 1, с помощью супервизора (точка 5) запускает операцию через канал 2.

В рассматриваемый момент времени канал 1 продолжает выполнять операцию ввода-вывода для программы 1, канал 2 начал выполнять операцию для программы 2, а процессор, как и ранее, через супервизор (точка 5) поступает в распоряжение программы 3. В нашем примере программа 3 выполняется до такого момента, пока канал 1 не закончит выполнение операции ввода-вывода первой программы. В этот момент от канала 1 поступает прерывание по классу ввода-вывода. В результате прерывания управление вновь получает супервизор (точка 6), который передает процессор в распоряжение программы 1 для продолжения решения задачи.

Далее возможно, что программа 1 потребует выполнения очередной операции ввода-вывода, тогда программа 1 будет прервана, например, каналом 2, который к этому моменту закончил выполнение своей операции. Происходит запуск операции ввода-вывода, и программа 1 завершает свою задачу.

Подводя итоги обсуждения рассмотренного примера, отметим, что любая программа для своей реализации требует определенных ресурсов: оперативной памяти, процессора, каналов и т. д. Эти ресурсы предоставляются в распоряжение программ только через супервизор управляющей программы.

Перейдем к рассмотрению вопроса о значении ввода-вывода в рамках всей операционной системы.

3.2. Ввод-вывод программ в операционных системах

Современная операционная система представляет собой комплекс программ, предназначенных облегчить и автоматизировать процессы использования ЭВМ. В структуре операционной системы можно выделить три функционально законченные части:

- 1) управляющая система;
- 2) система программирования;
- 3) прикладные средства.

Управляющая система представляет собой набор управляющих программ и системных средств программирования: средств супервизора, средств ввода-вывода и т. д.

Система программирования представляет собой набор языков программирования, компиляторов, редактирующих и отладочных программ.

Прикладные средства представляют собой набор готовых прикладных программ.

Поскольку объем программ операционных систем исчисляется от нескольких сотен тысяч до миллиона команд, то вопросы организации операционных систем приобретают первостепенное значение. Из них особо выделяются задачи организации, выполнения и взаимодействия различных компонент операционной системы.

Все программы операционной системы располагаются на внешних запоминающих устройствах прямого доступа, чаще всего на магнитных дисках. Программы находятся в библиотеках, каждая из которых соответствует той или иной части операционной системы. Причем программы хранятся в библиотеках различными способами. В одних

библиотеках находятся программы, написанные на символическом языке программирования. Такие библиотеки обычно называются библиотеками исходных модулей (БИМ). В других библиотеках хранятся программы, прошедшие стадию компилирования, но не прошедшие редактирование — программы в объектном виде. Это библиотеки объектных модулей (БОМ). И, наконец, в некоторых библиотеках находятся программы, готовые для выполнения. Это библиотеки абсолютных модулей (БАМ).

На конкретной ЭВМ имеется некоторый обязательный набор программ операционной системы, включающий в себя

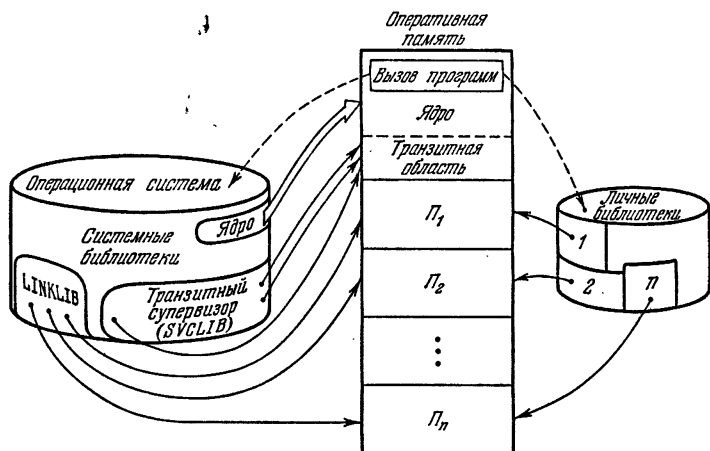


Рис. 3.2.

управляющую систему, системы программирования и стандартные обслуживающие программы, поставляемые вместе с машиной.

Кроме обязательного набора, пользователь, как правило, разрабатывает свои или использует уже готовые прикладные программы для решения конкретных задач. Эту группу программ пользователь может помещать либо в системные библиотеки, либо в личные библиотеки, которые могут быть в общем случае организованы в виде БИМ, БОМ или БАМ.

Для иллюстрации некоторых моментов выполнения и организации взаимодействия программ операционных систем обратимся к рис. 3.2. На рисунке приведена схема распределения оперативной памяти для мультипрограммного режима, носитель с программами операционной системы

и носитель с личными библиотеками. Библиотеки, содержащие программы операционной системы, составляют в общем случае совокупность библиотек типа БАМ, БОМ, БИМ.

Среди библиотек типа БАМ, как правило, обязательно имеются две библиотеки, содержащие копию ядра управляющей программы и супервизорные программы, не вошедшие в состав ядра. Дело в том, что весь супервизор, как правило, в оперативной памяти целиком не помещается, кроме того, его просто невыгодно целиком помещать в оперативную память, так как он содержит определенное количество редко используемых программ. Поэтому, в целях экономии оперативной памяти, указанные программы супервизора (транзитные модули) постоянно хранятся в системной библиотеке и вызываются в оперативную память только по мере необходимости. Для этой цели в оперативной памяти, занятой ядром, предусматривают специальную транзитную область.

Итак, оперативная память ЭВМ (см. рис. 3.2) разбита на области, предназначенные для размещения ядра управляющей программы, включая транзитную область, и прикладных программ (P_1, P_2, \dots, P_n).

При запуске операционной системы ядро с помощью специальной процедуры загрузки помещается в оперативную память, и с этого момента управляющая программа становится «хозяином положения» на ЭВМ.

Она вызывает для выполнения программы в отведенные им области оперативной памяти (разделы). Программы загружаются либо из системных, либо из личных библиотек, в зависимости от того, куда они были ранее записаны. Например, компиляторы и редактор связей вызываются из системных библиотек, а программы, написанные пользователем, могут вызываться из системных (если они туда были записаны) или из личных библиотек.

После загрузки соответствующих программ в оперативную память супервизор организует выполнение программ согласно принципам, рассмотренным в п. 3.1. Схема взаимодействия элементов операционной системы, приведенная на рис. 3.2, позволяет рассматривать операции ввода-вывода в качестве одного из наиболее важных аспектов функционирования управляющих программ системы.

Из схемы, в частности, видно, что во время работы операционной системы в машине постоянно происходит обмен программами между оперативной памятью и внешними запоминающими устройствами.

До сих пор рассматривались вопросы выполнения программ на ЭВМ, однако не менее существенными представляются вопросы ввода и подготовки программ к решению на ЭВМ в условиях мультипрограммирования. Рассмотрим их более детально.

3.3. Специфика пропуска задач в операционных системах

При использовании мультипрограммирования для решения конкретной задачи программист должен некоторым способом описать как свою программу, так и процесс ее решения. Программы операционной системы, выполняющие функции управления ходом решения программ, черпают необходимые для такого управления сведения из описания процесса решения задачи, составляемого пользователем.

Такое описание представляет собой некоторое задание операционной системе. Для написания заданий в любой операционной системе имеется специальный язык — язык управления заданиями (ЯУЗ). Обычно средства ЯУЗ построены по декларативному принципу: сделать то-то, привлечь для выполнения такие-то ресурсы и т. п.

Имеющейся в любой операционной системе ЯУЗ позволяет описывать требуемые для решения задачи ресурсы, а также основные этапы решения задачи.

Описание ресурсов содержит, например, сведения о минимальном объеме оперативной памяти, необходимом для решения конкретной программы, о количестве внешних устройств, об используемых каналах и т. д.

Описание основных этапов решения задачи содержит имена программ, которые нужно выполнить, последовательность их выполнения, а также различные условия, регулирующие заданную последовательность и т. д. Например, типовое описание процесса решения задачи состоит в том, что операционной системе предписывается выполнение трех программ: компилятора, редактора связей и программы пользователя, которая должна пройти стадии компилирования и редактирования.

Одно задание может содержать описание выполнения различных программ. Можно было бы потребовать для каждой выполняемой программы писать свое задание, однако в практической работе это не удобно. Программисту, если он работает сразу над несколькими программами, удобнее в одном задании описывать решение всех своих задач,

нежели для каждой программы создавать свое задание. Это удобно не только с точки зрения написания задания, но и с точки зрения выполнения.

Операционная система выполняет начатое задание непрерывно от начала до конца. Если программист напишет несколько отдельных заданий, и так как они выполняются независимо друг от друга, то результаты выполнения

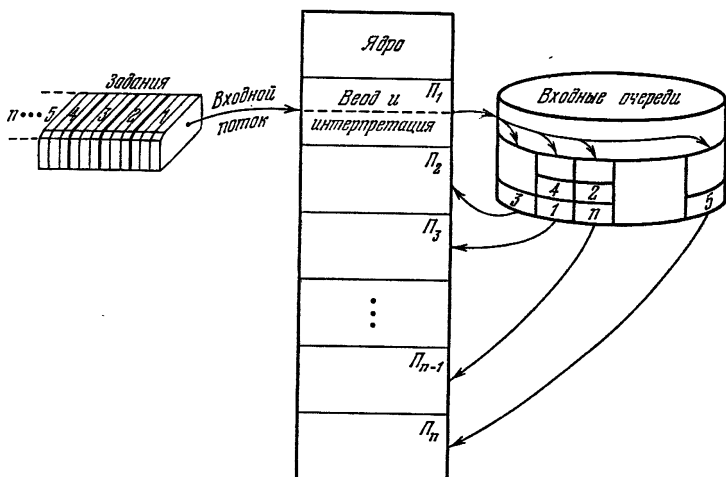


Рис. 3.3.

отдельных заданий поступят к программисту в различные моменты времени и, чаще всего, не в той последовательности, в какой ему бы хотелось.

Поэтому в операционной системе предусмотрено, что каждый программист имеет возможность в одном задании описать, практически, столько задач, сколько необходимо.

В режиме мультипрограммирования с одной ЭВМ работают сразу несколько пользователей, каждый из которых свои конкретные задачи оформляет в виде одного задания. Физически, каждое задание представляется в виде колоды перфокарт, в составе которых, кроме операторов языка управления заданиями, могут находиться перфокарты с программами и исходными данными для них.

Задания различных пользователей формируются в общий пакет, который затем вводится в ЭВМ специальными управляющими программами. Схематично процессы ввода и запуска заданий в операционной системе ОС ЕС изображены на рис. 3.3.

Пакет заданий образует входной поток, из которого задания с помощью системных программ ввода и интерпретации (P_1) поступают в распоряжение и под контроль операционной системы. Предварительно все задания, поступившие из входного потока, записываются во входные очереди (см. рис. 3.3), размещаемые на запоминающих устройствах прямого доступа.

Входные очереди создаются обычно с однозначной привязкой к некоторому разделу оперативной памяти из области задач. Таким образом, для каждого раздела имеется своя входная очередь, и находящиеся в ней задания будут выполняться только в соответствующем разделе.

Выполнение заданий организуется следующим образом. В каждом разделе запускается специальная системная программа (инициатор), которая планирует и обеспечивает через супервизор необходимые ресурсы для решения любой программы задания, вызывает эту программу и передает ей управление. По окончании выполнения отдельной программы задания управление снова получает инициатор, который завершает выполнение предыдущей программы и повторяет рассмотренные выше действия для остальных программ задания. Если задание полностью завершено, то инициатор приступает к выполнению следующего задания, находящегося в очереди. Если же очередь окажется полностью выбранной, то данный раздел переходит в состояние ожидания.

Описанные выше процессы происходят под постоянным управлением супервизора во всех разделах одновременно.

Из рис. 3.3 видно, что в процессах ввода и запуска заданий в операционной системе операции ввода-вывода занимают одно из важнейших мест и существенным образом могут определять скорость их выполнения.

В операционной системе ДОС ЕС ЭВМ схема запуска заданий несколько отличается от только что рассмотренной. Входной очередью в ДОС ЕС ЭВМ служит непосредственно входной поток заданий. Задания считываются и сразу выполняются в том разделе, для которого определен входной поток. Поскольку в ДОС ЕС ЭВМ возможен только трехпрограммный режим, то для каждого раздела при такой схеме должен быть свой отдельный поток, а это возможно только при наличии необходимого числа внешних устройств ввода.

Выше проанализированы некоторые функции управляющей программы операционных систем и отмечено, что процессы ввода-вывода в выполнении этих функций зани-

мают одно из центральных мест. К этому добавим, что значительное количество практических задач, особенно в сфере обработки данных, относятся к такому классу, в котором определяющая роль принадлежит также процессам ввода-вывода.

Подводя итоги обсуждения основных функций управляющей программы, рассмотрим ее укрупненную структуру.

3.4. Структура управляющей программы операционной системы

На рис. 3.4 схематично представлены элементы управляющей программы. Пунктирными линиями отмечены связи между отдельными компонентами, проявляющиеся в процессе реализации одной программы. Задания пользователя вводятся и запускаются с помощью системных программ,

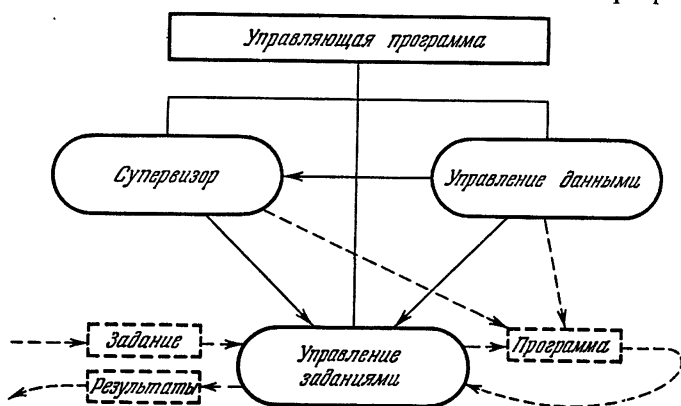


Рис. 3.4.

образующих компоненту управляющей программы, называемую «управление заданиями». Выдача результатов решения осуществляется также, как правило, программами этой компоненты.

Управление решением конкретной задачи осуществляется специальной программой, называемой «супервизором» управляющей программы, которая, главным образом, реализует функции распределения ресурсов между программами.

При функционировании «супервизора» и «управления заданиями», как было видно ранее, значительную долю времени занимают процессы ввода-вывода (см. рис. 3.1—3.3). Управление всеми процессами ввода-вывода в опера-

ционной системе осуществляет третья, не менее важная компонента управляющей программы, которую называют «управление данными». Эта компонента интенсивно используется также прикладными программами, особенно при решении информационных и экономических задач.

Программами «управления данными» пользуются все компоненты операционной системы, включая «супервизор» (вызов транзитных модулей, программ и т. д.) и «управление заданиями» (см. рис. 3.4). Рассмотрим детальнее участие некоторых компонент программ «управления данными» в выполнении операции ввода-вывода.

3.5. Выполнение операций ввода-вывода с помощью компонент управляющей программы

На рис. 3.5 приведены три центральных устройства ЭВМ: процессор, канал, оперативная память. В поле оперативной памяти выделены 4 участка: основная программа (1),

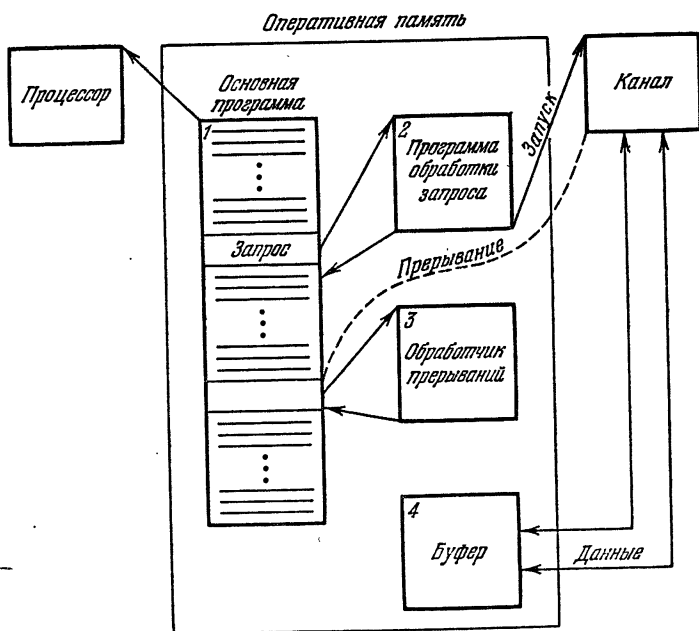


Рис. 3.5.

программа обработки запроса (2), обработчик прерываний (3) и буфер (4).

Из всего комплекса программ «управления данными» выделены только те, которые непосредственно участвуют в реализации одной операции ввода-вывода. Участки (2), (3), (4) являются частями системных программ. Рассмотрим принципиальную схему выполнения одной операции ввода-вывода, принятую в операционных системах ЕС ЭВМ, и программы, участвующие в этом процессе.

Как уже отмечалось, процессор выполняет программу пользователя до того момента, когда необходимо произвести обмен данными между оперативной памятью и внешними устройствами.

Пусть в некоторой точке программы ее составитель записал запрос на операцию ввода-вывода. По этому запросу управление передается специальной программе обработки запроса (2), которая выполняет целый ряд подготовительных функций для физического выполнения требуемой операции ввода-вывода. Среди этих функций следует отметить такие, как составление канальной программы, выделение, если необходимо, буферной области памяти, в которую будут вводиться данные, и т. д.

После проведенной подготовки программа осуществляет физический запуск операции ввода-вывода в канале и немедленно после отключения канала от процессора передает управление в ту точку основной программы, откуда пришел запрос. Начиная с этого момента, канал самостоятельно обеспечивает обмен данными между буфером (4) и внешним устройством, а процессор в то же время продолжает выполнение основной программы пользователя (1).

Такое совмещение работы процессора с каналом продолжается до тех пор, пока канал не закончит выполнение начатой операции обмена. По окончании операции канал вырабатывает сигнал прерывания, который вызывает передачу управления процессора из основной программы в программу, называемую обработчиком прерываний (3).

Здесь следует отметить одно обстоятельство, которое в программировании ввода-вывода играет немаловажную роль и состоит в следующем. Основная программа прерывается в произвольный момент времени, который, как показывает практика, точно рассчитать невозможно, а если и можно, то ценой неоправданно больших усилий. Естественно, программиста такое положение вряд ли может устроить. Ему точно нужно знать, когда закончится операция ввода-вывода. Если при этом еще учесть, что практически все сигналы прерывания находятся в непосредственном ведении управляющей программы, то для программиста

становится необходимым иметь специальные команды и соответствующие средства для синхронизации действий основной программы и канала.

Более подробно задачу синхронизации будем рассматривать в следующих главах, а сейчас вернемся к моменту, когда обработчик прерываний получил управление. Выполнив свои функции, которые в основном сводятся к анализу причины прерывания, проверке правильности окончания операции обмена и т. д., обработчик возвращает управление основной программе в точку, где она была прервана. На этом выполнение одной операции ввода-вывода заканчивается.

Нами рассмотрена укрупненная схема реализации операции ввода-вывода. Естественно, что подобная схема не отвечает на целый ряд вопросов, касающихся практического программирования операций ввода-вывода, однако она дает представление об основных функциях и составе системных программ, участвующих в выполнении только одной операции. Перечислим некоторые основные вопросы, на которые рассмотренная схема не дает ответа.

1. Каким образом обработчик запроса определяет конкретные канал и устройство, предназначенные для обмена?

2. Если на конкретном носителе располагаются несколько массивов информации, то к какому именно адресован выполняемый запрос?

3. Какой элемент массива будет участвовать в обмене?

Ответы на эти и некоторые другие вопросы будут даны в последующих главах.

ПРИНЦИПЫ ПОСТРОЕНИЯ СИСТЕМЫ УПРАВЛЕНИЯ ДАННЫМИ

4.1. Основные объекты системы управления данными

Объекты управления. Система управления данными (СУД) главным образом обеспечивает выполнение функций по реализации, запрограммированных с помощью системных макрокоманд, запросов к данным, находящихся в массивах информации и расположенных на внешних устройствах ЭВМ.

Основным объектом СУД является один или несколько массивов информации, называемых в операционных системах наборами данных или файлами. Набор данных (файл) представляет собой поименованную совокупность логически взаимосвязанных записей, располагаемых на одном или нескольких носителях. В дальнейшем будем пользоваться как тем, так и другим термином, обозначающим массив (набор данных или файл).

При обработке данных конкретного файла в любой заданный момент времени СУД взаимодействует практически только с некоторой частью файла. Такой частью может быть физическая или логическая запись. Указанные записи также можно рассматривать как объекты СУД, но объекты более мелкие, чем набор данных. Другими словами, система управления данными обеспечивает доступ не только к наборам данных, но и к отдельным физическим или логическим записям, из которых, в свою очередь, состоят обрабатываемые наборы данных.

Проиллюстрируем вышесказанное рис. 4.1. На нем представлены основные объекты СУД и уровни их детализации.

Верхний уровень — уровень наборов данных (НД). На этом уровне СУД, по существу, отслеживает только физические границы наборов данных на носителях различных типов, допуская внутри набора произвольную структуру данных. При этом в режиме мультипрограммирования обеспечивается доступ всех программ, использующих СУД, к своим файлам.

Программными средствами решается и задача защиты файлов от взаимного случайного обращения.

Второй уровень — уровень набора данных, представленного совокупностью физических записей, называемых блоками (НДБ). Этот уровень регламентирует некоторым способом внутреннюю структуру набора данных. На данном

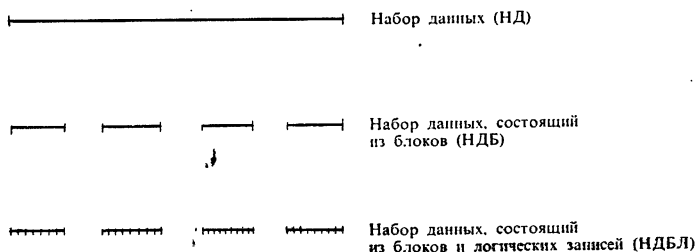


Рис. 4.1.

уровне набор данных представляется в виде множества физических записей (блоков). При этом СУД не только отслеживает границы набора данных на носителях (функции 1-го уровня сохраняются), но и обеспечивает необходимые средства программирования и их реализацию по обращению к отдельной составной части файла в виде физического блока.

Наконец, на третьем уровне, представленном наборами данных, состоящими из физических блоков и логических записей (НДБЛ), программы СУД обеспечивают средства и реализацию доступа к отдельным составным элементам физических блоков, называемых логическими записями.

Логические записи являются последними «кирпичиками» наборов данных, доступ к которым обеспечивает СУД в операционных системах ЕС ЭВМ. Обращение к отдельным составным частям логических записей, если это необходимо, должен осуществлять программист в своей программе несистемными средствами. Отметим также, что дальнейшее развитие и усовершенствование систем управления данными идет как по линии расширения границ файлов, так и по линии более детального, чем логические записи, представления элементов данных.

Физическое представление объектов. Любой набор данных физически располагается на носителях внешних устройств. Носители, которые представляют собой неделимые единицы, такие, как пакет магнитных дисков, катушка с магнитной лентой, называют томами. Другими словами, том —

это абстрактное понятие, используемое для обозначения физических носителей, которые могут быть применены для хранения информации.

Система управления данными в процессе обработки файлов взаимодействует с томами, на которых они расположены. В этом плане СУД выполняет такие функции, как: обеспечение средств надежного хранения файлов, поиск нужных файлов, защиту файлов от постороннего вмешательства и ряд других. Более детальное рассмотрение этих

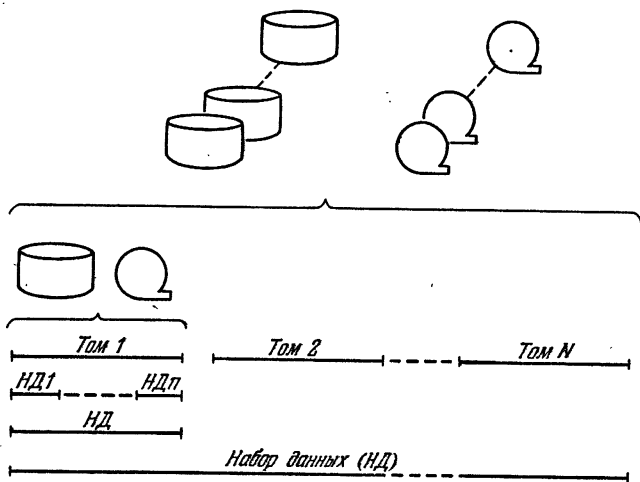


Рис. 4.2.

функций будет приведено в последующих главах. Набор данных на физических носителях может занимать часть тома, весь том или одновременно несколько томов (рис. 4.2).

Отметим, что если набор данных занимает только часть тома, то в оставшейся части могут быть дополнительно расположены несколько наборов данных, один набор данных или часть набора данных. Причем на количество наборов данных, записываемых на один том, никаких ограничений, кроме физических, не накладывается.

Если набор данных оказывается слишком большим, то его записывают на несколько однотипных томов. Количество однотипных томов для хранения одного набора данных не лимитируется. При обработке многотомных файлов система управления данными должна располагать средствами, обеспечивающими приемлемый способ переключения томов и динамическую смену носителей.

4.2. Уровни управления данными и средства их реализации

В предыдущем разделе было рассмотрено краткое описание объектов управления и некоторые вопросы их физического представления на носителях. Разбиение объектов управления данными на три уровня определяет соответствующее трехуровневое управление данными. На рис. 4.3 приведены три уровня управления данными, которые имеют место в операционных системах ОС и ДОС ЕС.

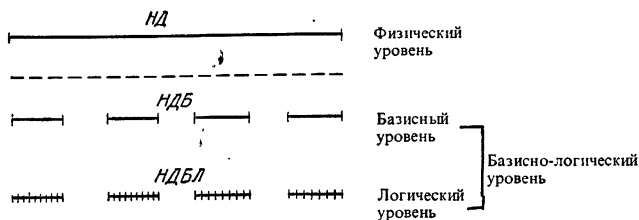


Рис. 4.3.

Физическое управление соответствует уровню НД, в котором основной единицей управления является файл. На этом уровне СУД никак не регламентирует внутреннее строение набора данных и обеспечивает, как уже отмечалось, в основном, отслеживание его границ на физических носителях. Управление обменом данными на этом уровне производится с помощью канальных программ пользователя, которые составляются программистом. СУД автоматически выполняет только процедуры запуска канала и обработку прерываний по окончании обмена.

Базисное управление соответствует уровню НДБ, в котором основной единицей управления является физический блок. На этом уровне обмен данными осуществляется физическими блоками. При таком способе управления программисту достаточно с помощью имеющихся средств запросить очередной блок, и программы СУД обеспечат доступ к этому блоку, т. е. нет необходимости составлять канальную программу. Указанный способ обработки данных называется базисным.

Логическое управление соответствует уровню НДБЛ, в котором основной единицей управления является логическая запись. На этом уровне обмен данными между программой и файлом осуществляется составными частями физического блока, называемыми логическими записями. Смысловое содержание логической записи определяет про-

граммист, исходя из смыслового содержания решаемой задачи. Говорят, что на этом уровне обработка данных осуществляется на более высоком, так называемом логическом уровне. Отметим, что на носителях набор данных логического уровня представляет собой совокупность физических блоков, поэтому программам СУД при чтении сначала приходится считывать блок (обмен между внешней и оперативной памятью), а затем выделять из него отдельные логические записи. При записи, наоборот, — сначала формировать блок из логических записей, а затем записывать его на носитель.

На рис. 4.3 пунктиром обозначена граница между физическим уровнем управления данными и двумя другими. Поскольку на базисном и логическом уровнях программист освобождается от необходимости составлять программы канала, то эти уровни управления данными следует особо выделить. Будем называть их базисно-логическими. На практике приходится использовать это понятие, чтобы отделить (по степени автоматизации процесса составления канальных программ) физический уровень от остальных.

Теперь кратко рассмотрим вопрос о местонахождении программ СУД в компонентах управляющей программы.

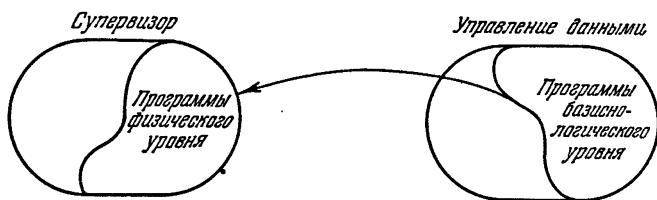


Рис. 4.4.

Программы физического уровня управления данными находятся в составе супервизора, а программы базисно-логического уровня — в компоненте управления данными (см. рис. 3.4 и 4.4). Попробуем объяснить имеющееся распределение программ СУД по компонентам управляющей программы. Для этого необходимо показать, что программы физического уровня СУД должны постоянно находиться в составе ядра управляющей программы и выполнять некоторые мониторинговые функции супервизора. Необходимо также показать, что программы базисно-логического уровня в значительной степени связаны с основными характеристиками программ пользователя и физически располагаются в области задач.

4.3. Физическая система управления данными

Сначала обратимся к физическому уровню управления данными, который часто называют физической системой ввода-вывода или физической системой управления данными. Если программист осуществляет обработку данных с использованием физического уровня, то он должен хорошо знать логическую структуру конкретного внешнего устройства, разбираться в специфических особенностях устройства и зачастую хорошо представлять себе его физическую реализацию. Это совершенно необходимо для правильного и качественного составления канальных программ при описании и программировании операций ввода-вывода.

Кроме того, программист в процессе написания программы не должен учитывать мультипрограммирование. Программист вовсе не должен знать, сколько других программ будет находиться в оперативной памяти при выполнении написанной им программы. При этом, в частности, он не обязан заботиться о том, что те внешние устройства, с которыми будет взаимодействовать написанная им программа, в условиях мультипрограммирования могут быть использованы другими программами.

Иными словами, система управления данными даже на физическом уровне должна располагать такими средствами программирования, которые, с одной стороны, позволили бы программисту составлять программу так, как будто бы только в его распоряжении находятся все ресурсы ЭВМ, а с другой стороны, позволили бы при реализации составленной программы выполнять на ЭВМ еще несколько других программ в режиме мультипрограммирования. Отметим, что это рассуждение применимо не только к системе управления данными, но и к другим компонентам управляющей программы.

В системе управления данными рассмотренные задачи решаются главным образом на физическом уровне. Базисно-логический уровень управления данными строится, базируясь на средствах физического уровня. Поэтому на базисно-логическом уровне решаются другие, не менее важные задачи. Однако эти задачи пока не будем рассматривать, а вернемся к физическому уровню управления данными.

Программы физической системы ввода-вывода выполняют следующие функции:

- 1) выделение ресурсов для выполнения операций;
- 2) запуск конкретных операций;
- 3) обработку прерываний по классу ввода-вывода.

Продemonстрируем выполнение физической системой ввода-вывода своих основных функций управления данными на схеме (рис. 4.5), иллюстрирующей основные принципы реализации физического уровня. На рис. 4.5 изображены

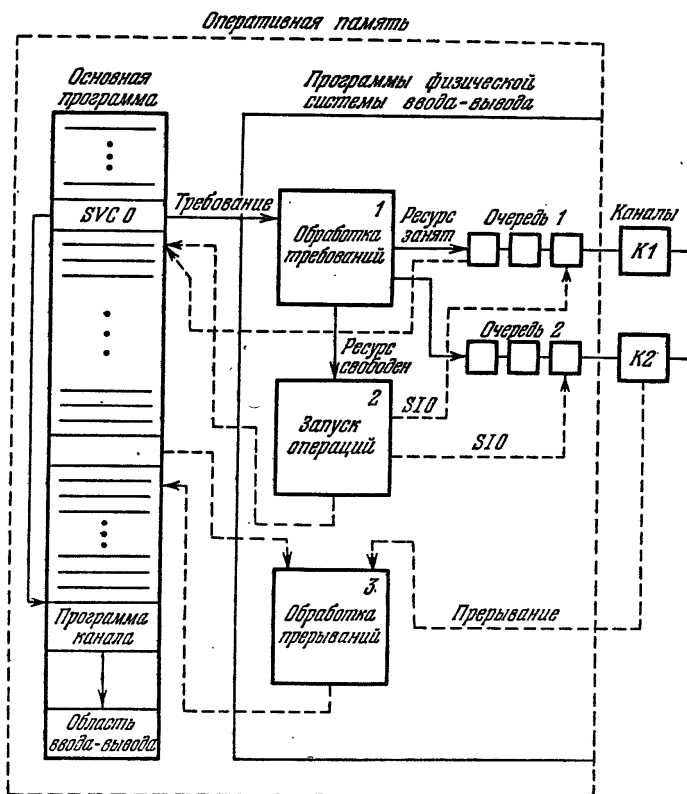


Рис. 4.5.

некоторый участок оперативной памяти и два канала. В выделенном участке оперативной памяти находится программа пользователя (основная программа) и программа физической системы ввода-вывода. Проследим взаимодействие задачи пользователя с программами физической СУД во времени. Основная программа выполняется обычным способом до того момента, пока не потребуется осуществить операцию ввода-вывода. Тогда программа выдает требование на реализацию операции ввода-вывода.

Поскольку физическая система ввода-вывода выполняет ряд мониторинговых функций, то основные программы физического уровня размещены в супервизоре и обращение к ним производится через команду «вызов супервизора». Например, в операционных системах ДОС и ОС ЕС обращение к физической СУД осуществляется через команду SVC 0 (см. рис. 4.5).

В точке выдачи требования SVC 0 должны быть указаны начальный адрес канальной программы и некоторые другие сведения. О составлении программы канала на этом уровне, как уже отмечалось, должен позаботиться программист. Поэтому на рис. 4.5 программа канала находится в области основной программы.

По указанному требованию управление передается одной из системных программ, которую условно назовем обработчиком требований (1). В конкретных операционных системах данная программа называется по-разному: планировщик каналов, супервизор EXCP. В основном эта программа выполняет функции по определению степени занятости того физического ресурса, который нужен для выполнения операции (в данном случае речь идет о канале).

Если ресурс свободен, то управление передается программе запуска операции (2), в функции которой входят запуск операции с помощью команды «начать ввод-вывод», проверка правильности запуска операции и некоторые другие. Эта программа по окончании выполнения своих функций возвращает управление в основную программу.

Если же ресурс окажется занятым, то обработчик требований ставит полученный запрос в очередь к соответствующему каналу и возвращает управление основной программе. Поставленный в очередь запрос будет принят к исполнению при обработке очередного прерывания немедленно после освобождения необходимого ресурса. Такие очереди запросов организуются к каждому каналу в отдельности (см. рис. 4.5). Обслуживание очередей производится программами физической системы ввода-вывода. Этот способ приема и исполнения требований в отношении операций ввода-вывода делает возможным их выполнение по запросам из различных программ и освобождает программиста от необходимости на этапе программирования учитывать многопрограммность.

Отметим еще одну важную функцию, которую выполняет физическая СУД. В момент получения требования обработчик проверяет его корректность. Основная функция проверки состоит в том, чтобы обеспечить защиту набора данных

от возможного случайного обращения к нему из других программ. По существу — это ранее упоминавшаяся нами функция отслеживания физических границ файлов. Обработчик требований, прежде чем начать выполнение, анализирует программу канала и определяет, «соответствует ли обращение к внешнему устройству тем границам, которые установлены для конкретного набора данных». Выполнение операции ввода-вывода разрешается лишь в случае получения положительного ответа на этот вопрос. Если же ответ отрицательный, то требование аннулируется, а «программа-нарушитель» удаляется из оперативной памяти ЭВМ.

Наконец, рассмотрим программу обработки прерываний (3), являющуюся самостоятельной составной частью физической СУД. Эта программа обеспечивает прием и обработку прерываний по классу ввода-вывода, а также возврат управления в точку основной программы, где она была прервана. Одновременно с этим обработчик прерываний имеет возможность время от времени производить запуск операций (2), стоящих в очереди требований на ввод-вывод. Возможность доступа к освободившимся каналам возникает в результате приема и обработки прерываний (очереди (1) и (2) на рис. 4.5). Завершает свою работу обработчик прерываний, как было отмечено ранее, возвратом управления основной программе.

В этом разделе были рассмотрены основные, можно сказать, принципиальные функции, выполняемые программами физической системы ввода-вывода. Отметим, что такие функции, как отслеживание границ файлов, обслуживание очередей требований, запуск операций на всех типах устройств, обработка прерываний и завершение операций ввода-вывода, не зависят от уровней представления объектов СУД и уровней управления ими. Появление отмеченных функций обусловлено возможностями аппаратных средств и средств управляющей программы. В значительной степени поэтому вышеуказанные функции являются супервизорными, и программы, их реализующие, являются составной частью супервизора операционной системы.

4.4. Базисно-логическая система управления данными

Главными недостатками физической СУД, с точки зрения программиста-пользователя, являются следующие: во-первых, программист обязан детально знать каждое внешнее устройство, с которым ему приходится работать; во-вто-

рых, программа, в которой непосредственно используются средства физической системы ввода-вывода, сильно зависит от конкретного устройства, в частности, она оказывается однозначно привязанной к типу устройства, к некоторому участку внешней памяти на магнитных дисках и т. п.

Для устранения указанных недостатков нужно построить систему управления данными таким образом, чтобы предоставить программисту возможность работать с устройством на более высоком, по сравнению с физическим, уровне и обеспечить возможность одной и той же программе, содержащей операции ввода-вывода, взаимодействовать с устройствами различных типов без каких бы то ни было переделок. Программы базисно-логического уровня управления данными, называемые также базисно-логической системой управления данными, в значительной мере решают поставленные задачи.

Чтобы успешно их решить, необходимо, главным образом, снять с программиста обязанности по составлению программ канала и переложить эти функции на систему управления данными. Однако такое решение, естественно, должно накладывать некоторые ограничения на внутреннюю структуру файлов. На физическом уровне управления данными от системы требуется обеспечить только «сохранность границ», имея в виду, что внутреннюю структуру наборов данных можно определить динамически с помощью канальных программ. На базисно-логическом уровне внутренняя структура наборов данных должна быть в известном смысле однородной, чтобы доступ к отдельным частям или элементам файлов можно было обеспечить с помощью типовых канальных программ.

В свою очередь типизация канальных программ создает предпосылки полной автоматизации процесса составления программ канала для каждого конкретного способа обращения к наборам данных.

В отношении независимости программ от типов устройств к вышесказанному следует добавить, что она может быть достигнута только при условии, что структура файлов не будет зависеть от типа внешних устройств, на носителях которых располагаются наборы данных. Программу можно сделать независимой от типа устройств лишь в том случае, если канальную программу для исполнения требуемой операции ввода-вывода оперативно формировать непосредственно в процессе ее выполнения.

Например, пусть в одном случае прикладная программа работает с файлом, расположенным на магнитной ленте,

в другом случае — с файлом на магнитном диске. Если канальную программу составить до начала работы программы, то, естественно, она окажется способной работать только либо с магнитной лентой, либо с магнитным диском. При переходе с одного типа устройства на другой возникает необходимость переделки канальной программы, что фактически означает переделку всей программы.

Таким образом, если канальная программа для реализации операций ввода-вывода будет формироваться для устройства динамически в момент работы с учетом типа устройства, то она может быть использована без переделок для работы с разнотипными устройствами.

Автоматическое составление канальных программ для реализации поступающих запросов на ввод-вывод является одной из основных функций базисно-логического уровня управления данными. На данном уровне за программистом закреплены только такие функции, которые позволяют ему составлять необходимые описания (описание файлов, устройств, запросов и т. д.). Составление описаний осуществляется программистом путем использования специальных средств — системных макрокоманд.

Рассмотрим схему исполнения запроса на ввод-вывод в базисно-логической системе ввода-вывода, которая приведена на рис. 4.6. На схеме изображены три поля оперативной памяти: основной программы, программы базисно-логической и физической систем ввода-вывода.

В отличие от реализации операции ввода-вывода в физической системе, где она осуществляется с помощью команды «вызов супервизора», запрос в базисно-логической системе производится через команду «передача управления с возвратом» (переход к подпрограмме). По этой команде управление передается программам базисно-логического уровня СУД. В поле оперативной памяти, где расположены эти программы, выделим 4 части (см. рис. 4.6).

Программа обработки запроса выполняет функции предварительной обработки команды ввода-вывода. Во время обработки анализируется тип запроса, выбирается режим выполнения операции, подготавливаются необходимые области ввода-вывода (буферы) и т. д.

Далее управление получает программа, в которой формируется канальная программа и выдается требование на ее выполнение. Это требование является обращением к программам ввода-вывода физической СУД. Физическая система после запуска операции возвращает управление программам базисно-логического уровня, которые, в свою очередь, пере-

дают управление основной программе в точку, откуда пришел запрос на операцию ввода-вывода. После этого основная программа продолжает свою работу.

В области программ базисно-логической системы на рис. 4.6 выделены еще два участка, в которых находятся

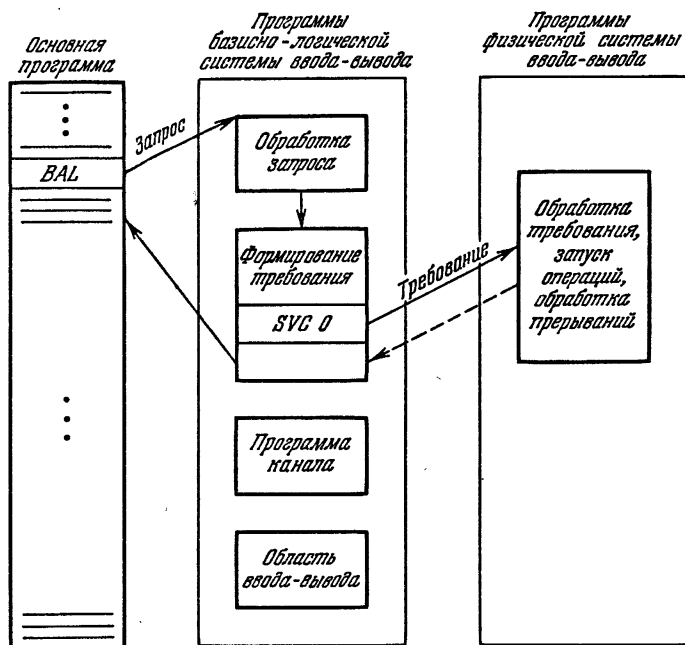


Рис. 4.6.

программа канала и область ввода-вывода (буфер). Программа канала конкретно настраивается для физического исполнения операции ввода-вывода каждый раз при формировании очередного требования. Из области ввода-вывода, в общем случае, выбирается один или несколько буферов для исполнения конкретной операции.

Кроме рассмотренного варианта схемы исполнения запроса на ввод-вывод программами базисно-логической системы управления данными, имеется еще один способ исполнения запроса. Этот способ отличается от рассмотренного, в сущности, только этапом окончания выполнения требования физической системой ввода-вывода. Если в способе, представленном на рис. 4.6, программы базисно-логической системы ввода-вывода возвращают управление

основной программе, то здесь основная программа не получает управления до тех пор, пока не завершится начатая операция ввода-вывода. В этом случае говорят, что основная задача находится в состоянии ожидания.

Для обоих способов исполнения запроса обработка прерываний по окончании ввода-вывода выполняется по различным схемам. Согласно первой схеме прерывается основная программа, которая получает управление в точке прерывания; согласно второй схеме в общем случае прерывается какая-то другая программа, а управление возвращается в основную программу в точке запроса. Можно отметить, что для основной программы, реализующей запрос на ввод-вывод по второй схеме, совмещение операции ввода-вывода со счетом отсутствует.

Появление двух схем исполнения запроса, в основном, обусловлено наличием двух уровней управления данными: базисного и логического. Отметим сразу, что исполнение запроса на ввод-вывод согласно схеме, приведенной на рис. 4.6, соответствует, как правило, базисной системе управления вводом-выводом, а другая схема (с ожиданием) соответствует логической системе.

На базисном уровне управления данными, в котором обрабатываемая информация представляет собой совокупность физических блоков, появляется возможность использовать свойство аппаратуры совмещать работу процессора с работой канала. Поэтому для базисной системы управления данными выбрана схема исполнения запросов на ввод-вывод с совмещением.

В логической системе управления данными один запрос на ввод-вывод, как правило, соответствует чтению одной конкретной части физического блока — логической записи. Так как физический блок обычно содержит несколько логических записей, то считывание физического блока происходит один раз на несколько логических записей. Совмещение работы канала с процессором на логическом уровне происходит не в процессе выполнения каждого запроса, а периодически через несколько однотипных запросов. В таком случае схема исполнения запроса с ожиданием оказывается лучше схемы с совмещением ввиду простоты реализации и большего удобства программисту, так как последнему в этом случае вообще не нужно думать о совмещении.

Наконец, рассмотрим вопрос о местонахождении программ базисно-логического уровня СУД на физических носителях и о расположении их в оперативной памяти на стадии выполнения.

Программы управления данными записаны в одной или нескольких библиотеках, отведенных для этой компоненты операционной системы. Часть из них, как, например, физическая система ввода-вывода, записывается в ядре и частично в библиотеке супервизора, а остальные либо в отдельной, либо в системной библиотеках.

Постоянно в оперативной памяти присутствуют только основные и наиболее часто используемые программы физической СУД, которые помещаются в ядро управляющей программы. Иногда в ядро помещают некоторые программы базисно-логической системы ввода-вывода. Остальные же программы в момент функционирования операционной системы постоянно находятся в библиотеках на магнитных дисках и вызываются в оперативную память только по мере необходимости.

Отметим только, что в различных операционных системах, например ОС и ДОС ЕС, способы вызова программ базисно-логического уровня в оперативную память различны. Сначала рассмотрим вопрос о том, в какую область памяти вызываются эти программы — в ядро или в область задач. В какой-то мере на этот вопрос помогает ответить схема, приведенная на рис. 4.6.

Содержательная часть программ обработки запросов и формирования требований определяется главным образом теми действиями, которые нужно выполнить, формируя каналную программу и выделяя буферы ввода-вывода. В свою очередь эти действия, способ их выполнения и конечный результат определяются только содержанием задачи и запрограммированными в ней средствами обработки данных.

Другими словами, программы канала, области ввода-вывода и содержание программ базисно-логического уровня очень сильно зависят от специфики задач, для которых они применяются. Причем зависимость настолько сильная, что содержимое участка оперативной памяти, где располагаются программы базисно-логического уровня, оказывается различным даже для однотипных запросов, относящихся к разным задачам.

Поэтому, в современных операционных системах программы базисно-логического уровня СУД в момент выполнения располагаются в области задач. Каждая задача, производящая обработку наборов данных, в своем разделе памяти имеет участок, в который помещаются системные программы базисно-логического уровня СУД.

Рассмотрим два способа вызова программ СУД в оперативную память: статический и динамический.

Статический способ состоит в том, что средства СУД становятся составной частью программы пользователя до ее исполнения. Обычно это происходит на этапах подготовки программы к выполнению (трансляция или редактирование). В этом случае программы базисно-логического уровня загружаются в оперативную память вместе с задачей пользователя. Этот способ является основным в операционной системе ДОС ЕС.

Динамический способ состоит в том, что программы СУД загружаются в область задачи, по мере необходимости, в момент ее выполнения. Этот способ характерен для операционной системы ОС ЕС.

4.5. Общая схема организации выполнения запросов в системе управления данными

Продолжая рассмотрение основных концепций, лежащих в основе системы управления данными, рассмотрим обобщенную схему организации исполнения запроса на базисно-логическом уровне. Такая схема приведена на рис. 4.7. По существу, эта схема мало чем отличается от схемы, изображенной на рис. 4.6. Разве только тем, что на рис. 4.7 имеются некоторые дополнительные элементы системы управления данными, ранее еще не упомянутые. Речь идет о различных таблицах, участвующих в работе программ СУД.

Область супервизора содержит целый ряд таблиц, описывающих системные ресурсы и задачи, выполняемые в режиме мультипрограммирования. К системным ресурсам относятся аппаратные (каналы, устройства) и программные ресурсы (обработчики прерываний, обработчики ошибок и т. д.), информация о которых находится в системных таблицах. Эти таблицы используются программами супервизора для выполнения своих функций. Некоторыми из таблиц пользуется физическая система ввода-вывода.

В области задачи находятся таблицы описания файла и ресурсов, содержащих сведения о физических характеристиках, используемых для обработки данных. Таблица файла описывает логические характеристики и свойства набора данных, подлежащего обработке. В таблице описания физических характеристик ресурсов содержатся сведения о типе носителей, о физических томах, на которых располагается набор данных, и т. д.

Приведенное распределение таблиц является довольно условным. Обычно таблицы ресурсов однозначно связаны

с описанием одного файла и являются как бы его продолжением, некоторые системные таблицы в момент исполнения запроса содержат адреса таблиц описания файлов и ресурсов.

Содержимое таблиц в области задач, их количество зависят от выполняемой программы обработки данных.

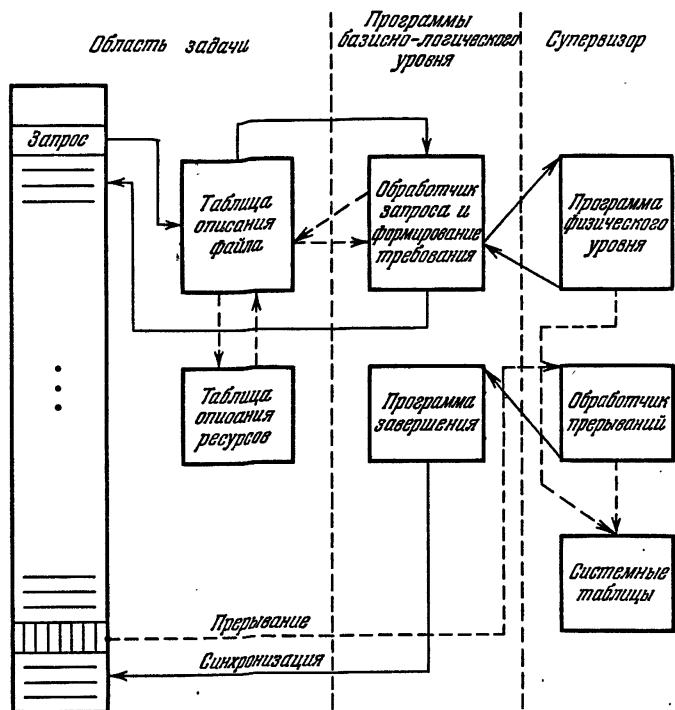


Рис. 4.7.

Эти таблицы создаются программистом, составляющим программу с помощью макрокоманд, входящих в систему управления данными. Отметим, что таблица описания файлов создается на этапе трансляции, а таблица описания ресурсов — обычно в момент выполнения программы.

Проследим ход исполнения запроса по рис. 4.7. Передача управления в программы базисно-логического уровня осуществляется по адресу, находящемуся в таблице описания файла. Программы базисно-логической системы ввода-вывода, используя соответствующие таблицы, обрабатывают запрос и формируют требование на выполнение опе-

рации в физической системе управления данными. Обращение к физической системе, как отмечалось ранее, происходит с помощью команды «вызов супервизора».

Программы физического уровня управления данными, пользуясь системными таблицами и таблицами задачи, обеспечивают запуск операции по канальной программе и через программы базисно-логического уровня возвращают управление задаче, которая продолжает счет. По сигналу прерывания управление вновь переходит к физической системе ввода-вывода в обработчик прерываний. Из обработчика прерываний через программы завершения операции управление возвращается в программу для контроля и синхронизации окончания ввода-вывода.

Еще раз отметим, что большую роль в описанных процессах играют таблицы описаний файлов и ресурсов, а также системные таблицы. Прежде чем начинать работу с конкретным файлом, нужно построить таблицу его описания, которая создается в момент проектирования программы, связать ее с другими таблицами, в том числе и с системными. Иными словами, требуется предварительная подготовка таблиц, в процессе которой выполняются некоторые действия с внешними устройствами. Если, например, файл находится на магнитном диске, то осуществляется его поиск и настройка на чтение первого блока данных.

Все подготовительные действия для работы с файлом выполняются специальной программой, являющейся частью системы управления данными, которая вызывается с помощью команды «вызов супервизора» и называется макрокомандой «открыть» (OPEN).

После окончания работы с файлом его нужно «закрыть» с помощью макрокоманды (CLOSE), которая выполняет действия, в некотором смысле обратные действиям макрокоманды «открыть». Более подробно функции макрокоманд «открыть» и «закрыть» будут рассмотрены позже.

Перейдем к рассмотрению некоторых специфических особенностей системы управления данными.

4.6. Особенности системы управления данными

Синхронизация операций счета и ввода-вывода. Появление задачи синхронизации обусловлено свойством моделей ЕС ЭВМ обеспечивать совмещение операций счета с операциями ввода-вывода. Эта задача имеет место для всех уровней управления данными. Различные уровни управления

данными отличаются друг от друга только способами решения задачи синхронизации. Рассмотрим этот вопрос более детально.

На рис. 4.8 изображен участок программы с одним запросом на ввод данных. По этому запросу через программы системы управления данными (на рисунке они не

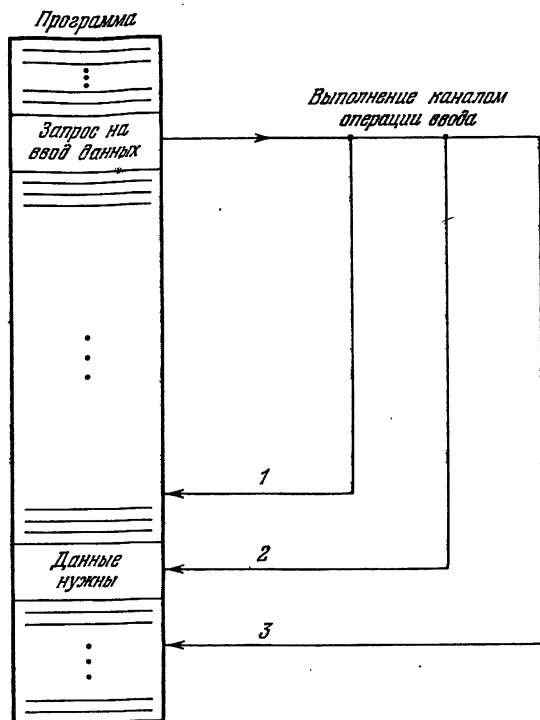


Рис. 4.8.

показаны) запускается канал для выполнения ввода данных. Пока канал осуществляет ввод, процессор продолжает выполнять программу.

Выделим в программе еще одну точку, в которой данные, вводимые по запросу, должны быть использованы для дальнейшей работы программы. Поскольку ввод данных осуществляется в общем случае в условиях мультипрограммирования автономным устройством — каналом, то теоретически имеются три возможные ситуации к моменту, когда данные нужно использовать.

1 — канал и система управления данными осуществили ввод данных раньше, чем они потребовались (1).

2 — канал и система управления данными закончили ввод точно к моменту, когда данные потребовались (2).

Оба эти случая означают, что программу можно просто продолжить.

3 — канал и система управления данными не успели завершить операцию ввода к моменту, когда потребовались данные (3). В этой ситуации необходимо предусмотреть «ожидание» завершения ввода. В противном случае результат работы программы окажется неверным, так как может оказаться, что она обрабатывает не те данные, которые должны быть введены.

Еще раз отметим, что момент завершения операции ввода-вывода для конкретного запроса является случайным событием. Практически определить, какой из трех случаев будет иметь место в конкретной ситуации, невозможно. Проследим путь выполнения запроса в условиях мультипрограммирования. Сначала, в общем случае, работают программы базисно-логического уровня, время выполнения которых поддается расчету. Затем поступает требование в программы физического уровня, где рассчитать точное время работы системных программ невозможно. Обусловлено это неоднозначностью их поведения в процессе выполнения требования, имеющем две альтернативы: первая — операция запущена и вторая — операция не запущена и запрос на ее исполнение поставлен в очередь.

Во второй ситуации не известны ни длина очереди, ни момент освобождения ресурса. Поэтому точный расчет времени нужного интервала практически невозможен. Отметим только, что на время выполнения одной операции ввода-вывода влияют такие факторы, как скорость работы внешних устройств, уровень управления данными, приоритет задачи, количество одновременно выполняемых задач при мультипрограммировании, число и частота возможных обращений к ресурсам и т. д.

Вообще говоря, задача синхронизации в условиях мультипрограммирования возникает не только при выполнении операций ввода-вывода. Супервизор, распределяя аппаратные и программные ресурсы, часто сталкивается с этой задачей. Поэтому, например, в ОС ЕС разработаны и включены в супервизор стандартные средства синхронизации — макрокоманды «ждать» (WAIT) и «установить» (POST), которые в совокупности можно представить как некоторый формальный аппарат для программирования

часто возникающих ситуаций синхронизации. Этот же аппарат используется и в системе управления данными для решения задачи синхронизации. Способ его использования определяется уровнем управления данными.

На физическом уровне используется аппарат супервизора в «чистом» виде. На базисном уровне он немного усовершенствован и реализован в виде макрокоманды «проверить» (СНЕСК), которая выполняет не только функции синхронизации, но и проверку, сколь успешно завершена операция ввода-вывода.

На логическом уровне аппарат синхронизации встроен в программы логической системы ввода-вывода, т. е. синхронизация осуществляется в программах логического уровня СУД, которые возвращают управление задаче только после выполнения заданной операции ввода-вывода. Таким образом, на логическом уровне управления точкой, где запрашиваемые данные можно использовать для обработки, является следующая после запроса команда.

Объединение записей в блоки. Как уже отмечалось, на логическом уровне управления данными физический блок представляет собой совокупность логических записей, к каждой из которых в отдельности разрешен доступ. При этом естественно возникает вопрос: «а нельзя ли каждую отдельную логическую запись оформлять в виде блока?» В принципе, конечно, можно, однако логический уровень управления данными, который является, пожалуй, самым популярным среди пользователей, располагает двумя существенными преимуществами, заложенными при объединении записей в блоки (блокирование):

- 1) более высокую скорость обработки;
- 2) высокий коэффициент использования объема памяти магнитных носителей (лент и дисков).

Рассмотрим указанные достоинства на примере, приведенном на рис. 4.9.

На рис. 4.9 изображены два участка магнитной ленты одинаковой длины с записанной информацией. На верхнем участке (вариант *а*) данные записаны в четырех блоках. Напомним, что блок от блока на магнитной ленте стоит на расстоянии примерно 1,5 см. Предположим, что каждый физический блок на верхнем отрезке соответствует логической записи. Время обработки информации, представленной в 4-х блоках, будет, в основном, определяться временем, необходимым для чтения блоков с магнитной ленты. Для чтения 4-х блоков нужно 4 раза запустить канал и устройство, на что уходит значительное время.

На этом отрезке магнитной ленты (вариант *а*) отношение объема записанной полезной информации к максимально возможному составляет порядка 0,1.

На нижнем отрезке магнитной ленты (вариант *б*) такие же логические записи объединены в блоки, каждый

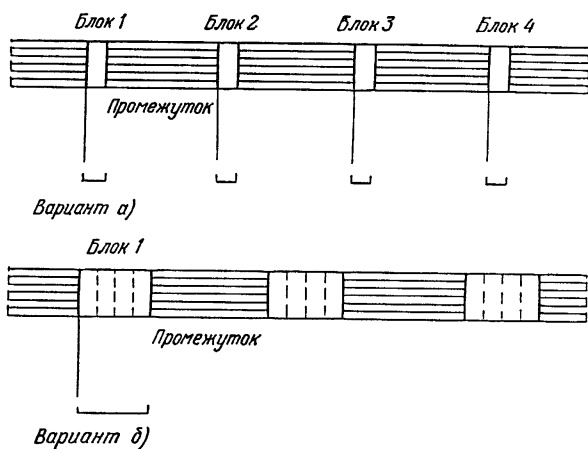


Рис. 4.9.

блок содержит 4 логические записи. При обработке информации, записанной таким способом, обращение к каналу и устройству происходит в 4 раза реже, чем для верхнего отрезка, т. е. скорость обработки возрастает практически в 4 раза. Коэффициент использования памяти магнитной ленты в данном случае увеличивается также в 4 раза. Примерно так же обстоит дело с накопителями прямого доступа (дисками).

Отметим только, какой ценой дается каждое из перечисленных преимуществ блокирования записей.

1) Более высокая скорость обработки данных, обеспечиваемая сокращением количества обращений к каналам и устройствам, требует выполнения операции деблокирования, т.е. выделения из блоков отдельных логических записей. На логическом уровне — это одна из стандартных функций программ системы ввода-вывода.

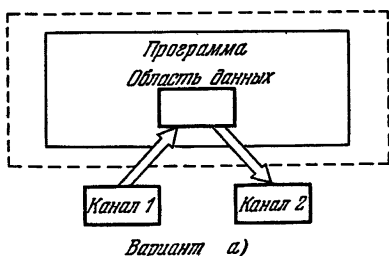
2) Увеличение коэффициента использования объема внешней памяти осуществляется за счет увеличения размера физических блоков. А это, в свою очередь, требует выделения большего объема оперативной памяти для области ввода-вывода. Безгранично увеличивать размер области

ввода-вывода не представляется возможным. Поэтому практически размер физического блока имеет некоторый оптимальный предел, определяемый конкретными условиями,

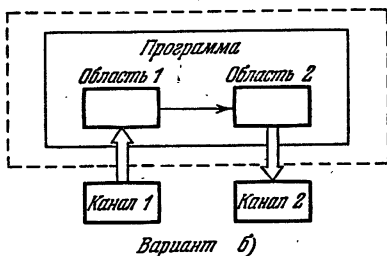
такими, как, например, объем оперативной памяти ЭВМ, размер области задачи и т. д.

Вопросы буферизации.

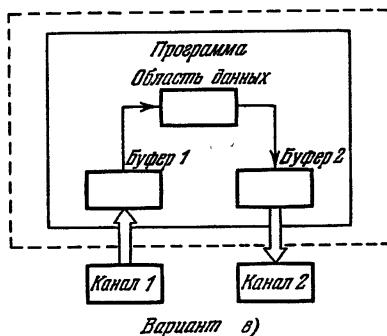
Большое влияние на скорость обработки данных, на методы программирования операций ввода-вывода оказывают вопросы буферизации. На примере простой задачи покажем, в чем состоят указанные вопросы. Рассмотрим их на задаче переноса данных с одного физического устройства на другое. Причем предполагаем, что оба устройства подсоединены к двум различным каналам ЭВМ. Обратимся к рис. 4.10. На рис. 4.10 приведены три различных варианта организации взаимодействия программы с каналами: а, б, в. Устройство, с которого считываются данные, подсоединено к каналу 1, а на которое записываются — к каналу 2. Пунктирный прямоугольник



Вариант а)



Вариант б)



Вариант в)

Рис. 4.10.

соответствует некоторому участку оперативной памяти, а сплошной — программе, выполняющей копирование.

Вариант а). В программе выделена область данных, которая используется как для ввода, так и для вывода информации. Очевидно, что использовать в программе принцип совмещения операций счета с вводом-выводом невозможно. Каналы работают последовательно без какого-либо перекрытия. Данный вариант соответствует самой низкой скорости функционирования программы.

В а р и а н т б). В программе имеются две области: область 1 для ввода и область 2 для вывода данных. При такой схеме организации ввода-вывода становится возможным использовать принцип совмещения для обработки данных, но, в основном, только для одного из каналов. После заполнения области 1 данные переносятся в область 2, при этом канал 1 запускается для чтения следующей порции, а в области 2 программа может обрабатывать данные. Когда в области 2 обработка закончится, программа запускает канал 2 для записи информации на другое устройство, т. е. при определенных условиях такая схема позволяет совмещать работу обоих каналов.

Приведенный вариант взаимодействия с каналами обеспечивает более высокую скорость выполнения программы. Однако этот вариант требует больше усилий со стороны программиста. Во-первых, он должен предусмотреть две области ввода-вывода, а, во-вторых, позаботиться о пересылке данных из одной области в другую.

В а р и а н т в). В программе имеются три области: область данных, буфер 1 и буфер 2. Канал 1 вводит данные в буфер 1, тем временем программа обрабатывает данные, находящиеся в области данных, а канал 2 выводит данные из буфера 2. Эта схема организации взаимодействия допускает совмещение операций счета с вводом-выводом на обоих каналах, причем оба канала могут работать одновременно. Программа, составленная таким способом, выполняется с наибольшей скоростью по сравнению с вариантами а и б. Однако этот вариант требует еще больше усилий со стороны программиста: он должен построить 3 области ввода-вывода и позаботиться о своевременной пересылке данных из буфера 1 в область данных и из области данных в буфер 2. Таким образом, к вопросам буферизации относятся:

- 1) построение методики и способов выделения необходимого количества областей ввода-вывода (буферов);
- 2) организация связей между буферами при взаимодействии программы с каналами.

На базисном и логическом уровнях системы управления данными эти задачи решаются по-разному.

Для базисного уровня характерно наличие только некоторых средств для построения нужного количества буферов. Связи между буферами организуются программистом.

На логическом уровне возможно автоматическое выполнение основных функций буферизации системными программами управления данными.

Завершение операций ввода-вывода и обработка ошибок. Операции ввода-вывода в системе управления данными операционной системы завершаются по сигналу прерывания от каналов. По этому сигналу управление передается сначала обработчику прерываний, находящемуся в составе физической системы ввода-вывода. Обработчик проверяет правильность выполнения операций каналом и устройством сначала по содержимому слова состояния канала (ССК), затем, если это необходимо, анализирует байты уточненного состояния, предварительно запросив их через канал. Анализ байтов уточненного состояния проводится только в случае обнаружения сбойной ситуации (ошибки ввода-вывода).

На физическом уровне управления данными обрабатываются ошибки, фиксируемые аппаратными средствами ЭВМ в управляющих словах. Если анализ ошибки показывает, что операцию можно попытаться повторить, то соответствующий обработчик передает управление программе запуска канала для повторного выполнения операции. Такое повторение возможно несколько раз. При неудачных попытках выполнения требуемой операции ввода-вывода обработчик возвращает управление программам более высокого уровня управления данными и сообщает им, что операция не завершена. Причины невыполнения также сообщаются соответствующим программам.

Если же операция ввода-вывода закончилась нормально, то программы базисно-логического уровня получают соответствующее подтверждение. В случае нормального завершения дальнейшими действиями программ СУД являются обеспечение синхронизации и деблокирования.

В сбойных ситуациях анализ ошибки продолжается на более высоком уровне способами, определяемыми программистом, который может, например, в случае сбоя: аварийно завершить задачу; проигнорировать сбой; пропустить данную операцию ввода-вывода.

В первом случае задача продолжаться не может. Во втором случае программа продолжает обработку данных, несмотря на сбой. В третьем случае операция, в процессе выполнения которой произошел сбой, игнорируется.

Кроме указанных возможностей, являющихся стандартными, программист может использовать полученную информацию о сбое для печати сообщения об ошибке и для смысловой обработки ошибки с помощью программы, специально предусмотренной им для этих целей. Программы базисно-логической СУД позволяют программисту обрабатывать ошибки всеми упомянутыми выше способами.

ВНЕШНИЕ ЗАПОМИНАЮЩИЕ УСТРОЙСТВА И ОРГАНИЗАЦИЯ РАЗМЕЩЕНИЯ ДАННЫХ

5.1. Особенности внешних запоминающих устройств ЕС ЭВМ

Накопители на магнитной ленте (НМЛ). Рассмотрим некоторые элементы НМЛ. Верхняя схема рис. 5.1 представляет тракт движения магнитной ленты в лентопротяжном механизме. На средней схеме рис. 5.1 изображена схема магнитной головки механизма. При записи информации на магнитную ленту левая головка (записывающая) осуществляет запись, а вторая (считывающая) читает только что записанную информацию. Такое устройство головки позволяет осуществлять динамический контроль записи. Если прочитанное не совпадает с тем, что должно быть записано, то выдается сигнал об ошибке. При чтении информации с магнитной ленты обе головки используются для чтения, контроль осуществляется сравнением результатов чтения разными головками.

Нижняя схема рис. 5.1 представляет собой последовательность блоков, записанных на магнитную ленту. На правом конце магнитной ленты выделен участок, названный физическим маркером, который представляет собой полосу алюминиевой фольги, отражающей свет и наклеиваемой на магнитную ленту примерно в 3—4 метрах от начала. Эта точка магнитной ленты называется точкой загрузки. После точки загрузки через некоторый промежуток записан блок информации, а за ним также через промежуток — другой блок информации и т. д. В конце последовательности блоков записывают специальный блок, который является индикатором того, что последовательность блоков закончилась. Этот специальный блок называют маркой ленты (ТМ). Марку ленты устройство управления отличает от блока данных.

Размеры блоков на магнитной ленте могут быть от 18 байтов (минимально допустимая длина) до величины, определяемой объемом оперативной памяти ЭВМ.

Промежутки между блоками, как правило, имеют длину 1,5 см.

Информация на магнитную ленту пишется девятидорожечным способом — побайтно (8 байтов информационных

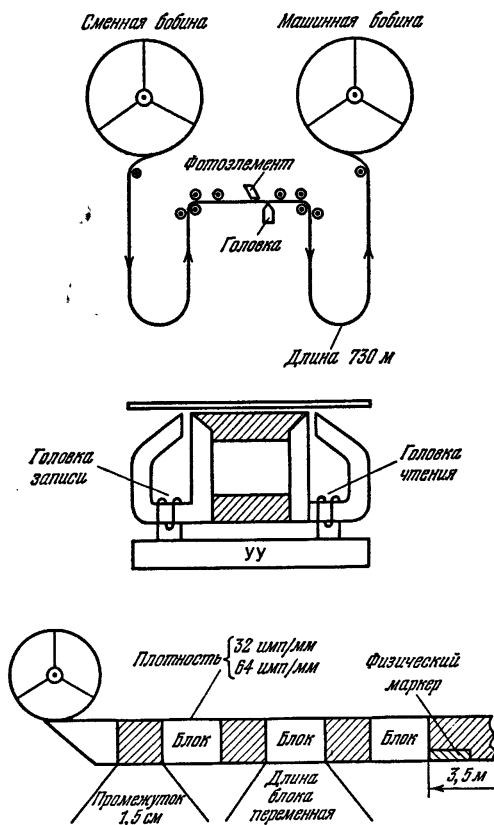


Рис. 5.1.

и 1 контрольный) с плотностями 8, 16, 32, 64 байта на миллиметр. На конце наклеивается такой же физический маркер, как и в начале, для фиксации физического конца магнитной ленты.

Устройство управления лентопротяжным механизмом позволяет выполнять следующие операции:

- чтение в прямом и в обратном направлении;
- запись информации в прямом направлении;
- перемотка магнитной ленты к точке загрузки;

- запись марки магнитной ленты;
- продвижение вперед и назад на один блок;
- продвижение вперед и назад на одну группу блоков (при этом магнитная лента движется до следующей марки);
- установка плотности записи информации на магнитную ленту и ряд других специальных действий.

Важная роль отводится ленточной марке. Этот блок используется для индикации конца данных. Например, если будет выдана команда «чтение» для ТМ, то марка, пройдя под головкой, вызовет установку бита «особый случай в устройстве» байта состояния устройства в ССК. Этот бит обычно используется как сигнал конца наборов данных.

Накопители на магнитном диске (НМД). Накопители на магнитной ленте обладают одним существенным недостатком. Время обращения к информации на магнитной ленте зависит от места ее расположения. Если данные записаны где-то в конце ленты, то потребуются много времени для их поиска (до нескольких десятков секунд). От указанного недостатка свободны «запоминающие устройства с прямым доступом» (ЗУПД). К этой категории относятся те устройства, у которых время, необходимое для обращения к любой информации, практически не зависит от места ее расположения (магнитные диски).

На магнитных дисках информация записывается на алюминиевые диски, покрытые ферромагнитным слоем, методами, аналогичными методам записи на магнитную ленту. Магнитные диски объединяются в пакеты, пакет дисков устанавливается на дисковом механизме (дисковом).

На рис. 5.2 схематично представлен семимегабайтный накопитель на магнитных дисках.

Пакет состоит из магнитных дисков. Каждый диск имеет две рабочие поверхности — верхнюю и нижнюю. Исключение составляют верхняя поверхность верхнего диска и нижняя — нижнего, так как они больше всего подвержены повреждениям. Информация на рабочих поверхностях дисков записывается по дорожкам, представленным концентрическими окружностями. Каждая поверхность диска рассчитана на 203 таких дорожки (трека). Все дорожки имеют физическое начало, фиксируемое отверстием на нижнем диске пакета.

После установки пакета на дисковод и включения механизма пакет дисков достигает скорости вращения 2400 оборотов в минуту, и специальные щетки очищают пакет от пыли, которая может ухудшить качество записи и чтения. Затем специальный держатель головок вдвигает их

между дисками так, что 10 головок располагаются друг над другом, образуя цилиндр (рис. 5.2).

Учитывая, что каждая рабочая поверхность содержит 203 дорожки, можно рассматривать пакет как запоминающее устройство, содержащее 203 цилиндра, в каждом из

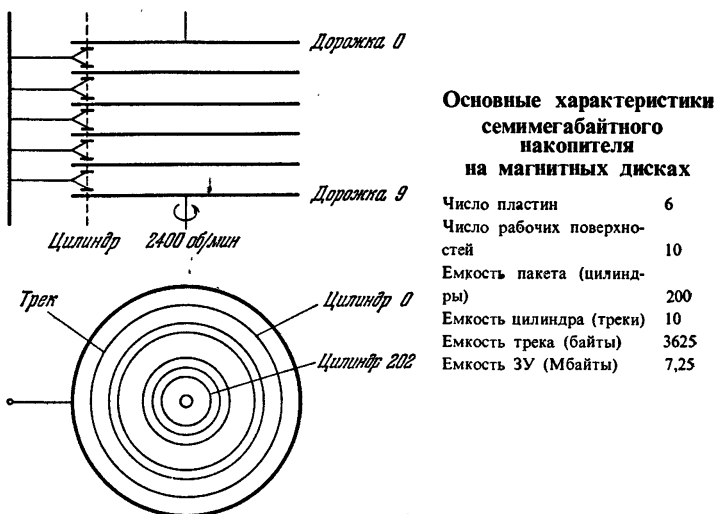


Рис. 5.2.

которых имеется 10 дорожек. Такое представление памяти позволяет уменьшить количество перемещений головок (от цилиндра к цилиндру) при обработке последовательно записанной информации. Записывается же информация сначала в пределах одного цилиндра, а затем головки перемещаются на следующий цилиндр. Внутри цилиндра каждая дорожка имеет свой номер от 0 до 9. Все цилиндры также перенумерованы от 0 до 202.

Обычно на диске используются в качестве основных 200 первых цилиндров. Остальные три являются запасными. Если какая-либо основная дорожка становится дефектной, то вместо нее используют одну из запасных, которую иногда называют альтернативной.

Любая дорожка пакета семимегабайтных магнитных дисков способна вместить 3625 байтов, таким образом, плотность записи на дорожках одной поверхности различна. Основные характеристики семимегабайтного накопителя на магнитных дисках приведены на рис. 5.2.

29-мегабайтные накопители на магнитных дисках, являющиеся в настоящее время наиболее распространенными, отличаются от семимегабайтных накопителей только количеством рабочих поверхностей (20) и емкостью одной дорожки (7250 байтов).

Рассмотрим теперь процессы обращения к данным, записанным на магнитных дисках. Прежде всего устройство управления обеспечивает подвод головок к нужному цилиндру, затем путем выбора одной из головок обеспечивает доступ к конкретной дорожке. После этого необходимо обеспечить поиск нужной информации в пределах одной дорожки.

Каждая дорожка диска имеет жестко заданную структуру хранения информации. Структура в целом и ее отдельные элементы выбраны, исходя из необходимости обеспечить:

- 1) жесткий контроль как правильности функционирования устройства, так и правильности записи и считывания информации;

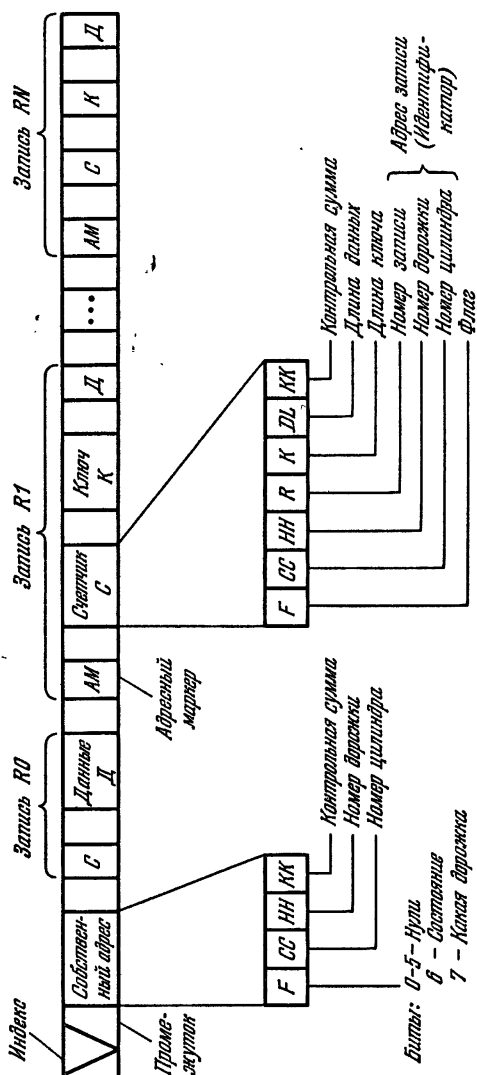
- 2) максимальную скорость поиска нужной информации на дорожке;

- 3) максимальные удобства при программировании процессов обработки данных, расположенных на магнитных дисках.

Структура дорожки изображена на рис. 5.3. В начале дорожки расположен индекс, соответствующий физическому началу. После индекса через промежуток записывается собственный адрес дорожки. Он содержит номер цилиндра и номер трека этой дорожки, которые используются для контроля устройства. Вслед за собственным адресом помещается запись R0, называемая описателем дорожки. Эта запись предусмотрена для программного отслеживания заполнения дорожки, она используется программами системы управления данными. Запись R0 состоит из двух элементов: счетчика и данных, разделенных небольшим стандартным промежутком (Π). Отметим, что все отдельные элементы дорожки диска разделяются промежутком длиной примерно в 18 байтов.

Далее (после R0) следует первая запись данных (R1), за ней вторая запись (R2) и т. д. В конце дорожки расположена запись RN. Количество записей на дорожке колеблется от 1 до 255. Любая запись, кроме (R0), может состоять из 4 элементов.

Адресный маркер (AM) — специальный символ, предшествующий всем записям. Счетчик содержит адрес записи



Биты: 0 - Чёт-нечёт
 1 - Переопределение
 2-5 - Нули
 6 - Системные
 7 - Канал дорожки

Рис. 5.3.

(идентификатор) и информацию о длине двух других элементов записи.

Ключ является не обязательным полем и может отсутствовать (как, например, в записи R0). Но если ключ имеется, то он содержит информацию пользователя длиной от 1 до 255 байтов, которая может быть использована для поиска данной записи (поиск по ключу).

Данные — поле, предназначенное для информации пользователя, длина его определяется доступным количеством места на дорожке.

Адресный маркер и счетчик являются служебной информацией, сопровождающей каждую порцию данных на дорожке, они используются устройством управления для контроля и ускорения поиска информации.

Структура счетчика приведена на рис. 5.3. Общая длина счетчика равна 11 байтов. Первый байт (флаг) используется только устройством управления, никогда не считывается и не записывается программой в оперативную память. Вторым и третьим байтами содержат номер цилиндра. Четвертый и пятый — номер дорожки внутри цилиндра. Шестой байт — номер записи на дорожке. Байты со второго по шестой включительно образуют идентификатор (адрес) записи, используемый для контроля и для поиска. Седьмой байт содержит информацию о длине ключа. Восьмой и девятый — указывают длину поля данных. Десятый и одиннадцатый байты — байты циклической проверки (контрольной суммы).

Устройство управления НМД вычисляет контрольную сумму байтов, записываемых на диск, и помещает ее в конце каждого элемента. При чтении элемента контрольная сумма считываемых байтов вновь вычисляется и сравнивается с записанной. Эти байты, как и первый байт счетчика, никогда не считываются и не записываются программой в оперативную память.

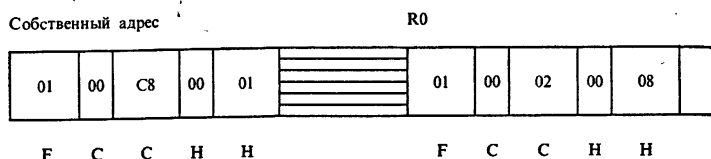
Если основная дорожка приходит в негодность, то вместо нее используют альтернативную дорожку. При этом собственный адрес основной дорожки соответствует этой дорожке, а номера цилиндров и треков в записях на такой дорожке совпадают с номерами цилиндров и треков альтернативной дорожки. Для альтернативной дорожки собственный адрес соответствует этой дорожке, а поле счетчика записей содержит адрес записей основной дорожки (рис. 5.4).

Перечислим основные команды, которые может выполнять устройство управления НМД.

Команды установки (ПОДВОД) головок обеспечивают перемещение механизма головок и установку головок на нужный цилиндр.

Команды поиска обеспечивают:

1) поиск идентификатора и выполнение одной из трех операций: $=$ или \leq или \geq (идентификатор счетчика сравнивается с заданным идентификатором);



Альтернативная дорожка

Рис. 5.4.

2) поиск ключа и выполнение одной из трех операций сравнения: $=$ или \leq или \geq ;

3) поиск ключа и данных и выполнение одной из трех операций сравнения: $=$ или \leq или \geq .

Команды чтения (записи) обеспечивают чтение (запись):

1) счетчика; 2) данных; 3) ключа и данных; 4) счетчика, ключа и данных. Кроме этого, имеются еще некоторые специальные команды управления.

Определим некоторые понятия, широко используемые в дальнейшем системой управления данными. Непрерывно адресуемая область памяти на диске называется *экстен-том*. Обычно файл располагается на диске, занимая один или несколько экстенгов.

В предыдущей главе были описаны уровни представления объектов системы ввода-вывода и, в частности, были рассмотрены файлы, состоящие из физических блоков. Что соответствует понятию блока в накопителях на магнитных дисках? Блок в НМД представлен записью, располагаемой на дорожке между двумя адресными маркерами. Адресный маркер является границей блока. Благодаря наличию адресного маркера время поиска записи на дорожке умень-

шается в среднем в 2 раза. Если бы адресный маркер отсутствовал, то для того, чтобы начать поиск конкретных записей, необходимо было бы ждать момента, когда под головкой пройдет индекс, т. е. начало дорожки. При наличии адресного маркера устройство может начинать операцию поиска сразу же после его прохождения под головкой, не дожидаясь появления начала дорожки. На этом в среднем экономится время, равное половине оборота диска.

Физический блок на НМД, кроме данных, содержит вспомогательную или служебную информацию (ключ, счетчик), эта информация используется программами системы управления данными для увеличения эффективности обработки данных.

5.2. Записи наборов данных и их форматы

Ранее набор данных был определен как совокупность взаимосвязанных записей. Записи, из которых состоят наборы данных, могут быть трех типов.

Записи фиксированной длины (формат F). В этом случае все записи файла имеют одинаковую длину.

Записи переменной длины (формат V). Логические записи файла имеют различную длину, однако внутри каждой записи содержится информация о длине записи.

Записи неопределенной длины (формат U). Эти записи характеризуются произвольной длиной и отсутствием точного указания ее длины.

Рассмотрим, как соотносятся эти три типа записей к различным уровням управления и представления объектов системы ввода-вывода.

Физический уровень. Размеры физических блоков определяются канальными программами и конкретными устройствами. Обычно заранее сказать о размерах блоков на этом уровне нельзя. Поэтому на физическом уровне используются записи неопределенной длины.

Базисный уровень. На этом уровне структура и состав файлов лимитируются некоторым способом, который необходим в основном для автоматизации процесса составления канальных программ. Файл на этом уровне состоит из однотипных записей. Используются записи (блоки) форматов F, V, U.

Логический уровень. Блоки данных, обрабатываемых на логическом уровне, состоят из однотипных логических записей. Логические записи могут быть фиксированной и переменной длины. Говорят, что блоки (фи-

зические записи) на логическом уровне имеют формат FB, если записи фиксированной длины, и формат VB, если записи переменной длины.

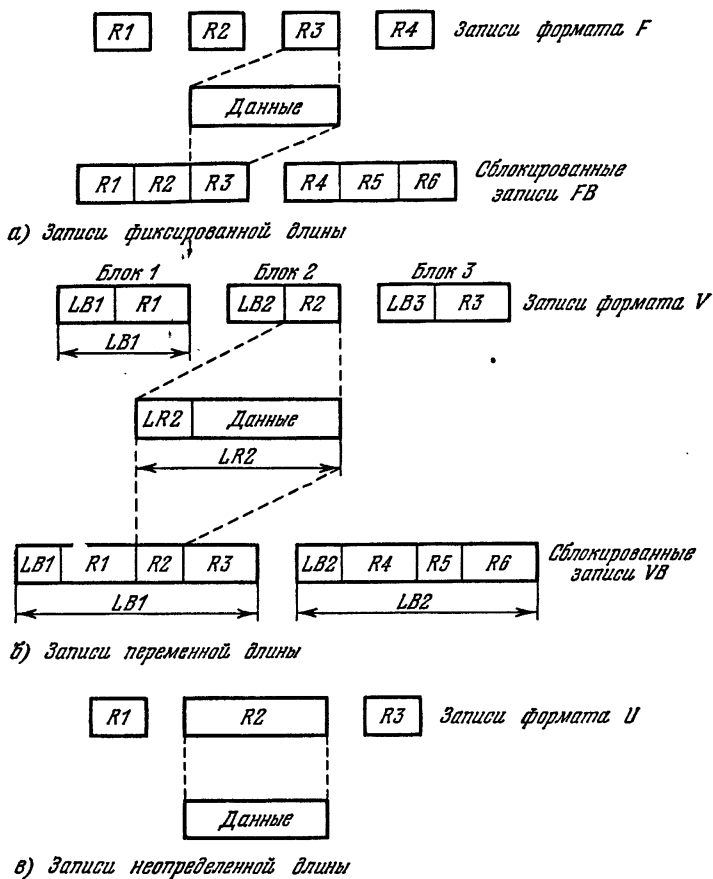


Рис. 5.5.

Форматы записей основных типов приведены на рис. 5.5. Записи фиксированной длины (рис. 5.5, а) пояснений не требуют. Записи переменной длины (рис. 5.5, б) представляются форматами V и VB. Для записей формата V каждый блок сопровождается дополнительным 4-байтовым указателем длины LB, каждая запись в блоке сопровождается также дополнительным 4-байтовым указателем длины LR (рис. 5.5). Количество дополнительных

байтов в блоке определяется выражением $(4 + 4 \times K)$, где K — коэффициент блокировки, указывающий число логических записей в одном блоке. На рис. 5.5 записи формата VB приведены в соответствии с коэффициентом блокировки, равным 3.

Записи неопределенной длины определяются только форматом U (рис. 5.5, в). Очевидно, что блокирование такого типа записей смысла не имеет.

5.3. Организация наборов данных

Последовательная организация файлов. Для последовательной организации файлов характерно последовательное расположение данных на физическом носителе, а также последовательный порядок их обработки. Такие файлы называют физическими последовательными, или стандартными последовательными, или просто последовательными. Файлы, расположенные на перфокартах, АЦПУ и магнитных лентах, всегда являются последовательными.

Как было отмечено, для подобных файлов последовательный характер обработки данных является определяющим. Такой характер обработки используется на внешних устройствах большинства типов, в том числе и на магнитных дисках. В последнем случае магнитный диск будет использоваться, в сущности, в режиме достаточно быстрого накопителя на магнитной ленте. В файлах с последовательной организацией можно использовать записи форматов всех типов. Операционные системы ЕС ЭВМ имеют мощные средства для обработки последовательных файлов.

Индексно-последовательная организация файлов. Запоминающее устройство с прямым доступом (ЗУПД) позволяет вести не только последовательную обработку данных, но допускает большое число разнообразных методов и способов организации данных. Одним из самых распространенных методов является метод индексно-последовательной организации. В этом методе используется свойство НМД записывать и осуществлять поиск записей по ключу. Если записи рассортировать в порядке возрастания значений их ключей, то становится возможным организовать быстрый поиск записей по ключам, не просматривая все записи файла. Для этого строится одна или несколько специальных таблиц, называемых *индексами*. В индексах с точностью до каждой дорожки записывается информация о ключах, по которой легко определить, на

какой дорожке находится запись с конкретным значением ключа.

Индексно-последовательный файл (ИПФ) строится обычно следующим образом. Записи с полем ключа помещаются в область экстенда, которая состоит из целого числа подряд расположенных цилиндров. Каждый цилиндр этого экстенда организован специальным способом. Нулевая дорожка не содержит записей с данными, а используется для хранения индекса дорожек. Данные обычно записываются на нескольких дорожках, начиная с первой. В особых случаях, предусматриваемых программистом, одна или несколько дорожек цилиндра используются в качестве области переполнения. Эта область используется тогда, когда в индексно-последовательный набор данных добавляются записи.

В индексе дорожек записываются адреса всех дорожек данных с указанием ключа последней записи на соответствующих дорожках. Для того чтобы определить местоположение любой записи в пределах данного цилиндра, требуется один оборот диска для поиска дорожек, следовательно, доступ к любой записи из области данных в цилиндре осуществляется за два оборота диска.

Если набор данных располагается на нескольких цилиндрах, то поиск удлиняется. Поэтому в индексно-последовательном файле обязательно строится индекс цилиндров, который располагается в другом экстенде. Индекс цилиндров содержит такую же информацию, что и индекс дорожек, с той лишь разницей, что эта информация относится к целому цилиндру.

Наконец, для очень больших наборов данных применяется один или даже несколько главных индексов, в которых находится информация для небольших групп цилиндров или главных индексов. Например, в операционной системе ОС ЕС допускаются четыре главных индекса, а в системе ДОС ЕС только один.

Для иллюстрации принципа индексно-последовательной организации обратимся к примеру, приведенному на рис. 5.6.

На рис. 5.6 изображена часть индексно-последовательного файла, расположенная на цилиндрах 6 и 7. На каждом цилиндре область данных занимает дорожки 1, 2, 3, а область переполнения цилиндра — 8 и 9. Нулевая дорожка хранит индекс дорожек.

В индексе дорожек каждого цилиндра находится по четыре элемента. Первые три описывают дорожки области данных, а последний является признаком конца данного

	6/37	7/63		Индекс цилиндров
--	------	------	--	---------------------

Цилиндр 6

Дорожка

0	1/17	2/23	3/37	FFF	Индекс дорожек
1	11	14	17		
2	20	21	23		
3	29	31	37		
.					
.					
.					
9	Область переполнения цилиндра 6				

Цилиндр 7

Дорожка

0	1/42	2/46	3/63	FFF	Индекс дорожек
1	39	41	42		
2	44	45	46		
3	51	62	63		
.					
.					
.					
9	Область переполнения цилиндра 7				

Рис. 5.6.

индекса. Содержимое элемента индекса дорожки определяет номер соответствующей дорожки и значение ключа последней записи на этой дорожке. Например, второй элемент индекса дорожек цилиндра 7 содержит номер дорожки, равный 2, и значение ключа 46 (см. рис. 5.6).

На рис. 5.6 выделена также часть области индекса цилиндров, в которой находится индекс цилиндров 6 и 7. Соответствующие им элементы содержат номера цилиндров и значения ключей последних записей в цилиндрах. Пусть требуется найти в ИПФ на рис. 5.6 запись с номером 51. Для этого сначала просматривается индекс цилиндров, по которому определяется, что искомая запись находится на цилиндре 7. Затем по содержимому индекса дорожек цилиндра 7 находится адрес дорожки, на которой расположена нужная запись — дорожка 3. После этого запись 51 может быть считана в оперативную память для обработки.

В режиме дополнения записей в ИПФ используются области переполнения цилиндров, а если и они заполняются до предела, то используется независимая область переполнения, располагаемая в отдельном экстенсте.

В индексно-последовательных наборах данных допустимо применение записей только форматов типа F, V, FB и VB. Отметим, что в операционной системе ДОС ЕС для ИПФ могут использоваться только записи типа F и FB, в ОС ЕС — все четыре указанных выше формата.

Прямая организация файлов. Свойства ЗУПД позволяют создавать и обрабатывать не только наборы данных с индексно-последовательной организацией. Гибким методом организации, приспособленным к свойствам дисковых накопителей, является прямой способ организации файлов.

Прямой способ организации, как и индексно-последовательный, делает возможной эффективную обработку данных путем значительного сокращения времени поиска отдельных записей в файле. Однако, в отличие от индексно-последовательной, при прямой организации не используются служебные таблицы (индексы), облегчающие процесс и сокращающие время поиска. Основной отличительной особенностью прямой организации является то, что фактическую организацию полностью определяет программист-пользователь. Для этого имеется несколько типовых способов указания и определения местоположения записей в файле. Как уже упоминалось, эти способы определяются свойствами ЗУПД.

При указании местоположения записей при прямой организации используются либо абсолютная, либо относитель-

ная адресация. Абсолютный способ адресации состоит в использовании физических адресов для указания места расположения записей на диске. Относительный способ адресации состоит в использовании специальных приемов, позволяющих однозначно указывать место расположения записей в файле. Основное достоинство набора данных с прямой организацией, в котором применена относительная

<i>Номер дорожки</i>	<i>Номер записи</i>			
	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
<i>0</i>	<i>100</i>	<i>101</i>	<i>102</i>	<i>103</i>
<i>1</i>	<i>104</i>	<i>105</i>	<i>106</i>	<i>107</i>
<i>2</i>	<i>108</i>	<i>109</i>	<i>110</i>	<i>111</i>
<i>3</i>	<i>112</i>	<i>113</i>	<i>114</i>	<i>115</i>

Рис. 5.7.

адресация, — перемещаемость, т. е. независимость от физических адресов записей.

Рассмотрим один из способов относительной адресации, называемый прямым. На рис. 5.7 схематично изображено расположение некоторых записей в файле с прямой организацией. Все записи одинакового размера, на одной дорожке можно уместить ровно четыре записи. Одновременно с этим каждая запись в файле может быть однозначно идентифицирована с помощью некоторого, например, трехзначного номера, начиная со 100. При указанных выше условиях массив данных, содержащий записи с номерами 100, 101, 103, 105, 107, 108, 111, 112, 114, 115 и организованный прямым способом, располагается на 4 дорожках дискового запоминающего устройства так, как это изображено на рис. 5.7. Местоположение конкретной записи вычисляется следующим способом:

1) вычисляется относительный номер записи в файле $M = N - C$, где N — номер записи, C — начальный номер,

2) вычисляется порядковый номер дорожки (D), на которой будет располагаться данная запись, с помощью опе-

рации деления нацело: $D = M \div N$ (N — число записей на дорожке).

3) вычисляется порядковый номер физической записи (R) на дорожке, номер которой вычислен в п. 2 ($R = \text{res } D + 1$, где $\text{res } D$ — остаток от деления).

В нашем примере $C = 100$, $N = 4$. Вычислим местоположение записи с номером 114.

1) $M = 114 - 100 = 14$;

2) $D = 14 \div 4 = 3$;

3) $R = \text{res}(14 : 4) + 1 = 3$.

Таким образом, запись с номером 114 размещается на 3-й дорожке файла в блоке с номером 3 (см. рис. 5.7).

Очевидно, что записи, находящиеся в файле с прямой организацией и адресуемые рассмотренным выше способом, можно обрабатывать в произвольном порядке. Отметим однако, что вышеуказанный способ относительной адресации не является единственным и универсальным. Этот способ практически не пригоден в тех случаях, когда номера двух соседних записей имеют значительный разброс, что равнозначно недоиспользованию памяти на магнитных дисках. Даже в примере, приведенном на рис. 5.7, можно видеть, что отсутствие записей с номерами 102, 104, 106, 109, 110, 113 означает наличие пустых записей в файле, каждая из которых занимает столько же места, сколько запись, несущая конкретную информацию. Если представить себе файл, организованный таким образом и содержащий, к примеру, записи с номерами 100, 1001, 10002, 11132, 20099, то он должен занимать 20000 физических блоков, из которых использованы будут только 5(!), т. е. меньше 0,03 %.

Поэтому для массивов, обладающих только что рассмотренным свойством, используются другие способы адресации записей, которые более детально будут описаны в дальнейшем.

Поскольку фактическая организация наборов данных при прямом способе определяется самим программистом на уровне физических блоков, то в файлах с прямой организацией могут использоваться записи форматов F, V, U. При этом отметим, что записи формата U могут быть применены только в ОС ЕС.

Библиотечная организация файлов. Запоминающие устройства с прямым доступом позволяют создавать файлы с более сложной организацией, чем рассмотренные выше. К одному из таких способов организации относится

библиотечный способ. Этот способ в системе управления данными ОС ЕС является стандартным, наряду с прямым, последовательным и индексно-последовательным способами.

В принципе, библиотечный файл состоит из конечного множества последовательных файлов, объединенных в общий набор данных. Чаще всего такой способ организации используется для создания программных библиотек, в которых каждый последовательный файл представляет собой программу или подпрограмму. В библиотечных наборах данных такие последовательные файлы называют разделами.

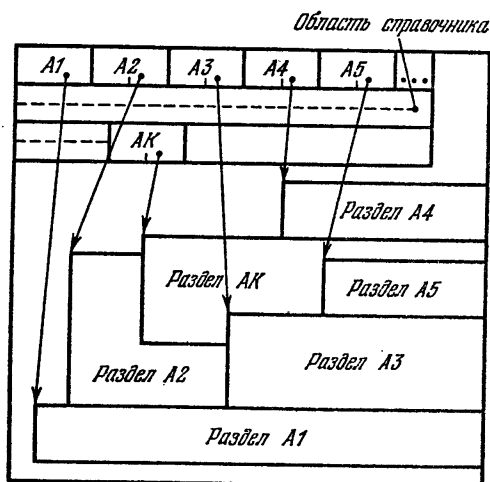


Рис. 5.8.

Основной организующей частью в библиотечном файле является специальный раздел, называемый справочником набора данных. Справочник представляет собой последовательно-организованный раздел, помещаемый в начале набора данных и содержащий для каждого раздела соответствующую запись.

Принцип библиотечной организации иллюстрируется схемой, представленной на рис. 5.8.

В области справочника находятся элементы, соответствующие имеющимся разделам файла: A1, A2, A3, A4, A5, ..., AK. В каждом элементе содержатся данные, необходимые для указания названия и начального адреса первой записи раздела. Элементы в справочнике располагаются в определенном порядке, который облегчает поиск

нужного раздела. Сами разделы могут располагаться внутри файла не в том порядке, в котором записаны описывающие их элементы в области справочника (рис. 5.8).

Наборы данных, организованные библиотечным способом, обычно обрабатываются по разделам. Каждый раздел представляет собой последовательно-организованный файл, поэтому в библиотечных наборах данных могут быть использованы записи форматов всех видов.

5.4. Методы доступа к данным

Понятие метода доступа. Метод доступа представляет собой комплекс программных средств, обеспечивающих конкретный способ обращения к данным, образующим файл. Метод доступа и его основные свойства определяются многочисленными факторами, среди которых нужно выделить два главных — уровень управления и способ организации данных. Очень часто метод доступа определяют через эти понятия, а именно, как сочетание конкретного способа организации и уровня управления данными.

Это в значительной мере обусловлено тем, что уровень управления и способ организации данных решающим образом оказывают влияние на большинство свойств метода доступа. К ним следует отнести способы решения задачи буферизации, синхронизации, обработки ошибок ввода-вывода и ряд других.

Примем основное определение метода доступа как сочетание способа организации и уровня управления данными. Однако при таком определении метода доступа следует рассматривать только базисный и логический уровни управления данными. Вызвано это тем, что физический уровень является несколько специфичным, и для него понятие «способ организации данных» не имеет смысла, так как организация и средства обращения к данным на этом уровне полностью определяются программистом.

Вышесказанное дает некоторое (правда, слабое) основание для введения понятия «физический метод доступа», не связывая физический уровень управления с какими-либо способами организации данных.

Основные методы доступа, их полные названия и принятые сокращения в ОС ЕС приведены в табл. 5.1.

С точки зрения программиста, использующего методы доступа для решения конкретных задач, каждый метод доступа имеет свой набор макрокоманд.

Таблица 5.1

Основные методы доступа в ОС ЕС

Организация данных	Метод доступа	
	при базисном уровне управления данными	при логическом уровне управления данными
Последовательная	базисный последовательный (BSAM)	логический последовательный (QSAM)
Индексно-последовательная	базисный индексно-последовательный (BISAM)	логический индексно-последовательный (QISAM)
Прямая	прямой (BDAM)	
Библиотечная	библиотечный (BPAM)	

Основные характеристики методов доступа.

Физический метод доступа, называемый иногда элементарным, предполагает свободное и порой виртуозное владение программистом навыками управления внешними устройствами на уровне канальных программ, что обычно доступно только наиболее опытным и высококвалифицированным специалистам. Использование других методов доступа освобождает программиста от необходимости досконального знания устройства и составления канальных программ для управления его работой.

Логический последовательный метод доступа предлагает строго последовательную обработку наборов данных. При использовании этого метода доступа могут быть применены 18 макрокоманд. Данный метод доступа осуществляет автоматическую синхронизацию операций ввода-вывода с работой программы, автоматическое выделение буферов и организацию их взаимодействия. На этом уровне производится также автоматическое объединение записей в блоки и выделение их из блоков. Считается, что данный метод доступа является наиболее простым и отработанным.

Базисный последовательный метод доступа предполагает также в основном последовательную обработку, но с более гибкими возможностями. В принципе, этот метод доступа дает возможность производить обработку данных файла в произвольном порядке. На базисном уровне обмен данными осуществляется физическими блоками. При этом программист обеспечивает функции

по синхронизации операций ввода-вывода, по выделению и организации взаимодействия буферов. Если в одном блоке файла имеется несколько логических записей, то выделение их, так же как и объединение в блоки, выполняется вручную. Базисный последовательный метод доступа содержит 20 макрокоманд и является наиболее развитым в системе ОС ЕС.

Логический индексно-последовательный метод доступа предназначен в основном для последовательной обработки индексно-последовательного файла (ИПФ), допуская при этом обработку либо всего ИПФ, либо произвольной его части. В других чертах этот метод доступа подобен логическому последовательному методу доступа. В его составе 15 макрокоманд.

Базисный индексно-последовательный метод доступа предназначен главным образом для осуществления обработки данных в произвольном порядке. С помощью средств этого метода доступа можно выбирать или вставлять в ИПФ отдельные записи по ключу, обновлять конкретные записи. В остальном базисный индексно-последовательный метод доступа подобен базисному последовательному методу доступа. Указанный метод доступа содержит 15 макрокоманд.

Прямой метод доступа предназначен для обработки файлов с прямой организацией, имеет в своем составе 16 макрокоманд. Обмен данными осуществляет по блочно. Для указания местоположения записей допускает использовать: а) относительный номер блока; б) относительный адрес дорожки; в) относительный адрес дорожки и ключ; г) физический адрес блока.

Прямой метод доступа используется с уже созданными файлами, имеющими прямую организацию. Создание набора данных с прямой организацией производится с помощью специального режима базисного последовательного метода доступа. В остальном рассматриваемый метод доступа аналогичен базисному последовательному методу доступа.

Библиотечный метод доступа предназначен для обработки библиотечных наборов данных. В названии метода доступа опущено слово «базисный» (следовало бы писать «базисный библиотечный метод доступа») потому, что данный метод доступа, так же как и прямой, может быть выполнен только при базисном уровне управления данными. Библиотечный метод доступа содержит в своем составе 19 макрокоманд. С их помощью можно обрабо-

тать одновременно один или несколько разделов библиотеки, производя при этом обмен данными в виде физических блоков, решая задачи синхронизации и буферизации способами, идентичными способам базисного последовательного метода доступа. Отдельные разделы библиотек можно обрабатывать, используя последовательные методы доступа (базисный и логический), так как каждый раздел имеет последовательную организацию.

На этом закончим краткое обсуждение методов доступа ОС ЕС и перейдем к рассмотрению отдельных свойств и характеристик методов доступа.

5.5. Хранение и поиск файлов

Общие соображения. Кроме рассмотренных выше сведений, следует получить представление о методах решения задач, связанных с организацией хранения и обеспечивающих эффективные способы поиска файлов на внешних носителях. Что касается файлов на перфолентах, перфокартах, АЦПУ и т. д., то задачи хранения и поиска этих файлов должны решаться непосредственно оператором или программистом. Обусловливается это в основном однократностью использования таких файлов. Например, при считывании массива данных, представленного на перфокартах, для повторной обработки того же массива его следует снова установить на устройство для повторного считывания. При этом поиск массива осуществляет пользователь, и никаких специальных программных средств для этого предусматривать, естественно, не нужно. Хранение и обслуживание таких массивов осуществляет либо программист, либо персонал, выделенный специально для этих целей.

Иначе обстоит дело с данными, записанными на магнитных носителях (НМЛ, НМД). Полностью обеспечить выполнение функций хранения и поиска силами обслуживающего персонала без привлечения специальных программных средств в этом случае не удастся. Такое положение обусловлено тем, что информация, записываемая на магнитный носитель, невидима. Поэтому для специалиста, работающего с магнитным носителем, необходимо иметь комплекс программных средств, позволяющих ему «почувствовать» записываемую или считываемую информацию.

В значительной мере решению указанных задач способствует аппарат меток. Каждый носитель, каждый файл должны сопровождаться специальной служебной записью, называемой *меткой*. Различают два типа меток — метка

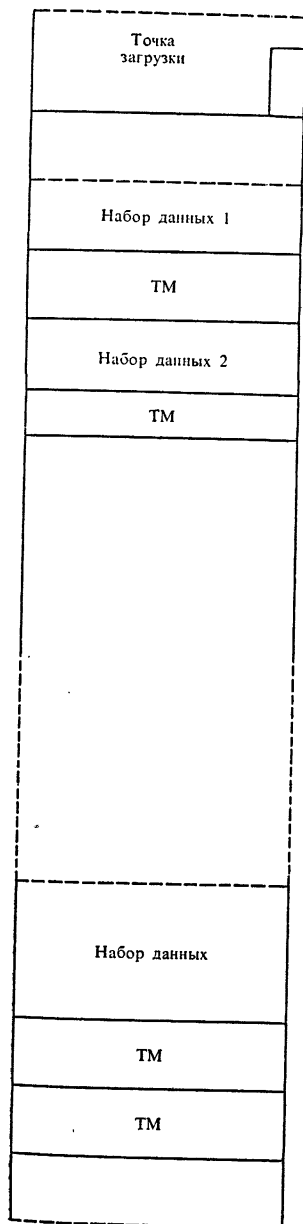
(носителя) тома и метка файла. Метка тома содержит необходимую информацию, используемую при организации хранения и поиска физического носителя. Метка файла содержит необходимую информацию, предназначенную для организации хранения и поиска наборов данных, записываемых на магнитные носители.

Отметим, что для магнитных лент, а также файлов, размещаемых на них, аппарат меток несколько отличается от аналогичного аппарата, разработанного для накопителей и наборов данных на магнитных дисках. Рассмотрим отдельно оба типа файлов.

Файлы на магнитной ленте. Основные данные о физическом размещении информации на магнитных лентах были рассмотрены в п. 5.1. Информация на магнитной ленте представляется в виде совокупности блоков, заключенных между двумя ленточными марками или между точкой загрузки и ленточной маркой (рис. 5.9, а). Режим работы с файлами, представленными на рис. 5.9, а, называется режимом работы без меток. Обработка таких файлов допустима, но имеет много существенных недостатков. К одному из главных недостатков следует отнести непосредственное участие оператора или программиста в выборе нужной бобины магнитной ленты и поиске на магнитной ленте соответствующего набора данных (подвод головок). При этом велика вероятность ошибочных действий со стороны пользователя, который может или неправильно выбрать бобину, или неправильно подвести головки считывания накопителя. Неправильные действия могут привести либо к большим потерям машинного времени, либо вообще к потере (затиранию) информации, записанной ранее на магнитную ленту.

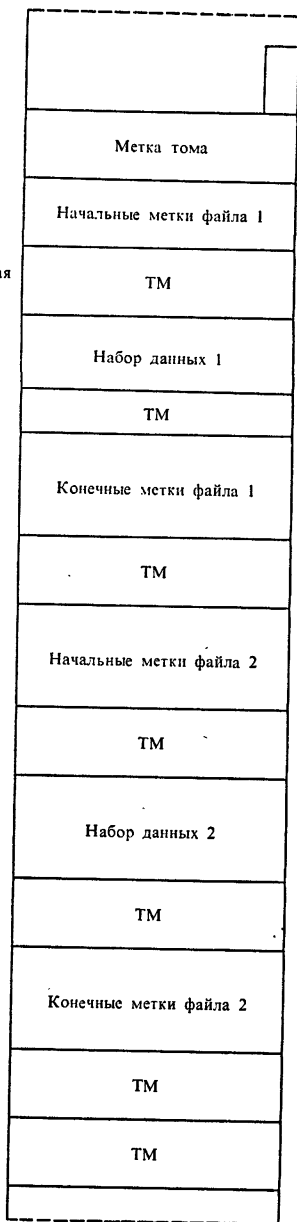
В связи с этим на практике не рекомендуется использовать режим работы без меток.

Рекомендуется использовать режим с применением стандартных меток. В этом режиме наборы данных на магнитной ленте должны быть представлены несколько иначе (рис. 5.9, б). Непосредственно после точки загрузки должна быть записана стандартная метка тома, за ней — стандартные начальные метки первого файла, после чего — ленточная марка. После блоков, представляющих собой набор данных, записывается следующая ленточная марка, за ней — стандартные конечные метки первого файла и, наконец, последняя ленточная марка. Если на магнитной ленте записано несколько файлов, то вся последовательность, за исключением метки тома, повторяется. В конце участка



а) Файлы без меток

Ленточная
марка



б) Файлы со стандартными метками

Рис. 5.9.

магнитной ленты, на которой записана информация, ставятся две ленточные марки (рис. 5.9).

Стандартные метки представляют собой физические записи длиной 80 байтов. Каждая метка имеет свой 4-байтовый идентификатор. Метка тома имеет идентификатор VOL1, две начальные метки файлов, которые чаще всего используются, имеют идентификаторы HDR1 и HDR2.

Две стандартные конечные метки файлов имеют идентификаторы или EOF1 и EOF2, или EOV1 и EOV2. Метки EOV1 и EOV2 используются лишь тогда, когда файл не умещается на одном томе и имеет продолжение на другом томе.

Основные способы организации томов магнитной ленты со стандартными метками приведены на рис. 5.10. На томе 1 записаны два файла. На томе 2 записан файл 3 и начало файла 4. Конечные метки файла 4 в этом случае имеют идентификаторы EOV1 и EOV2. Том 3 содержит последнюю часть файла 4.

Содержимое стандартных меток в значительной степени позволяет упростить процедуры поиска томов и файлов. Форматы меток представлены на рис. 5.11. Информация в метках записывается в коде ДКОИ. Стандартная метка тома используется для идентификации тома и его владельца, если последнее необходимо.

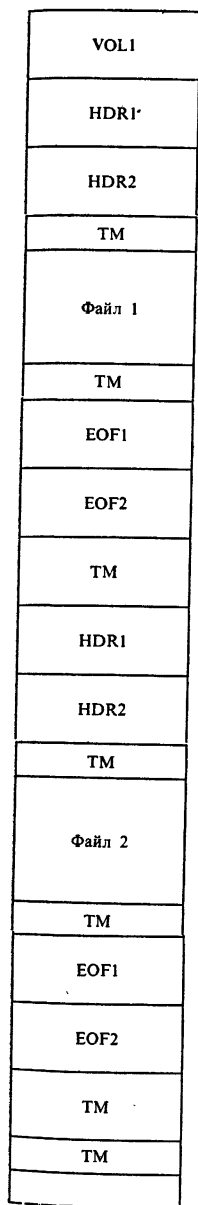
Рассмотрим отдельно поля метки тома. Поле «идентификатор метки» тома содержит всего 4 символа VOL1.

Поле «регистрационный номер тома» обычно соответствует внешней маркировке тома на его наружной поверхности. Регистрационный номер, который часто называют *серийным*, является идентификатором бобины магнитной ленты, находящейся в архиве ВЦ. Для написания регистрационного номера обычно используется 6-разрядный алфавитно-цифровой код.

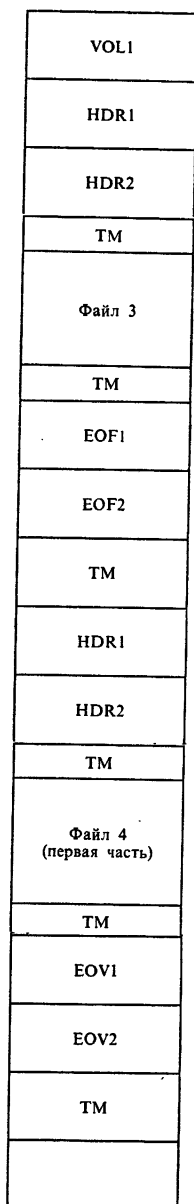
Поле «указатель оглавления тома» используется только для томов прямого доступа. На ленточных томах это поле не используется.

В поле «имя владельца тома и код адреса» указывают обычно, что том принадлежит либо некоторому пользователю, либо отделу и т. д. Данные в это поле метки записываются по желанию пользователя, который может задать любое имя и любой код.

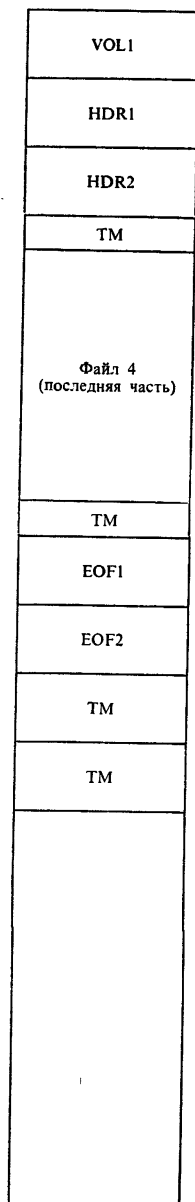
Стандартная метка тома записывается на носитель чаще всего специальной служебной программой *утилитой*, предназначенной для этих целей.



Том 1



Том 2



Том 3

Рис. 5.10.

Позиция

1	Идентификатор метки VOL1
5	Регистрационный номер тома
11	Резерв
12	Указатель оглавления тома
22	Резерв
42	Имя владельца тома и код адреса
52	Резерв
80	Метка тома

1	Идентификатор метки HDR1, EOF1, EOVI
5	Имя набора данных
22	Регистрационный номер тома
28	Номер тома
32	Номер файла
36	Поколение данных
42	Дата создания
48	Дата уничтожения
54	Защита файла
55	Счетчик блоков
61	Не используется
80	Первая стандартная метка файла

1	Идентификатор метки HDR2, EOF2, EOVI
5	Формат записи
6	Длина блока
11	Длина записи
16	Плотность записи
17	Положение файла
18	Название задания и шага задания
35	Пробелы
37	Управляющие символы
38	Резерв
39	Блокирование
40	Резерв
80	Вторая стандартная метка файла

Метка тома

Первая стандартная метка файла

Вторая стандартная метка файла

Рис. 5.11

Первая стандартная метка файла описывает некоторые данные, характеризующие файл. Рассмотрим отдельные поля первой стандартной метки файла.

Поле «идентификатор метки» содержит: HDR1 для начальной метки файла, EOVI для конечной метки тома (в том случае, если набор данных переходит на другой том) и EOF1 для конца файла. Наличие в файлах на магнитных лентах конечных меток позволяет вести обработку таких файлов с конца.

Поле «имя набора данных» может состоять из 17 символов, записываемых в отведенное поле метки. Это поле используется для проверки, соответствует ли имя обрабатываемого набора данных имени, заданному пользователем.

Поле «регистрационный номер тома» идентифицирует номер тома, на котором расположен данный файл.

Поле «номер тома» указывает порядковый номер тома внутри группы томов, на которых располагается данный файл. Этот номер всегда равен 0001 для файла, записанного на одном томе.

Поле «номер файла» указывает на относительное расположение файла на магнитной ленте среди множества файлов, записанных на одном томе. Для первого файла этот номер равен 0001. Относительный номер файла используется программами управления данными для начального подвода головок, этот номер при вызове того или иного файла всегда следует указывать.

Поле «поколение данных» используется в том случае, если набор данных является частью группы поколений данных, применяемых при обработке этого файла. Если набор данных входит в группу поколений, то данное поле содержит номер поколения (4-значный код) и номер варианта поколения (2-значный код).

Поле «дата создания» файла содержит год и день, когда набор данных был создан.

Поле «дата уничтожения» файла содержит год и день истечения срока хранения файла, после которого он может быть затерт. Обычно дата задается пользователем. Если дата не указывается, то это поле заполняется нулями, и считается, что срок хранения данных уже истек.

Поле «защита файла» заполняется следующим образом: 0 — набор не защищен; 1 — набор защищен, и для считывания, записи или уничтожения данных требуется дополнительная информация; 3 — набор защищен только от записи или уничтожения. При обращении к данным такого

файла программы управления данными сначала проверяют его защищенность. Если файл защищен (1 или 3 в поле защиты), то проверяется пароль, задаваемый оператором. Заданный пароль должен соответствовать паролю, установленному для этого набора данных в операционной системе. Если пароль задан неправильно, то обработка файла не допускается.

Поле «счетчик блоков» используется только в конечных метках и указывает количество информационных блоков, помещенных в набор данных. В начальной метке это поле всегда содержит нули.

Рассмотрим содержимое полей второй стандартной метки файла. Обычно она содержит дополнительную информацию о наборе данных.

Поле «идентификатор метки» может содержать HDR2, EOF2, EOV2.

Поле «формат записи» содержит следующий символ, указывающий формат записей в файле: F — записи фиксированной длины; V — записи переменной длины; U — записи неопределенной длины.

В поле «длина блока» указывают размер блока в байтах. Для записей формата FB длина должна быть кратна длине логической записи; для записей формата V указывается максимальная длина блока, включая 4-байтовый указатель длины; для записей формата U всегда указывается максимальная длина.

Поле «длина записи» содержит размер логической записи. Для записей формата V указывается максимальная длина логической записи, включая 4-байтовый указатель длины; для записей формата U в этом поле всегда нули.

Поле «плотность записи» содержит следующие коды: 0 — плотность 8 бит/мм, 2 — плотность 32 бит/мм, 3 — плотность 64 бит/мм.

Поле «положение файла» содержит код, определяющий, происходило ли переключение тома: 0 — переключение тома не происходило; 1 — переключение тома произошло. Это поле в начальной метке всегда равно 0, а в конечных метках заполняется в зависимости от того, происходило переключение или нет.

Поле «название задания и шага задания» содержит имя задания и шага задания, в процессе выполнения которых набор данных был создан.

Поле «управляющие символы» содержит код, указывающий использование управляющих символов при выводе

набора данных: А — управляющие символы в алфавитном виде; М — управляющие символы в двоичном виде.

Поле «блокирование» содержит характеристику блокирования записей: В — записи блокированные; S — записи расширенные; BS — записи блокированные и расширенные.

Блокированные записи были рассмотрены ранее. К расширенным записям относятся такие записи, размер которых больше размера физического блока на носителе. Расширенные записи используются довольно редко.

Отметим, что логическую организацию томов со стандартными метками осуществляют системные программы управления данными. Они формируют содержимое меток и записывают их в нужной последовательности на магнитную ленту.

В практической работе знание содержимого стандартных меток является необходимым при выполнении ряда функций по обеспечению хранения и обслуживания файлов. Кроме того, максимально используя содержимое стандартных меток, можно экономным способом описывать обрабатываемые файлы.

Носители и файлы на запоминающих устройствах прямого доступа. Основные сведения по устройству и способам физической записи информации на накопителе с прямым доступом были рассмотрены в п. 5.1. С носителями ЗУПД в отличие от магнитных лент работать, не используя аппарата меток, нельзя. Единственным режимом работы с носителями прямого доступа, обеспечиваемым программами управления данными, является режим обработки со стандартными метками.

Организация тома и файлов, расположенных на носителях прямого доступа, определяется свойствами и спецификой функционирования ЗУПД. В качестве типичного ЗУПД будем рассматривать накопитель на магнитном диске типа ЕС-5050. Схема представления файлов на магнитном диске приведена на рис. 5.12. Основные отличия организации файлов на диске от ленточных файлов состоят в следующем:

- 1) все стандартные метки файлов собраны в специальном экстенде тома, называемом «оглавление тома»;
- 2) каждый файл имеет только одну стандартную метку. На магнитном диске файл не имеет конечной метки, так как нет необходимости считывать данные в обратном порядке;
- 3) границы наборов данных отмечаются следующим образом. Начало набора данных указывается в стандарт-

ной метке в виде физического адреса первого блока. Конец набора данных отмечается специальным блоком, представляющим собой однобайтовый фиктивный блок с нулевой длиной ключа и нулевой длиной данных. Этот

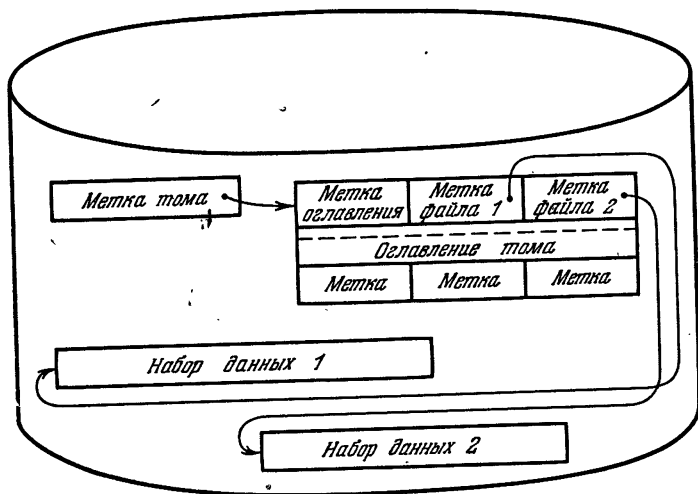


Рис. 5.12.

фиктивный блок играет роль своеобразного дискового маркера, аналогичного ленточной марке;

4) начальный адрес оглавления содержится в стандартной метке тома. Формат стандартной метки тома приведен на рис. 5.11. Назначение и содержание этой метки аналогично метке тома магнитной ленты. Оглавление тома является набором данных, который содержит стандартные метки всех файлов, находящихся на данном томе.

Форматы и назначение меток, содержащихся в оглавлении, различны. Все типы меток представляют собой блоки, имеющие 44-байтовую область ключа и 96-байтовую область данных. Следует выделить 5 основных типов (форматов) меток.

Прежде чем приступить к работе с томом прямого доступа, его необходимо подготовить или, как говорят, проинициализировать. Инициализация тома состоит в том, что на каждой дорожке тома записывается ее собственный адрес и запись R0. Затем на нулевой дорожке нулевого цилиндра формируется метка тома с указателем оглавления тома. Местоположение оглавления тома опре-

деляется оператором или программистом, осуществляющим инициализацию.

В момент инициализации оглавление формируется следующим образом. Первой в оглавлении записывается метка формата 4. Эта метка описывает само оглавление. Она содержит информацию о местоположении оглавления, его размерах, о запасных дорожках тома и т. д. Вслед за меткой формата 4 записывается метка формата 5. Она содержит информацию о свободных областях (экстентах) на данном томе. По мере заполнения тома наборами данных обновляется содержимое метки формата 5. Метка формата 5 в общей сложности может описать 26 свободных экстенгов. Если число свободных экстенгов больше 26, то на томе в оглавлении записывается еще одна метка формата 5. Несколько меток этого формата связываются между собой в одну цепочку при помощи ссылочных адресов.

Непосредственно после инициализации на томе имеется только один набор данных — оглавление тома, а остальная часть тома указана в метке формата 5 как свободная. В оглавлении после инициализации, кроме меток формата 4 и 5, записываются метки так называемого нулевого формата, которые не содержат полезной информации, а заполняются пробелами с целью форматизации оглавления тома.

Для любого набора данных, размещенного на томе, в оглавлении находится стандартная метка формата 1. Это основной тип метки, используемый для описания наборов данных. В метке формата 1 содержится информация, необходимая для обеспечения хранения, поиска и обработки данных. Формат этого типа метки приведен на рис. 5.13. Рассмотрим отдельные поля этой метки.

Поле «имя файла» содержит 44 байта для указания наименования набора данных, включая номер и вариант поколения, если файл является частью группы поколений данных. В последнем случае для имени используется 35 байтов.

Поле «формат метки» содержит 1 для метки данного типа.

Поле «регистрационный номер тома» содержит регистрационный номер тома, на котором расположен данный файл. Для файла, расположенного на нескольких томах, это поле содержит регистрационный номер первого тома (так же как для ленточного файла).

Поле ключа	
Позиция	
1	Имя файла
.	
.	
.	
44	
Поле данных	
45	Формат метки
46	Регистрационный номер тома
.	
.	
52	
.	Порядковый номер
.	
.	
54	
.	Дата создания файла
.	
.	
57	
.	Дата уничтожения файла
.	
.	
60	
.	Число экстенгов
.	
.	
61	
.	Число байтов, использованных в последнем блоке справочника
.	
.	
62	
.	Резерв
.	
.	
63	
.	Нули
.	
.	
76	
.	Резерв
.	
.	
83	
.	Способ организации файла
.	
.	
.	

85	Формат записи
86	Резерв
87	Длина блока
.	
.	
.	
89	Длина записи
.	
.	
.	
91	Длина ключа
92	
.	
.	
.	Расположение ключа
.	
.	
.	
94	Индикаторы набора данных
95	
.	
.	
.	Вторичное распределение
.	
.	
.	
99	Адрес последней записи
.	
.	
.	
104	Число оставшихся байтов на дорожке
.	
.	
.	
106	Первый экстент
.	
.	
.	
116	Второй экстент
.	
.	
.	
126	Третий экстент
.	
.	
.	
136	Указатель на следующую метку файла
.	
.	
.	
140	

Рис. 5.13.

Поле «порядковый номер тома» содержит номер тома в двоичном коде. Это поле позволяет определить порядок томов в многотомном файле.

Поле «дата создания файла» содержит дату первоначального создания набора данных.

Поле «дата уничтожения файла» содержит дату окончания срока хранения набора данных. Используется это поле аналогично одноименному полю метки магнитной ленты.

Поле «число экстенгов» содержит количество экстенгов, занимаемых файлом на данном томе.

Поле «число байтов, использованных в последнем блоке справочника» заполняется только для файлов с библиотечной организацией. Содержимое этого поля позволяет вычислить количество элементов, описывающих разделы, которые могут быть добавлены в справочник библиотеки.

Поля, заполненные нулями, а также резервные, могут быть использованы по мере развития операционных систем.

Поле «способ организации файла» содержит информацию о типе организации файла: последовательная, индексно-последовательная, прямая или библиотечная.

Поле «формат записи» содержит информацию о формате записей, используемых в данном файле, а также способах их блокирования.

Поле «длина блока» содержит максимальный размер блока в байтах, представленный в двоичном коде.

Поле «длина записи» содержит максимальный размер логической записи в байтах, представленный в двоичном коде.

Поле «длина ключа» содержит длину (в двоичном коде) области ключа в записях данного файла. Это поле используется для тех файлов, в которых имеется поле ключа.

Поле «расположение ключа» содержит начальный адрес первого байта ключа, размещенного в поле логической записи. Применяется только для индексно-последовательных файлов.

Поле «индикаторы набора данных» показывает, было ли переключение томов применительно к описываемому файлу.

Поле «вторичное распределение» используется для указания объема памяти, запрашиваемого после окончания обработки каждого экстенга файла. Поле «адрес последней записи» для последовательного или библиотечного файла содержит адрес последней записи.

Поле «число оставшихся байтов на дорожке» связано с предыдущим полем. Оно позволяет вычислить количество записей, которые можно поместить на частично заполненную дорожку.

Следующие три поля описывают экстен-ты, занимаемые данным файлом. Описание экстен-та содержит: тип экстен-та (область индексов, переполнения и т. п.), порядковый номер экстен-та, местоположение и объем экстен-та.

Ранее отмечалось, что файлы на магнитном диске могут быть расположены на нескольких экстен-тах одновременно. В одной метке формата 1 могут быть описаны только три экстен-та (см. рис. 5.13). Если же число экстен-тов, отводимых для описываемого файла, превышает 3, то в оглавление помещается еще одна метка, относящаяся к данному файлу, — метка формата 3. В метке формата 3 описываются последующие экстен-ты тома, занимаемые файлом. Метки формата 1 и формата 3 связываются между собой в цепочку с помощью специального поля, имеющегося как в метке формата 1, так и в метке формата 3 — «указатель на следующую метку файла» (см. рис. 5.13). Одна метка формата 3 может одновременно описывать до 13 экстен-тов тома, выделяемых под данный файл.

Метка формата 2 является дополнительной к метке формата 1. Этот тип метки используется только для наборов данных с индексно-последовательной организацией. Он описывает физическое расположение областей набора данных (индекс цилиндров, главные индексы различных уровней). Метка формата 2 также сцепляется с меткой формата 1 с помощью поля «указатель на следующую метку файла».

Метка формата 1 была рассмотрена так подробно, потому что этот тип метки используется для всех файлов и употребляется наиболее часто. Метки других форматов являются менее распространенными. Для облегчения поиска файла метки в оглавлении рассортированы по содержимому поля ключей, т. е. по именам. Обслуживание оглавления тома, т. е. запись или поиск меток, а также обновление содержимого меток, осуществляют системные программы управления данными.

Для пользователя обработка наборов данных выглядит как обработка файлов, не содержащих меток, так как чтение и запись меток осуществляются системными программами. Однако это вовсе не означает, что пользователь полностью освобожден от необходимости указания

некоторой информации, используемой системными программами в процессе обработки меток. Кроме того, знание форматов меток может оказаться полезным при создании экономного описания файлов.

Рассмотрение стандартных меток как ленточных, так и дисковых файлов позволяет сделать вывод о том, что содержимое меток является дополнительным источником описания обработки ранее созданных файлов.

Это свойство аппарата стандартных меток можно использовать для создания программ, выполняющих обработку файлов с различной структурой. Например, одной и той же программой можно обрабатывать файлы с различными именами и с различной структурой блоков и записей, если соответствующую информацию описания файла черпать прямо из меток, относящихся к нужному набору данных.

ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ ВВОДА-ВЫВОДА

6.1. Организация системы программирования операций ввода-вывода

Как упоминалось ранее, система программирования операций ввода-вывода на уровне языка ассемблера, который является базовым для соответствующих средств языков более высокого уровня, состоит из конечного множества макрокоманд управления данными. Наборы макрокоманд основных методов доступа базисно-логического уровня управления данными операционных систем ОС и ДОС приведены соответственно в табл. 6.1 и 6.2. В них показано использование макрокоманд для различных методов доступа. Сравнительный анализ наборов макрокоманд ввода-вывода показывает, что системы управления данными в ОС и ДОС различны. Набор макрокоманд ввода-вывода любой рассматриваемой операционной системы можно разделить на 4 группы:

- 1) макрокоманды описания наборов данных;
- 2) макрокоманды начала и завершения обработки наборов данных;
- 3) макрокоманды формирования запросов на выполнение операций ввода-вывода;
- 4) макрокоманды для формирования запросов на выполнение дополнительных вспомогательных функций.

Макрокоманды, входящие в состав группы описания наборов данных, будем называть *декларативными*, а остальные макрокоманды — *императивными*.

Средства описания файлов предоставляют возможность каждому программисту определять наборы данных в процессе программирования, формулируя собственные требования как к данным, так и к динамике их обработки. В процессе написания программ обработки данных сначала создается полное формальное описание объекта системы управления данными. Описание объекта включает в себя такой набор сведений, который допускает управление им в условиях мультипрограммирования, в усло-

Таблица 6.1

Макрокоманды основных методов доступа ОС ЕС ЭВМ

№ п/п	Макрокоманды	Методы доступа					
		QSAM	BSAM	BPAM	QISAM	BISAM	BDAM
1	BLDL	-	-	+	-	-	-
2	BSP	-	+	-	-	-	-
3	BUILD	+	+	+	+	+	+
4	BUILDRCD	+	-	-	-	-	-
5	CHECK	-	+	+	-	+	+
6	CLOSE	+	+	+	+	+	+
7	CNTRL	+	+	-	-	-	-
8	DCB	+	+	+	+	+	+
9	DCBD	+	+	+	+	+	+
10	ESETL	-	-	-	+	-	-
11	FEOV	+	+	-	-	-	-
12	FIND	-	-	+	-	-	-
13	FREEDBUF	-	+	+	-	+	+
14	FREEDBUF	-	-	-	-	+	+
15	FREEPOOL	+	+	+	-	+	+
16	GET	+	-	-	+	-	-
17	GETBUF	-	+	+	-	+	+
18	GETPOOL	+	+	+	+	+	+
19	NOTE	-	+	+	-	-	-
20	OPEN	+	+	+	+	+	+
21	POINT	-	+	+	-	-	-
22	PRTOV	+	+	-	-	-	-
23	PUT	+	-	-	+	-	-
24	PUTX	+	-	-	+	-	-
25	READ	-	+	+	-	+	+
26	RELEX	-	-	-	-	-	+
27	RELSE	+	-	-	+	-	-
28	SETL	-	-	-	+	-	-
29	STOW	-	-	+	-	-	-
30	SYNADAF	+	+	+	+	+	+
31	SYNADRLS	+	+	+	+	+	+
32	TRUNC	+	-	-	-	-	-
33	WAIT	-	+	+	-	+	+
34	WRITE	-	+	+	-	+	+

Макрокоманды основных методов доступа ДОС ЕС ЭВМ

№ п/п	Макрокоманды	Методы доступа				
		QSAM	BSAM	QISAM	BISAM	BDAM
1	CDMOD	+	+	-	-	-
2	CHECK	-	+	-	-	-
3	CLOSE	+	+	+	+	+
4	CLOSER	+	+	+	+	+
5	CNTRL	-	+	-	-	+
6	DAMOD	-	-	-	-	+
7	DIMOD	+	+	-	-	-
8	DTFCD	+	+	-	-	-
9	DTFCN	+	+	-	-	-
10	DTFDA	-	-	-	-	+
11	DTFDI	+	+	-	-	-
12	DTFIS	-	-	+	+	-
13	DTFMT	+	+	-	-	-
14	DTFPR	+	+	-	-	-
15	DTFPT	+	+	-	-	-
16	DTFSD	+	+	-	-	-
17	ENDFL	-	-	+	+	-
18	ESETL	-	-	+	-	-
19	FEOV	+	+	-	-	-
20	GET	+	-	+	-	-
21	ISMOD	-	-	+	+	-
22	LBRET	+	+	-	-	-
23	MTMOD	+	+	-	-	-
24	NOTE	-	+	-	-	-
25	OPEN	+	+	+	+	+
26	OPENR	+	+	+	+	+
27	POINTR	-	+	-	-	-
28	POINTS	-	+	-	-	-
29	POINTW	-	+	-	-	-
30	PRMOD	+	+	-	-	-
31	PRTOV	+	+	-	-	-
32	PTMOD	+	+	-	-	-
33	PUT	+	-	+	-	-
34	READ	-	+	-	+	+
35	RELSE	+	-	-	-	-
36	SDMOD	+	+	-	-	-
37	SETFL	-	-	+	+	-
38	SETL	-	-	+	-	-
39	TRUNC	+	-	-	-	-
40	WAITF	-	-	-	+	+
41	WRITE	-	+	-	+	+

виях широко изменяемой конфигурации ЭВМ, но в пределах изменений типов внешних устройств.

Кроме того, описание файлов осуществляется таким способом, при котором написание императивных команд ввода-вывода требует минимальных усилий со стороны программиста. Другими словами, количество операндов, указываемых в макрокомандах описаний, значительно больше числа операндов в императивных макрокомандах.

Например, в ОС ЕС макрокоманда DCB содержит минимум 14 операндов, тогда как наиболее часто используемая макрокоманда ввода GET содержит максимум 2 операнда. Это свидетельствует о том, что основные сведения о наборе данных и способах его обработки должны указываться в макрокомандах описания. Большинство стандартных действий в процессе обработки данных, а также способы их реализации должны быть указаны в макрокомандах описаний. Среди них можно отметить определение способов буферизации, синхронизации, обработки ошибок и ряд других.

Множество императивных макрокоманд разбито на три группы. Группа макрокоманд, обеспечивающая начало и завершение обработки наборов данных, предназначена для выполнения таких функций, которые перед началом обработки приводят в готовность таблицы описания файлов, создаваемые в результате трансляции соответствующих макрокоманд. Кроме того, выполняются функции поиска нужных файлов, обработки их меток и ряд других подготовительных операций. Другие макрокоманды этой группы выполняют ряд функций после завершения процесса обработки данных. К этой группе в основном относятся макрокоманды OPEN и CLOSE.

Основной группой среди императивных макрокоманд является группа, формирующая запросы на выполнение отдельных операций ввода-вывода. К ним относятся макрокоманды GET, PUT, READ, WRITE.

Последняя группа императивных макрокоманд — макрокоманды, формирующие запросы на выполнение дополнительных вспомогательных функций. Эти макрокоманды расширяют набор средств программирования операций ввода-вывода и представляют возможность использовать специфические особенности аппаратуры. Среди таких макрокоманд можно отметить макрокоманды CHECK (проверить), CNTRL (управление устройством) и т. д.

Перейдем к рассмотрению вопросов применения макрокоманд при программировании операций ввода-вывода.

6.2. Основные этапы программирования операций ввода-вывода

В соответствии с вышеописанной классификацией множества макрокоманд становится возможным описать основные этапы программирования операций ввода-вывода.

Во-первых, начинать программирование операций обмена необходимо с описания файла (одного или нескольких). В распоряжении программиста имеется одна или несколько декларативных, предназначенных для этого, макрокоманд. Естественно, что прежде чем приступить непосредственно к описанию набора данных, необходимо иметь полную информацию о нем, включая выбор соответствующего метода доступа. Закончив написание необходимых декларативных макрокоманд, следует помнить, что в макрокомандах описываются в основном логические свойства (характеристики) файлов.

Физические характеристики файлов, такие как тип устройства, конкретный носитель (регистрационный номер), адрес устройства и ряд других, описываются с помощью средств системы управления заданиями, а именно, с помощью операторов языка управления заданиями. Связь между декларативными макрокомандами системы управления данными и некоторыми операторами языка управления заданиями будет описана в дальнейшем. Подчеркнем лишь, что такая связь существует. Устанавливается она следующим образом.

В процессе выполнения операций ввода-вывода системные программы управления данными должны располагать полной информацией о наборе данных, включающей в себя как логические, так и физические характеристики набора данных. Но, как было отмечено ранее, физические характеристики можно описать только при помощи специальных операторов языка управления заданиями. Следовательно, необходимая информация, взятая из операторов языка управления заданиями, должна находиться в таблице описания файлов.

Обычно для описания файлов и способов их обработки в ОС ЕС используется одна макрокоманда DCB, а в ДОС ЕС макрокоманды DTFaа и ааMOD, где аа — переменная, принимающая значения в зависимости от типа устройства и способов организации данных. Например: DTFCД и CDMOD — макрокоманды, используемые при работе с устройством считывания перфокарт. Основной перечень макрокоманд DTFaа и ааMOD дан в табл. 6.2.

Во-вторых, непосредственно перед началом обработки описанных данных программист должен предусмотреть выполнение процедуры начала обработки, называемой *процедурой открытия*. Исполнение ее вызывается, главным образом, макрокомандой OPEN в операционных системах ОС и DOS. Использование данной макрокоманды, за очень редким исключением, является обязательным.

В-третьих, после завершения описания набора данных и завершения написания процедуры открытия можно приступить к программированию необходимых операций ввода-вывода. В зависимости от выбранного метода доступа на этом этапе программирования могут быть использованы следующие макрокоманды:

GET, PUT — для логического уровня;

READ, WRITE — для базисного уровня управления данными.

На этом этапе программирования для логических методов доступа соответствующие макрокоманды являются единственными. Обычно использование их не вызывает особых затруднений, так как макрокоманда GET в ОС ЕС может иметь, как уже отмечалось выше, максимум два параметра, а макрокоманда READ, например, — максимум семь параметров.

Как правило, для базисных методов доступа, кроме основных макрокоманд, формирующих команды чтения или записи, характерным является использование ряда вспомогательных макрокоманд. Наиболее типичной вспомогательной макрокомандой, используемой вместе с основными, является макрокоманда CHECK. Данная макрокоманда применяется для совмещения операций ввода-вывода со счетом и синхронизации указанных событий (счет и ввод-вывод).

На базисном уровне управления данными программирование отдельной операции ввода-вывода состоит в написании сразу двух макрокоманд, например, WRITE и после нее CHECK. Чаще всего операция ввода записывается парой

READ <операнды>

CHECK <операнды>

Операция вывода — другой парой

WRITE <операнды>

CHECK <операнды>

Кроме команд синхронизации, могут быть использованы и другие вспомогательные макрокоманды из множества макрокоманд, образующих конкретный метод доступа. Например, в методе доступа BSAM можно избежать строго последовательной обработки файлов на магнитной ленте или диске, используя для этого макрокоманды NOTE и POINT.

Первая из указанных макрокоманд позволяет запомнить (отметить) физический адрес блока при его записи на носитель, а вторая по полученному ранее адресу позволяет подвести отмеченный блок под головки считывания для того, чтобы его можно было прочитать очередной командой чтения.

Еще один пример. В методе доступа QISAM для того, чтобы считывать записи не с самого начала файла, а с записи, имеющей вполне конкретное значение ключа, используется макрокоманда SETL. С помощью этой макрокоманды осуществляется подвод головок считывания в нужное место файла для последующего считывания требуемой последовательности записей.

В-четвертых, после окончания программирования вышеописанных трех этапов необходимо предусмотреть выполнение процедуры окончания обработки файлов, называемой *процедурой закрытия файла*. На этом этапе используется макрокоманда CLOSE в системе ОС и ДОС. Макрокомандой закрытия файлов программируются действия, отменяющие действия макрокоманды открытия. Делается это с целью облегчения работы системных программ по управлению наборами данных в условиях мультипрограммирования. Поскольку каждый набор данных, обрабатываемый программами, должен стоять «на учете» системы управления данными, которая обслуживает их, то своевременное снятие «с учета» наборов данных, обработка которых уже закончилась, приводит к более рациональной загрузке системных программ. Кроме системных программ, с обрабатываемыми файлами тесно связан целый ряд системных таблиц, освобождение которых от взаимодействия с таблицами, описывающими уже обработанные файлы, также имеет немаловажное значение.

Наконец, в-пятых, может появиться необходимость составить специальные программы или подпрограммы, предназначенные для выполнения вспомогательных действий, обеспечивающих и поддерживающих успешное выполнение операций ввода-вывода. Отметим сразу, что пятый этап является не обязательным, в отличие от вышеописанных

ранее, и доступен программистам, имеющим достаточный опыт работы с системой управления данными.

К специальным подпрограммам можно отнести подпрограммы обработки и коррекции ошибок ввода-вывода, отличающиеся от стандартных системных обработчиков ошибок ввода-вывода.

Как правило, для достижения максимальной достоверности ввода-вывода больших массивов данных требуется осуществлять более детальный контроль записей и блоков, чем это предусмотрено системными средствами. Это может быть сделано, в частности, специально составленными программами, позволяющими контролировать (а может быть, и корректировать при необходимости) данные в полном соответствии с семантическими отношениями внутри записей согласно смысловому содержанию задачи.

В качестве еще одного примера составления специальных программ, предусматриваемых пятым этапом, можно привести прием, который рекомендуется использовать в том случае, если внутри одной задачи обрабатывается значительное количество файлов. Речь идет о более интенсивном использовании таблиц описания данных.

После использования и закрытия таблицы описания файла ее можно применять для обработки другого набора данных. Однако такое использование таблиц, как правило, требует написания нестандартных программ, осуществляющих динамическую «настройку» таблиц при переходе с обслуживания одного файла на другой. Эти программы пишутся на пятом этапе.

В заключение подчеркнем, что программирование операций обмена с использованием средств системы управления данными не является простой задачей.

На первых порах, когда программист только начинает осваивать систему управления данными, можно рекомендовать следующее.

1. Использовать только четыре первых этапа программирования операций ввода-вывода, описанных выше, причем третий этап выполнять по возможности без применения специфических макрокоманд.

2. Среди множества макрокоманд основных методов доступа выделить такие, которые являются общими для всех методов доступа (см. табл. 6.1 и 6.2).

3. Среди множества макрокоманд основных методов доступа выделить 2 подмножества макрокоманд. В одно из подмножеств входят макрокоманды, которые являются общими для логических уровней управления данными.

В другое — макрокоманды, являющиеся общими для базисных уровней управления данными (см. табл. 6.1 и 6.2).

4. Первые программы следует составлять только с использованием макрокоманд, выделенных в пп. 2 и 3. Причем начинать следует с использования логических методов доступа, постепенно переходя к базисным.

Выделим минимально необходимое количество макрокоманд, приведенных в табл. 6.1, используя вышеописанные рекомендации, для работы с методом доступа QSAM. Применяя п. 2, выделим макрокоманды DCB, OPEN и CLOSE. Другие общие макрокоманды, например, DCBD, BUILD, могут быть использованы только на пятом этапе (см. п. 1 рекомендаций). Применяя п. 3, выделим макрокоманды GET и PUT. Выделенное подмножество макрокоманд рекомендуется использовать начинающему программисту. Аналогичные действия можно произвести применительно и к другим методам доступа.

Перейдем к рассмотрению вопроса установления связей между макрокомандами системы управления данными и программой пользователя.

6.3. Связи макрокоманд ввода-вывода с программами системы управления данными

Ранее отмечалось, что в системе программирования операций ввода-вывода на уровне языка ассемблера применяются стандартные средства макроязыка. Поэтому для понимания излагаемого далее материала требуется предварительное знакомство с основными элементами макроязыка ассемблера.

В предыдущих параграфах был приведен список макрокоманд, используемых при программировании операций обмена, и порядок их применения в системах управления данными. После составления программы, предназначенной для обработки данных, необходимо, чтобы макрокоманды, использованные в программе пользователя, были заменены на соответствующие системные программы или таблицы, обеспечивающие правильное выполнение запрограммированных операций ввода-вывода.

Замена макрокоманд и ввод в действие указанного выше механизма осуществляется стандартными средствами системы программирования, имеющимися в составе программного обеспечения ЭВМ. Общая схема обработки макрокоманд в процессе трансляции программ приведена на

рис. 6.1. Данная схема иллюстрирует процесс преобразования макрокоманд в соответствующие им последовательности машинных команд, которые называются *макрорасширениями*. Для того чтобы конкретная макрокоманда в результате трансляции имела соответствующее ее содержанию

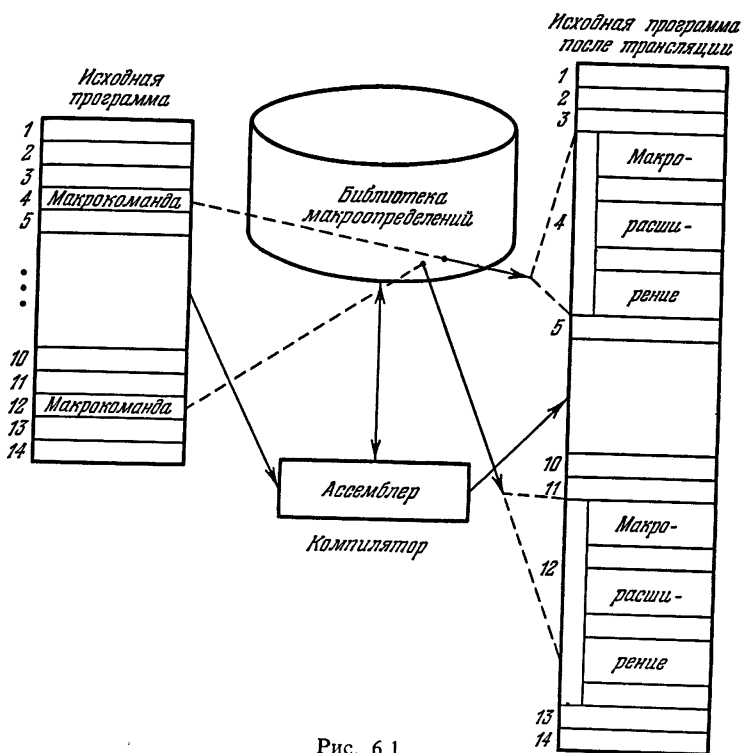


Рис. 6.1.

макрорасширение, нужно в специальной библиотеке ассемблера, предназначенной для обработки макрокоманд, поместить *макроопределения*.

Из макроопределения в процессе трансляции, используя значения заданных в макрокоманде операндов, ассемблер вырабатывает подходящее макрорасширение, которое вставляется в текст основной программы вместо макрокоманды (см. рис. 6.1).

Совокупность макрокоманд системы управления данными вместе с соответствующими им макроопределениями образуют *макросистему ввода-вывода*. Через макросистему осуществляется доступ программ пользователя к програм-

мам системы управления данными. Макрокоманды ввода-вывода в результате трансляции исходной программы заменяются конкретными последовательностями машинных команд или констант.

В современных операционных системах стадия трансляции программы не является единственной, когда в исходную программу могут быть автоматически вставлены отдельные

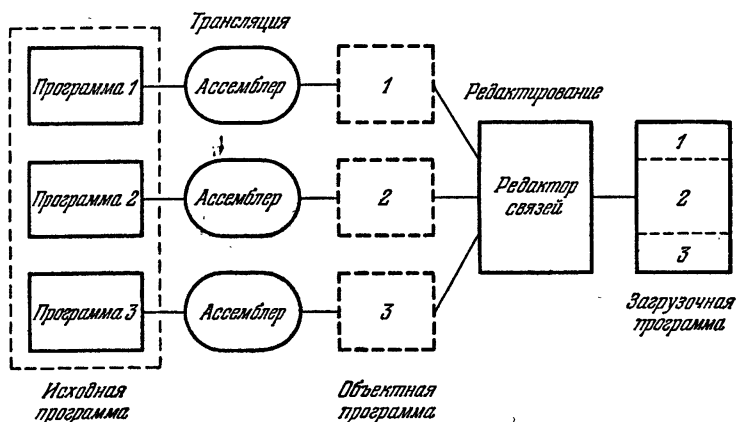


Рис. 6.2.

части, например, макрорасширения, написанные ранее. Основными недостатками такого формирования программы на этапе трансляции являются следующие:

- 1) увеличение времени компиляции программы;
- 2) увеличение расхода внешней и оперативной памяти для хранения рабочих, промежуточных массивов и таблиц;
- 3) увеличение размера выходного модуля.

По желанию программиста отмеченные недостатки можно устранить путем активного использования возможностей специальной системной программы — Редактора связей. Одной из важнейших функций программы Редактора связей является осуществление сборки программы по частям, заранее написанным и прошедшим стадию трансляции (рис. 6.2). Упомянутые выше макросредства ассемблера и некоторые возможности Редактора связей могут быть использованы системой управления данными при установлении связей программ пользователей с системными программами. Так, например, можно заранее по желанию пользователя подготовить объектные модули, полученные

путем трансляции соответствующих макроопределений, входящих в состав макросистемы ввода-вывода.

Макроопределения транслируются при задании конкретных сочетаний и значений операндов соответствующих макрокоманд. Если учесть, что при редактировании программ Редактор связей может осуществлять автоматический поиск и присоединение к программе недостающих модулей, находящихся в библиотеке объектных модулей, то можно использовать указанное свойство для организации связей программы пользователя с некоторыми макрокомандами системы ввода-вывода.

Вышесказанное реализовано в макросистеме ввода-вывода ДОС ЕС для главных макрокоманд DTFaа и ааMOD. Связь между этими макрокомандами устанавливается с помощью адресных констант, помещаемых в таблицу DTF, которая получается в результате трансляции одноименной макрокоманды.

Макроопределения этих макрокоманд составлены таким образом, что по желанию программиста можно макрокоманду ааMOD в программе не писать, а в соответствующей макрокоманде DTFaа указать имя объектного модуля, содержание которого эквивалентно содержанию макрорасширения опускаемой макрокоманды. После трансляции такой программы в ней появляется *неразрешенная ссылка*, которая указывает на неописанную в тексте программы переменную.

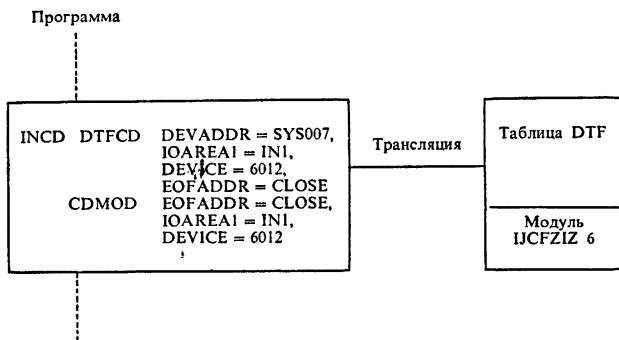
При редактировании полученного объектного модуля Редактор связей, работая в режиме автоматического поиска, осуществляет необходимые действия для «разрешения» вышеуказанной ссылки. Эти действия состоят в том, что Редактор связей просматривает библиотеку объектных модулей, отыскивая в ней подходящий модуль для «разрешения» неопределенной ссылки, и автоматически присоединяет данный модуль к основной программе.

Подтверждением вышесказанного является наличие в макрокомандах DTFaа и ааMOD операндов, предназначенных для указания такого режима использования макрокоманд, при котором они транслируются отдельно от программ, их использующих.

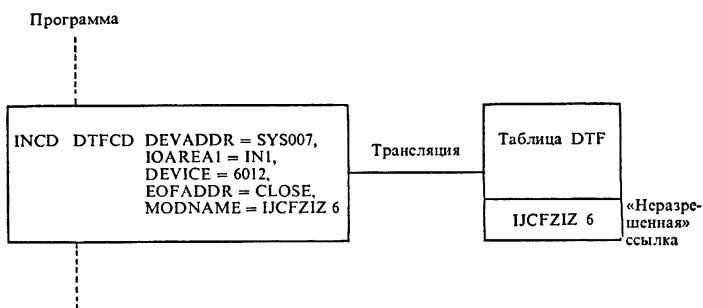
Например, операнд SEPASMB можно использовать для предварительной подготовки «заготовок» макрорасширений. «Заготовки» следует помещать в системную библиотеку объектных модулей для их последующего присоединения к программам пользователя в процессе их редактирования.

Два наиболее распространенных способа применения рассмотренных макрокоманд в DOC ЕС приведены на рис. 6.3 (вариант а, б).

В варианте а при написании программы были использованы макрокоманды DTFCД и CDMOD. После трансляции макрокоманда DTFCД превратилась в таблицу



Вариант а)



Вариант б)

Рис. 6.3.

DTF, а CDMOD — в модуль, который предназначен для обработки запросов, формирования канальных программ и запуска операции ввода с помощью физической системы ввода-вывода, т.е. для обеспечения выполнения операций ввода.

В варианте б (рис. 6.3) при написании программы использована только макрокоманда DTFCД. После трансляции макрокоманда превращается в таблицу, а программа,

кроме этого, имеет еще «неразрешенную» ссылку. Эта ссылка соответствует названию системного модуля, обеспечивающего непосредственное выполнение операций ввода. Присоединение модуля к основной программе происходит в процессе ее редактирования.

До сих пор рассматривались способы организации связи макрокоманд, употребляемых в программах пользователя, с программами системы управления данными на этапах трансляции и редактирования. Эти способы являются основными в СУД ДОС ЕС. Однако установление рассматриваемых связей принципиально возможно и на этапе выполнения программ. Этот способ является основным в операционной системе ОС ЕС.

Как известно, программа на уровне ассемблера в ДОС ЕС почти всегда зависит от типов внешних устройств, с которыми она взаимодействует в процессе решения. Обуславливается это главным образом тем, что основные макрокоманды описания файлов и модулей, обеспечивающих доступ к ним, предназначены для описания обработки данных, расположенных на внешних устройствах только одного типа, т.е. для каждого типа внешних устройств в макросистеме ввода-вывода ДОС ЕС имеются свои макрокоманды.

Таким образом, если программу пользователя необходимо использовать с другим типом устройств, то ее необходимо переделать. Переделка состоит в замене одних макрокоманд другими. К таким макрокомандам относятся в основном макрокоманды описания. Например, на рис. 6.3 описывается файл, который будет вводиться с перфокарт (устройство 6012). Для того чтобы данная программа смогла считывать данные с магнитной ленты, необходимо макрокоманды DTFCД и CDMOD заменить на DTFMТ и MTMOD соответственно.

Вышеприведенные рассуждения подводят к следующему. Если в макросистеме ввода-вывода унифицировать макрокоманду описания файлов, т.е. сделать ее единой для использования в работе с различными типами устройств, и если ликвидировать макрокоманды, описывающие системные модули, а присоединять соответствующие им стандартные системные подпрограммы непосредственно в процессе выполнения программы, то программа может стать независимой от типов внешних устройств, с которыми она будет взаимодействовать.

Макрокоманда описания в этом случае используется для описания файла на некотором абстрактном, логичес-

ком уровне. В процессе выполнения основной программы становится известным конкретный тип устройства, на котором располагается описанный выше файл. В соответствии с этим, при выполнении ранее упоминавшейся процедуры «открыть», вводятся в оперативную память и присоединяются к основной программе необходимые системные модули.

Описанный способ организации связи макрокоманд с программами системы управления данными в ОС ЕС принят

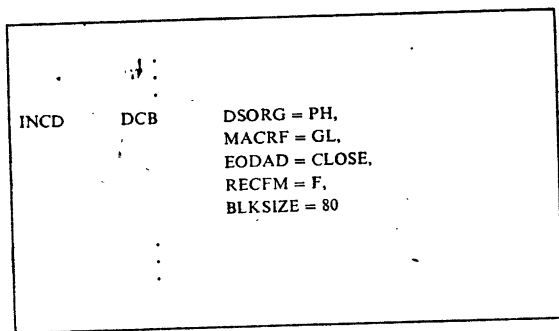


Рис. 6.4.

в качестве основного. Для описания файлов, размещаемых на стандартных устройствах всех типов, используется макрокоманда DCB. Например, описание входного файла на перфокартах, эквивалентное описанию, приведенному на рис. 6.3, дано на рис. 6.4. В приведенном примере тип устройства определяется системой в момент выполнения программы.

Отметим в заключение, что декларативные макрокоманды макросистемы ввода-вывода являются по сравнению с императивными более сложными в использовании.

Следующий параграф посвящен главным образом описанию файлов, а также процедурам открытия и закрытия файлов, как наиболее важным элементам системы управления данными в ОС ЕС.

6.4. Описание и обеспечение доступа к наборам данных

Общие положения. Практически написание любой программы, предназначенной для обработки данных, в рамках операционных систем начинается с описания обрабатываемых наборов данных. Рассмотрим основные элементы

описания файлов, составление которых входит в обязанности программиста. Что же должен указывать программист в описании отдельного файла?

1. Должны быть указаны логические характеристики файла. К ним относятся способ организации, уровень управления данными (метод доступа) и структура данных. Методы доступа были рассмотрены ранее. Структура данных определяет внутреннее строение файла. Конкретно структура описываемых данных задается значениями таких параметров, как формат записей, максимальный размер блоков, максимальный размер логических записей, размер ключей, сопровождающих записи на магнитном диске, и ряд других.

2. Должны быть описаны основные характеристики, касающиеся динамических процессов обработки данных. К таким характеристикам, называемым в дальнейшем характеристиками обработки, относятся: методы буферизации и способы работы с ними, методы обработки ошибок ввода-вывода, методы использования специальных программ, написанных пользователем, и особые случаи обработки, указываемые с помощью специфических параметров. В качестве примера использования специальных программ, написанных пользователем, можно привести программу, с помощью которой завершается обработка данных. Эта программа получает управление от системы управления данными в момент обнаружения физического конца данных. К особым случаям обработки данных можно отнести, например, осуществление контроля правильности записи данных на магнитный носитель.

3. Должны быть описаны физические характеристики файлов, к которым следует отнести: имя файла, записываемое на физическом носителе; устройство, на котором описываемый файл будет установлен; физический том, содержащий набор данных; состояние файла в момент обработки; объем вспомогательной памяти, необходимой для размещения набора данных; параметры для занесения в метки (для файлов, которые должны содержать метки).

Все указанные физические характеристики должны быть описаны для файлов, впервые записываемых на магнитные диски. При считывании таких файлов обычно не указывают объем вспомогательной памяти и параметры метки. Для файлов, записываемых на магнитные ленты, не указывают только объем вспомогательной памяти, а при считывании не указывают также параметры метки. Для других типов носителей информации при описании файлов чаще

всего используют только тип или адрес устройства и текущее состояние.

Текущее состояние определяет, является ли набор данных новым, старым, модифицируемым и т.д. По заданному текущему состоянию файла система управления данными выполняет конкретные действия, связанные с созданием метки файла (если он относится к категории новых), с осуществлением операций поиска набора данных и считыванием его метки (если файл относится к категории уже существующих), и ряд других. Кроме того, текущее состояние определяет действия СУД в момент завершения работы с данным файлом. Например, по окончании обработки набора данных его можно уничтожить или сохранить.

Более детально логические, физические характеристики, а также характеристики обработки будут рассмотрены далее.

Средства описания наборов данных. Ранее упоминалось о том, что управление конкретным набором данных СУД осуществляется с помощью одной или нескольких таблиц, содержащих полную информацию о данном файле.

Центральной и основной является таблица описания файла, называемая *блоком управления данными* (кратко DCB в ОС ЕС и DTF в DOC ЕС). Другие таблицы являются системными и используются для внутренних нужд программами СУД. Программисту приходится иметь дело главным образом с основной таблицей — блоком управления данными. Более того, создание такого блока и определение конкретных значений основных его элементов является одной из главных задач программиста, составляющего программу, предназначенную для обработки файла.

Система управления данными располагает нужными средствами, которые позволяют программисту «строить» блок управления данными. Упомянутые средства представляют собой довольно сложный комплекс, полное и эффективное использование которого требует от программиста большого опыта, определенного навыка и высокой квалификации. Из всего комплекса средств выделим три главных и определяющих.

Первое — макрокоманда DCB. С помощью этой макрокоманды, входящей в макросистему ввода-вывода ОС ЕС, программист выделяет (резервирует) область оперативной памяти нужного размера для размещения таблицы. Место расположения таблицы в программе определяется также программистом по ходу ее написания. Значения основных элементов таблицы записывают в виде параметров этой

макрокоманды. Однако только с помощью операндов макрокоманды DCB полностью заполнить таблицу нельзя. Незаполненными остаются те поля таблицы, которые описывают физические характеристики файла, такие, как тип устройства, его адрес и т.д. Это дает возможность, как уже упоминалось ранее, создавать программы, обладающие определенной независимостью от типа устройств.

КАР	DCB	DSORG = PS,	1 параметр X
		MACRF = (GM),	2 параметр X
		RECFM = FB,	3 параметр X
		BLKSIZE = 80,	4 параметр X
		LRECL = 80,	5 параметр X
		EODAD = EOND,	6 параметр X
		DDNAME = BBD	7 параметр

Рис. 6.5.

Для указания недостающих значений полей таблицы управления данными используются другие средства описания данных. На рис. 6.5 приведен пример описания последовательного набора данных с помощью макрокоманды DCB.

В примере логические характеристики файла определены с помощью значений параметров 1–5. Характеристики обработки, необходимые в данном конкретном случае, определены значениями параметров 2 и 6. О значении 7-го параметра будет сказано ниже.

Логические характеристики определяют метод доступа и структуру файла.

Первый параметр указывает, что описываемый файл использует последовательный способ организации. Вторым и третьим параметрами указывают, что используется логический уровень управления данными. Параметры 3, 4, 5 указывают, что структура файла организована блоками фиксированной длины по 80 байтов в каждом.

Буква G во втором параметре DCB указывает на использование при обращении к файлу за очередной порцией данных макрокоманды GET. Характеристики обработки, указанные в данном примере, относятся к определению способа буферизации и определению адреса под-

программы, которой будет передано управление после обнаружения физического конца данных. Буква М во втором параметре указывает на конкретный режим обработки данных при некотором подразумеваемом способе буферизации. Более конкретно об этом будет сказано ниже. Адрес подпрограммы выхода указан в 6-м параметре.

На основании вышесказанного отметим, что 2-й параметр содержит величины, которые влияют на определение как логических характеристик файла, так и характеристик его обработки. Это, естественно, заставляет предположить, что с точки зрения использования параметров макрокоманды описания параметр MACRF наиболее сложен, а, следовательно, является источником значительного количества ошибок при программировании. Поэтому программистам необходимо более внимательно относиться к использованию этого параметра.

Однако вернемся к рассмотрению примера. По описанию, приведенному на рис. 6.5, нельзя судить о том, на каком типе внешних устройств располагается файл. Таким устройством может быть и устройство ввода с перфокарт, и магнитная лента, и накопитель на магнитном диске. Тип внешнего устройства должен быть определен к моменту начала обработки файла. В макрокоманде DCB нет параметров, с помощью которых можно определить тип внешнего устройства.

Однако, чтобы иметь возможность описать, а затем и дополнить таблицу управления данными сведениями об основных физических характеристиках файла, в макрокоманде DCB предусмотрен специальный обязательный параметр — DDNAME (7-й параметр на рис. 6.5). Этот параметр указывает, где будут определены основные физические характеристики описываемого файла. Физические характеристики файла в операционных системах ЕС ЭВМ указываются при помощи средств языка управления заданиями (ЯУЗ). В ОС ЕС к ним относится один из основных операторов ЯУЗ — оператор DD.

Оператор DD относится ко второму средству, с помощью которого программист «строит» блок управления данными. В этом операторе могут быть указаны рассмотренные ранее физические характеристики файла. Продолжение описания рассматриваемого файла и определение основных физических характеристик приведены на рис. 6.6, где даны три различных варианта написания оператора DD с именем BBD. Название оператора ЯУЗ, описывающего физические характеристики набора данных, должно

быть указано в макрокоманде DCB (см. параметр 7 на рис. 6.5). Таким образом, оператор DD содержит операнды, которые как бы продолжают описание набора данных, начатого в макрокоманде DCB.

Вариант *а* (рис. 6.6) может быть использован для обработки файла, расположенного на устройстве ввода

//BBD DD UNIT = 6012			
а) Для устройства ввода перфокарт			
//BBD	DD	DSNAME = A.BNOD,	1 параметр X
//		UNIT = 5012,	2 параметр X
//		VOL = SER = L00012,	3 параметр X
//		DISP = OLD	4 параметр
б) Для накопителя на магнитной ленте			
// BBD	DD	DSNAME = A.BNOD,	1 параметр X
//		UNIT = 5056,	2 параметр X
//		VOL = SER = D00023,	3 параметр X
//		DISP = OLD	4 параметр
в) Для накопителя на магнитном диске			

Рис. 6.6.

с перфокарт. Для этого в операторе DD необходимо указать либо тип устройства, либо его физический адрес.

Вариант *б* (рис. 6.6) может быть использован для обработки файла, расположенного на магнитной ленте. Первый параметр определяет название файла, которое записывается в поле стандартной метки, 2-й параметр указывает тип накопителя на магнитной ленте, 3-й — серийный номер тома, на котором записан файл. И, наконец, 4-й параметр определяет состояние файла к моменту обработки. В данном случае файл определен как старый, т.е. записанный ранее на носитель. Указанное состояние, кроме этого, требует сохранения файла после окончания обработки.

Вариант *в* (рис. 6.6) отличается от варианта *б* значениями двух параметров. 2-й и 3-й параметры оператора DD задают соответственно тип накопителя на магнитном диске и серийный номер пакета дисков, используемого для хранения файла.

Естественно, любой из приведенных вариантов описания физических характеристик файла может быть применен при

обработке указанного набора данных. Известно, что операторы ЯУЗ не являются составными частями тела программы и, следовательно, не подлежат компиляции и редактированию. Обработка операторов ЯУЗ осуществляется программами системы управления заданиями. В связи с этим возникает логичный вопрос: «Почему описание физических характеристик вынесено из тела программы и подлежит реализации на уровне языка управления заданиями?». В качестве ответа на поставленный вопрос отметим два основных преимущества при таком способе описания физических характеристик файлов.

1. Программы обработки файлов становятся независимыми от типа устройств. Примеры, приведенные на рис. 6.5 и 6.6, подтверждают это. Действительно, с помощью фрагмента программы, содержащей описание файла (рис. 6.5), можно производить обработку данных, записанных на устройствах различных типов. Для этого не надо ни переделывать программу, ни заново ее транслировать и редактировать. Достаточно в задании на исполнение указанной программы заменить один вариант оператора DD на другой, чтобы программа была перенастроена на иной тип устройства.

2. Физические ресурсы, необходимые для выполнения программы, становятся известными операционной системе до начала исполнения программы. Это позволяет упростить процессы постановки задач на ЭВМ, давая возможность планировать использование внешних устройств и носителей информации до запуска программы, что, в свою очередь, уменьшает вероятность блокирования системы.

Блокировка системы представляет собой такое состояние вычислительной машины, при котором все решаемые задачи, находящиеся в оперативной памяти, не могут быть решены до конца без удаления одной или нескольких из них. Если производится удаление начатой задачи, то время, затраченное на ее запуск и счет, теряется впустую.

Проиллюстрируем состояние блокировки на примере. Пусть необходимо решить две задачи. Для одной требуются три файла с номерами 1, 2, 3, а для другой — два файла с номерами 2 и 3. Причем одновременно использовать файлы 2 и 3 задачи не могут. Очевидно, если эти сведения известны системным программистам, осуществляющим запуск задач на ЭВМ, до их исполнения, то одну из вышеописанных задач можно задержать с запуском до тех пор, пока не освободятся необходимые ресурсы. Если же необходимые ресурсы будут известны

только в процессе решения, то легко может возникнуть ситуация блокировки.

Пусть при выполнении двух вышеуказанных программ первая сначала использует файлы 1 и 3, а вторая — файл 2. Выполнение этих программ будет продолжаться до того момента, пока первой потребуется файл 2, а второй — файл 3. Далее произойдет блокирование решения обеих задач. Ликвидировать создавшуюся ситуацию можно путем удаления одной из рассматриваемых программ и освобождения необходимых ресурсов.

Ясно, что полное знание всех ресурсов, необходимых программе до ее запуска, позволяет практически исключить ситуации, аналогичные описанной.

Наконец, третье средство, которым пользуется программист для заполнения блока управления данными, основано на использовании стандартных меток при работе с файлами на магнитных носителях. Известно, что стандартная метка файла представляет собой некоторую таблицу, записываемую на носитель и сопровождающую файл. Эта таблица содержит информацию как о логических, так и о физических характеристиках файла. Однако использование упомянутой информации для построения блока управления данными сопряжено с некоторыми особенностями:

- 1) применять метку можно только при описании файлов, расположенных на магнитных носителях и имеющих текущее состояние, — старый, т. е. ранее записанный;

- 2) применять метку для указанной цели можно только неявным способом; для этого необходимо точно представлять те действия системы управления данными, которые предшествуют окончательному формированию блока управления данными.

Для иллюстрации указанных особенностей рассмотрим тот же самый файл (рис. 6.7), что был описан на рис. 6.5 и 6.6.

Первую особенность, очевидно, пояснять нет необходимости, так как стандартные метки файлов используются только в наборах данных, располагаемых на магнитных носителях, а также только после того, как файл будет размещен на носителе. Напомним, что непосредственно перед занесением набора данных на магнитный носитель туда помещается стандартная метка, информация в которую заносится из блока управления данными, описывающего этот файл.

Если вернуться к примеру (рис. 6.7), то можно заметить в описании файла отсутствие некоторых параметров,

определяющих такие логические характеристики файлов, как формат, размеры записей и блоков. Что это? Ошибка? Нет, в приведенном описании нет ошибки. Более того, следует подчеркнуть, что в данном примере использованы

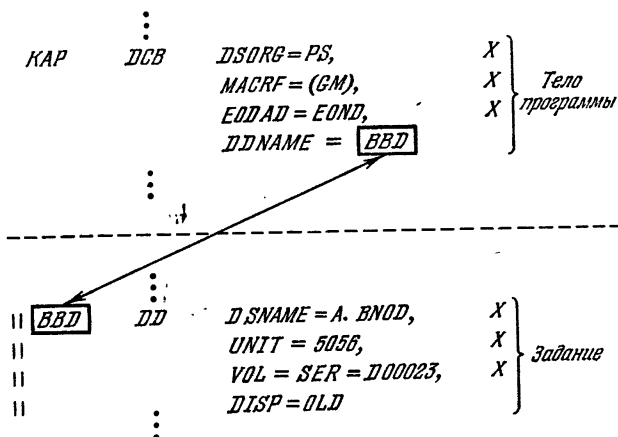


Рис. 6.7.

все средства, позволяющие определять конкретное содержание основных полей блока управления данными. Здесь неявным способом указана информация о логических характеристиках файла, извлекаемая из меток. Таким образом, система управления данными формирует основные параметры таблицы описания данных в три стадии.

На первой стадии с помощью макрокоманды DCB резервируется необходимая область оперативной памяти и формируются значения основных элементов логических характеристик и характеристик обработки блока управления данными, задаваемых параметрами макрокоманды.

На второй стадии в таблицу описания данных переносится информация об основных физических характеристиках, заданных в операторе DD языка управления заданиями.

И, наконец, на третьей стадии для файлов, ранее записанных на магнитные носители, недостающую информацию система управления данными может извлекать из стандартных меток в процессе их обработки.

Основные параметры файлов, которые могут быть заданы с помощью трех рассмотренных средств описания, распределенных по методам доступа, приведены в табл. 6.3.

Основные средства описания файлов

Таблица 6.3

Средство описания	Характеристика	Операнд	Методы доступа					
			QSAM	BSAM	BPAM	QISAM	BISAM	BDAM
Макрокоманда DCB	Логическая	BLKSIZE*	+	+	+	+	+	+
		CYLOFL*	-	-	-	+	-	-
		DSORG*	+	+	+	+	+	+
		KEYLEN*	-	+	+	+	-	+
		LRECL*	+	+	+	+	-	+
		NTM*	+	+	+	+	-	+
		RECFM*	+	+	+	+	-	+
		RKP*	-	-	-	+	-	-
	Логическая и обработки	MACRF	+	+	+	+	+	+
	Обработки	BFALN*	+	+	+	+	+	+
		BFTEK*	+	+	-	-	-	+
		BUFCB	+	+	+	+	+	+
		BUFL*	+	+	+	+	+	+
		BUFNO*	+	+	+	+	+	+
		EODAD	+	+	+	+	-	-
		EROPT*	+	-	-	-	-	-
		EXLST	+	+	+	+	+	+
		LIMCT*	-	-	-	-	-	+
		MSHI	-	-	-	-	+	-
		MSWA	-	-	-	-	+	-
		NCP*	-	+	+	-	+	-
		OPTCD*	+	+	+	+	-	+
		SMSI	-	-	-	-	+	-
		SMSW	-	-	-	-	+	-
		SYNAD	+	+	+	+	+	+
	Физическая	DDNAME	+	+	+	+	+	+
		DEVD	+	+	-	-	-	-
Оператор DD	Физическая	DSNAME	+	+	+	+	+	+
		UNIT	+	+	+	+	+	+
		VOLUME	+	+	+	+	+	+
		DISP	+	+	+	+	+	+
		SPACE	+	+	+	+	+	+
		LABEL	+	+	+	+	+	+

Для стандартных меток файлов выделены основные параметры блока управления данными, которые можно задавать неявно.

Открытие и закрытие наборов данных. Ранее отмечалось, что непосредственно в программе программист может только выделить участок оперативной памяти и заполнить часть полей блока управления данными. Для использования блока управления данными его необходимо «достраивать», используя подходящие значения операндов, задаваемые частично в операторах языка управления заданиями, частично в стандартных метках файла, частично вырабатываемые системными программами управления данными.

«Достраивание» блока управления данными, предназначенного для обслуживания отдельного файла, выполняется комплексом системных программ, образующих специальную процедуру, которая называется *процедурой открытия наборов данных*. Вызов этой процедуры осуществляется непосредственно программистом. Процедура открытия наборов данных вызывается с помощью макрокоманды OPEN (открыть), входящей в состав макросистемы управления данными.

Система управления данными может рассматриваться как некая система обслуживания, выполняющая заявки-запросы на выполнение операций обмена данными между внешними устройствами ЦВМ и ее оперативной памятью. Как было сказано выше, заявки-запросы могут исходить как от программ, написанных пользователем, так и от ряда программ, входящих в состав операционной системы.

Программы управления данными, образующие значительную часть операционной системы, выполняют конкретные вышеупомянутые запросы-заявки, используя для этого главным образом информацию, находящуюся в блоках управления данными и описывающую текущее состояние файла, к которому относится запрос. Поэтому системой программирования ввода-вывода предусмотрено, что перед тем, как помещать в программе запросы на обмен (команды ввода-вывода), необходимо, с одной стороны, поместить в нужные блоки управления данными информацию о начальном состоянии соответствующих файлов, а с другой — связать вышеуказанные блоки с системными программами управления данными для обеспечения последующего обслуживания запрограммированных запросов.

Описанные выше действия выполняются стандартной процедурой открытия наборов данных, выполнение которой должно программироваться с помощью макрокоманды OPEN. Макрокоманда может использоваться таким образом,

что с ее помощью могут быть открыты сразу несколько блоков управления данными. В дальнейшем будем говорить также, что макрокоманда OPEN открывает наборы данных или файлы, так как результатом исполнения процедуры открытия наборов данных является готовность системы управления данными обеспечить доступ к конкретным файлам.

Процедура открытия наборов данных используется программистами очень часто. В процессе использования следует иметь ввиду следующие особенности ее применения:

1) блок управления данными полностью строится только в процессе работы процедуры открытия файлов;

2) исполнение процедуры открытия в значительной мере зависит от правильности и корректности описаний открываемых наборов данных, записанных с помощью средств, рассмотренных выше;

3) понимание физического смысла выполнения процедуры открытия значительно облегчает программирование описания, уменьшает вероятность появления ошибок в описаниях и, наконец, как показывает опыт, оказывает существенную помощь на этапах отладки программ.

Перейдем к рассмотрению основных функций процедуры «открыть».

1. Заполнение основных полей блока управления данными, содержание которых должно быть указано программистом. Так как информацию о конкретном файле программист вынужден рассредоточивать по различным источникам (DCB, DD, метка), то задачей процедуры «открыть» является процесс собирания и согласования (увязки) всех составных элементов описаний, задаваемых различными средствами. Проблема согласования значений отдельных параметров появляется тогда, когда различные средства (например, DCB и DD) содержат значения одних и тех же операндов (полей) таблицы управления данными. Если значения таких операндов совпадают, то никакого согласования не требуется. А если средства описания содержат различные значения одних и тех же операндов, то в процедуре открытия файлов предусмотрена определенная очередность, задающая приоритет выбора значений того или иного операнда между средствами макросистемы управления данными (DCB), операторами языка управления заданиями (DD) и использованием информации из стандартных меток.

Например, размер блока некоторого файла, находящегося на магнитной ленте, может быть указан и в макро-

команде DCB, и в операторе DD, и, наконец, в стандартной метке. Значение этого операнда в блоке управления данными будет выбрано таким, каким оно указано в макрокоманде DCB. Если значение того же операнда в макрокоманде DCB не указано, то оно будет взято из оператора DD. Если значение операнда не указано и в макрокоманде DCB, и в операторе DD, то оно будет взято из соответствующего поля стандартной метки в момент открытия файла. Естественно, что все сказанное о стандартной метке имеет смысл тогда, когда речь идет о файлах, записанных на магнитные носители, т.е. когда стандартная метка существует

Таким образом, при формировании основных полей блока управления данными наивысший приоритет имеет информация из макрокоманды DCB, следующей по приоритету является информация из DD оператора и самый низкий приоритет имеет информация из метки файла.

Попутно отметим, что такой способ формирования блока управления данными позволяет создавать программы с достаточной степенью независимости как от некоторых логических, так и от большинства физических характеристик файлов. Иллюстрируется это уже рассмотренными примерами. Выделим только пример, приведенный на рис. 6.7 В этом описании файла отсутствуют некоторые логические характеристики: формат записей, размеры блоков и логических записей. Подразумевается, что значения этих характеристик будут взяты из метки файла. Поэтому данную программу (с некоторыми ограничениями), в принципе, можно использовать для обработки файлов, имеющих различные форматы записей, размеры блоков и логических записей.

2. Связывание блока управления с системными программами управления данными. Для обеспечения выполнения этой функции процедура «открыть» должна произвести следующие действия.

- установить необходимые связи с системными таблицами,

- поместить в область задачи нужные системные программы методов доступа,

- построить, если необходимо, нужное количество буферов;

- построить канальные программы для осуществления непосредственного доступа к данным.

Связь с системными таблицами устанавливается для того, чтобы сделать доступным СУД блок управления

данными, а через него собственно и набор данных. При выполнении этих действий таблица управления данными «прикрепляется» к системной таблице, описывающей физическое устройство, на котором должен находиться обрабатываемый файл. «Прикрепление» осуществляется путем установки в обе таблицы адресных констант, указывающих место взаимного расположения таблиц в оперативной памяти.

Ранее отмечалось, что в СУД ОС ЕС программы методов доступа помещаются в область памяти, где находится решаемая задача, только на этапе ее выполнения. Эта операция осуществляется процедурой «открыть», в момент выполнения которой, во-первых, определяется метод доступа и соответственно ему перечень необходимых, обеспечивающих его, системных программ-модулей. Во-вторых, программы-модули методов доступа, включенные в вышеуказанный перечень, вызываются в свободную область оперативной памяти, принадлежащую задаче. В-третьих, адреса, определяющие точки входа загруженных модулей, помещаются в блок управления данными. Обращение к большинству программ-модулей методов доступа осуществляется только через блок управления данными. Этим обстоятельством в основном и вызвана необходимость записи начальных адресов программ-модулей в таблицу.

Во многих методах доступа программисты используют автоматические способы буферизации, т. е. такие, при которых построение необходимого количества буферов и реализация выбранных режимов работы с буферами обеспечиваются программами СУД.

Соответствующая подготовка производится процедурой «открыть», во время исполнения которой выделяется область оперативной памяти, предназначенная для буферов ввода-вывода и называемая *буферным пулом*. Буферный пул используется для обеспечения нужным количеством буферов планируемых операций обмена.

Наконец, процедура «открыть» конструирует подходящие каналные программы, с помощью которых должно осуществляться программное управление внешним устройством, обеспечивающее выполнение команд ввода-вывода. Точнее говоря, в процессе открытия файла в составе загруженных модулей доступа имеются заготовки каналных программ, а окончательное построение их состоит в том, что на основании информации о методе доступа, о режимах работы с файлом, о месте нахождения буферов и их количества заготовки насыщаются соответствующими данными. Эти

данные помещаются в такие поля канальных команд, как код операции, адрес буфера и счетчик, а также в поле признаков.

В результате выполнения вышеописанных действий система управления данными как бы принимает на учет открываемый файл, что позволяет ей выполнять запросы на операции ввода-вывода. Однако приступить к исполнению запросов можно только после выполнения некоторых обязательных физических действий, составляющих третью основную функцию процедуры «открыть».

3. Физическая подготовка устройства и тома. Главное содержание этой функции состоит в том, что процедура «открыть» осуществляет:

- проверку фактической готовности внешнего устройства;
- проверку правильности установки физического тома, на котором записан или будет записываться файл;
- поиск открываемого файла на носителе;
- подготовку внешнего устройства к чтению или записи первого блока на носитель.

Проверка фактической готовности внешнего устройства состоит в установлении работоспособности требуемого устройства. Эта проверка, в принципе, должна производиться для всех типов внешних устройств. Если для устройств ввода-вывода этой проверки оказывается достаточно, чтобы переходить к обработке данных, то для внешних запоминающих устройств необходимо выполнить еще целый ряд действий, упомянутых выше. К ним относится проверка правильности установки тома (или томов), на котором должен храниться открываемый файл.

На этом этапе проверяется совпадение метки тома, установленного на устройстве, с меткой тома, описанной в операторе DD языка управления заданиями, соответствующего данному файлу. Если метки не совпадают, то может быть осуществлена смена тома, которую должен выполнить оператор системы согласно сообщению, посылаемому процедурой «открыть».

● После установки нужного тома процедура «открыть» производит поиск открываемого файла на установленном носителе. Для файлов, записанных ранее, эта операция состоит в том, что на основании данных о названии и метке файла, помещенных в операторе DD, отыскивается и считывается стандартная метка файла. Для вновь создаваемых файлов на основании информации, находящейся в таблице DSB, на носитель записывается стандартная метка. Кроме того, если необходимо, отмечаются физи-

ческие границы расположения файла, используемые в последующем для защиты данных файла от некорректного обращения.

После проверки правильности установки тома, поиска файла или подготовки области внешней памяти для файла производится подвод головок в то место физического тома, с которого будет начинаться обработка набора данных. Чаще всего головки устанавливаются для чтения первого блока данных, но иногда первый блок (блоки) во время открытия файла считывается в выделенный для него буфер (буферы).

Рассмотренные нами три функций составляют главное содержание процедуры открытия файлов. Отметим только, что выполнение указанных функций осуществляется не совсем в том порядке, в каком они были рассмотрены. Например, информация, находящаяся в стандартной метке, может быть помещена в DCB только после того, как стандартная метка будет считана в оперативную память.

После завершения процедуры открытия файлов можно приступать к обработке данных, находящихся в файлах. Обработка обычно производится программами, составленными программистом, а обмен данными — программами методов доступа. Программирование последних осуществляется с помощью основных макрокоманд, входящих в состав макросистемы управления данными. Чаще всего для логического уровня управления используются макрокоманды GET и PUT, а для базисного — READ и WRITE.

Когда обработка того или иного набора данных заканчивается, необходимо предусмотреть в программе выполнение процедуры, называемой *процедурой закрытия файлов*. Эта процедура выполняет функции, в некотором смысле обратные функциям процедуры «открыть». А именно, разрушает логические связи блока управления данными с конкретным файлом, разрушает связи, установленные с СУД, освобождая при этом те области основной памяти, которые были использованы для установления этих связей, освобождает блок управления данными, делая его доступным для использования в других целях.

Процедура «закрыть» вызывается с помощью макрокоманды CLOSE, которую программист должен предусмотреть в своей программе сразу же после завершения обработки файла. В принципе, конечно, можно и не программировать выполнение этой процедуры, так как после решения задачи все открытые наборы данных будут автоматически закрыты программами операционной системы.

Однако при этом следует иметь в виду, что каждый файл, обработка которого закончена, потенциально снижает производительность машины. Обуславливается это множеством факторов. Отметим некоторые из них:

1) незакрытый файл занимает физические ресурсы, которые могли бы быть использованы для решения других задач;

2) увеличивается расход занятой оперативной памяти, которая также могла бы быть использована для других целей;

3) увеличиваются накладные расходы по времени выполнения запросов на операции ввода-вывода, получаемых даже от других задач, так как на «учете» системы управления данными стоит большее количество файлов, чем это необходимо для обслуживания.

По тем же причинам программирование процедуры «открыть» нужно начинать непосредственно перед началом обработки файлов, т. е. следует избегать открытия файлов заранее. Следовать упомянутым рекомендациям не так уж сложно, поскольку написание макрокоманд OPEN и CLOSE не вызывает затруднений даже у начинающих программистов.

Перейдем к рассмотрению конкретного примера, иллюстрирующего применение описанных в данной главе средств описания и открытия файлов.

Пример обработки данных. В качестве примера рассмотрим программирование простейшей задачи, суть которой состоит в следующем. На магнитную ленту, имеющую регистрационный и соответственно серийный номер L00013, записан некоторый массив перфокарт с именем КАР. Необходимо, используя логический уровень управления данными, вывести содержимое всех перфокарт на АЦПУ в том же порядке и в том же коде, в каком они записаны на магнитной ленте. Другие недостающие исходные данные будут указаны по ходу решения. В этой задаче в процессе ее решения отсутствует какая-либо обработка. Сделано это с целью более наглядной иллюстрации применения средств системы управления данными. Текст программы, обеспечивающий решение задачи, приведен на рис. 6.8.

Текст программы написан на языке ассемблер. Полный перечень наименований инструкций языка ассемблер, использованных в этом и последующих примерах, приведен в [11].

Первые восемь команд являются стандартными для программ, составляемых в рамках операционной системы

ОС ЕС, поэтому рассматривать их не будем. Отметим только, что указание и загрузка базового регистра (в нашем примере — 2) должны осуществляться программистом.

В рассматриваемой задаче необходимо описать два набора данных. Первый — исходный набор данных, второй —

№ п/п	Имя	КОП	Операнды и комментарии
1		START	
2		SAVE	(14, 12)
3		USING	HALT, 2
4		BALK	2,0
5	HALT	ST	13, SU + 4
6		LR	10, 13
7		LA	13, SU
8		ST	13,8 (10)
9		OPEN	(BNOD), (INPUT), BEND, (OUTPUT)
10	BEGIN	GET	BNOD, BU ВВОД ПЕРФОКАРТЫ
11		PUT	BEND, BU ВЫВОД НА АЦПУ
12		B	BEGIN
13	KON	CLOSE	(BNOD, BEND)
14		L	13, SU + 4
15		RETURN	(14, 12)
16	SU	DS	18F ОБЛАСТЬ СОХРАНЕНИЯ РЕГИСТРОВ
17	BU	DS	CL80 РАБОЧАЯ ОБЛАСТЬ
18		DC	48C РЕЗЕРВНАЯ ОБЛАСТЬ
19	BNOD	DCB	DSORG = PS, ОПИСАНИЕ
20			MACRF = (GM), ВХОДНОГО
21			EODAD = KON, ФАЙЛА
22			DDNAME = BBD,
23	BEND	DCB	DSORG = PS, ОПИСАНИЕ
24			MACRF = (PM), ВЫХОДНОГО
25			RECEM = FB, ФАЙЛА
26			BLKSIZE = 128,
27			LRECL = 128,
28			DDNAME = BYEND
29		END	

Рис. 6.8.

набор данных, выводимый на АЦПУ. Описание обоих файлов осуществлено в программе путем использования макрокоманд DCB, имеющих имена BNOD для исходного и BEND для результирующего наборов данных.

Описание блока управления данными BNOD содержит четыре операнда, находящихся в строках 19–22 текста программы. Операнд, указанный в строке 19, имеет значение, определяющее последовательный способ организации файла.

Следующий операнд является более сложным и требует более детального рассмотрения. Значением этого операнда являются две буквы G и M. Первая буква обозначает, что для чтения данных из описываемого файла будет использоваться макрокоманда GET. Вторая буква обозначает сразу несколько моментов, относящихся к буферизации. Во-первых, для исходного файла будет использован только один буфер, во-вторых, этот буфер будет построен автоматически процедурой «открыть», и, в-третьих, данные, введенные в буфер, будут автоматически пересылаться в рабочую область программы, адрес которой указывается в макрокоманде GET.

Значение операнда, находящегося в строке 21, определяет адрес программы, по которому будет передано управление после обнаружения физического конца файла. Управление по указанному адресу передается программами метода доступа в основном тогда, когда операция чтения данных с устройства заканчивается установкой признака «особый случай» в байте состояния устройства.

Наконец, значение операнда, расположенного в строке 22, указывает имя оператора DD, помещаемого в задание

PROGR	JOB	
	EXEC	PROC = ASMFCLG
ASM.SYSIN	DD	*
	.] текст исходной программы
	.	
	.	
GO.BBD	DD	DSNAME = KAP,
		UNIT = 5012,
		VOL = SER = L00013,
		DISP = OLD
GO.BYEND	DD	UNIT = 00F

Рис. 6.9

на выполнение составленной программы и предназначенного для описания физических характеристик описываемого файла. Операторы языка управления заданиями, записанные в нужном порядке для обеспечения прогона рассматриваемой программы, приведены на рис. 6.9.

Имена операторов DD, записанные в макрокомандах DCSB и соответствующие им имена в задании выделены для лучшей ориентации.

Чтобы завершить рассмотрение описания первого файла, проанализируем параметры, указанные в операторе DD, с именем BBD. Значения этих параметров аналогичны

значениям параметров DD, приведенным на рис. 6.6 (вариант б). В нашем примере имя набора данных и серийный номер тома указаны в соответствии с условиями задачи.

Как видно, описание исходного файла не содержит сведений о логической структуре файла. Предполагается, что эти данные СУД получит в процессе обработки стандартной метки исходного файла. Конкретно значения упомянутых параметров можно принять следующими: запись формата FB, размер блока — 800, размер логической записи — 80 байтов.

Перейдем к описанию выходного набора данных, представленного другой макрокомандой, записанной в строках 23—28 исходной программы. Значение первого операнда не требует пояснений. Значение операнда в строке 24 обозначает использование макрокоманды PUT для вывода данных и использование рабочей области, из которой данные перед выводом будут перенесены в буфер.

Имя рабочей области должно быть сообщено в макрокоманде PUT. Параметры, значения которых помещены в строках 25—27, очевидно, не требуют специального пояснения. Наконец, в последней 28-й строке описания выходного файла содержится имя оператора DD, содержащего физические характеристики файла и помещенного в задание (см. рис. 6.9). Оба имени выделены для облегчения их поиска в тексте программы и задания. В операторе DD для выходного файла указан только один операнд, значение которого соответствует указанию типа и адреса устройства одновременно (00F — физический адрес АЦПУ).

Теперь рассмотрим участок программы, содержащий операции ввода-вывода. Это участок со строками 9—13. В строке 9 записана макрокоманда, вызывающая выполнение процедуры открытия сразу для обоих файлов. В операндах макрокоманды указаны имена блоков управления данными и режимы обработки открываемых файлов. Для исходного файла определен режим считывания, для выходного — режим записи. С помощью следующих трех строк 10—12 запрограммированы следующие действия.

По макрокоманде GET программы методов доступа вызовут в оперативную память блок данных, находящийся на магнитной ленте и подготовленный для чтения, выделяют в буфере первую запись, передадут ее содержимое в рабочую область, имеющую имя BU, и настроят модуль метода доступа на чтение следующей по порядку

логической записи. Следовательно, при повторном выполнении макрокоманды GET будет считана следующая логическая запись.

По макрокоманде PUT данные, находящиеся в рабочей области, будут пересылаться в буфер выходного файла и выводиться на печать. По окончании печати с помощью команды В (безусловный переход) управление будет передано снова команде GET. Таким образом, организован цикл «чтение — печать одной перфокарты», этот цикл будет выполняться до тех пор, пока не будет зафиксирован конец исходного массива.

После обнаружения физического конца данных в исходном файле система управления данными передаст управление процессору для выполнения участка программы, начинающегося со строки 13. В этой строке записана макрокоманда, выполняющая процедуру закрытия файлов.

На этом решение поставленной задачи заканчивается. После макрокоманды закрытия файлов записаны две команды, обеспечивающие завершение работы приведенной программы (строки 14, 15). Так как эти команды являются стандартными, как и первые восемь, то их рассматривать не будем.

Закончив обсуждение основных моментов системы управления данными, связанных с описанием и обеспечением доступа к наборам данных, перейдем к анализу основных режимов управления данными, рассредоточенных по методам доступа.

7.1. Управление последовательно-организованными файлами

Логический последовательный метод доступа. Основной режим управления этого метода доступа был рассмотрен в гл. 6 (см. рис. 6.8 и 6.9). Другие режимы обработки данных на этом уровне управления являются по существу различными модификациями основного и связаны главным образом с модификациями способов буферизации и режимами обработки записей, находящихся в буферах.

Логический последовательный метод доступа предусматривает два способа буферизации — простую и обменную.

1. Простая буферизация. При простой буферизации для каждого обрабатываемого файла выделяется своя область памяти, предназначенная для построения одного или нескольких буферов. Обмен данными между каждым буфером и внешним устройством производится с помощью одной команды канала. Каждый буфер используется для размещения одного блока данных. Блок данных содержит одну или несколько логических записей. Часть буфера, которая содержит одну логическую запись, будем называть *сегментом буфера*. Этим понятием воспользуемся главным образом при рассмотрении основных свойств обменной буферизации.

Проиллюстрируем некоторые свойства простой буферизации. Для примера рассмотрим (рис. 7.1) данный тип буферизации, применяемый в процессе обработки двух файлов — А и В. Причем файл А является входным, т. е. предназначенным для ввода данных, а файл В — выходным. Каждому файлу в некоторой области оперативной памяти (ОП) выделено по одному буферу — буфер А и буфер В. Буфер А может содержать один физический блок файла А, а буфер В — блок файла В. Теперь проанализируем некоторые моменты, связанные с обработкой данных, осуществляемой согласно схеме рис. 7.1. По этой схеме сначала вводятся данные из файла А, затем эти данные исполь-

зуются для обработки, а потом выводятся в файл В. Обработка данных осуществляется последовательно логическими записями. При указанном способе обработки двух файлов (см. рис. 7.1) можно выделить по крайней мере три варианта организации обработки логических записей.

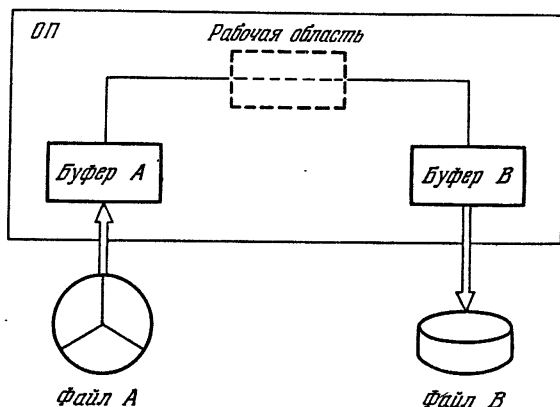


Рис. 7.1.

Первый — каждая логическая запись обрабатывается в отдельном сегменте буфера А и для вывода пересылается в соответствующий сегмент буфера В.

Второй — каждая логическая запись файла А предварительно пересылается в соответствующий сегмент буфера В, в нем обрабатывается и оттуда будет в последующем выведена в файл В.

Третий — каждая логическая запись файла А предварительно из буфера А помещается в некоторую рабочую область (см. рис. 7.1), в ней обрабатывается, а затем пересылается в соответствующий сегмент буфера В для вывода.

Каждый из указанных вариантов организации обработки записей имеет свои преимущества и недостатки. Выделим из них только один общий и характерный для простой буферизации недостаток, который заключается в необходимости перемещения логических записей из одного буфера в другой. Очевидно, что способ выделения памяти под буферы, принятый для простой буферизации, в принципе, обуславливает необходимость как минимум одной пересылки для каждой записи, подвергающейся обработке по схеме: ввод — обработка — вывод.

Рассмотрим вышеуказанные три варианта организации обработки и определим особенности каждого из них при осуществлении пересылок логических записей в оперативной памяти.

При выполнении обработки по первому способу операции пересылки осуществляются после обработки непосредственно перед выводом логической записи.

При выполнении обработки по второму способу операции пересылки осуществляются непосредственно после ввода перед обработкой логической записи. Особенностью первого и второго способов обработки является необходимость выполнения одного перемещения логической записи и использование при этом только тех участков оперативной памяти, которые отведены для буферов.

При выполнении обработки по третьему способу пересылка логической записи осуществляется дважды: одна — до обработки, другая — после обработки. При этом, кроме буферов, дополнительно используется участок оперативной памяти, называемый *рабочей областью*.

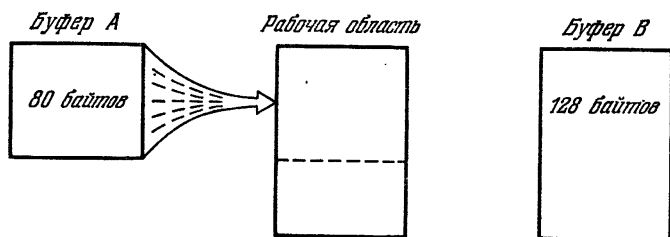
Теперь рассмотрим вопрос, касающийся выбора или составления тех программ, которые должны выполнять действия по пересылке логических записей. Необходимые перемещения логических записей при обработке с использованием простой буферизации могут быть выполнены либо системными программами управления данными (конкретно — программами метода доступа), либо программой, написанной пользователем.

Методы доступа обеспечивают перемещение логической записи во время выполнения макрокоманд ввода-вывода. Для этого в макрокоманде ввода-вывода должна быть помещена информация о начальном адресе области оперативной памяти, в которую или из которой должна будет пересылаться логическая запись. Если макрокоманда ввода-вывода выполняет пересылку логической записи, то говорят, что она обеспечивает обработку записи в режиме пересылки.

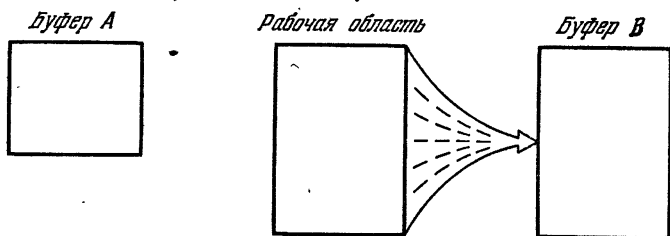
В противном случае говорят, что макрокоманда ввода-вывода обеспечивает обработку записи в режиме указания, т. е. макрокоманда в этом режиме «указывает» начальный адрес того сегмента буфера, в котором находится логическая запись, предназначенная для обработки. Принято, что адрес логической записи сообщается в качестве результата выполнения макрокоманды ввода-вывода в регистре общего назначения с номером 1.

Подводя итог анализу некоторых свойств простой буферизации, отметим, что для простой буферизации характерными являются два режима обработки логических записей: режим пересылки и режим указания. Отметим, что информация об указании конкретного режима обработки при составлении пользовательских программ должна быть помещена в макрокоманде описания файла с помощью операнда MACRF.

Рассмотрим несколько примеров. Сначала вернемся к программе, приведенной на рис. 6.8. В этом примере макрокоманды ввода-вывода для обоих файлов используются в режиме пересылки, задаваемом значением (буква



а) Выполнение макрокоманды GET



б) Выполнение макрокоманды PUT

Рис. 7.2.

М) операндов с номерами 20 и 24 соответственно для входного и выходного файлов. Схема, иллюстрирующая режим обработки логических записей, соответствующий вышеуказанному примеру, приведена на рис. 7.2. Подчеркнем, что пересылку записей осуществляют программы метода доступа, выполняющие макрокоманды ввода-вывода. На рис. 7.2 вариант *а* соответствует операции ввода, а вариант *б* — операции вывода. Размер рабочей области должен выбираться таким, чтобы она вмещала логическую запись максимальной длины.

Рассмотрим два режима обработки записей без использования рабочей области. Схема использования первого режима изображена на рис. 7.3.

В примере, приведенном на рис. 7.3, рассмотрена программа, с помощью которой все логические записи файла BNOD без обработки могут быть переписаны в файл BEND. Текст программы дан на рис. 7.3, в. Макрокоманда GET используется в режиме указания, а макрокоманда PUT — в режиме пересылки (см. выделенные в тексте программы операторы и операнды макрокоманд DCB). После выполнения очередной макрокоманды GET в регистре 1 находится адрес выбранной логической записи (см. рис. 7.3, а), а в результате выполнения последующей макрокоманды PUT выбранная логическая запись пересылается программами метода доступа в выходной буфер (см. рис. 7.3, б). Непосредственно перед выполнением макрокоманды вывода ей сообщается через регистр 0 адрес сегмента, в котором находится нужная запись (см. текст программы на рис. 7.3, в).

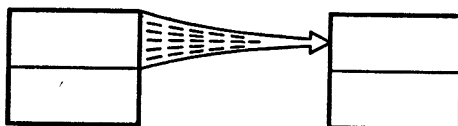
Использование еще одного режима обработки дано на рис. 7.4. Этот рисунок иллюстрирует ту же программу, что и рис. 7.3. Отличаются они режимами использования макрокоманд ввода-вывода. Если в программе, приведенной на рис. 7.3, макрокоманда ввода используется в режиме указания, а макрокоманда вывода — в режиме пересылки, то в рассматриваемом примере макрокоманда ввода используется в режиме пересылки, а вывода — в режиме указания. Использование макрокоманды PUT в режиме указания обладает одной особенностью, заключающейся в способе получения адреса сегмента, предназначенного для выводимой записи. Макрокоманда PUT после выполнения сообщает через регистр 1 адрес следующего сегмента (см. рис. 7.4, а). Поэтому эта макрокоманда в тексте программы стоит раньше макрокоманды ввода. В остальном пример аналогичен примеру рис. 7.3.

В заключение подчеркнем, что во всех рассмотренных выше примерах действия по пересылке записей в процессе обработки выполняются программами метода доступа. Однако, если макрокоманды ввода и вывода использовать в режиме указания, то все необходимые операции по пересылке логических записей должны выполняться в программе, написанной пользователем.

2. Обменная буферизация. Основной недостаток простой буферизации, заключающийся в принципиальной невозможности выполнить обработку логических записей



а) Выполнение макрокоманды GET



б) Выполнение макрокоманды PUT

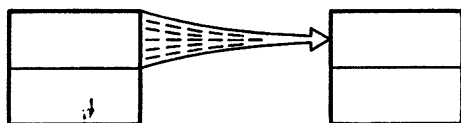
	START	
	SAVE	(14, 12)
	USING	BEGIN, 2
	BALR	2, 0
BEGIN	ST	13, SU + 4
	LR	10, 13
	LA	13, SU
	ST	13,8 (10)
INPUT	OPEN	(BNOD, ,BEND, (OUTPUT))
	GET	BNOD
	LR	0,1
	PUT	BEND, (0)
KON	B	INPUT
	CLOSE	(BNOD, ,BEND)
	L	13, SU + 4
SU	RETURN	(14, 12)
BNOD	DS	18F
	DCB	MACRF = (GL), DSORG = PS,
		RECFM = FB, LRECL = 80,
		BLKSIZE = 160, EODAD = KON,
		DDNAME = BBD
BEND	DCB	MACRF = (PM), DSORG = PS,
		RECFM = FB, LRECL = 80,
		BLKSIZE = 160,
		DDNAME = BYEND
	END	

в) Текст программы перезаписи данных из файла BNOD в файл BEND.

Рис. 7.3.



а) Выполнение макрокоманды PUT



б) Выполнение макрокоманды GET

	START	
	SAVE	(14, 12)
	USING	BEGIN, 2
	BALR	2, 0
BEGIN	ST	13, SU + 4
	LR	10, 13
	LA	13, SU
	ST	13, 8 (10)
INPUT	OPEN	(BNOD, ,BEND, (OUTPUT))
	PUT	BEND
	LR	0,1
	GET	BNOD, (0)
	B	INPUT
KON	CLOSE	(BNOD, ,BEND)
	L	13, SU + 4
SU	RETURN	(14, 12)
BNOD	DS	18F
	DCB	MACRF = (GM), DSORG = PS,
		RECFM = FB, LRECL = 80,
		BLKSIZE = 160,
		DDNAME = BBD,
		EODAD = KON
BEND	DCB	MACRF = (PL), DSORG = PS,
		RECFM = FB, LRECL = 80,
		BLKSIZE = 160,
		DDNAME = BYEND
	END	

в) Текст программы

Рис. 7.4.

без их дополнительного перемещения в оперативной памяти, является существенным. Обменная буферизация позволяет устранить этот недостаток. Однако ее использование возможно с некоторыми ограничениями:

- применение только в схеме ввод — обработка — вывод, т. е. для обработки двух файлов;
- размеры логических записей в обоих файлах должны быть фиксированными и одинаковыми;
- коэффициент блокирования записей в обоих файлах должен быть одинаковым.

Основная цель обменной буферизации — исключение дополнительных «лишних» перемещений логических записей — достигается путем специальной организации сегментов внутри буфера. Если при простой буферизации буфер представляет собой совокупность смежных сегментов, то при обменной буферизации это условие может не соблюдаться. Для того чтобы такое стало возможным, программы методов доступа формируют отдельную команду канала на каждый сегмент, в отличие от простой буферизации, где команда формируется на каждый буфер.

В результате этого становится возможной операция обмена сегментов между входным и выходным буферами. Поэтому при обменной буферизации поле буферов используется в равной степени как для ввода, так и для вывода, т. е. является общим. Организация обработки логических записей при использовании обменной буферизации осуществляется только в одном режиме, называемом режимом подстановки (Т).

Конкретную организацию обработки записей в режиме подстановки с применением обменной буферизации рассмотрим на примере программы, выполняющей те же функции, что и программа, приведенная на рис. 7.3. Обратимся к рис. 7.5. Варианты схем *a — d* (рис. 7.5) иллюстрируют изменение состояния поля буферов и рабочей области по мере последовательного выполнения команд ввода и вывода.

Для того чтобы не происходило нарушения цепочек сегментов, образующих входной и выходной буферы, выделяется рабочая область размером в один сегмент. Эта рабочая область будет использоваться в качестве подмены одного из сегментов либо входного, либо выходного буферов в моменты обработки той или иной записи. Подмена осуществляется путем замены адресов в командах канала. Команды ввода-вывода работают в режиме указания, т. е. после их выполнения в регистре 1 содержится

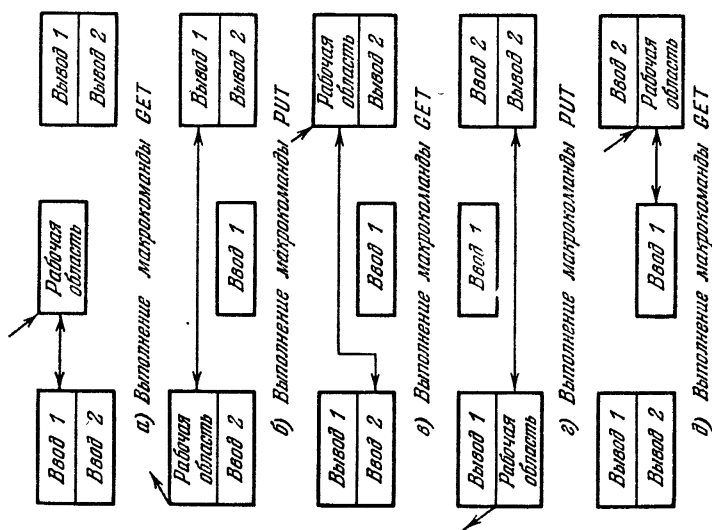


Рис. 7.5.

START
 SAVE (14, 12)
 USING BEGIN, 2
 BALR 2, 0
 ST 13, SU + 4
 LR 10, 13
 LA 13, SU
 ST 13,8 (10)
 OPEN (BNOD, BEND, (OUTPUT))
 LA 6, RAB
 LR 0, 6
 GET BNOD, (0)
 LR 0, 1
 PUT BEND, (0)
 LR 6, 1
 B INPUT
 CLOSE (BNOD, BEND)
 L 13, SU + 4
 RETURN (14, 12)
 DS 18F
 DS CL 80
 DCB MACRF = (GT), DSORG = PS,
 BUFEK = E, RECFM = FB,
 LRECL = 80, BLKSIZE = 160,
 EODAD = KON, DDNAME = BBD
 DCB MACRF = (PT), DSORG = PS,
 BUFEK = E, RECFM = FB, LRECL = 80,
 BLKSIZE = 160, DDNAME = BYEND
 END

е) Текст программы

адрес логической записи, к которой произошло обращение. Подменный адрес должен сообщаться в операнде макрокоманды ввода-вывода аналогично тому, как это делается при простой буферизации в режиме пересылки.

В качестве подменного адреса должен всегда сообщаться адрес сегмента, выполняющего функции рабочей области. В рассматриваемом примере этот адрес должен находиться в регистре 0 (см. рис. 7.5). При описании файлов обменная буферизация и режим подстановки определяются с помощью подчеркнутых на рис. 7.5 операндов макрокоманд DCB.

Операнд BUFTEK определяет тип буферизации. Если этот операнд опущен, то его значение предполагается равным S, т. е. соответствующим определению простой буферизации.

Очевидно, что рассмотренный режим обработки записей при обменной буферизации исключает физическое перемещение записи в оперативной памяти. Благодаря этому достигается существенная экономия времени обработки файлов.

Базисный последовательный метод доступа. При составлении программ, использующих данный метод доступа, на программиста возлагается гораздо больше обязанностей по сравнению с логическим последовательным методом доступа. Как уже отмечалось ранее, этот метод доступа предусматривает обмен данными — физическими блоками. Если в программе необходимо производить действия с логическими записями, то выполнять соответствующие операции по блокированию или деблокированию необходимо в составляемой программе.

Макрокоманды ввода-вывода рассматриваемого метода доступа, равно как и всех других базисных методов, только иницируют операции ввода-вывода. Проверка окончания иницизированной операции, а также проверка правильности выполнения операции ввода-вывода являются одной из главных обязанностей программиста. Для выполнения указанных функций в макросистеме базисного последовательного метода доступа имеется специальная макрокоманда СНЕСК.

Вышеописанные средства программирования операций ввода-вывода позволяют программировать совмещение обработки записей с их вводом-выводом. Добиться значительной степени совмещения счета с обменом можно, например, путем инициирования подряд нескольких операций ввода-вывода, не ожидая их завершения. Проверку завершения иницизированных таким способом операций об-

мена можно производить путем использования одной или нескольких макрокоманд СHECK.

Управление буферами должно осуществляться программой пользователя. Данный метод доступа предусматривает два способа буферизации.

Первый способ состоит в том, что программист может самостоятельно выделить буфер в своей программе и указать его адрес непосредственно в макрокомандах ввода-вывода. Будем называть этот способ буферизации *ручным*.

Второй способ состоит в том, что программист может использовать специальные макрокоманды, предназначенные для построения буферного пула и выделения из него буферов ввода-вывода. Такой способ управления буферами будем называть *непосредственным*. Например, к командам, с помощью которых можно выделять буферы и возвращать их в буферный пул, относятся соответственно макрокоманды GETBUF и FREEBUF.

Рассмотрим в качестве примера текст программы, использующей базисный последовательный метод доступа и выполняющей те же функции, что и программа, приведенная на рис. 7.3. Написанная указанным способом программа дана на рис. 7.6.

Проанализируем текст данной программы немного подробнее. Начнем с описаний. Отличие от ранее рассмотренных примеров (см. рис. 7.3—7.5) состоит только в способе кодирования значений операндов MACRF, выделенных на рисунке. Значения этого операнда *R* и *W* обозначают использование макрокоманд ввода и вывода соответственно. Кодирование других операндов макрокоманд описания файлов BNOD и BEND ничем не отличается от кодирования соответствующих операндов, приведенных на рис. 7.4.

Написание стандартной начальной части программы (см. рис. 7.6) и макрокоманды открытия файлов совпадает с ранее рассмотренным. Опишем макрокоманду ввода (см. № 10) детальнее. В макрокоманде даны 4 операнда.

Первый идентифицирует специальный блок, называемый *блоком управления событием данных*, который используется для проверки окончания и правильности завершения операции ввода.

Второй указывает тип операции, заданный в нашем случае кодом, определяющим нормальную выборку.

Третий содержит адрес блока описания данных (DCB).

Четвертый указывает начальный адрес построенного в программе буфера.

Описанная макрокоманда осуществляет инициирование операции ввода данных в область памяти, выделенной для использования в качестве буфера.

№ п/п	Имя	КОП	Операнды
1		START	
2		SAVE	(14, 12)
3		USING	BEGIN, 2
4		BALR	2, 0
5	BEGIN	ST	13, SU + 4
6		LR	10, 13
7		LA	13, SU
8		ST	13, 8 (10)
9		OPEN	(BNOD, .BEND, (OUTPUT))
10	INPUT	READ	CB1, SF, BNOD, BUF
11		CHECK	CB1
12		WRITE	CB2, SF, BEND, BUF
13		CHECK	CB2
14		B	INPUT
15	KON	CLOSE	(BNOD, .BEND)
16		L	13, SU + 4
17		RETURN	(14, 12)
18	SU	DS	18F
19	BUF	DS	CL80
20	BNOD	DCB	DSORG = PS, MACRF = R, RECFM = F, BLKSIZE = 80, EODAD = KON, DDNAME = BBD
21			
22			
23	BEND	DCB	DSORG = PS, MACRF = W, RECFM = F, BLKSIZE = 80, DDNAME = BYEND
24			
25			
26		END	

Рис. 7.6.

Следующая макрокоманда (№ 11) выполняет функции синхронизации и проверки правильности завершения инициированной ранее операции ввода. Свои функции макрокоманда СНЕСК осуществляет при помощи использования блока управления событием данных, адрес которого является единственным операндом этой макрокоманды и соответствует адресу аналогичного блока, созданного соответствующей макрокомандой ввода. Выполнение этой макрокоманды осуществляется программами супервизора в отличие от выполнения макрокоманды ввода, которое производится программами метода доступа, расположенными в области задачи.

Последующие две макрокоманды (см. №№ 12 и 13) предназначены для выполнения операции вывода данных, находящихся в буфере, использованном при вводе. Дополнительных комментариев эти макрокоманды не требуют.

В заключение анализа базисного последовательного метода доступа отметим, что главным его достоинством является большая гибкость при работе с последовательным набором данных. В подтверждение вышесказанного можно, например, отметить гибкость данного метода доступа, используемого при обработке набора данных, который располагается на магнитном носителе. Состав макрокоманд метода доступа содержит две специальные макрокоманды, позволяющие в принципе программировать не только последовательный вид обработки, но также и произвольный. К ним относятся NOTE и POINT. С помощью первой макрокоманды программист может получить относительный адрес любого блока данных, а затем, используя вторую макрокоманду, вернуться к обработке этого блока в любой момент времени, но в пределах одного физического тома.

7.2. Управление индексно-последовательными файлами

Логический индексно-последовательный метод доступа. Указанный метод доступа может быть использован для управления данными главным образом в двух режимах: в режиме загрузки и в режиме сканирования.

1. Режим загрузки индексно-последовательного файла (ИПФ). Данный режим главным образом используется при создании индексно-последовательного набора данных. Для создания ИПФ предварительно подготавливается последовательный набор данных, в котором логические записи рассортированы в порядке возрастания по содержимому поля записи, выбранному и предназначенному для использования в качестве ключа. Подготовленный таким образом файл переписывается последовательно запись за записью в область памяти на магнитном диске, отведенную под ИПФ.

Конкретный пример программы, осуществляющей создание (загрузку) индексно-последовательного набора данных, приведен на рис. 7.7.

Текст программы до оператора № 22 практически почти не отличается от текста программ, рассмотренных выше (см., например, рис. 7.4). Программа предназначена для

переписывания записей из одного файла (BNOD) в другой (BEND). Входной файл определен и описан как последовательный, а выходной — как индексно-последовательный.

Проанализируем описание индексно-последовательного набора данных. В тексте программы ему соответствует

№ п/п	Имя	КОП	Операнды
1		START	
2		SAVE	(14, 12)
3		USING	BEGIN, 2
4		BALR	2, 0
5	BEGIN	ST	13, SU + 4
6		LR	10, 13
7		LA	13, SU
8		ST	13, 8 (10)
9		OPEN	(BNOD, ,BEND, (OUTPUT))
10	INPUT	GET	BNOD, BU
11		PUT	BEND, BU
12		B	INPUT
13	KON	CLOSE	(BNOD, ,BEND)
14		L	13, SU + 4
15		RETURN	(14, 12)
16	SU	DS	18F
17	BU	DS	CL80
18	BNOD	DCB	DSORG = PS, MACRF = (GM),
19			RECFM = FB, BLKSIZE = 80,
20			LRECL = 80, EODAD = KON,
21			DDNAME = BBD
22	BEND	DCB	DSORG = IS, MACRF = (PM),
23			DDNAME = BYEND
24		END	

Рис. 7.7.

только две строки (22 и 23). Строка 22 определяет логический индексно-последовательный метод доступа, а также использование макрокоманды записи в режиме пересылки. Сразу оговоримся, что рассматриваемый метод доступа использует только простую буферизацию. Поэтому все, что было сказано о простой буферизации в п. 7.1, в равной степени относится и к данному методу доступа. Вернемся к описанию ИПФ.

Строка 23 содержит наименование DD оператора, используемого в составе операторов задания, предназначенного для запуска программы. Указанный оператор содержит основную часть описания ИПФ как в части определения

ряда параметров таблицы DCB, так и в части определения основных физических параметров.

Задание для осуществления запуска рассматриваемой программы приведено на рис. 7.8.

Как уже известно, ИПФ может занимать на носителе прямого доступа три независимые области: область индексов, которая предназначена для индекса цилиндров и

PROG	JOB	
	EXEC	PROC = ASMFCLG
ASM.SYSIN	DD	*
	.] текст исходной программы, приведенной на рис. 7.7;
	.	
	.	
GO.BBD	DD	DSNAME = KAP, UNIT = 5012, VOL = SER = L00013, DISP = OLD
	DD	DSN = KAH (INDEX), UNIT = 5052, VOL = SER = D00012, DISP = (,KEEP), DCB = (DSORG = IS, RECFM = FB, BLKSIZE = 80, LRECL = 80, RKP = 19, KEYLEN = 10, OPTCD = Y, CYLOFL = 1), SPACE = (CYL, 1, ,CONTIG)
GO.BYEND	DD	DSN = KAH (PRIME), UNIT = 5052, VOL = SER = D00012, DISP = (, KEEP), DCB = (DSORG = IS, RECFM = FB, BLKSIZE = 80, LRECL = 80, RKP = 19, KEYLEN = 10, OPTCD = Y, CYLOFL = 1), SPACE = (CYL, 10, ,CONTIG)

Рис. 7.8.

главных индексов, основная область, которая предназначена для данных, и область переполнения, предназначенная для отдельных записей, которые не умецаются на дорожке.

Последняя область — область переполнения может отсутствовать. Однако, если при добавлении записей все-таки потребуется область переполнения, то она может быть рассредоточена по всем цилиндрам области данных. Каждый цилиндр основной области организован следующим образом: нулевая дорожка отводится для размещения индекса дорожек, все следующие дорожки цилиндра либо их часть отводятся для данных, и, наконец, одна или несколько последних дорожек могут быть отведены для области переполнения, называемой *областью переполнения цилиндра*. Последним способом выделения области переполнения воспользуемся в примере (рис. 7.8). Память для каждой области (области индексов и основной области)

запрашивается с помощью отдельного оператора DD. Располагаться в задании эти операторы должны друг за другом, и только первый из них может иметь имя.

В примере использованы два оператора DD. Один предназначен для описания области индексов, второй — для описания основной области. Тип области описывается в операнде, который определяет имя набора данных, с помощью ключевого слова, помещаемого в скобках после имени набора данных. Слово INDEX обозначает область индексов, а слово PRIME — основную область.

Параметры, определяющие тип устройства, серийный номер тома, диспозицию, были уже рассмотрены ранее (см., например, рис. 6.9). Для определения ряда параметров таблицы DCB использованы значения подпараметров операнда DCB, заданные в операторах описания данных, приведенных в задании и относящихся к описанию ИПФ. Значения некоторых подпараметров, описывающих способ организации, формат записей, размеры блоков и логических записей (см. рис. 7.8), не нуждаются в пояснениях.

Подпараметры RKP и KEYLEN определяют местоположение ключа в логической записи и его размер в байтах соответственно. В примере поле ключа начинается с 20-го байта и имеет длину 10 байтов. С помощью значений последних двух подпараметров (OPTCD и CYLOFL) определяется способ выделения области переполнения. На каждом цилиндре основной области одна дорожка (последняя) выделяется в качестве области переполнения цилиндра.

Параметр SPACE в операторах DD предназначен для описания запроса на внешнюю память, требуемую для размещения соответствующих областей индексно-последовательного набора данных. Смысл приведенных описаний этого параметра состоит в следующем.

Память на магнитных дисках, в соответствии с принципом формирования ИПФ, запрашивается в единице измерения, именуемой *цилиндром*. Для области индексов затребован один цилиндр, для основной области — десять. Ключевое слово CONTIG означает, что выделяемые по запросу цилиндры обязательно должны быть смежными.

На этом анализ описания индексно-последовательного файла заканчиваем. Отметим, что описание ИПФ является несколько громоздким, поэтому можно рекомендовать следующее. Подробно описывать ИПФ следует только в момент его создания, а в дальнейшем при обработке следует пользоваться аппаратом стандартных меток.

Подводя итог обсуждению режима загрузки, отметим, что этот режим может применяться для добавления записей в конец уже существующего ИПФ.

2. Режим сканирования индексно-последовательного файла. Этот режим главным образом используется при обработке группы смежных логических записей, находящихся в составе ИПФ. Внутри такой группы записей обработка осуществляется последовательным способом. В частности, этот режим может быть использован для последовательной обработки или обновления всего набора данных.

Проиллюстрируем режим сканирования на простом примере. Составим программу распечатки содержимого всех

	START	
	SAVE	(14, 12)
	USING	BEGIN, 2
	BALR	2, 0
BEGIN	ST	13, SU + 4
	LR	10, 13
	LA	13, SU
	ST	13, 8 (10)
	OPEN	(BNOD, ,BEND, (OUTPUT))
	SETL	BNOD, KC, KEYAD
INPUT	GET	BNOD
	LR	0, 1
	PUT	BEND, (0)
	B	INPUT
KON	CLOSE	(BNOD, ,BEND)
	L	13, SU + 4
	RETURN	(14, 12)
SU	DS	18F
KEYAD	DC	C'915'
	DC	CL7'0'
BNOD	DCB	DSORG = IS, MACRF = (GL, SK), EODAD = KON, DDNAME = BBD
BEND	DCB	DSORG = PS, MACRF = (PM), RECFM = FB, BLKSIZE = 128, LRECL = 128, DDNAME = BYEND
	END	

Рис. 7.9.

логических записей индексно-последовательного файла, начиная с записи, у которой первые три цифры ключа (при общей длине 10) равны 915, и кончая последней записью файла.

Текст программы, соответствующий поставленной задаче, приведен на рис. 7.9.

Для этого примера в качестве исходного взят ИПФ, создаваемый при помощи программы, рассмотренной выше (см. рис. 7.7).

Проведем краткий анализ выделенной макрокоманды SETL и выделенного операнда макрокоманды DCB, описывающей индексно-последовательный файл.

В общем случае макрокоманда SETL используется для установки начальной точки, начиная с которой будет осуществляться последовательная обработка некоторой части ИПФ.

В примере по этой макрокоманде производится установка головок устройства для чтения первой записи ИПФ, у которой первые три цифры ключа — 915.

Если необходимо обрабатывать несколько последовательных участков, то перед тем, как с помощью вышеуказанной макрокоманды установить начальную точку очередного участка, необходимо отменить предыдущую установку с помощью макрокоманды ESETL.

Отметим также, что если последовательные участки состоят каждый из одной записи, то имеет место произвольная обработка индексно-последовательного файла.

Рассмотрим значения операнда MACRF в описании ИПФ (BNOD). Символы GL обозначают применение для чтения логических записей макрокоманды GET в режиме указания; символы SK обозначают соответственно: режим сканирования и обеспечение установки начальной точки по ключу.

Заканчивая обсуждение примера, подчеркнем, что в операторе DD должны быть указаны только физические характеристики файла, такие, как имя набора данных, тип устройства, серийный номер носителя и диспозиция, определяющая состояние файла — «старый». Недостающие характеристики описания файла будут взяты программами процедуры «открыть» из стандартных меток, сопровождающих ИПФ на носителе.

В заключение рассмотрения логического индексно-последовательного метода доступа отметим, что он в основных чертах подобен логическому последовательному методу доступа.

Базисный индексно-последовательный метод доступа. Данный метод доступа предназначен для программирования произвольных режимов обработки записей ИПФ. В произвольном режиме макрокоманды рассматриваемого метода доступа могут использоваться, в частности, для добавления новых записей в конце ИПФ, для занесения новых записей

между существующими записями, для выборочной обработки и обновления записей.

В качестве иллюстрации возможностей базисного индексно-последовательного метода доступа рассмотрим наиболее «тяжелый» режим управления, при котором в ИПФ

	START	
	SAVE	(14, 12)
	USING	BEGIN, 2
	BALR	2, 0
BEGIN	ST	13, SU + 4
	LR	10, 13
	LA	13, SU
	ST	13, 8(10)
	OPEN	(BNOD, .BEND)
INPUT	GET	BNOD, KEY
	LA	0, KEY + 19
	WRITE	BLOK, KN, BEND, KEY, 'S', (0)
	CHECK	BLOK, DSORG = IS
	B	INPUT
KON	CLOSE	(BNOD, .BEND)
	L	13, SU + 4
	RETURN	(14, 12)
SU	DS	18F
KEY	DS	CL80
BNOD	DCB	DSORG = PS, MACRF = (GM), RECFM = FB, BLKSIZE = 800, LRECL = 80, EODAD = KON, DDNAME = BBD
BEND	DCB	DSORG = IS, MACRF = (WA), DDNAME = BYEND
	END	

Рис. 7.10.

помещаются новые записи, предназначенные для вставки между существующими.

Основным индексно-последовательным файлом в примере является ИПФ, создаваемый программой, приведенной на рис. 7.7.

Новые записи, предназначенные для занесения в основной файл, находятся в последовательно-организованном наборе данных, расположенном на магнитной ленте со стандартными метками.

Программа, с помощью которой можно вставить новые записи в существующий индексно-последовательный файл, приведена на рис. 7.10.

Задание для обеспечения запуска этой программы дано на рис. 7.11.

Проанализируем текст программы. Описание входного файла (BNOD) аналогично описанию последовательных файлов, рассмотренных ранее. В описании индексно-последовательного набора данных приведены только три операнда.

PROGR	JOB	
	EXEC	ASMFCLG
ASM.SYSIN	DD	.
	.] текст исходной программы, приведенный на рис 7.10
	.	
	.	
GO.BBD	DD	DSN = KAP, UNIT = 5012, VOL = SER = L00015, DISP = OLD
GO.BYEND	DD	DSN = KAH, UNIT = 5052, VOL = SER = D00012, DISP = OLD

Рис. 7.11.

Наибольший интерес представляет выделенный операнд. Его содержимое, равное WA, указывает на то, что будет использоваться только макрокоманда записи и что она предназначена только для занесения новых записей. Другие операнды и их значения не требуют пояснений. Выделенная макрокоманда осуществляет занесение очередной записи в ИПФ. Новая запись, подготовленная для занесения в ИПФ, располагается после считывания из файла BNOD в рабочей области, имеющей начальный адрес KEY.

Макрокоманда записи в программе имеет 6 операндов. Первый определяет адрес блока управления событием данных. Второй определяет режим занесения в ИПФ новых записей с использованием поля ключа. Третий определяет адрес таблицы DCB (BEND). Четвертый — адрес логической записи (KEY). Пятый — длину логической записи, которая содержится в DCB и помещается в него после выполнения процедуры «открыть» из стандартной метки. Шестой — адрес ключа, по которому будет определяться место занесения новой записи в ИПФ.

Макрокоманда СНЕСК выполняет те же функции, что и в программе на рис. 7.6. Работа описанной выше

программы заканчивается после чтения всех записей входного набора данных.

Для более наглядного восприятия процесса занесения новых записей в ИПФ проведем схематичный анализ внутренних действий программ метода доступа, осуществляющих управление данными в указанном режиме.

Обратимся к рис. 7.12, на котором изображены четыре последовательных состояния некоторого участка (фрагмента) индексно-последовательного набора данных, возникающих в результате поочередного занесения в файл трех записей.

В качестве фрагмента выделены четыре последних дорожки (трека) некоторого цилиндра, из которых треки 6, 7, 8 принадлежат области данных, а 9-й отведен для области переполнения цилиндра. Из области, содержащей индекс дорожек и располагающейся на нулевом треке цилиндра, выделены элементы, описывающие дорожки 6, 7 и 8 данного цилиндра.

На указанных дорожках в исходном состоянии расположены 12 записей, значения поля ключа которых приведены на рис. 7.12 (вариант а) и выбраны начальными

Трек								
Индекс трека	0	120,6	120,6	185,7	185, 7	900,8	900,8	FFF
Область данных	6	100		101		105		120
	7	150		180		181		185
	8	600		697		698		900
Область переполнения цилиндра								
	9							

а) Исходное состояние

Рис. 7.12 а.

для дальнейшего рассмотрения. Каждый элемент индекса, описывающий дорожку, состоит из двух частей — основной и дополнительной. Дополнительная часть используется только в случае применения области переполнения. Конкретный механизм ее использования будет показан по ходу анализа. В исходном состоянии основная и дополнительная части элемента индекса дорожек содержат одну и ту же информацию, обозначающую значение ключа последней записи, расположенной на дорожке, и ее физический номер. Например, пара чисел 185, 7 (см. рис. 7.12, а) означает,

что на дорожке № 7 значение ключа последней (старшей) записи равно 185. Элемент, содержимое которого равно FFF, является признаком конца индекса дорожек данного цилиндра.

0	• • •	120,6	120,6	181,7	185,9/1	900,8	900,8	FFF
:								
6	100	101		105		120		
7	125	150		180		181		
8	600	697		698		900		
9	185	FFF						

б) Добавлена запись 125

Рис. 7.12 б.

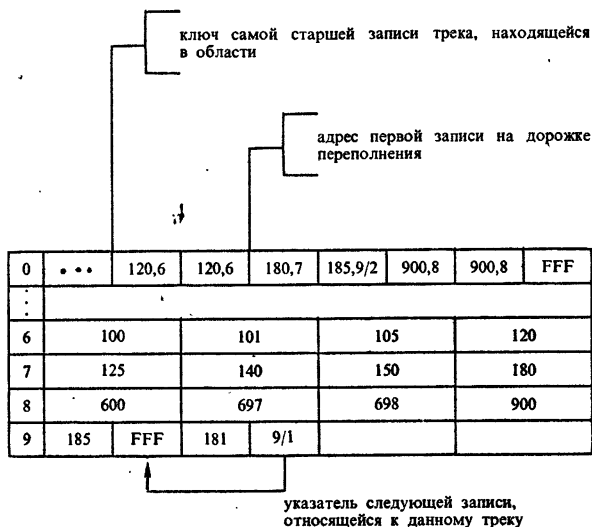
Состояние рассматриваемого фрагмента ИПФ, изображенное на рис. 7.12 (вариант б), явилось результатом действий системных программ, осуществивших добавление в набор данных новой записи, имеющей значение ключа, равное 125. Изменения состояния фрагмента коснулись содержимого дорожки № 7 и дорожки переполнения № 9, содержимого основной и дополнительной частей индекса седьмого трека.

Записи на 7-й дорожке расположились так, как это показано на рис. 7.12 (см. вариант б). Запись с номером 125 заняла полагающееся ей место, записи 150, 180, 181 уместились на дорожке, а запись с номером 185, которая на данной дорожке не уместается, помещена на дорожку переполнения. В области переполнения к записи добавляется некоторый идентификатор (FFF), назначение которого станет ясно по мере дальнейшего изложения.

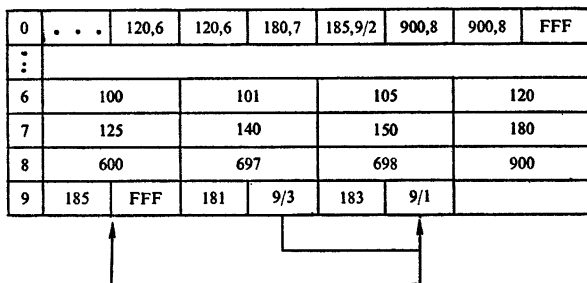
Помещенная на дорожку переполнения запись имеет адрес 9/1, что означает: первая запись на дорожке № 9. Содержимое основной части индекса заменяется на пару (181, 7), в которой указывается номер дорожки (7) и ключ последней записи на этой дорожке (181). Дополнительная часть индекса содержит пару (185, 9/1). Числа в паре соответствуют: ключу старшей записи, относящейся к 7-й дорожке и находящейся в области переполнения цилиндра (185); адресу первой записи дорожки, помещенной в область переполнения (9/1).

Теперь рассмотрим состояние фрагмента ИПФ, приведенное на рис. 7.12 (вариант в) и полученное в результате добавления записи с номером, равным 140. Занесенная

запись расположилась на дорожке 7 между записями 125 и 150, в результате чего запись с номером 181 оказалась вытесненной с дорожки 7 и записанной на свободное место в области переполнения. Адрес записи с номером 181 равен 9/2.



е) Добавлена запись 140



г) Добавлена запись 183

Рис. 7.12 в, г.

Поскольку номер записи, помещенный на дорожку переполнения, меньше 185, то в дополнительную часть элемента индекса, описывающего записи, находящиеся в области переполнения, помещен адрес 9/2, а в качестве идентификатора записи 181 — адрес 9/1. В основной части

элемента индекса 7-й дорожки вместо номера 181 помещен номер 180, означающий, что на дорожке 7 самой старшей записью является запись с указанным номером.

Наконец, рассмотрим состояние фрагмента ИПФ, приведенное на рис. 7.12 (вариант *г*), которое получилось в результате добавления в набор данных новой записи с номером 183. Согласно логике построения индексно-последовательных файлов эта запись помещена в область переполнения на свободное место по адресу 9/3. Изменение состояния набора коснулось только области переполнения. У записи с номером 181 идентификатор заменен на адрес записи с номером 183 (9/3), а у записи 183 в качестве идентификатора помещен адрес 9/1, который являлся в предыдущем состоянии ИПФ идентификатором записи 181.

Идентификаторы предназначены для образования цепочек записей в необходимой последовательности. Начало такой цепочки указывается в дополнительной части элемента индекса дорожки, например, 9/2 для дорожки 7. Данный адрес указывает запись (181), которая по принципу рассортированности должна находиться сразу после старшей записи, расположенной на основной дорожке (180). Идентификатор записи 181 указывает местоположение следующей по порядку записи с номером 183, а адрес последней — местоположение записи 185. Конец рассматриваемой цепочки обозначается идентификатором FFF.

Вышеуказанный способ размещения записей в ИПФ позволяет осуществлять эту операцию без каких-либо значительных перемещений информации в наборе данных. В самом неблагоприятном случае изменения ИПФ касаются только трех локальных частей файла: основной дорожки, дорожки переполнения и элемента индекса дорожки.

Закончим рассмотрение основного режима управления данными программами базисного индексно-последовательного метода доступа тем, что отметим его значительную схожесть с базисным последовательным методом доступа. В этом легко убедиться, рассмотрев процесс программирования обработки данных средствами упомянутых методов доступа.

7.3. Управление библиотечными файлами

Библиотечный файл предназначается для хранения некоторого количества последовательно-организованных разделов, каждый из которых имеет свое имя, состоящее не более чем из 8 символов.

Для управления такими файлами создан базисный библиотечный метод доступа, который позволяет выполнять следующие функции:

- обработку одного, нескольких или всех разделов файла;
- удаление и добавление отдельных разделов файла;
- замену одного, нескольких или всех разделов файла.

Для того чтобы указанные функции могли быть осуществлены с использованием только одного блока DCB и однократного применения к файлу процедуры «открыть», в макросистему рассматриваемого метода доступа введены специальные характерные для него макрокоманды FIND «найти» и STOW «запомнить».

Макрокоманда «найти» должна использоваться непосредственно перед обработкой отдельного раздела библиотеки. Ее функции состоят в том, чтобы, во-первых, найти по имени раздел, предназначенный для обработки, и, во-вторых, настроить головки дисковода для чтения первого блока найденного раздела.

Рассмотрим простейший пример, в котором требуется распечатать содержимое одного раздела библиотеки. Имя раздела будет для простоты указано в тексте программы. Основные характеристики файлов также приведены в программе. Обратимся к рис. 7.13.

Первые девять команд были уже рассмотрены. Десятой написана макрокоманда «найти», с помощью которой метод доступа настраивается на обработку начального блока искомого раздела. Имя раздела помещено в команде 21. Отметим, что начальный адрес имени раздела должен быть выставлен на границу двойного слова. Об этом требовании свидетельствует третий операнд макрокоманды «найти» (D), а выполнение указанного требования обеспечивается с помощью команды 20 (CNOP). По поводу остальных операторов приведем только краткие комментарии. Команды 11 и 12 осуществляют ввод и синхронизацию операции ввода соответственно. Команда 13 производит печать содержимого буфера, в который считан предыдущий блок раздела. Последующие команды пояснений не требуют.

Из данного примера видно, что базисный библиотечный метод доступа очень схож с базисным последовательным методом доступа, особенно в той части, которая касается программирования операций ввода-вывода, синхронизации и ряда других.

Макрокоманда «запомнить» используется в таких режимах, при которых в файл записываются новые разделы. Ее функции состоят в том, чтобы записать в справочник библиотеки новый элемент, описывающий конкретный

№ п/п	Имя	КОП	Операнды
1	BEGIN	START	
2		SAVE	(14, 12)
3		USING	BEGIN, 2
4		BALR	2, 0
5		ST	13, SU + 4
6		LR	10, 13
7		LA	13, SU
8		ST	13, 8(10)
9		OPEN	(BNOD, BEND, (OUTPUT))
10	START	FIND	BNOD, IMJA, D
11		READ	URA, SF, BNOD, BUF
12		CHECK	URA
13		PUT	BEND, BUF
14		B	START
15	KON	CLOSE	(BNOD, BEND)
16		L	13, SU + 4
17		RETURN	(14, 12)
18	SU	DS	18F
19	BUF	DS	CL80
20		CNOP	(0, 8)
21	IMJA	DC	C'ISKOM 100'
22	BNOD	DCB	DSORG = PO, MACRF = (R), RECFM = F, BLKSIZE = 80, EODAD = KON, DDNAME = BBD
23			
24			
25	BEND	DCB	DSORG = PS, MACRF = (PM), RECFM = FB, BLKSIZE = 80, LRECL = 80, DDNAME = BYEND
26			
27			
28		END	

Рис. 7.13.

раздел. Чаще всего эта макрокоманда используется в процессе создания библиотечного файла.

В ряде случаев, используя последовательный способ организации разделов, библиотечный набор данных может обрабатываться с помощью программ как логического, так и базисного последовательных методов доступа.

Применение указанного режима управления библиотечным файлом иллюстрируется программой и заданием на ее выполнение, тексты которых соответственно даны на рис. 7.13 и 7.14. Программа в процессе выполнения должна переписать некоторый раздел одного библиотечного файла в другой. Имя первого файла и переписываемого

раздела записано в оператор DD с меткой BBD, а имя второго файла и название нового раздела дано в операторе DD с меткой BYEND (см. последнее задание). Для решения поставленной задачи использовалась программа, которая ранее применялась совсем для других целей (см. рис. 7.3).

Описание конкретных физических характеристик файлов приведено в задании на выполнение программы. Рассмотрим

PROG	JOB	
	EXEC	PROC = ASMFCLG
ASM.SYSIN	DD] текст программы, приведенный на рис. 7.13
GO.BBD	DD	DSNAME = BIBL1 (PAZDL01),
		VOL = SER = D00012, UNIT = 5052,
		DISP = OLD
GO.BYEND	DD	DSNAME = BIBL 2 (PAZDL0N),
		VOL = SER = D00012, UNIT = 5052,
		DISP = MOD

Рис. 7.14.

их детальнее. Оператор BBD описывает библиотечный набор данных с указанием конкретного раздела, подлежащего обработке. Приведенное описание свидетельствует о том, что файл записан на магнитном диске, имеющем серийный номер D00012 и устанавливаемом на дисковом устройстве типа EC-5052. Оператор BYEND описывает другой библиотечный файл, который записан на том же диске, что и первый. Название, указанное в скобках, должно отличаться от названий, уже имеющихся в наборе данных разделов. Для того чтобы новый раздел был записан в конец набора данных, указывается состояние файла как модифицируемое (значение параметра, задающего диспозицию, равно MOD).

Способ указания имени набора данных с написанием названия раздела в скобках существенным образом влияет на работу процедур открытия и закрытия файлов. В процессе открытия первого файла программы этой процедуры обеспечивают поиск на томе не только самого набора данных, но также поиск указанного раздела, и настраивают метод доступа на чтение первого блока этого раздела. Процедура открытия второго файла настраивает метод доступа для записи данных на свободное место

в конце участка внешней памяти, выделенной под размещение набора данных.

Процедура закрытия первого файла практически ничем не отличается от процедуры закрытия последовательного набора данных. В процессе выполнения операции закрытия второго файла сначала записывается в справочник библиотеки название нового раздела с указанием основных параметров, характеризующих его местоположение и размер, а затем осуществляется обычное закрытие последовательного набора данных.

Завершая анализ основных способов обработки библиотечных наборов данных, нужно подчеркнуть одну практически очень важную особенность, связанную с необходимостью частого удаления разделов и одновременной записи новых разделов.

Обработка файла в указанном режиме может очень быстро привести к тому, что в набор данных нельзя будет записать ни одного нового раздела, даже если объем всех разделов, находящихся в данный момент в библиотеке, меньше объема внешней памяти, выделенной для размещения файла.

Такая ситуация может возникнуть при частом удалении разделов, не являющихся последними в библиотеке. Дело в том, что операция удаления с целью экономии времени осуществляется таким образом, что из справочника удаляется только элемент, описывающий соответствующий раздел, а физические записи раздела остаются на диске. Однако при этом нигде не фиксируется информация о том, что область памяти, занимаемая ранее удаленным разделом, стала свободной. Поэтому любой новый раздел, помещаемый в библиотеку, записывается всегда в конец набора данных. При этом, конечно, отведенная внешняя память может оказаться достаточно быстро исчерпанной.

Для предотвращения неэффективного использования внешней памяти при обработке библиотечных файлов в составе вспомогательных программ операционной системы имеется специальная программа, с помощью которой можно осуществить так называемое «сжатие» библиотеки.

Выполнение операций «сжатия» обуславливает физическое удаление из файла тех блоков, которые входили в состав ранее удаленных разделов, в результате чего вся свободная память как бы «перемещается» в конец набора данных, становясь тем самым доступной для записи новых разделов.

7.4. Управление файлами с прямой организацией

Базисный прямой метод доступа, с помощью которого осуществляются режимы управления файлами с прямой организацией, является одним из самых сложных. Его использование предполагает наличие у программиста солидного опыта работы с различными методами доступа, знаний в области обработки данных, полного понимания физических процессов обмена данными между внешними устройствами и оперативной памятью ЭВМ.

Учитывая вышесказанное, ограничимся в данном разделе рассмотрением основных принципов обработки, определяющих главные режимы управления файлов с прямой организацией, а также рассмотрением некоторых примеров.

Все режимы управления прямыми наборами данных обеспечивают достижение одной цели — предоставить программисту тот или иной способ упрощения программирования операций поиска нужных записей при произвольной обработке. В общем-то для этого же созданы индексно-последовательные методы доступа. Однако они реализуют только один способ, упрощающий поиск нужных записей и основанный на принципе рассортированности записей в порядке возрастания ключей и создания специальной таблицы (индексов), с помощью которой можно осуществлять достаточно быстрый поиск записей.

В отличие от индексно-последовательного прямой метод доступа предназначен для реализации нескольких способов упрощения поиска, из которых каждый в свою очередь может быть использован различным образом по усмотрению программиста для программирования быстрого поиска нужных записей в режиме произвольной обработки. Для прямого метода доступа характерны пять способов, облегчающих программирование операций поиска и основанных на пяти различных методах адресации физических записей (блоков) в области внешней памяти, предназначенной для хранения всего набора данных. Соответственно пяти методам адресации данных внутри файла с прямой организацией можно выделить пять способов внутренней организации таких наборов данных.

Поскольку конкретный способ адресации записей внутри файла выбирается и определяется пользователем — программистом, то поэтому в какой-то мере понятно, почему такие файлы называются файлами с прямой (непосредственной, т.е. определяемой пользователем) организацией. Про-

ведем краткий анализ всех пяти способов адресации записей в прямом наборе данных.

1. Физическая адресация. При таком способе адресации адрес каждой записи файла должен указываться с помощью физического адреса, содержащего для тома на магнитном диске конкретную информацию о номере цилиндра, номере дорожки на цилиндре и номере записи на дорожке. Эффективная организация обработки прямого набора данных с указанным способом адресации затруднена и, как правило, требует программной организации специальных таблиц, устанавливающих соответствие между содержимым записей и физическими адресами их местоположения. При этом набор данных оказывается жестко привязанным к определенному участку внешней памяти тома (неперемещаемым), что является значительным неудобством в процессе эксплуатации файла.

В силу указанных недостатков физическая адресация данных в файлах с прямой организацией используется очень редко. В отдельных случаях рассмотренный выше способ адресации называют *абсолютным*.

Все остальные способы относительной адресации иллюстрируются рис. 7.15.

Иллюстрация преследует цель на простых примерах показать принципы рассматриваемых способов адресации записей. Выполнена она путем показа небольшого фрагмента файла с прямой организацией. Фрагмент файла содержит три дорожки (0, 1, 2), взятые из основной области данных, и одну дорожку (X), принадлежащую области переполнения. Область переполнения, как и для ИПФ, может быть независимой, а может быть выделена для каждого цилиндра тома прямого доступа. Примеры будем рассматривать в предположении, что файл содержит записи фиксированной длины, причем на одну дорожку умещается восемь записей.

2. Прямой способ адресации (рис. 7.15, а). Данный способ является наиболее простым, распространенным и эффективным. Эти качества прямого способа адресации обусловлены тем, что с его помощью программисту представляются наиболее естественные и простые средства произвольной обработки данных. Адресом каждого блока в файле с прямой организацией, использующего прямую адресацию, является некоторое натуральное число, которое указывает относительный порядковый номер блока. В таком файле каждый блок имеет свой относительный номер, причем любому номеру однозначно соответствует конкретный

Номер трека	Номера записей							
	1	2	3	4	5	6	7	8
0	200	201						207
1		209		211	212			215
2	216		218	219	220	221	222	223
X								

основная область

область переполнения

а) Прямой способ адресации

0	3400		249033	(309003) 193			1234	441
1	201	717	2016	3126	(1417) 2345	1818	4567	6721
2			1860		340		9786	
X	309003	1417						

б) Косвенный способ адресации (с указанием номера записи)

0	3400	249033	193	1234	441	309003		
1	201	717	2016	3126	2345	1818	4567	6721
2	1860	340	9786					
X	1417							

в) Косвенный способ адресации (без указания номера записи)

0	3400	249003	193	1234	441	309003		
1	201	717	2016	3126	2345	1818	4567	6721
2	1860	340	9786	1417				
X								

г) Косвенный способ адресации (без указания номера записи с использованием принципа расширенного поиска)

Рис. 7.15.

физический адрес записи. При прямой адресации соответствие между относительным номером записи и ее относительным физическим адресом является взаимно однозначным. Очень часто целесообразно в качестве относительного номера записи выбирать конкретную смысловую часть записи, являющуюся характерной для обрабатываемого файла. В качестве упомянутых частей могут служить такие реквизиты, как табельный номер, шифр изделия, номер документа и т.д. Указанные реквизиты очень удобно использовать при адресации записей рассматриваемым способом. Непосредственно в процессе обработки приходится выполнять некоторые расчеты, связанные с вычислением относительного физического адреса записи по ее относительному номеру.

Относительный физический адрес блока записывается в виде TTR , где TT — относительный номер дорожки, на которой помещается блок, а R — порядковый номер записи на указанной дорожке. Отметим, что относительные номера дорожек указываются с помощью ряда натуральных чисел, включая нуль (0, 1, 2, 3,...), а номера записей — с помощью «чистого» ряда натуральных чисел (1, 2, 3,...). Относительные номера блоков записываются с помощью такого же ряда чисел, как и относительные номера дорожек (0, 1, 2, 3,...).

Обратимся к рис. 7.15, а. В приведенном фрагменте прямого файла записаны блоки с признаками 200, 201, 207, 209 и т.д. Блоки с выделенными признаками имеют относительные номера 0, 1, 7, 9 и т.д.

В процессе обработки, которая обычно осуществляется с использованием смысловых признаков, программой пользователя предварительно должен быть вычислен относительный номер блока, содержащий запись с выделенным признаком. Например, для считывания блока, содержащего запись с признаком 209, вычисляется относительный номер ($209 - 200 = 9$). В дальнейшем это число должно быть указано при обращении к файлу в качестве адреса считываемого блока. Программы прямого метода доступа, использующие прямой способ адресации, автоматически переводят заданное число, идентифицирующее относительный номер блока, в относительный физический адрес, используя при этом следующий алгоритм.

1. Относительный номер блока делится на число, указывающее количество записей, размещаемых на одной дорожке (в нашем случае это число равно 8), $9 : 8 = 1(1)$. В результате получается в частном 1 и в остатке 1.

2. Полученное частное используется в качестве относительного номера дорожки (1).

3. К полученному остатку прибавляется 1, и результат этого действия берется в качестве порядкового номера записи (2).

В примере блок, имеющий относительный номер 9 (признак 209), расположен на дорожке № 1, на месте записи № 2 (см. рис. 7.15, а).

Таким образом, как для записи, так и для считывания необходимых блоков при использовании прямого способа адресации указывается только их относительный номер, а все последующие вычисления и действия выполняют системные программы прямого метода доступа. Рассмотренный способ прямой адресации позволяет эффективно организовать обработку записей по признакам в самом произвольном порядке. Например, можно за один проход упорядочить, рассортировать в порядке возрастания признаков наборов данных, в то время как ни один из других известных методов сортировки не позволяет этого сделать.

Несмотря на существенное преимущество прямого способа адресации, он имеет ряд ограничений и недостатков. Основные ограничения и недостатки сводятся к следующему:

1) могут использоваться записи только одного формата (F);

2) требуется предварительная разметка участка внешней памяти, предназначенного для записи файла;

3) не допускается большой разброс относительных номеров блоков.

Последний недостаток является наиболее существенным. Проанализируем его последствия, возникающие в процессе обработки. Предположим, что обрабатываемый файл рассчитан на 500 записей. Ряд относительных адресов блоков, которые могут быть записаны в такой набор данных, представляет собой множество чисел 0, 1, 2, ..., 499. Если, например, фактически будет использована только половина из всех допустимых номеров, то объем внешней памяти, выделенный для набора данных, будет недоиспользован наполовину. При меньшей степени недоиспользования диапазона адресов коэффициент использования объема внешней памяти резко падает. На рис. 7.15, а это показано путем выделения пустых записей, которые появились из-за того, что соответствующие их адресам признаки не были использованы.

На практике значительное количество задач обработки данных использует способы обработки по некоторым смыс-

ловым признакам, шифрам и т.п. Причем в большинстве из них применяются такие наборы шифров, которые не позволяют использовать прямой способ адресации. Это объясняется тем, что применяемые шифры, признаки строятся не по принципу последовательной нумерации, а путем использования отдельных разрядных групп, отождествляемых с конкретными смысловыми характеристиками. В качестве примера можно представить систему шифрации материалов, применяемую на некотором производстве. Обычной для упомянутого класса задач является ситуация, когда шифр, используемый для обработки, представляет собой шестизначное (или более) число, а общее количество шифров, применяемых в задаче, не превышает нескольких тысяч.

Указанный класс задач обработки данных может с достаточной степенью эффективности решаться путем использования трех косвенных способов адресации, к рассмотрению которых перейдем.

3. Косвенный способ адресации (с указанием номера записи). Данный способ иллюстрируется рис. 7.15, вариантом б. Для системных программ методов доступа, обеспечивающих операции чтения или записи данных, при использовании рассматриваемого способа адресации указывается только относительный физический адрес записи. Этот адрес должен быть указан в виде, уже рассмотренном ранее (ТТР). Если прямой способ адресации не требует многосложных и многоступенчатых пересчетов смысловых признаков или шифров, то анализируемый здесь косвенный способ адресации требует предварительного программирования некоторого способа преобразования шифров в относительный физический адрес указанного вида.

В этом параграфе рассмотрен наиболее распространенный способ такого преобразования, основанный на принципе рандомизации. *Рандомизацией* называется способ отображения множества N -разрядных чисел в множество M -разрядных чисел, при котором $M < N$. Очень распространенным способом рандомизации является способ деления исходного множества чисел на некоторый заранее и специально подбираемый делитель. Обычно в качестве такого делителя выбирается простое число.

Результатом операции отображения при этом считается остаток от выполняемого деления. Делитель должен быть таким, чтобы значение остатка не превосходило некоторое число, определяющее максимальное количество чисел в исходном множестве. Вычисленный остаток может быть использован в качестве относительного номера блока

для записи его на выделенное для хранения файла поле внешней памяти, аналогично тому, как это делается при прямой адресации. Однако все необходимые для этого расчеты выполняются в программе пользователя.

Таким образом, на основании вышесказанного можно предложить следующий алгоритм вычисления адресов блоков при использовании косвенного способа адресации с использованием номера записи:

- 1) исходный заданный шифр делится на заранее подобранный делитель, в дальнейшем во внимание принимается только остаток от деления;

- 2) полученный остаток от деления нужно поделить на число, указывающее количество записей, размещаемых на одной дорожке. В дальнейшем используется частное от деления и остаток;

- 3) полученное частное берется в качестве относительного номера дорожки;

- 4) к полученному в п. 2 остатку прибавляется единица и результат используется в качестве номера записи.

На основании данных, полученных в пп. 3 и 4 приведенного алгоритма, формируется искомый адрес в виде TTR, по которому соответствующие программы прямого метода доступа обеспечат необходимый поиск блока.

Рассмотренный способ адресации во многом похож на прямой. Отличие состоит в следующем:

- 1) для прямого способа адресации вычисление относительного физического адреса по номеру блока выполняется автоматически программами метода доступа, а для косвенного способа — прикладной программой;

- 2) при использовании прямого способа адресации можно однозначно по значению относительного физического адреса блока восстановить значение смыслового признака, а при использовании косвенного способа — нельзя.

Если первое отличие налагает некоторые дополнительные обязанности на программиста, то второе имеет более серьезные последствия. Эти последствия состоят не только в том, что создают дополнительную нагрузку программисту, но и усложняют системные программы методов доступа. С целью уточнения данного обстоятельства обратимся к рис. 7.15, 6.

Не упоминая о способе рандомизации, отметим главные особенности расположения записей с конкретными шифрами в пределах фрагмента файла с прямой организацией:

- 1) записи располагаются не в порядке возрастания шифров,
- 2) имеется довольно много свободных мест для разме-

щения записей, 3) несмотря на наличие свободных мест, некоторые записи, например, с шифрами 309003 и 1417, не могут разместиться в основной области, так как их относительные физические адреса совпадают с адресами записей 193 и 2345 соответственно, 4) наличие области переполнения, предназначенной для размещения записей, которым не «нашлось» места в основной области. В нашем случае ими являются записи с шифрами 309003 и 1417.

Подводя итог обсуждению косвенного способа адресации с указанием номера записи, отметим, что он позволяет значительно расширить диапазон смысловых признаков, шифров в процессе их использования при обработке. Однако при этом повышается коэффициент использования области переполнения, что снижает эффективность поиска нужных записей. Снижению коэффициента использования области переполнения способствуют два следующих способа косвенной адресации.

4. Косвенный способ адресации (без указания номера записи).

Данный способ иллюстрируется рис. 7.15, вариантом *в*. Отличие от предыдущего способа косвенной адресации состоит в двух обстоятельствах. Во-первых, исходным относительным физическим адресом является адрес не в виде TTR, а в виде TT. Второе обстоятельство вытекает из первого и означает обязательное использование поля ключа во всех записях набора данных.

Таким образом, алгоритм вычисления относительных физических адресов блоков, рассмотренный в предыдущем случае, используется полностью, за исключением п. 4. При косвенном способе адресации без указания номера записи вместо вычисления порядкового номера блока нужно формировать ключ записи, равный значению смыслового признака, принятого в качестве исходного для необходимых вычислений по рандомизации и определению относительного адреса дорожки.

В процессе формирования прямого набора данных с применением рассматриваемого метода адресации записи, попадающие на одну дорожку, располагаются на ней без пропусков друг за другом в порядке поступления. Поэтому этот способ менее критичен к неудачному выбору метода рандомизации. Обратимся к рис. 7.15, *в*.

В приведенном фрагменте использованы те же блоки, но если в предыдущем случае блок с шифром 309003 располагался в области переполнения, то в рассматриваемом случае этот блок размещается на основной дорожке (0). В область переполнения записи будут попадать

только тогда, когда дорожка, адрес которой вычислен, окажется полностью заполненной. На рис. 7.15, в приведен блок с шифром 1417.

5. Косвенный способ адресации (без указания номера записи с использованием принципа расширенного поиска). Данный способ косвенной адресации представляет собой незначительную модификацию предыдущего. Если для четвертого способа адресации характерным является поиск записи по ключу на одной дорожке, то для только что названного поиск записи по ключу может осуществляться не только по одной дорожке, а сразу по нескольким, причем количество дорожек для такого «расширенного» поиска определяется программистом.

Исходной информацией для организации выполнения запроса на операцию ввода-вывода программам прямого метода доступа, реализующим данный способ адресации, являются, как и в четвертом способе, относительный адрес дорожки (ТТ) и ключ записи. Собственно «расширенный» поиск может быть использован только в том случае, когда дорожка полностью заполнена, либо когда нужной записи на указанной дорожке нет. Программы автоматически «переключаются» на соседние дорожки, просматривая их друг за другом до тех пор, пока не будут найдены либо свободное место, либо искомая запись.

На рис. 7.15, г, иллюстрирующем анализируемый способ адресации, отмечено, что блок с шифром 1417 вместо того, чтобы попасть в область переполнения (см. рис. 7.15, в), записан на следующую по порядку дорожку (2).

Принцип расширенного поиска увеличивает эффективность использования основной области и уменьшает коэффициент использования области переполнения. В остальном он идентичен четвертому способу адресации.

В заключение анализа прямого базисного метода доступа рассмотрим пример создания прямого набора данных путем использования режима разметки. Тексты программы и задания для ее выполнения приведены на рис. 7.16 и 7.17.

С помощью данного примера создается файл с прямой организацией, содержащий 8000 блоков длиной по 200 байтов каждый. Все блоки записываются последовательно с использованием адресации типа TTR (см. параметр MACRF, значение J). В остальном пояснения не нужны.

В завершение рассмотрения основных режимов управления наборами данных и способов их реализации отметим некоторые обобщающие характеристики СУД.

№ п/п	Имя	КОП	Операнды
1	BEGIN	START	
2		SAVE	(14, 12)
3		USING	BEGIN, 2
4		BALR	2, 0
5		ST	13, SU + 4
6		LR	10, 13
7		LA	13, SU
8		ST	13, 8(10)
9		OPEN	(DALOAD, (OUTPUT))
10		L	9, = F'8000'
11	START	WRITE	URA, SF, DALOAD, FIX
12		CHECK	URA
13		BCT	9, START
14		CLOSE	(DALOAD)
15		L	13, SU + 4
16	SU	RETURN	(14, 12)
17		DS	18F
18		DS	CL200
19		DCB	DSORG = DA, MACRF = (WJC),
20			DDNAME = DABEND
21		END	

Рис. 7.16.

PROGR	JOB	
	EXEC	PROC = ASMFCLG
ASM.SYSIN	DD	*
	.] текст программы, приведенный на рис. 7.16
	.	
	.	
GO.DABEND	DD	DSNAME = DAHAB,
		DISP = (,KEEP),
		VOL = SER = D00012,
		UNIT = 5052,
		SPACE = (200, 8000),
		DCB = (DSORG = DA,
		BLKSIZE = 200. RECFM = F)

Рис. 7.17.

Обращает на себя внимание прежде всего обилие макрокоманд и значительное количество операндов отдельных макрокоманд, включая множество значений параметров.

* * *

Значительную роль в правильном выборе и определении значений параметров большинства макрокоманд методов доступа играют средства описания наборов данных. Поэтому главное внимание на этапе программирования следует уделять правильному описанию набора данных с активным

Таблица 7.1

Основные характеристики описаний файлов

Характеристики	Способ организации					библиотечный
	последовательный	индексно-последовательный	прямой			
Тип носителя	магнитная лента, диск	диск	диск			диск
Специальные средства адресации блоков	нет	таблицы индексов	абсолютная	относительная		указатели TTR
			физический адрес	прямая	косвенная	
				адрес записи	адрес дорожки	
Наличие поля ключа	нет	обязательно	возможно		обязательно	нет
Наличие области переполнения	нет	возможно	нет	возможно		нет
Блокирование записей	возможно	возможно	нет	нет	нет	возможно
Макрокоманды обработки	GET (PUT) READ (WRITE)	GET (PUT) READ (WRITE)	READ (WRITE)			GET (PUT) READ (WRITE)
Наличие многозначности	возможно	возможно	возможно	возможно	возможно	нет
Форматы записей	F (FB) V (VB) U	F (FB) V (VB)	F, V, U	F	F, V, U	F (FB) V (VB) U
Специальные действия	—	реорганизация	—	предварительная разметка	—	сжатие

использованием всех имеющихся для этого средств. При этом важно сразу в комплексе представлять себе все описание в целом. Некоторую помощь программисту, составляющему комплексное описание файла, могут оказать данные, характеризующие основные элементы описаний, приведенные в табл. 7.1.

Среди приемов, увеличивающих эффективность выполнения операций ввода-вывода, чуть ли не главным является правильный и обоснованный выбор способа буферизации данных. Объясняется это тем, что способы буферизации определяют такие важнейшие показатели обработки файлов, как число обращений к внешнему устройству, уровень и возможность максимального совмещения операций ввода-вывода с операциями счета и ряд других. Доступные операционной системе ОС ЕС способы буферизации и распределение их по методам доступа даны в табл. 7.2.

Таблица 7.2

Основные способы буферизации методов доступа

Методы доступа	Тип буферизации					
	простая		обменная	динамическая	непосредственная	ручная
	режим пересылки	режим указания				
QSAM	+	+	+	—	—	—
BSAM	—	—	—	—	+	+
QISAM	+	+	—	—	—	—
BISAM	—	—	—	+	+	+
BDAM	—	—	—	+	+	+
BPAM	—	—	—	—	+	+

В той или иной степени, за исключением динамической, были рассмотрены все виды буферизации данных.

Однако в рамках одной книги, конечно, не представляется возможным дать всеобобщающий анализ систем управления данными ЕС ЭВМ. Авторы будут считать свою задачу выполненной, если материал этой книги окажет существенную помощь читателям в освоении системы управления данными операционных систем ЕС ЭВМ на основе технической и эксплуатационной документации.

ЛИТЕРАТУРА

1. Единая система ЭВМ./Под ред. А. М. Ларионова. — М.: Статистика, 1974.
2. Система математического обеспечения ЕС ЭВМ./Под ред. А. М. Ларионова. — М.: Статистика, 1974.
3. Вычислительная система IBM/360./Пер. с англ. под ред. Штаркмана В. С. — М.: Сов. радио, 1969.
4. Джермейн К. Программирование на IBM/360./Пер. с англ. под ред. Штаркмана В. С. — М.: Мир, 1971.
5. Стэбли Д. Логическое программирование в системе IBM/360./Пер. с англ. под ред. Райкова Л. Д. и Шура-Бура М. Р. — М.: Мир, 1974.
6. Электронная вычислительная машина ЕС-1020./Под ред. А. М. Ларионова. — М.: Статистика, 1975.
7. Электронная вычислительная машина ЕС-1030./Под ред. А. М. Ларионова. — М.: Статистика, 1977.
8. Электронная вычислительная машина ЕС-1050./Под ред. А. М. Ларионова. — М.: Статистика, 1976.
9. Вычислительная техника социалистических стран./Сб. статей, вып. 1./Под ред. М. Е. Раковского. — М.: Статистика, 1977.
10. Лебедев В. Н., Соколов А. П. Введение в систему программирования ОС ЕС. — М.: Статистика, 1978.
11. Программирование на языке ассемблера ЕС ЭВМ. — М.: Статистика, 1975.

85 коп.

501 60
Б948

