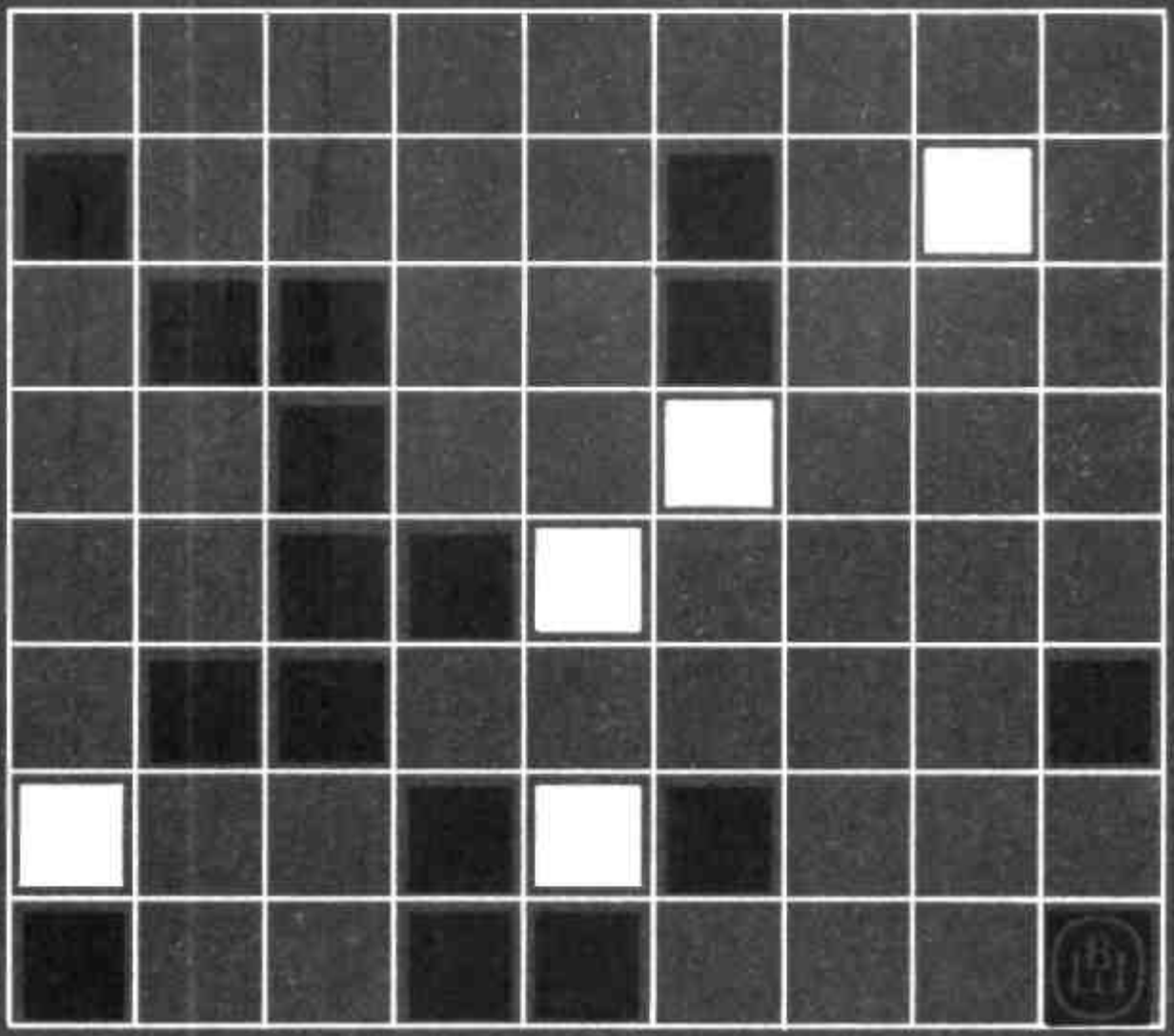
В. Ю. Дьяконов В. А. Китов И. А. Калинчев

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ



В.Ю. Дьяконов,
В.А. Китов, И.А. Калинчев

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Под редакцией д-ра техн. наук, проф. А.Л. Горелика

Допущено Государственным комитетом СССР
по народному образованию
в качестве учебного пособия для студентов
высших технических учебных заведений

Москва «Высшая школа» 1990

ББК 32.973-01
Д93
УДК 681.3.06

Рецензенты:

кафедра математического обеспечения вычислительных систем Московского института радиотехники, электроники и автоматики (зав. кафедрой - проф. Н.А. Криницкий), проф. Г.А. Миронов (Научно-исследовательский информационный центр систем управления)

Дьяконов В.Ю., Китов В.А., Калинчев И.А.
Д 93 Системное программирование: Учеб. пособие для втузов/Под ред. А.Л. Горелика. - М.: Высш. шк., 1990. -221 с: ил.

ISBN 5-06-000732-4

В книге рассмотрены архитектура, состав технических средств и основные функции системного программного обеспечения ЭВМ третьего поколения. Описаны управляющая программа, системы программирования, средства ввода-вывода, процедуры и комплексы системных вспомогательных программ ЕС ЭВМ. Приведено описание основных компонентов микропроцессорных ЭВМ.

Д 2404090000(4309000000) - 319 / 001(01)-90 193-90

ББК 32-973-01
6Ф7.3

ISBN 5-06-000732-4

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Функционирование современных универсальных ЭВМ невозможно без использования развитых системных программных средств, основу которых составляют операционные системы (ОС). Цель настоящего учебного пособия - подготовка студентов к пониманию основ системного программирования.

Теоретический материал иллюстрируют примеры, ориентированные на использование электронных вычислительных машин Единой Системы (ЕС ЭВМ). Во введении и первой главе описываются основные понятия и базовые компоненты ЭВМ третьего поколения. Во второй, третьей и четвертой главах излагается материал, посвященный составу и функциям системного программного обеспечения, структуре управляющей программы и составу языков программирования ОС ЕС ЭВМ. В пятой главе основное внимание уделяется описанию языка процедур и технологии их использования при организации вычислительного процесса. Отдельно рассматриваются процедуры трансляции, редактирования и исполнения, а также обслуживания наборов и томов данных. В шестой и седьмой главах описывается разработка программ сложной структуры (межмодульный обмен информацией, динамическая и оверлейная структуры программ, организация подзадач и т. д.) и программное обеспечение ввода-вывода информации. Помимо традиционного материала, относящегося к обработке прерываний, супервизору ввода-вывода, методам доступа и т. д., в седьмой главе описывается программное обеспечение систем телеобработки данных. Это обусловлено стремительным развитием и использованием технологии безбумажного информирования, базирующейся на системах терминального доступа и сетях ЭВМ. В частности, рассматриваются методы теледоступа и широко используемые отечественные мониторы телеобработки данных КАМА и ОБЬ, программные средства автоматизированного рабочего места программиста. В восьмой главе излагаются структура и программное обеспечение персональных ЭВМ. Внедрение практически во все области человеческой деятельности персональных электронных вычислительных машин (ПЭВМ) требует написания отдельного учебного пособия по этой актуальной тематике.

Введение и гл. 1, 2, 4 написаны В.А. Китовым, гл. 3, 5, 6 – В.Ю. Дьяконовым, гл. 8 – И.А.Калинчевым, гл. 7 – совместно.

Авторы выражают признательность коллективу кафедры системного программирования Московского института радиотехники, электроники и автоматики (зав. кафедрой – проф. Н.А.Криницкий) и проф. Г.А. Миронову (Научно-исследовательский информационный центр систем управления) за внимательный просмотр рукописи и ценные советы, способствовавшие улучшению учебного пособия.

Все замечания и пожелания по совершенствованию книги просим направлять по адресу: *101430, Москва, ГСП-4, Неглинная ул., 29/14, издательство "Высшая школа".*

Авторы

ВВЕДЕНИЕ

Одна из отличительных черт современной научно-технической революции – широкое использование методов и средств обработки информации на ЭВМ практически во всех сферах человеческой деятельности: научных исследованиях; экономике; медицине; военном деле; проектировании электронных схем и сложных строительных конструкций; управлении работой промышленных предприятий, научных организаций и государственных учреждений; редакционно-издательском деле; обучении и т. д.

На настоящем этапе развития общества важнейшим производственным ресурсом становится информация. Все большее количество информации обрабатывается на ЭВМ, на базе которых создаются сложные автоматизированные системы сбора и переработки данных. К началу следующего столетия во всех технически развитых странах основные объемы информации, и прежде всего научно-технической, социально-политической и экономической, будут храниться в памяти ЭВМ [1]. Предпосылкой этого является широкое распространение диалоговых многотерминальных систем интерактивного взаимодействия и информационно-логических сетей ЭВМ, а также постоянное совершенствование структуры и возможностей электронно-вычислительной техники и аппаратуры передачи данных. Теоретической основой создания систем автоматизированного сбора и переработки информации на основе средств электронно-вычислительной техники является кибернетика. Автоматизированные системы сбора и переработки информации предполагают участие человека в процессе управления. Задачи человека сводятся к формированию и корректировке целей и критериев управления, внесению творческого элемента в процесс определения оптимальных путей достижения поставленных целей, отбору решений из числа предложенных автоматизированной системой и приданию отобранным решениям юридического статуса, выполнению процедур, которые невозможно или нерационально автоматизировать (например, снабжение системы некоторыми типами исходных данных). Благодаря автоматизации процессов управления и информационного обеспечения увеличивается количество и объем решаемых задач, качественно изменяется характер использования систем организационного управления.

Применение современных средств вычислительной техники обеспечивает более эффективное использование автоматизированных управленческих систем за счет значительного сокращения временного интервала между моментами ввода исходных данных и получением результата; повышения актуальности хранящейся в базе данных информации; оперативного контроля вводимых данных и устранения ошибок в режиме диалога; обеспечения информационного обмена по каналам связи между автоматизированными системами различных уровней и назначений; своевременного получения пользователями информации и на основе этого – принятия ими более обоснованных решений. Развитие терминальной техники – и аппаратуры передачи данных создают предпосылки для перехода от пакетной технологии обработки информации, громоздкость которой снижает во многих случаях эффективность использования автоматизированных систем переработки информации, к принципиально новой технологии, основанной на принципах безбумажной информатики.

Внедрение безбумажной технологии обработки информации предполагает частичное, а в ряде случаев полное исключение таких трудоемких операций с бумажными носителями, как заполнение специальных бланков, ввод, обновление и пополнение баз данных при помощи перфокарт и перфолент, выдача пользователям результирующей информации в виде распечаток на алфавитно-цифровое печатающее устройство (АЦПУ), обмен распечатками между предприятиями или учреждениями, а также между их отдельными подразделениями и т. д. Безбумажная технология обработки данных подразумевает осуществление информационной обеспеченности всех уровней управления, индивидуальную информационную ориентированность, использование прямого диалога при работе с ЭВМ, разграничение доступа к управленческой информации и т.д.

Информационная обеспеченность работников всех уровней управления позволяет устранить дефицит информации, существовавший при традиционном пакетном способе обработки данных, когда полное информационное обслуживание было невозможно, в первую очередь в силу экономической нецелесообразности. Раньше считалось, что для оптимального управления производством необходимо и достаточно наличие всей информации о производстве, т. е. для

разрешения возникшего вопроса руководителю достаточно найти нужную справку или отчет. На практике оказывается, что информация, представляемая руководителю, должна быть сжата, отфильтрована и по возможности соответствовать стилю работы того или иного руководителя, т. е. должен быть реализован принцип индивидуальной ориентированности информации.

Требование обеспечения прямого диалога при работе с ЭВМ необходимо потому, что для эффективной управленческой деятельности пользователи на всех уровнях автоматизированной системы управления (АСУ) должны получать информацию непосредственно от ЭВМ, минуя промежуточные звенья (программист, оператор, секретарь, работники вспомогательных служб и т. д.). Следствием этого является повышение оперативности при принятии управленческих решений. Реализовать указанный режим можно только в рамках безбумажной технологии при наличии программных средств, обеспечивающих язык диалога, доступный пользователю-непрограммисту. Глобальная доступность информационной базы реализуется в том случае, когда программные средства систем безбумажной информатики обеспечивают пользователям вышестоящих уровней доступ к информации пользователей нижних уровней. Это позволяет проверять достоверность передаваемой информации и получать данные с любой степенью детализации, что повышает объективность принимаемых решений.

Пользователи не должны иметь возможности несанкционированно получать информацию, предназначенную для высших уровней и других подразделений данного уровня. Несоблюдение принципа разграничения доступа к управленческой информации может привести к ее бесконтрольному тиражированию или умышленному искажению.

Электронные вычислительные машины первого поколения появились более сорока лет тому назад. Первая электронная вычислительная машина, получившая название ЭНИАК, была разработана в 1945 г. в Пенсильванском университете в США. В Европе первая ЭВМ была создана в 1947 г. в Англии. В 1951 г. в СССР коллективом ученых, возглавляемым акад. *С.А. Лебедевым,* была разработана первая советская ЭВМ – Малая электронная счетная машина (МЭСМ). Начиная с 1953 г. в нашей стране начинается серийное производство Большой электронной счетной машины (БЭСМ-1) и созданной под руководством *Ю.Я. Базилевского* машины "Стрела". К отечественным машинам первого поколения также относятся М-1, М-20, "Урал-1", "Урал-2", "Урал-4"; они были выполнены на электронных лампах. Достоинство первых ЭВМ – сравнительно неплохое быстродействие (10...20 тыс. оп/с). Они наглядно продемонстрировали неоспоримые преимущества использования вычислительной техники. Первые ЭВМ оказали существенную помощь человеку в проведении научных исследований и прежде всего в атомной физике, механике, освоении космоса. Основные недостатки машин первого поколения обусловливались использованием электронных ламп и заключались в большом потреблении энергии, чрезмерной громоздкости, незначительном объеме оперативной памяти, низкой надежности и т. д. Существенный недостаток ЭВМ первого поколения – слабо развитое программное обеспечение. Разработчики прикладных программ были вынуждены программировать в машинных кодах.

Электронные вычислительные машины второго поколения появились в конце 50-х годов. Их элементной базой являлись транзисторы. В СССР наибольшее распространение получили ЭВМ типа "Урал-14", "Урал-16", БЭСМ-3, БЭСМ-4, БЭСМ-6, "Минск-2", "Минск-22", "Минск-32", М-220, М-222, НАИРИ, РАЗДАН, МИР. Быстродействие таких машин – от нескольких десятков тысяч до нескольких сотен тысяч операций в секунду. Среди ЭВМ второго поколения выделяется БЭСМ-6, обладающая быстродействием 1 млн. оп/с и объемом оперативной памяти в 122 тыс. машинных слов. Программное обеспечение ЭВМ второго поколения включает в свой состав ряд трансляторов с алгоритмических языков высокого уровня (АЛГОЛ, ФОРТРАН, АЛГЭМ и др.), пакеты стандартных программ, сервисные программные средства и управляющие мониторные системы. Более развитое по сравнению с ЭВМ первого поколения программное обеспечение позволило существенно расширить сферу применения ЭВМ путем их использования не только для решения научных задач, но и при проведении инженерных расчетов, планировании и управлении производственными процессами.

Электронные вычислительные машины третьего поколения, выполненные на интегральных схемах, имеют развитое программное обеспечение, широкий спектр периферийных устройств и представляют собой семейства программно-совместимых вычислительных машин. Первым образцом ЭВМ третьего поколения были машины IBM/360,

серийный выпуск которых начался в 1965 г. Их разработка коренным образом повлияла на процессы создания и промышленного выпуска ЭВМ во всем мире. Они явились прообразом вычислительных машин, выпущенных позднее в большинстве промышленно развитых стран, таких, как ICL (Англия), HITACHI (Япония), OLIVETTI (Италия), SIEMENS (ФРГ), ЕС ЭВМ (страны СЭВ) и др. Вычислительные машины третьего поколения по сравнению с ЭВМ второго поколения обладают значительно большими возможностями, предоставляемыми пользователям. Произошло перераспределение функций между программной и аппаратной частями ЭВМ, т. е. многие функции, которые в ЭВМ второго поколения могли быть реализованы лишь программным путем, в ЭВМ третьего поколения возложены на аппаратуру. Коренное отличие ЭВМ третьего поколения – реализация режима разделения времени центрального вычислителя ЭВМ между несколькими программами, т. е. *мультипрограммный режим* работы. Следствием его применения стали более эффективная эксплуатация дорогостоящего электронного оборудования и значительное уменьшение общего времени, затрачиваемого ЭВМ на решение пакета программ. Одна из главных характеристик ЭВМ третьего поколения – появление семейств программно-совместимых вычислительных машин, что обеспечивает разумную расстановку ЭВМ на предприятиях и в отраслях, экономное расходование мощностей вычислительных средств, возможность накопления и обмена пакетов программ, а также исключения дублирования в их разработках. Программная совместимость семейства ЭВМ обеспечивается снизу вверх, т. е. программы, разработанные для менее производительных машин, можно выполнить на всех более производительных машинах. Появление ЭВМ третьего поколения сделало реальной возможность создания крупных вычислительных центров коллективного пользования (ВЦКП), способных предоставлять свои вычислительные услуги большому числу абонентов, а также распределенных информационно-вычислительных сетей ЭВМ, имеющих отраслевой, национальный и международный характер использования.

Типичные представители вычислительных машин третьего поколения – ЕС ЭВМ; их промышленный выпуск начался в 1972 г. Первые модели ЕС ЭВМ представляют собой семейство вычислительных машин, известных под общим названием *Ряд 1;* оно включает ЭВМ ЕС-1010, ЕС-1020, ЕС-1030, ЕС-1040, ЕС-1050, ЕС-1060. Модели ЕС-1012, ЕС-1022, ЕС-1032, ЕС-1033, ЕС-1052 являются модифицированными образцами первых моделей Ряда 1 [10]. Основные характеристики наиболее распространенных программно-совместимых ЭВМ Ряда 1 занимают первые восемь строк в табл. В.1.

Таблица В. 1

Тип модели ЕС ЭВМ	Страна разра-ботчик	Скорость вычислений тыс. оп/с	Объем оперативной памяти, К байт	Число и скорость мульти-плексных каналов, К байт/с	Число и скорость селекторных каналов, М байт/с
ЕС-1020	СССР, НРБ	10...20	64...256	1x16	2x0,3
ЕС-1022	СССР	80...90	256...512	1x80	2x0,5
ЕС-1030	СССР, ПНР	60	128...512	1x40	3x0,8
ЕС-1032	ПНР	200	128...1024	1x40	3x0,4
ЕС-1033	СССР	200	256...512	1x70	3x0,8
ЕС-1040	ГДР	400	256...1024	1x50	6x1,25
ЕС-1050	СССР	500	512...1024	1x110	6x1,3
ЕС-1060	СССР	1000	2048...8192	2x110	6x3,0
ЕС-1015	ВНР	12	160	1x20	–
ЕС-1025	ЧССР	60	256	1x24	1x0,8
ЕС-1035	СССР, НРБ	140	512	1x30	4x0,8
ЕС-1045	СССР, ПНР	700	1024...4096	1x40	5x1,3
ЕС-1055	ГДР	450	512...3072	2x40	1x3,0 либо 3x1,5
ЕС-1065	СССР	4000	4096...16384	2x110	14x3,0

Наряду с производством и освоением созданных моделей ЕС ЭВМ постоянно осуществлялось их совершенствование в плане увеличения быстродействия и объемов оперативной памяти, расширения спектра терминальных устройств, а также другой периферийной техники, создания новых системных и прикладных программных средств, разработки более емких внешних запоминающих устройств, повышения надежности функционирования технических и программных средств, создания многомашинных комплексов и систем сетевого взаимодействия ЭВМ и т. д. В 1977 г. начат промышленный выпуск вычислительных машин ЕС-1015, ЕС-1025, ЕС-1035, ЕС-1045, ЕС-1055, объединенных общим названием *Ряд 2.* Характеристики ЭВМ Ряда 2 занимают нижние шесть строк в табл. В.1.

Электронные вычислительные машины Ряда 2 сохраняют программную совместимость с вычислительными машинами Ряда 1, но наряду с этим обладают дополнительными возможностями. В первую очередь это наличие аппарата виртуальной памяти, большие вычислительные мощности, расширенный набор команд, возможность подключения большего числа периферийных устройств.

Простота, удобство и эффективность эксплуатации современных ЭВМ достигаются при помощи специального класса программ, называемых *системными.* Такие программы (например, операционные системы, компиляторы, загрузчики, редакторы, макропроцессоры) были разработаны для лучшего приспособления ЭВМ к нуждам пользователей [7]. Системные программы являются объектом рассмотрения настоящего учебного пособия.

Глава 1. АРХИТЕКТУРА ЭВМ ТРЕТЬЕГО ПОКОЛЕНИЯ. СОСТАВ ТЕХНИЧЕСКИХ СРЕДСТВ ЕС ЭВМ

В главе рассматриваются базовые компоненты архитектуры ЭВМ третьего поколения. Описывается назначение центрального процессора, каналов, оперативной памяти и основных периферийных устройств. Приводятся сведения о формате управляющего поля слова состояния программы, играющего важную роль при выполнении в ЭВМ машинных команд, а также о назначении полей слова состояния канала и адресного слова канала, идентифицирующих соответственно состояние канала и адрес первой команды канальной программы. В главе особое внимание уделено механизму обработки прерываний, служащему для передачи управления между выполняемыми на ЭВМ программами. Приводятся структуры машинных и канальных команд, а также описание принципа базовой адресации. В качестве основных периферийных технических средств ЭВМ третьего поколения рассматриваются накопители на магнитных носителях, алфавитно-цифровые печатающие устройства, устройства телеобработки данных и т. д., разработанные в СССР и странах СЭВ и входящие в состав номенклатуры устройств ЭВМ Единой Системы.

§ 1.1. ОСНОВНЫЕ КОМПОНЕНТЫ ЭВМ ТРЕТЬЕГО ПОКОЛЕНИЯ

Любая модель ЭВМ третьего поколения состоит из следующих основных компонентов: центрального процессора, оперативной памяти, каналов, набора внешних устройств ввода-вывода информации.

Центральный процессор (ЦП). Это устройство, обеспечивающее выполнение команд программы и управление выполнением этих команд. Выполняются арифметические и логические операции, причем большинство из них осуществляется с использованием регистров. Имеется 16 общих регистров и 4 регистра для выполнения операций с плавающей точкой. Общие регистры, применяемые в основном для хранения целых значений, имеют длину в 32 двоичных разряда. Регистры с плавающей точкой, используемые для хранения вещественных значений, имеют длину в 64 двоичных разряда. Один двоичный разряд информации называется *битом.* Бит – наименьшая единица информации, с которой оперирует ЭВМ; может принимать одно из двух следующих значений: 1 или 0.

Оперативная память (ОП). Такая память представляет собой последовательность битов. Восемь бит информации составляют один *байт.* В оперативной памяти все байты последовательно пронумерованы, причем номер байта ОП называется его *адресом.* В качестве единиц объемов оперативной памяти используются килобайт (К) и мегабайт (М). Один К равен 1024 байт, а один М – 1024 К. Используются также такие единицы памяти ЭВМ, как поле, полуслово, слово и двойное слово. *Полем* называется последовательность из n байт, где $1 \le n \le 256$. *Полусловом* называется двухбайтовое поле с адресом, кратным двум. *Словом* называется поле из 4 байт с адресом, кратным четырем. *Двойным словом* называется поле из 8 байт с адресом, кратным восьми.

Канал. Это устройство, обеспечивающее выполнение операций ввода-вывода данных параллельно с работой центрального процессора. Наличие в составе ЭВМ каналов позволяет согласовывать высокую скорость вычислений ЦП со сравнительно низкими скоростями работы внешних периферийных устройств. Помимо пересылки символов канал осуществляет функцию их учета [8], тем самым освобождая от этой операции центральный процессор. Канал также контролирует приращение адресов размещения в оперативной памяти пересылаемых данных, что позволяет избежать затирания соседних областей. Канал представляет собой малую специализированную вычислительную машину, выполняющую так называемые канальные программы. Команды канальной программы хранятся в оперативной памяти и реализуют операции пересылки данных между ОП и внешними устройствами минуя ЦП. Функция центрального процессора заключается в том, чтобы указать каналу адрес первой команды требуемой канальной программы при необходимости выполнить какую-либо операцию ввода-вывода. После этого осуществляется загрузка в канал указанной канальной программы, которую канал выполняет автономно. Затем путем посылки специальных сигналов канал извещает центральный процессор об окончании канальной программы. Благодаря наличию канала устраняется присущий ЭВМ предыдущих поколений недостаток, когда быстрый дорогостоящий ЦП простаивал в ожидании окончания очередной операции обмена информацией с каким-либо сравнительно медленно действующим периферийным устройством. Помимо совмещения процессов счета и ввода–вывода повышение производительности в ЭВМ третьего поколения достигается за счет параллельного выполнения нескольких операций по взаимодействию с

внешними устройствами. При обслуживании быстродействующих внешних устройств, таких, как магнитные ленты и диски, производительность повышается благодаря использованию сразу нескольких каналов, называемых *селекторными.* Селекторные каналы в каждый момент времени могут работать только с одним внешним устройством. Они работают в импульсном режиме, предусматривающем передачу в одном импульсе непрерывного потока байтов от конкретного устройства [8]. Селекторный канал используется устройством монопольно. При обслуживании медленных внешних устройств, таких, как устройство чтения перфокарт или алфавитно-цифровое печатающее устройство, параллельная работа с несколькими внешними устройствами достигается путем образования нескольких подканалов в одном канале, называемом *мультиплексным.* Данный канал работает в мультиплексном режиме, при котором его аппаратные средства поочередно обслуживают все подканалы. Так как подключенные к мультиплексному каналу внешние устройства обладают невысокими скоростями обмена данными, появляется возможность организации режима мультиплексирования, при котором происходит последовательный опрос всех подканалов, инициирующий побайтный обмен информацией с внешними устройствами ввода-вывода. Мультиплексный канал будет после передачи 1 байта алфавитно-цифровому печатающему устройству принимать очередной байт от устройства чтения перфокарт и т. д. Скорость опроса подканалов значительно выше скорости работы каждого из подключенных внешних устройств, поэтому работа каждого из них не ущемляется параллельным функционированием других. Таким образом, каждое из внешних устройств обслуживается мультиплексным каналом так, как если бы оно было подключено к нему одно. При подключении к мультиплексному каналу высокоскоростного внешнего устройства скорость его работы и скорость опроса подканалов уравняются и обслуживание устройства будет осуществляться монопольно (как селекторным каналом). Канал идентифицирует внешние устройства по их однобайтовым адресам.

Набор внешних устройств ввода-вывода. Такой набор устройств ввода-вывода информации ЭВМ третьего поколения очень разнообразен. Это устройства ввода с перфокарт и перфолент, алфавитно-цифровые печатающие устройства, графопостроители, телетайпы, дисплеи, абонентские пункты, устройства вывода на перфокарты и перфоленты, накопители на магнитных лентах, дисках, дискетах, кассетных магнитных лентах и т. д. По скоростям обмена информацией внешние устройства ввода-вывода разделяются на три группы: высокоскоростные (магнитные диски, ленты, барабаны, карты и т. д.); среднескоростные (алфавитно-цифровые печатающие устройства, устройства чтения с перфокарт, оптические печатающие устройства и т. д.); низкоскоростные (телетайпы, дисплеи и т. д.). Работой внешнего устройства управляет специальное устройство управления или контроллер. В одних внешних устройствах контроллеры являются неотъемлемой аппаратной частью, а в других, например при управлении накопителями на магнитных лентах и дисках, – самостоятельными электронными устройствами. В последнем случае один контроллер может управлять работой нескольких накопителей.

§ 1.2. СЛОВО СОСТОЯНИЯ ПРОГРАММЫ

Слово состояния программы (PSW) представляет собой 64-битовое поле, размещенное в оперативной памяти с нулевого байта по седьмой и служащее для управления выполнением последовательности машинных команд программы. Наличие PSW позволяет запомнить состояние центрального процессора в момент возникновения прерывания. Выполнение машинной команды ЦП состоит из этапов выборки команды из оперативной памяти, ее декодирования и собственно исполнения. Существует ряд команд, служащих для работы с PSW, которые могут использоваться только в программах операционной системы. Эти команды в отличие от обычных, используемых в программах пользователей, называются *привилегированными.* К числу привилегированных относятся следующие команды: ОБРАЩЕНИЕ К СУПЕРВИЗОРУ (SVC); УСТАНОВИТЬ КЛЮЧ ПАМЯТИ (SSK); ПРОЧИТАТЬ КЛЮЧ ПАМЯТИ (ISK); УСТАНОВИТЬ МАСКУ ПРОГРАММЫ (SPM); ЗАГРУЗИТЬ PSW (LPSW) и т. д. Формат слова состояния программы приведен на рис. 1.1, его поля рассматриваются ниже.

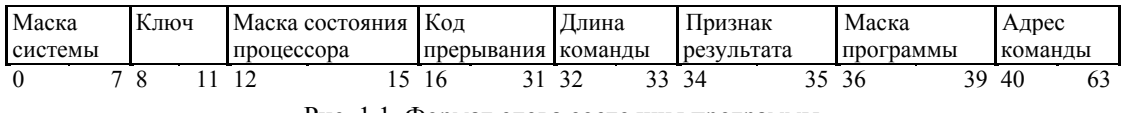


Рис. 1.1. Формат слова состояния программы

Поле маски системы. Это поле служит для указания того, разрешены или запрещены прерывания ввода-вывода и внешние прерывания. Если прерывание запрещено, то в соответствующий двоичный разряд поля маски системы помещается нуль, если разрешено - единица. Нулевой бит идентифицирует прерывания от мультиплексного канала, т. е. если нулевой бит содержит единицу, то центральный процессор может в данный момент принять прерывание от мультиплексного канала; если в нулевой бит помещен нуль, то сигнал прерывания должен быть задержан. Двоичные разряды с первого по шестой соответствуют шести селекторным каналам, для которых, аналогично мультиплексному каналу, наличие единицы в соответствующем двоичном разряде разрешает обработку прерывания, а наличие нуля предписывает задержание прерывания в селекторном канале. Если все рассмотренные двоичные разряды содержат нули, то обработка задержанных в каналах прерываний будет осуществлена только после того, как ОС поместит старое PSW на место текущего. Седьмой двоичный разряд слова состояния программы играет роль семафора для внешних прерываний.

Поле ключа. Это поле, занимающее в PSW двоичные разряды с восьмого по одиннадцатый, служит для защиты области памяти одной программы от искажений со стороны других программ. Для этого используется разработанный фирмой «IBM» метод, предписывающий идентификацию каждого из занимаемых программой двухкилобайтовых блоков, на которые разбита ОП, при помощи специального восьмибитового поля, в котором первые четыре бита играют роль ключа защиты памяти от записи, последние три всегда равны нулю, а пятый предназначен для защиты от записи и чтения. Биты ключа защиты памяти соответствуют полю ключа в слове состояния программы. Если пятый бит указанного восьмибитового поля равен единице, то операции чтения-записи для данного блока запрещены. Если же он равен нулю, то осуществляется сравнение значения поля ключа PSW со значением, записанным в первых четырех битах восьмибайтового поля блока. При их совпадении в данный блок может быть осуществлена запись информации, а операция выборки допускается в этом случае даже если значения в упомянутых полях не совпадут, т. е. для любой программы. При распределении для загружаемой программы области оперативной памяти каждый из занимаемых ею блоков имеет одинаковый ключ защиты, значение которого определяется первым из блоков, занимаемых данной программой. Это значит, что если программа занимает 80 К оперативной памяти и первый из занимаемых ею блоков имеет ключ защиты памяти, равный (0010)2, то и все остальные 39 блоков будут иметь такой же ключ защиты. Ключ защиты памяти помещается в поле слова состояния программы, которое ограничено 8-м и 11-м битами. Так как блоки одной программы имеют один и тот же ключ защиты, то при попытке обращения к блоку, имеющему другой ключ защиты (т. е. отличный от ключа, имеющегося в текущем PSW), происходит немедленное прекращение выполнения программы с последующей выдачей диагностического сообщения о наличии программной ошибки. Присвоение блокам оперативной памяти ключей защиты, соответствующих их адресам, осуществляется при помощи привилегированной команды УСТАНОВИТЬ КЛЮЧ ПАМЯТИ (SSK). Чтение ключа защиты происходит при помощи привилегированной команды ПРОЧИТАТЬ КЛЮЧ ПАМЯТИ (ISK). Эти команды предназначены для работы с ключами защиты и в качестве своих операндов используют два общих регистра, первый из которых применяется для хранения ключа защиты, второй – для задания адреса. Блоки, предназначенные для хранения модулей управляющей программы (супервизора), имеют ключ защиты, равный $(0000)_2$, и не защищены от операции чтения. Блоки оперативной памяти, занимаемой другими программами, имеют значения ключей памяти, отличные от нулевых. Причем блоки, занимаемые одной программой, не обязательно должны находиться в памяти рядом.

Поле маски состояний процессора (МСП). Это поле занимает в PSW двоичные разряды в 12-го по 15-й. Первый бит МСП идентифицирует используемый код, следующий указывает на то, маскируются или нет прерывания от схем контроля. Если он содержит единицу, то прерывания от схем контроля машины будут обрабатываться, если нуль – игнорироваться. Нулевое значение второго бита МСП используется при проведении специальных профилактических работ техническим персоналом обслуживания ЭВМ. В обычном режиме работы машины значение второго бита равно единице. Третий бит МСП идентифицирует состояние процессора, т. е. ожидает ли он завершения операций ввода-вывода (единичное значение) либо выполняет команды (нулевое значение). Четвертый бит МСП идентифицирует, какую из программ – управляющую или прикладную – в настоящий момент выполняет машина, т. е. находится ли она в состоянии

СУПЕРВИЗОР или ЗАДАЧА. Состоянию СУПЕРВИЗОР, в котором разрешено выполнение привилегированных команд, соответствует нулевое значение четвертого бита МСП. Состоянию ЗАДАЧА, в котором запрещено выполнение привилегированных команд, соответствует единичное значение. Таким образом, наличие четвертого бита МСП позволяет контролировать правомочность использования привилегированных команд. Указанный контроль также может быть осуществлен путем проверки ключа защиты памяти.

Поле код прерывания. Этот код находится в битах в 16-го по 31-й слова состояния программы. С его помощью можно определить одну из следующих причин прерывания работы ЦП: ошибку в выполняющейся программе; необходимость обращения к супервизору; прерывание по вводу-выводу; машинный сбой (сигнал от схем контроля ЭВМ); поступление внешнего сигнала. Каждый из типов прерываний идентифицируется определенным значением, записываемым в поле кода прерывания.

Поля длины команды и признака результата. Поле длины команды (ДК) находится в 32-м и 33-м двоичных разрядах PSW и указывает длину машинной команды, которая выполнялась последней, т. е. прежде чем произошло прерывание.

Поле признака результата (ПР) образуют следующие два двоичных разряда PSW. Это поле содержит такие коды условий, характеризующие результаты выполнения машинных команд, как "Произошло переполнение", "Результат равен нулю", "Результат сравнения подтвердил равенство операндов", "Полученное значение является отрицательной величиной" и т.д. Программист имеет возможность организовать дальнейшее выполнение программы в зависимости от значения признака результата путем использования команд, анализирующих это значение и принимающих соответствующие логические решения по переходу на исполнение того или иного блока программы. В зависимости от результата выполнения предыдущей команды принимается решение, какая из команд программы будет выполняться следующей. Таким образом, наличие в слове состояния программы поля признак результата и специальных команд переходов позволяет создавать программы с разветвленной многовариантной логикой вычислений. Команды переходов разделяются на команды условного перехода и перехода с возвратом. Команды условного перехода служат для определения той ветви программы, по которой должно продолжаться ее выполнение, в зависимости от признака результата или от величины, хранящейся в одном из заданных общих регистров. Команды перехода с возвратом используются в том случае, если надо осуществить обращение к подпрограмме, а затем продолжить выполнение с точки программы, откуда было произведено обращение.

Поле маски программы. Это поле занимает четыре двоичных разряда и служит для маскирования прерываний, являющихся результатом так называемых особых случаев в программе. Четыре бита поля маски программы идентифицируют ситуации, когда происходят соответственно: переполнение при выполнении операции с фиксированной точкой; переполнение при выполнении операции десятичной арифметики; исчезновение порядка; потеря значимости. Если в какой-либо двоичный разряд поля маски программы помещена единица, то при возникновении соответствующего особого случая в программе происходит прерывание. Если помещен нуль, то прерывание не происходит, так как оно считается замаскированным.

Поле адреса команды. Это поле указывает местоположение команды программы, которая должна быть выполнена следующей. После выполнения очередной команды программы значение, хранящееся в поле адреса команды, увеличивается на длину только что выполненной команды и соответствует адресу следующей команды программы. Изменение последовательности выполнения центральным процессором машинных команд может произойти при возникновении прерывания или в результате выполнения команды передачи управления. В последнем случае в поле адреса команды PSW записывается адрес команды, которой передается управление. В слове состояния программы под поле адреса машинной команды отведено 24 бит, что позволяет осуществлять адресацию более 16 М байт оперативной памяти.

§ 1.3. ОБРАБОТКА ПРЕРЫВАНИЙ

Прерывание представляет собой передачу управления в ответ на сигналы, асинхронные по отношению к выполнению команд [7]. При этом в счетчик адреса команд записывается новый адрес, а старое значение счетчика и информация, необходимая для идентификации текущего состояния прерываемой программы, запоминаются. После этого управление получает программа

обработки прерываний. *Прерыванием* называется ситуация, возникающая в результате воздействия какого-либо независимого события, приводящего к временному прекращению выполнения последовательности команд одной программы с целью выполнения последовательности команд другой программы [12].

Типы прерываний. В ЕС ЭВМ существуют следующие типы прерываний: внешние; для обращений к супервизору; программные; от схем контроля ЭВМ; ввода-вывода; повторного пуска.

Внешние прерывания возникают в результате получения сигналов с пульта оператора ЭВМ, от таймера, другого процессора или иного управляющего устройства, подсоединенного при помощи линий связи. Наличие прерываний типа прерывания по таймеру позволяет контролировать работу программы, для которой было введено ограничение на время исполнения. Это устраняет непроизводительные затраты времени центрального процессора в случае зацикливания программ пользователей. Сигналы с пульта оператора ЭВМ могут поступать в результате нажатия специальной кнопки ПРЕРЫВАНИЕ.

Прерывания для обращений к супервизору возникают при необходимости выполнения каких-либо действий со стороны операционной системы для обеспечения стандартной технологии вычислений.

Программные прерывания возникают в том случае, если в выполняющейся программе обнаружена ошибка. Это может быть указание неправильного кода операции, ошибка в данных, деление на нуль, переполнение, использование привилегированной команды, попытка нарушения защиты памяти и т. д. Определив наличие программного прерывания, супервизор прекращает выполнение обрабатывающей программы или передает ей управление, если в ее составе имеется специально разработанный модуль обработки программных сбоев. Обрабатывающая программа сообщает супервизору ОС ЕС о наличии модуля обработки программных сбоев при помощи макрокоманды SPIE. Если специальные средства обработки программных сбоев отсутствуют, супервизор ОС ЕС прекращает выполнение задания и распечатывает содержимое регистров оперативной памяти, занимаемой выполняемой программой.

Прерывания от схем контроля происходят в случае возникновения сбоев в каких-либо блоках ЭВМ. Эти прерывания инициируют процедуру диагностики основных узлов машины. При этом запрещается обработка всех других типов прерываний. После анализа ситуации, возникшей в результате прерывания от схем контроля, обычное возобновление работы вычислительной системы становится невозможным. Для этого необходимо произвести перезагрузку операционной системы.

Прерывания ввода-вывода обеспечивают обработку центральным процессором ситуаций, связанных с процедурами ввода-вывода данных на внешние устройства. Прерывания этого типа возникают при необходимости обращений к ЦП от устройств ввода-вывода и каналов. Поток запросов на прерывания от устройств ввода-вывода диспетчеризируется в соответствии с приоритетами, присваиваемыми в порядке поступления запросов, т. е. более высокий приоритет имеет ранее поступивший запрос. Запросы на прерывания ввода-вывода сохраняются в очереди до тех пор, пока центральный процессор их не обработает.

Прерывания повторного пуска возникают в результате действий со стороны оператора ЭВМ или другого процессора. Прерывания данного типа не маскируются и являются средством, с помощью которого инициируется выполнение некоторой программы [13]; оно возникает в результате нажатия кнопки повторного пуска на пульте управления ЭВМ.

Для разрешения конфликтной ситуации, которая может возникнуть при одновременном возникновении нескольких прерываний, организуется очередь с аппаратурным механизмом распределения приоритетов. В ЕС ЭВМ высший приоритет имеют прерывания от схем контроля, следующий приоритет имеют взаимоисключающие программные прерывания и прерывания для обращения к супервизору; далее идут подавляемые прерывания от схем контроля. На следующем уровне приоритетности находятся внешние прерывания, после которых идут прерывания ввода-вывода. Низший приоритет имеют прерывания повторного пуска. Возникновение неотложного условия прерываний от схем контроля предписывает их внеочередную обработку по отношению к прерываниям других типов. Если процессор находится в состоянии СТОП, то в плане приоритетности прерывания повторного пуска находятся вслед за программными прерываниями [13]. Процессор может находиться в одном из следующих четырех состояний: Р1, Р2, Р3, Р4. В

состоянии Р1 выполняются обрабатывающие программы, в состоянии Р2 обрабатываются прерывания. Состояние Р3 классифицируется как анализ прерываний; в состоянии Р4 обрабатываются прерывания от схем контроля. В состоянии Р1 недопустимо использование привилегированных команд, так как их применение приводит к возникновению прерывания программного типа. Выполнение привилегированных команд допускается только программами супервизора. В случае возникновения прерываний (кроме прерываний от схем контроля ЭВМ) происходит автоматическое переключение в состояние Р3, в котором программы супервизора определяют тип возникшего прерывания и соответствующую программу обработки этого прерывания. Переключение ЦП в состояние Р4 происходит в том случае, если возникает прерывание от схем контроля.

Каждому из приведенных выше типов прерываний соответствуют два PSW, одно из которых называется старым PSW, а другое – новым PSW. Старое и новое PSW хранятся в определенных ячейках оперативной памяти ЭВМ. В случае возникновения прерывания осуществляются определение его типа и запись аппаратным путем текущего PSW по адресу старого PSW. В качестве текущего загружается новое PSW, которое определяет порядок дальнейшего выполнения команд центральным процессором. Наличие новых PSW обеспечивает идентификацию программ супервизора, соответствующих каждому из типов прерываний. Старое PSW и специальная область сохранения позволяют запомнить текущее состояние прерванной проблемной программы с целью нормального возобновления ее работы после того, как будет обработано возникшее прерывание. Свое состояние, предшествующее моменту возникновения прерывания, система может восстановить путем загрузки старого PSW на место текущего на последней стадии работы программы обработки прерывания. Операции по пересылке полей PSW осуществляются после завершения цикла выполнения команд центральным процессором, т. е. после окончания одной команды и до начала следующей.

Причину возникновения прерывания устанавливает программа обработки прерываний, используя для этого поле кода прерывания старого PSW. Место в программе пользователя, в котором возникло прерывание, идентифицируется в старом PSW полем адреса команды.

Информация о возникшем прерывании записывается в слове состояния программы и в специальном регистре прерываний, каждый двоичный разряд которого идентифицирует соответствующую причину прерывания. Наличие регистра прерываний позволяет упорядочить прерывания в очередь, так как позиционность каждого бита устанавливает приоритет, соответствующий причине прерывания. Какой-либо бит в регистре прерываний устанавливается в единицу тогда, когда появляется соответствующий ему запрос на прерывание. Порядок обработки прерываний зависит от приоритетов, при этом после поступления прерывания на обработку соответствующий ему двоичный разряд в регистре прерываний обнуляется.

Для предотвращения ситуаций, при которых возможны потери прерываний, используется аппаратно реализованный механизм маскирования, запрещающий временно прерывание работы центрального процессора в случае возникновения прерываний указанных типов. Для реализации маскирования существуют следующие маски: системы, программы, прерываний от схем контроля. Каждому типу прерываний соответствуют два двойных слова, предназначенных для хранения старого и нового PSW и имеющих фиксированные адреса в начальной области оперативной памяти (табл. 1.1).

Таблица 1.1.

Адреса размещения PSW		Типы прерываний
старого	нового	
24	88	Внешние
32	96	Обращения к супервизору
40	104	Программные
48	112	От схем контроля
56	120	Ввода-вывода
0	8	Повторного пуска

Маскирование того или иного прерывания означает введение запрета на его обработку. В этом случае поступивший на замаскированное прерывание запрос помещается в очередь на ожидание или теряется, а ЦП выполняет следующую команду.

§ 1.4. СЛОВО СОСТОЯНИЯ КАНАЛА

Слово состояния канала (CSW) – специальное служебное поле, в котором хранится информация, характеризующая состояние канала. Слово состояния канала является двойным словом и занимает фиксированную область оперативной памяти, начинающуюся с 64-го байта.

Формат слова состояния канала. Формат CSW представлен на рис. 1.2.

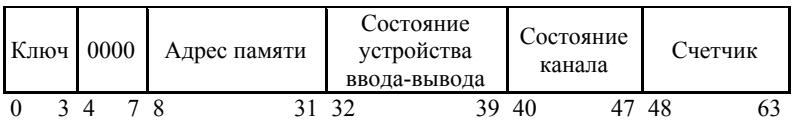


Рис. 1.2. Формат слова состояния канала

П о л е к л ю ч а соответствует аналогичному полю в адресном слове канала и используется для защиты оперативной памяти при выполнении операций ввода-вывода.

П о л е а д р е с а памяти характеризуется тем, что в него заносится адрес канальной команды, которая должна быть выполнена следующей. Значение этого поля формируется путем прибавления длины канальной команды (8 байт) к старому содержимому поля адреса памяти.

Поля состояния устройства ввода-вывода (УВВ) и состояния канала используются для хранения байтов состояний УВВ и канала. Они содержат информацию, идентифицирующую окончание работы канала и устройства ввода-вывода, занятость устройства, конец программной проверки.

Поле счетчика имеет 2 байта и соответствует значению счетчика байтов в последней выполненной канальной команде.

Операции ввода-вывода осуществляются следующими привилегированными командами: НАЧАТЬ ВВОД-ВЫВОД (SIO); ОПРОСИТЬ ВВОД-ВЫВОД (TIO); ОСТАНОВИТЬ ВВОД-ВЫВОД (НЮ); ОПРОСИТЬ КАНАЛ (TCH). Операнд каждой из перечисленных команд задает адрес устройства ввода-вывода путем указания поля, в котором биты с 21-го по 23-й идентифицируют канал, а биты с 24 по 31-й служат для указания устройства ввода-вывода. Работа канальной программы начинается по команде НАЧАТЬ ВВОД-ВЫВОД, которая по заданному адресу канала и устройства ввода-вывода определяет возможность осуществления операции ввода-вывода. Если такая возможность имеется, то активизируется, параллельно с работой центрального процессора, работа указанного канала.

Известно, что канал - это малая специализированная ЭВМ, управляемая канальной программой, состоящей из *команд канала* (CCW). Команда канала занимает двойное слово и состоит из кода операции, поля адреса данных, счетчика и признаков.

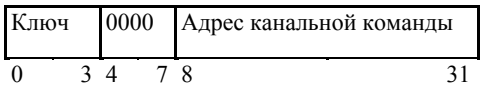


Рис. 1.3. Формат адресного слова канала

Для указания адреса команды канальной программы, которая должна выполняться первой, определено *адресное слово канала* (CAW). Адресное слово канала (рис. 1.3) является четырехбайтовым полем, расположенным в фиксированной памяти ЭВМ начиная с 72-го байта. Адресное слово канала состоит из трех полей. Первое поле (четыре двоичных разряда) содержит ключ защиты памяти; нулевое значение этого поля свидетельствует о том, что защита не используется или вообще не предусмотрена в ЭВМ. Второе поле всегда состоит из четырех двоичных нулей. Третье поле длиной в 24 двоичных разряда служит для указания адреса ОП первой команды канальной программы, т. е. первой CCW. В начале операции ввода-вывода адрес первой команды канальной программы пересылается в CAW. Последующие команды канальной программы определяются каналом без участия центрального процессора.

§ 1.5. МАШИННЫЕ И КАНАЛЬНЫЕ КОМАНДЫ

Машинная команда ЕС ЭВМ обеспечивает выполнение одной элементарной операции по обработке данных машинного языка и занимает в зависимости от формата одно, два или три полуслова. Законченная последовательность машинных команд составляет *программу.*

Форматы машинных команд. Существует пять форматов машинных команд: RR, RX, RS, SI, SS. Команды формата RR занимают одно полуслово, форматов RX, RS и SI – два полуслова, формата

SS – три полуслова. Во всех форматах левый крайний байт служит для хранения кода операции (КОП) машинной команды. Код операции служит для идентификации операции, требуемой для исполнения, и является признаком длины команды и типа адресации исходных и результирующих данных. Если операнд команды размещается в общем регистре, то для хранения номера этого регистра используется один полубайт, так как номер максимального (пятнадцатого) регистра в шестнадцатеричной системе счисления равен F. При описании машинных команд регистры для хранения одного из операндов машинной команды обозначаются в большинстве случаев буквой R. Цифра, следующая за буквой R, указывает на то, каким по счету является операнд, хранящийся в общем регистре: первым, вторым, третьим.

В машинных командах форматов RX, RS, SI и SS обязательно имеется операнд, хранящийся в оперативной памяти ЭВМ и адресуемый средствами базовой адресации. Суть *базовой адресации* сводится к следующему: операнд, хранящийся в оперативной памяти, адресуется с помощью базового регистра (базы) и смещения. В дальнейшем базовый регистр будем обозначать буквой B, а смещение – D. В качестве базового регистра используется один из шестнадцати общих регистров. Из 32 бит базового регистра для хранения базового адреса используются правые 24 бит. Часто базовый регистр обозначают B_i, где i = 1,2 – индекс, указывающий, какой из операндов машинной команды (первый или второй) адресуется по методу база – смещение. Обозначим через $[B_i]$ число, хранящееся в правых 24 бит общего регистра B. Смещение адреса операнда, хранящегося в оперативной памяти, обычно обозначают D_i (i=1,2). Смещение занимает три полубайта и может принимать значения в диапазоне от 000 до $(FFF)_{16} = (4095)_{10}$. Таким образом, адрес операнда вычисляется по формуле $A_i=[B_i] + D_i$. Если в качестве базового регистра используется регистр с номером 0, то его значение принимается равным нулю независимо от содержимого 24 правых бит.

Использование базовой (составной) адресации вызвано необходимостью оперировать с адресами, принимающими большие значения. Абсолютные адреса операндов равны сотням килобайтов или мегабайтам и в установленные размеры машинных команд уместиться не могут. Базовая адресация позволяет осуществить это при помощи одного полубайта для размещения номера базового регистра и трех полубайт для размещения значения смещения. Полный адрес составляется из содержимого 24 бит базового регистра, идентифицирующего отправную базовую точку адресации, и относительного (по отношению к базовому значению) адреса, заданного в 12-битовом поле смещения машинной команды. Укорочение кода адреса операнда в машинной команде происходит за счет того, что длина поля, отводимого для размещения смещения, относительно невелика. Применение базовой адресации позволяет сократить длину машинных команд за счет уменьшения полей, отводимых под хранение кодов адресов операндов, находящихся в оперативной памяти. Это приводит к более экономному и рациональному использованию оперативной памяти ЭВМ. Экономия достигается за счет того, что укорочение длин машинных команд приводит к общему уменьшению размеров программ, а следовательно, в один и тот же объем оперативной памяти можно записать либо большее количество обрабатывающих программ, либо больший объем исходных и результирующих данных.

Базовая адресация обеспечивает также позиционную независимость обрабатывающих программ. Это достигается в результате того, что присвоение соответствующих служебных адресов происходит не на этапах составления, трансляции или компоновки программы (что привело бы к ее жесткому позиционному размещению в оперативной памяти), а на этапе ее загрузки для исполнения. Таким образом, обрабатывающая программа и связанные с ней данные могут загружаться в произвольные, свободные на текущий момент времени участки оперативной памяти.

В случае размещения программы на другом участке оперативной памяти *безусловная адресация* вызывает необходимость изменения фиксированных адресов в командах программы. В отличие от этого при базовой адресации достаточно лишь произвести изменение значения, содержащегося в базовом регистре: эта процедура выполняется ЭВМ автоматически.

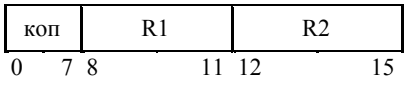


Рис. 1.4. Поля машинной команды формата RR

Машинные команды формата RR (регистр-регистр) имеют длину 2 байт (одно

полуслово) (рис. 1.4). Регистр R1 определяет первый операнд команды, регистр R2 – второй. Каждый из операндов команды занимает по 4 бит. После выполнения предписанной КОП операции результат помещается в R1, а содержимое регистра R2 остается прежним. В качестве примера рассмотрим случай сложения значений, размещенных в 6-м и 14-м регистрах. В этом случае машинная команда будет иметь вид: 1A6E, где 1A – КОП; 6 и E – обозначение соответственно 6-го и 14-го регистров. Результат данной операции будет помещен в 6-й регистр.

В ряде машинных команд поле первого операнда используется как 4-битовый индикатор, служащий для управления выполнением программы.

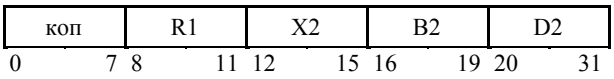


Рис. 1.5. Поля машинной команды формата RX

Машинная команда формата RX (регистр – индексированная память) использует два операнда (рис. 1.5). Первый операнд помещается в R1, а второй – в оперативной памяти по адресу A2=[B2] + [X2] + D2. Результат операции, как правило, записывается по адресу первого операнда. При этом второй операнд остается неизменным. Регистр X2, называемый *индексным,* служит для записи индексной части адреса второго операнда. Использование *индексной адресации* эффективно в тех случаях, когда обработка элементов массивов или последовательностей данных осуществляется одним и тем же набором операций (процедур) в программе. Чтобы не повторять многократно в программе однотипные блоки команд, выполняется поочередная выборка для обработки элементов последовательности данных путем соответствующего изменения индексной части адреса операнда, хранящейся в индексном регистре X2. Значения базового адреса и смещения адреса элемента последовательности данных для всех команд, входящих в соответствующую процедуру обработки, остаются неизменными. Модификация адреса, хранящегося в X2, производится при цикле обработки на заданный шаг изменения адресов обрабатываемых элементов. Таким образом, адрес второго операнда в машинной команде формата RX получается путем сложения трех значений: содержимого базового регистра, смещения (D2) и содержимого индексного регистра.

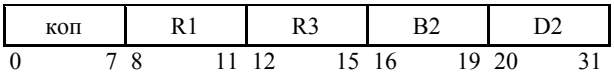


Рис. 1.6. Поля машинной команды формата RS

Машинные команды формата RS (регистр – оперативная память) оперируют с тремя операндами; структура их полей изображена на рис. 1.6. Первый операнд команды содержится в регистре R1. Второй операнд хранится в оперативной памяти в ячейке с адресом A2=[B2]+D2. Третий операнд адресуется при помощи регистра R3 и служит для хранения результата операции над первыми двумя операндами. Таким образом, использование регистра R3 позволяет не только получить результат выполнения заданной операции, но и сохранить два исходных операнда.

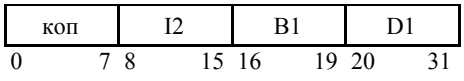


Рис. 1.7. Поля машинной команды формата SI

Машинные команды формата SI (оперативная память – непосредственный операнд) позволяют осуществлять непосредственное задание в поле второго операнда символьных значений. Структура машинной команды формата SI показана на рис. 1.7. Длина непосредственно представляемого операнда I2 равна 1 байт.

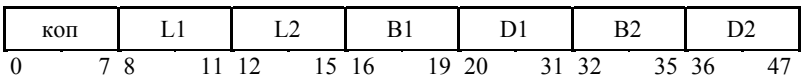


Рис. 1.8. Поля машинной команды формата SS

Машинные команды формата SS (оперативная память – оперативная память) имеют длину, равную 6 байт (рис. 1.8). Оба операнда идентифицированы путем задания их длин и адресов в оперативной памяти. Если операнды имеют различные длины (L1≠L2), то они могут

состоять не более чем из 16 символов. Под каждое из полей машинной команды длин операндов L1 и L2 отводится 4 бит. Если длины обоих операндов совпадают (L1 = L2), то для указания длины каждого из них отводится однобайтовое поле, что позволяет оперировать с последовательностями символов длиной до 256 символов.

Код канальной команды	Адрес данных	Флажки	Не используется	Счетчик передаваемых байтов

Рис. 1.9. Формат канальной команды

Канальные команды (CCW). Управление работой канала осуществляют канальные программы, состоящие из *канальных команд* (CCW). Формат канальной команды показан на рис. 1.9. Каждая CCW занимает двойное слово. Тип операции идентифицируется кодом канальной команды, который занимает первое поле в ее формате. Поле адреса данных определяет местоположение данных в оперативной памяти. Поле флажков содержит ряд признаков, характеризующих канал и соответствующую канальную операцию.

Поле счетчика содержит количество байтов, которое должно быть передано. Канальные программы хранятся в оперативной памяти. Адрес канальной программы, требуемой в каждый конкретный момент, помещается ОС ЕС в адресное слово канала CAW.

При необходимости выполнения операции ввода-вывода специальные средства ОС ЕС формируют в командах соответствующей канальной программы значения счетчика байтов и адреса данных. Адрес, идентифицирующий расположение соответствующей канальной программы в оперативной памяти машины, помещается в CAW, расположенном в фиксированной области ОП (72-й байт). Каждое из полей (код канальной команды, флажки, счетчик) занимает 1 байт. Адрес данных хранится в трехбайтовом поле канальной команды. Следующий после поля флажков пятый байт CCW не используется. Наличие единицы в первом бите поля флажков идентифицирует наличие цепочки данных, при которой после передачи количества байтов, указанных в поле счетчика, осуществляется переход к передаче данных, адрес которых указан в следующей CCW. В новой CCW также используются поля счетчика и флажков, а поле кода канальной команды игнорируется (применяется код канальной команды, задавшей цепочку данных). Наличие единицы во втором бите поля флажков идентифицирует наличие цепочки команд. В этом случае после передачи данных, количество которых задано в счетчике, будет выполняться следующая канальная команда. Наличие единицы в третьем бите поля флажков вызывает блокировку сигнала об ошибке, являющейся результатом несовпадения между количеством передаваемых байтов и аналогичным параметром, установленным для данного устройства ввода-вывода. Четвертый бит поля флажков служит для установления блокировки записи информации в память машины после ее считывания. Пятый бит поля флажков указывает на возникновение программно-управляемого прерывания. Последние три бита поля флажков всегда должны быть равны нулю. Во всех командах канальной программы, кроме последней, должен быть идентифицирован либо первый, либо второй бит поля флажков. Передача данных осуществляется побайтно. При этом, после передачи очередного байта, значение, хранящееся в поле CCW адреса данных, увеличивается на единицу, а значение, хранящееся в поле счетчика переданных байтов, уменьшается на единицу.

§ 1.6. НАКОПИТЕЛИ НА МАГНИТНЫХ ЛЕНТАХ И ДИСКАХ

В состав внешних запоминающих устройств ЕС ЭВМ входят накопители на магнитных лентах (НМЛ), магнитных дисках (НМД), кассетных магнитных лентах (НКМЛ), дискетах или гибких магнитных дисках (НГМД), кассетных магнитных дисках (НКМД) и т. д., а также устройства управления. Наибольшее распространение получили разработанные ранее других накопители на магнитных лентах и магнитных дисках.

Накопители на магнитных лентах. Эти накопители относятся к классу внешних запоминающих устройств последовательного доступа. В них доступ к требуемому набору данных происходит только после завершения перемотки всей предшествующей части магнитной ленты (МЛ). Такие накопители благодаря низкой стоимости, простоте эксплуатации и хранения, компактности и долговременности использования обладают несомненными преимуществами в тех случаях, когда порции данных обрабатываются последовательно друг за другом. В ЕС ЭВМ

применяется девятидорожечная магнитная лента шириной 12,7 мм и длиной до 750 м (на одной бобине). Информация записывается одновременно девятью магнитными головками. Из девяти одновременно записываемых битов информации восемь являются информационными и один - контрольным битом четности. Начало области магнитной ленты, в которую записывается информация, называется точкой загрузки и помечается специальным физическим маркером. Физический маркер представляет собой кусочек алюминиевой фольги, наклеиваемый на расстоянии 3 м от начала магнитной ленты. Конец информационной области МЛ помечается таким же физическим маркером, наклеиваемым на расстоянии 4 м от конца МЛ. Наличие указанных специальных маркеров, распознавание которых производится фотоэлектрическим способом, позволяет осуществить перемотку МЛ к началу информационной области и автоматический останов по достижении ее конца.

Плотность записи на магнитную ленту в зависимости от используемого метода записи может составлять 8, 22, 32, 63 и 64 бит/мм. В случае записи с плотностью 64 бит/м максимальная емкость МЛ составляет 48 М байт. Во избежание случайного искажения информации на МЛ применяется специальное кольцо; это механическая защита данных. Наличие кольца на бобине разрешает выполнение для этой МЛ операций чтения и записи. Без защитного кольца выполнение операции записи запрещается, чтение информации разрешается.

Помимо физических маркеров имеются специальные служебные блоки, называемые *ленточными марками;* обозначаются ТМ. Ленточные марки служат для выделения области на МЛ, занимаемой одним набором данных (файлом). Они используются также в дополнение к физическому маркеру конца МЛ. Две подряд записанные ленточные марки идентифицируют логическое окончание области магнитной ленты, содержащей информационные массивы пользователей. Данные на МЛ записываются блоками. Специальный блок ТМ применяется для отделения одного набора данных от другого.

С помощью *специальных промежутков* производится разделение отдельных записей друг от друга, что упорядочивает процесс обращения к одной записи за счет перемещения магнитной ленты от одного промежутка к другому. Размер промежутка между записями выбирается достаточным для разгона ленты до установленной скорости и остановки ее точно на следующем промежутке. Недостаток промежутков между записями – уменьшение полезного объема МЛ, так как области, отведенные под промежутки, нельзя использовать для хранения данных.

Частично указанный недостаток устраняет *процесс блокирования,* суть которого состоит в объединении нескольких записей в блоки.

Минимально допустимый размер одного блока на МЛ равен 18 байт. Максимальное ограничение на размер блока зависит от размера доступной оперативной памяти. Блокирование увеличивает полезный объем магнитной ленты за счет сокращения числа промежутков между записями. Кроме того, уменьшается количество операций ввода-вывода, так как за одну операцию производится пересылка не одной записи, а сразу нескольких. Преимущества блокирования, заключающиеся в увеличении полезного объема МЛ и уменьшений общего времени работы программы на ввод-вывод данных, значительно превосходят возникающие при этом недостатки, связанные с увеличением объемов данных в программе пользователя и необходимостью выполнения процедур по формированию блоков и разделению принятых блоков на записи.

Значение контрольного бита четности выбирается в зависимости от значений восьми информационных битов. Если число единиц в восьми информационных битах нечетное, то в контрольных бит четности заносится единица, а если четное – нуль. Таким образом, общее число единиц в девяти записываемых битах всегда должно быть четным, это контролируется в процессе чтения данных.

Индикатором возникшей ошибки является нечетное число битов в считываемом с МЛ символе. Причинами ошибок часто бывают дефекты покрытия МЛ и налипание на ее поверхности пыли.

Частота возникновения ошибок при работе с магнитными лентами существенно зависит от условий их хранения. Необходимо предусмотреть защиту МЛ от случайных магнитных полей и пыли, следить за соблюдением заданных значений Уровня влажности и температуры. Предусмотрена возможность пропуска выявленных дефектных участков на МЛ. Помимо посимвольного контроля, производимого при помощи контрольного бита четности, существует поблочный контроль данных. Его суть заключается в том, что в конце каждого блока записывается контрольная комбинация из 9 бит. Значение каждого из девяти битов формируется

так, чтобы общее количество единиц в каждой из девяти дорожек МЛ было нечетным. В случае возникновения в блоке единичной ошибки посимвольный и поблочный контроль позволяет определить ее местонахождение и выполнить автоматическое исправление. Для этого перед байтом поблочного контроля записывается байт циклического контроля. После обнаружения ошибки делается предположение о наличии временного дефекта МЛ и осуществляется повторная попытка записи информации на то же место. Если и последующие попытки заканчиваются неудачей, то дефектный участок просто пропускается. В целях контроля правильности выполнения операций записи – чтения помимо основного набора магнитных головок используется дополнительный.

С помощью дополнительного набора магнитных головок считываются только что записанные на МЛ биты информации, в случае их несовпадения идентифицируется состояние ошибки. Оба набора магнитных головок считывают данные с МЛ и при несовпадении какой-либо пары битов также будет выработан сигнал об ошибке.

Магнитные ленты в силу ряда своих положительных достоинств играют важную роль при организации больших информационных архивов и фондов пакетов программ. На каждом вычислительном центре имеется большое число МЛ. Это диктует необходимость санкционирования их использования, т. е. выдачу из хранилищ следует производить только тем пользователям, которые обладают правом доступа к записанной информации.

Для синхронизации работы быстродействующих каналов с более медленными накопителями на магнитных лентах применяются устройства управления НМЛ, или *ленточные контроллеры*. К 1987 г. освоено производство семи типов ленточных контроллеров ЕС ЭВМ. Каждый контроллер может управлять работой до восьми НМЛ, при этом он выполняет полученные от канала команды. Информационный обмен между каналом и НМЛ производится командами ЧТЕНИЕ, ЧТЕНИЕ НАЗАД, ЗАПИСЬ и УТОЧНИТЬ СОСТОЯНИЕ. Другие команды носят служебный характер и обеспечивают перемотку ленты, запись метки, установку требуемого уровня плотности записи, а также перемещения на шаг назад на файл или блок, на шаг вперед на файл или блок и т. д. Наличие специального инженерного пульта обеспечивает автономный режим работы контроллера.

В накопителях на магнитной ленте кассетного типа (НКМЛ) носителем информации является стандартная компакт-кассета с магнитной лентой шириной 3,81 мм и длиной 90 м. По сравнению с бобинными магнитными лентами преимущества НКМЛ состоят в компактности и менее высокой стоимости. Их недостатки заключаются в меньшей скорости обмена данными и меньшей в десятки раз информационной емкости.

Основной недостаток внешних запоминающих устройств на магнитных лентах – значительное время ожидания на помещение требуемой области магнитной ленты в зону магнитных головок для выполнения операции записи (считывания). Это занимает в большинстве случаев несколько десятков секунд, что существенно замедляет процесс обработки данных. Прогресс в этой области был достигнут путем разработки запоминающих устройств прямого доступа, включающих в свой состав накопители на магнитных барабанах и дисках (НМД), на гибком магнитном диске (НГМД) и кассетном магнитном диске (НКМД).

Накопители на магнитных дисках получили наибольшее распространение. В них каждая запись данных имеет свой собственный уникальный адрес, обеспечивающий непосредственный (минуя все остальные записи) доступ к ней. В НМД предусмотрена аналогичная НМЛ возможность последовательного доступа к информации. Накопитель на магнитных дисках сочетает в себе несколько устройств последовательного доступа, причем сокращение времени поиска данных обеспечивается за счет независимости доступа к записи от ее расположения относительно других записей. Конструкция НМД сложнее, чем у НМЛ, а следовательно, выше их стоимость. В НМД в качестве носителей данных используется пакет магнитных дисков, закрепленных на одном стержне, вокруг которого они вращаются с постоянной скоростью. Поверхность магнитного диска, покрытия ферромагнитным слоем, называется *рабочей.* Каждый магнитный диск пакета, кроме верхнего и нижнего, имеет две рабочие поверхности. Верхний и нижний магнитные диски обладают по одной рабочей поверхности, расположенной соответственно на нижней и верхней частях указанных дисков. Каждая рабочая поверхность магнитного диска разбита на 203 окружности (дорожки), пронумерованные от 0 до 202 от края к центру. На каждой из дорожек начало области данных механически идентифицировано при помощи маркера начала оборота. Дорожки, расположенные одна под другой на разных магнитных

дисках, образуют соответственно 203 так называемых цилиндра. Из пакетов магнитных дисков, аналогично магнитным лентам, организуют информационные архивы.

Несмотря на то что стоимость одного пакета магнитных дисков в несколько раз превышает стоимость бобины магнитной ленты, НМД благодаря незначительному времени доступа к информационным записям широко применяются для обращения к часто используемым данным и программам. Из 203 цилиндров 3 являются резервными и 200 - основными. Дорожки резервных цилиндров пользователям недоступны. Системные средства обеспечивают замену дорожки основного цилиндра, ставшей дефектной, на дорожку запасного цилиндра. Запись и считывание информации в НМД производит механизм доступа, состоящий из держателей магнитных головок. Количество магнитных головок равно числу рабочих поверхностей на одном пакете дисков. Если пакет состоит из 11 дисков, то механизм доступа состоит из 10 держателей с двумя магнитными головками на каждом из них. Держатели магнитных головок объединены в единый блок таким образом, чтобы обеспечить их синхронное перемещение вдоль всех цилиндров. Фиксируя блок механизма доступа на каком-либо из цилиндров с помощью только электронного переключения головок, можно сделать переход с одной дорожки на другую данного цилиндра. При фиксированном положении блока механизма доступа возможно обращение к любой из записей, находящихся на дорожках текущего цилиндра. Дорожки в цилиндре нумеруются начиная с верхних. Как правило, обращение к дорожкам происходит с нулевой по последнюю одного цилиндра, потом с нулевой дорожки следующего цилиндра и т. д. Каждый пакет магнитных дисков находится в специальном футляре, предохраняющем от механических загрязнений и пыли.

Любая операция чтения (записи) информации с (на) магнитного диска состоит из трех этапов [8]. На первом этапе происходит механический подвод магнитной головки к дорожке, содержащей требуемые данные. На втором этапе обеспечивается ожидание момента, пока требуемая запись не окажется в зоне магнитной головки. На третьем этапе осуществляется собственно процесс обмена информацией между вычислительной машиной и магнитным диском. Таким образом, общее время, затрачиваемое на операцию записи-считывания, состоит из суммы времен поиска соответствующей дорожки, ожидания подвода записи (так называемое время ротационного запаздывания) и обмена с ЭВМ. Максимальное значение времени ротационного запаздывания равно времени, за которое совершается полный оборот магнитного диска.

Работой накопителей на сменных магнитных дисках ЕС-5066 управляет устройство управления ЕС-5566 (СССР). Между устройством управления и НМД информация передается поразрядно, а между устройством управления и каналом – побайтно. В устройстве, выполненном на базе интегральных схем, обеспечивается контроль пересылаемых данных и определение работоспособности НМД. Одно устройство может обслуживать до восьми НМД.

§ 1.7. АЛФАВИТНО-ЦИФРОВЫЕ ПЕЧАТАЮЩИЕ УСТРОЙСТВА. УСТРОЙСТВА РАБОТЫ С ПЕРФОНОСИТЕЛЯМИ

Устройства вывода информации на печать бывают последовательного и параллельного типов. Устройства последовательного типа обеспечивают распечатку обработанной на ЭВМ информации со скоростью от 30 до 180 зн/с. Для большинства печатающих устройств параллельного типа скорость выдачи информации 700...1000 строк/мин. Эти устройства характеризуются построчным способом печати, который обеспечивается благодаря наличию специального буфера, емкость которого равна длине выдаваемой строки. Длина строки и используемый набор символов – это существенные факторы, от которых зависит скорость распечатки информации. В качестве носителя информации используется перфорированная бумага.

Алфавитно-цифровое печатающее устройство. В алфавитно-цифровом печатающем устройстве (АЦПУ) перемещение бумаги осуществляется по командам, поступающим от канала. Данные, требующие распечатки, поступают в буфер устройства побайтно. На АЦПУ возложено выполнение ряда контрольных функций: идентификация окончания и разрыва перфорированной бумаги; контроль за уходом бумаги в результате программной ошибки и т. д.

Основные компоненты механизма печати в АЦПУ следующие: набор молоточков; валик; красящая лента. Молоточки функционируют независимо друг от друга и приводятся в действие тогда, когда необходимые знаки находятся под ними. Валик закреплен на жесткой оси, вокруг которой он вращается с постоянной скоростью. Длина валика равна длине строки, которую можно напечатать на АЦПУ. На внешней стороне валика находятся параллельные его оси продольные

дорожки выпуклых литер. Литеры одной дорожки отображают только один печатаемый знак. Полное количество печатаемых на АЦПУ знаков равно количеству продольных дорожек валика, а количество литер в дорожке равно количеству символов, печатаемых в строке.

Устройства работы с перфоносителями. Информация на перфокартах кодируется в виду пробитых и непробитых позиций. В ЕС ЭВМ используются преимущественно 80-колоночные перфокарты. Каждая из позиций перфокарты находится на пересечении одной из 80 колонок и одной из 12 строк. Строки перфокарты нумеруются сверху вниз следующим образом: 12, 11, 0, 1, 2, ..., 9. Логической единицей информации является символ, кодируемый в виде комбинации пробивок в одной из колонок, что позволяет хранить на одной перфокарте до 80 символов. При этом символы набиваются на перфокарты (перфорируются) в коде КПК-12. Перфорация производится на специальных устройствах – перфораторах, обеспечивающих в целях контроля печать данных. В ЕС ЭВМ разработано более десяти типов устройств ввода информации с перфокарт и шесть типов устройств вывода информации на перфокарты. В частности, устройство ввода информации с перфокарт ЕС-6019 (СССР) обеспечивает скорость функционирования 1200 карт/мин. Подключать такое устройство к ЭВМ можно через мультиплексный или селекторный каналы. Перфокарты закладываются оператором ЭВМ в подающий карман устройства, после чего одна за другой через считывающий механизм переносятся в приемный карман. Информация с перфокарт считывается при помощи фотоэлектрического поколонного способа. Емкости подающего и приемного карманов рассчитаны на 1950 перфокарт. В процессе считывания данных устройство осуществляет их посимвольный контроль на предмет появления комбинаций пробивок, не - соответствующих коду КПК-12.

В номенклатуру устройств ЕС ЭВМ входят устройства ввода информации с перфолент (ЕС-6022, различные исполнения ЕС-6121, ЕС-6122, ЕС-6191) и вывода информации на перфоленту (ЕС-7022). Перфоленты широко применялись в телеграфии, а также в ЭВМ первого и второго поколений. Ширина перфоленты порядка 2 см. Ее технической основой является плотная бумага или пластмасса. Как и в перфокартах, данные на перфоленту заносятся поколонно в виде комбинаций пробивок. Помимо синхродорожки на перфоленте может быть от 5 до 8 информационных дорожек. Устройство ввода информации с перфоленты ЕС-6122 (ЧССР) обеспечивает считывание данных со скоростью 1500 строк/с.

§ 1.8. ТЕХНИЧЕСКИЕ СРЕДСТВА СИСТЕМ ТЕЛЕОБРАБОТКИ ДАННЫХ

Режим телеобработки данных подразумевает обработку на ЭВМ информации, поступающей посредством аппаратуры передачи данных от пользователей удаленных абонентских пунктов (терминалов) или сопряженных вычислительных машин. Системы телеобработки данных представляют собой совокупность технических и программных средств. Комплекс технических средств телеобработки данных включает в себя абонентские пункты, мультиплексоры передачи данных, модемы, каналы связи и т. д.

Абонентский пункт. Абонентский пункт (АП) – устройство, обеспечивающее дистанционно удаленным от ЭВМ пользователям возможность ввода исходных данных или запросов на обработку с последующим получением результирующей информации. Абонентский пункт обычно состоит из устройства управления (УУ) и одного или нескольких устройств ввода-вывода (УВВ) [16].

Разнообразные типы АП отличаются способами представления исходных и результирующих данных, режимами и скоростями передачи информации, средствами программной поддержки, областями использования, режимами взаимодействия с ЭВМ и т. д. Если АП обеспечивает одновременную работу сразу нескольких пользователей терминалов, то это абонентский пункт *группового использования.* Он должен иметь в своем составе несколько УВВ. Если АП ориентирован на обслуживание в течение сеанса работы одного пользователя, то это абонентский пункт *индивидуального использования.*

Применяется два типа кодов передачи информации: семиразрядный КОИ-7 (для большинства АП) и международный телеграфный пятибитовый МТК-2 (для телеграфных аппаратов). В состав АП могут входить следующие периферийные устройства: дисплеи, телетайпы, АЦПУ, НМЛ, НМД, устройства ввода-вывода с перфокарт и перфолент и т. д. По скорости передачи информации устройства телеобработки данных делятся на низкоскоростные (обмен данными осуществляется по телеграфным линиям связи со скоростью до 200 бит/с), среднескоростные (от

200 до 4800 бит/с) и высокоскоростные (свыше 4800 бит/с).

Мультиплексор передачи данных (МПД). Он служит для управления обменом информацией между ЭВМ и АП и подключается к мультиплексному каналу. Мультиплексор включает в свой состав несколько специальных адаптеров, каждый из которых в зависимости от его типа обслуживает телеграфные или телефонные выделенные или коммутируемые линии связи. Основные функции мультиплексора передачи данных следующие: преобразование параллельной кодовой комбинации в последовательную и наоборот; удаление и вставка стартовых и стоповых битов; добавление и удаление контрольных битов; формирование и проверка для каждого блока контрольной знаковой последовательности; распознавание специальных управляющих символов; измерение временных интервалов (тайм-аутов); синхронизация процессов передачи информации по каналам связи различных типов; выполнение служебных команд каналов и т. д.

В ЕС ЭВМ используются МПД ЕС-8400 (СССР), ЕС-8401 (НРБ), ЕС-8402 (СССР), ЕС-8403 (СССР), ЕС-8404 (ГДР), ЕС-8410 (ВНР).

Мультиплексор передачи данных ЕС-8400 (МПД-1А) обеспечивает одновременную работу по каналам связи с 15 различными АП. В качестве АП могут применяться ЕС-8561, ЕС-8563, ЕС-8570, телеграфный аппарат РТА-60. При помощи МПД-1А можно осуществить режим межмашинного обмена информацией между двумя ЕС ЭВМ, оснащенными МПД данного типа. Мультиплексор МПД-1А имеет специальный переключатель, позволяющий поочередное его использование двумя ЕС ЭВМ. Обеспечивается скорость передачи информации до 75 бит/с по телеграфным линиям связи и до 2400 бит/с по телефонным. При этом можно подключить 8 дуплексных каналов связи или 16 полудуплексных.

Мультиплексор передачи данных ЕС-8401 (МПД-1) предназначен для обмена данными по некоммутируемым и коммутируемым телеграфным и некоммутируемым четырехпроводным телефонным каналам связи. При помощи МПД-1 ЭВМ может обмениваться данными с абонентскими пунктами ЕС-8501, ЕС-8570, ЕС-8562, ЕС-8564 и стартстопными телеграфными аппаратами Т-63. Используются дуплексный и полудуплексный режимы обмена информацией. Обеспечивается управление 8 дуплексными и 63 – полудуплексными каналами связи. При помощи МПД-1 можно передавать данные со скоростью от 50 до 2400 бит/с.

Мультиплексор передачи данных ЕС-8402 (МПД-2) позволяет сопряжение с ЕС ЭВМ абонентских пунктов и удаленных вычислительных комплексов. Обмен информацией с АП происходит в коде КОИ-7, а с телеграфными аппаратами - в коде МТК-2. Мультиплексор МПД-2 обладает двухканальным переключателем, управление которым производится вручную или программно от ЭВМ. Такой переключатель обеспечивает попеременное подключение МПД-2 к двум разным ЕС ЭВМ.

Мультиплексор передачи данных МПД-2 в зависимости от комплектации может управлять полудуплексными (от 8 до 176) и дуплексными (от 4 до 88) линиями связи. Информационный обмен может вестись со скоростями, находящимися в диапазоне от 50 до 4800 бит/с. Специальное оборудование позволяет автономно проверять работоспособность компонентов МПД-2.

Мультиплексор передачи данных ЕС-8403 (МПД-3) обеспечивает подключение к ЕС ЭВМ абонентских пунктов ЕС-8501, ЕС-8570, телеграфного аппарата РТА-60 и др., а также функционирование режима межмашинного обмена информацией между удаленными вычислительными комплексами, созданными на базе ЕС ЭВМ. Обмен информацией может осуществляться по двум дуплексным или четырем полудуплексным каналам связи. В качестве каналов связи используются коммутируемые или выделенные телефонные, или выделенные телеграфные линии. В МПД-3 предусмотрены блок автономного тестирования и двухканальный переключатель, управляемый программно или вручную и обеспечивающий попеременное применение устройства двумя ЕС ЭВМ.

Мультиплексор передачи данных ЕС-8404 (МПД-4) предназначен для сопряжения ЕС-1040 с АП ЕС-8505. В полудуплексном режиме обмена МПД-4 обеспечивает передачу данных со скоростью от 100 до 1200 бит/с; максимально можно подключить 12 каналов связи.

Мультиплексор передачи данных ЕС-8410 (МПД-10) позволяет управление абонентскими пунктами ЕС-8501, ЕС-8504, ЕС-8S61, ЕС-8562, ЕС-8563, ЕС-8564, ЕС-8570. Обмен данным происходит по некоммутируемым телеграфным и коммутируемым или некоммутируемым телефонным каналам связи. Обеспечивается обслуживание до 32 каналов

связи. Максимальная скорость передачи информации 2400 бит/с.

М о д е м ы . Эти устройства в системах телеобработки данных выполняют функции частотно-модулированного преобразования сигналов при удаленном информационном обмене между пользователями терминалов и ЭВМ. Модем – устройство сопряжения АП с линией связи. Для сопряжения удаленной терминальной станции с ЭВМ помимо мультиплексора передачи данных и линии связи необходимо наличие модема на каждом из концов этой линии.

В качестве примера рассмотрим характеристики модема ЕС-8010 (СССР). Устройство обеспечивает частотно-модулированное преобразование сигналов, передаваемых в дуплексном или полудуплексном режимах по четырехпроводным линиям связи. Модем ЕС-8010 может функционировать в режимах настройки, переговоров, передачи информации и шлейфа. В режиме «Настройка» с помощью встроенного датчика сигналов обеспечивается установка частотных характеристик канала связи. В режиме «Переговоры» происходит обмен служебными сообщениями в промежутках между сеансами передачи информации. Собственно информационный обмен осуществляется в режиме ″Данные″. В режиме ″Шлейф″ в канал связи передается только что из него полученная информация. Этот режим очень удобен при отладке и тестировании программного обеспечения межмашинного обмена данными. Модем ЕС-8010 сопрягается с устройством защиты от ошибок; обеспечивается обмен данными со скоростями 75, 600, 1200 и 2400 бит/с.

К а н а л ы с в я з и . Под каналом связи понимается передающая среда, обеспечивающая перемещение в пространстве информационных сигналов [24]. В реальных применениях в качестве физической передающей среды применяются телефонные и телеграфные линии связи, витые пары проводов, коаксиальные кабели, волоконно-оптические кабели, радиоканалы и др.

Из широкого перечня терминальных устройств значительное распространение в нашей стране получили дисплейные комплексы типа ЕС-7920, поставляемые заказчиком в виде трех вариантов модификаций: группового локального, одиночного удаленного и группового удаленного.

Г р у п п о в о й л о к а л ь н ы й в а р и а н т дисплейного комплекса ЕС-7920 состоит из набора дисплеев ЕС-7927 и устройств печати ЕС-7934 (или ЕС-7936), управляемых устройством группового управления ЕС-7922. Устройство группового управления, к которому можно подсоединить максимально до 32 дисплеев и устройств печати, подключается непосредственно к мультиплексному, блок-мультиплексному или селекторному каналам. Дисплеи и устройства печати должны быть удалены от ЕС-7922 не более чем на 1200 м, расстояние между ЭВМ и самим ЕС-7922 не должно превышать 60 м.

О д и н о ч н ы й у д а л е н н ы й в а р и а н т дисплейного комплекса состоит из дисплея ЕС-7925, снабженного устройством печати ЕС-7934. Подключение к каналу ЭВМ осуществляется посредством мультиплексора передачи данных (например, МПД-3) и телефонной линии связи с модемами на обоих концах.

Г р у п п о в о й у д а л е н н ы й в а р и а н т дисплейного комплекса ЕС-7920 состоит из устройства группового управления ЕС-7922, дисплеев ЕС-7927 и устройств печати ЕС-7934 или ЕС-7936. Допускается произвольное соотношение между количеством дисплеев и устройств печати при условии, что их суммарное количество не превышает 32. Аппаратура передачи данных используется та же, что и для подключения к ЭВМ одиночного дистанционного дисплейного комплекса ЕС-7920.

Дисплей ЕС-7927 состоит из экрана и клавиатуры. На экране дисплея могут отображаться буквы русского и латинского алфавита, знаки препинания, служебные символы. В зависимости от заданной кодировки на экране дисплея можно организовать информационные поля следующих типов: защищенные или не защищенные от вмешательства оператора; алфавитно-цифровые; цифровые; отображаемые с нормальной или повышенной яркостью; неотображаемые; отображаемые с миганием; выбираемые фотоселектором.

В дисплее ЕС-7927 используется специальный символ, называемый *курсором.* Он изображается под знаковой позицией для указания места на экране, на которое будет помещен очередной вводимый с клавиатуры знак. При включении устройства курсор автоматически генерируется в первой позиции (на пересечении первой строки и первого столбца). Положение курсора может изменить оператор дисплея при помощи специальных служебных клавиш клавиатуры или программа, взаимодействующая в настоящий момент с данным устройством. Экран дисплея

состоит из 24 строк и 80 столбцов, что позволяет отображать 1920 символов.
На клавиатуре дисплея ЕС-7927 можно выделить десять групп клавиш. Первая группа состоит из 47 клавиш. При помощи клавиш верхнего регистра первой группы можно задавать символы русского алфавита и цифры. Нижний регистр составляют клавиши латинского алфавита и служебные символы. Вторая группа включает четыре клавиши, обеспечивающие передвижение курсора вверх, вниз, влево и вправо. Эти клавиши относятся к типу клавиш с возможностью автоматического повторения. Третья группа содержит 12 клавиш, называемых функциональными. Они обеспечивают краткую форму задания запросов на обработку данных. Остальные группы клавиш клавиатуры устройства ЕС-7927 служат для задания команды ввода данных, набранных на экране дисплея, их редактирования и табуляции. С помощью этих клавиш можно стирать всю набранную на экране информацию или только ту часть, которая расположена от позиции, указанной курсором, до конца экрана. Возможно также посимвольное вычеркивание, когда стирается символ, находящийся на позиции курсора, путем сдвига всей оставшейся части строки влево. Клавиша ”Новая строка” обеспечивает перемещение курсора в первую позицию следующей строки, а при использовании незащищенных полей - в начало первого незащищенного поля, находящегося на следующей строке. Клавиши табуляции позволяют перемещать курсор с одного незащищенного поля к другому. Использование клавиши ″Табуляция влево″ позволяет переместить курсор в первую позицию предыдущего незащищенного поля, а клавиши ″Табуляция вправо″ - в первую позицию следующего незащищенного поля.

В странах СЭВ в рамках ЕС ЭВМ выпускается более 20 типов абонентских пунктов, дисплейных станций и других разновидностей терминальной техники. К числу их основных производителей относятся СССР, ГДР, ВНР, ПНР.

Глава 2. СОСТАВ И ОСНОВНЫЕ ФУНКЦИИ СИСТЕМНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В главе описываются назначение и состав программного обеспечения ЭВМ третьего поколения, разделяемого на системное и прикладное. Основные компоненты системного программного обеспечения - операционные системы, системы программирования, специальные тестовые программы, наборы вспомогательных сервисных программ и т. д. Важным качеством системного программного обеспечения ЭВМ третьего поколения является его открытость, благодаря которой оно постоянно может развиваться и пополняться, обеспечивая функционирование новых типов технических средств, расширение областей использования, модернизацию способов обработки данных и т. д. На базе системного программного обеспечения создается прикладное программное обеспечение, ориентированное на решение конкретных экономических, научно-технических и других задач.

§ 2.1. СИСТЕМНОЕ И ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Известно, что любая современная ЭВМ состоит из аппаратных средств и программного обеспечения. *Программное обеспечение* выполняет функцию посредника между пользователями и ЭВМ, расширяет возможности аппаратуры вычислительной машины, являясь логическим ее продолжением. Использование развитого программного обеспечения позволяет увеличить производительность вычислительных комплексов, автоматизировать многочисленные рутинные информационные процессы в различных областях человеческой деятельности, повысить производительность труда разработчиков различных систем автоматизированной переработки информации, сократить общие сроки разработок и т. д.

Программное обеспечение. Программное обеспечение ЭВМ третьего поколения можно разделить на системное и прикладное. *Системное программное обеспечение* представляет собой комплекс управляющих и обрабатывающих программ, описаний и инструкций, обеспечивающих функционирование вычислительной системы, а также разработку и исполнение программ пользователей. Состав системного программного обеспечения мало зависит от характера решаемых задач пользователей. *Прикладное программное обеспечение* представляет собой совокупность программ решения конкретных задач из различных сфер применения ЭВМ. Специализированные комплексы программ решения конкретных задач вместе с сопровождающей документацией называют *пакетами прикладных программ* (ППП).

Объем программного обеспечения современных вычислительных систем непрерывно возрастает, несмотря на то что его стоимость остается довольно высокой даже при использовании промышленных методов разработки.

Особенно велика роль системного программного обеспечения, так как на его базе разрабатывается специальное программное обеспечение. Например, объем системных программ ЕС ЭВМ превышает 1,5 млн. команд; системное программное обеспечение вычислительной машины БЭСМ-6 имеет объем около 250 тыс. команд. Нередко доля стоимости системного программного обеспечения от общей стоимости вычислительной системы достигает 50 % и выше.

Системное программное обеспечение. Программы системного программного обеспечения различаются по функциональному назначению и характеру исполнения. Они делятся на испытательные программы, системы программирования и операционные системы.

Испытательные программы предназначены для проверки исправности блоков ЭВМ, обнаружения и локализации отказов устройств и устранения их влияния на работу системы в целом.

Система программирования представляет собой комплекс программных средств, обеспечивающих автоматизацию программирования и отладки программ. К системе программирования относятся языки программирования, трансляторы, отладочные программы. Эти программные средства предназначены для повышения производительности труда программистов. Программы системы программирования, как и программы пользователей, выполняются под управлением операционной системы.

Операционная система представляет собой совокупность программ, управляющих ходом работы вычислительной машины, идентифицирующих программы и данные и осуществляющих связь между машиной и оператором. Операционная система повышает производительность вычислительного комплекса за счет гибкой организации прохождения потока задач через машину, равномерной загрузки оборудования, оптимального использования всех ресурсов ЭВМ, стандартной организации хранения в машине больших массивов данных при наличии разнообразных способов доступа к ним.

К числу наиболее известных первых управляющих программ относятся комплексы SAGE, SABRE, MERCURY реализованные на ЭВМ второго поколения. Для ЭВМ IBM/360 были разработаны операционные системы, обеспечивающие пакетную технологию обработки данных и работу в реальном масштабе времени, а также реализацию многомашинных и мультипроцессорных комплексов. Первые версии отечественных операционных систем для ЭВМ третьего поколения (дисковая операционная система (ДОС) ЕС и ОС ЕС) были сданы в эксплуатацию в начале 70-х годов. Системное программное обеспечение ЭВМ предназначено для осуществления адаптируемости программ пользователей к изменениям состава ресурсов ЭВМ. Высокая производительность вычислительной системы обеспечивается операционной системой благодаря применению режимов пакетной обработки и мультипрограммного, и наличию специальных программных средств для выполнения трудоемких операций ввода-вывода информации. Высокая производительность труда программиста достигается за счет предоставления ему большого числа языков программирования; специальных библиотек программ; удобных средств ввода-вывода, отладки программ и оформления заданий.

Системы программирования в зависимости от уровня формализации входного языка и назначения делятся на машинно-ориентированные, процедурные, проблемные и вспомогательные.

Машинно-ориентированные системы программирования характеризуются взаимосвязью языков программирования с типом ЭВМ, для которой разработана данная система. Для систем символического кодирования, относящихся к машинно-ориентированным системам, характерно, что одной символической инструкции входного языка соответствует одна машинная команда, причем распределение памяти производится автоматически. Язык символического кодирования максимально приближен к машинному, поэтому позволяет создавать очень эффективные программы. С другой стороны, трудоемкость программирования на таком языке очень велика. Системы символического кодирования используются как база для создания более совершенных систем программирования. К автокодам относятся такие машинные системы программирования, в которых одна инструкция входного языка (макрокоманда) заменяется при трансляции несколькими машинными командами.

Процедурные системы программирования используют алгоритмические машинно-независимые языки. Процедурная система включает в себя входной язык и транслятор, переводящий программу с входного языка в символические коды или машинные команды. Процедурные системы программирования позволяют существенно повысить производительность труда программистов,

однако качество оттранслированных программ (объем программы и время счета) ниже, чем у программ, созданных с использованием машинных систем программирования.

Первым алгоритмическим языком высокого уровня явился ФОРТРАН, разработанный в середине 50-х годов для распространенной в то время вычислительной машины IBM/704. На начало 70-х годов в мире насчитывалось свыше 700 алгоритмических языков и около 300 трансляторов. Наибольшее распространение получили следующие алгоритмические языки: КОБОЛ, АЛГОЛ-60, ПЛ/1, ФОРТРАН, АЛГОЛ-68, БЕЙСИК. Широко известны исследования в области создания функционально-ориентированных алгоритмических языков, рассчитанных на пользователей определенных специальностей: экономистов, медиков, проектировщиков и др., не являющихся профессионалами в области программирования.

Основные преимущества языков высокого уровня перед машинно-ориентированными – проблемная ориентированность, компактность записи программ, необязательность подробных знаний ЭВМ и т. д. В то же время машинно-ориентированным языкам присущи универсальность применения и экономное расходование оперативной памяти вычислительной машины. Использование языков низкого уровня вызывает необходимость подробных знаний эксплуатируемой ЭВМ; разработанные с их помощью программы состоят из большего числа операторов. Языки низкого уровня применяются преимущественно при разработке системных программ, а языки высокого уровня – при разработке прикладных программ. Развитие процедурных систем программирования идет в основном по пути универсализации алгоритмических языков.

Проблемные системы программирования ориентированы на узкий класс однотипных задач. К их созданию привело стремление максимально снизить требования к уровню программистской подготовки пользователей. При использовании проблемной системы программирования пользователь избавлен от необходимости разрабатывать алгоритм решения задачи, ему достаточно описать постановку задачи. В этом отношении к проблемным системам программирования примыкают некоторые пакеты прикладных программ.

Вспомогательные системы программирования включают в себя наборы программ, выполняющих при обработке данных ряд сервисных функций, необходимых всем или большинству пользователей. К ним относятся программы отладки, редактирования, обслуживания библиотек, копирования массивов данных и т. д.

Система программного обеспечения ЕС ЭВМ включает в свой состав операционные системы, комплексы программ технического обслуживания, пакеты прикладных программ. Эта система открытая, т. е. ее состав может пополняться, обеспечивая развитие технических средств, совершенствование методов обработки информации, расширение сфер применения.

На базе операционных систем строятся ППП, расширяющие функции ОС, и пакеты общего назначения для решения различных научных, технических, экономических и других задач. Такие пакеты не входят в ОС и поставляются отдельно. Многие ППП имеют собственные средства генерации.

К пакетам, расширяющим функции ОС, относятся ППП, обеспечивающие работу многопроцессорных и многомашинных комплексов, диалоговые системы, системы работы в реальном масштабе времени, пакеты телеобработки данных и т. д.

К пакетам общего назначения относятся ППП, обеспечивающие применение графопостроителей, системы моделирования, пакеты для научно-технических расчетов, решения задач математического программирования, теории массового обслуживания, обработки матриц и т. п.

Для улучшения организации вычислительных работ, более эффективного использования разработанных программ и ликвидации параллелизма в разработке программного обеспечения в нашей стране создан Государственный фонд алгоритмов и программ (ГосФАП). В ГосФАП принимаются для хранения и распространения пакеты прикладных программ для решения научно-технических, экономических и других задач на ЭВМ различных типов, а также методические и инструктивные материалы по программированию, алгоритмическим языкам и организации вычислительных работ, информационные и справочно-библиографические материалы. С ГосФАП взаимодействуют отраслевые фонды алгоритмов и программ (ОФАП), опирающиеся на фонды алгоритмов и программ предприятий.

Одно из основных требований к разработке программного обеспечения ЭВМ – *модульность*.

Модульная структура программ и программных комплексов облегчает организацию работы больших коллективов программистов по созданию программного обеспечения. Другое важное требование к программному обеспечению – *возможность развития* программной системы. Выполнению этого требования способствует модульная организация программ. Существенным является требование *простоты* освоения, поддержания, эксплуатации и совершенствования возможностей программного обеспечения. Это позволяет обходиться небольшим числом специалистов, обслуживающих принятое к эксплуатации программное обеспечение.

Система программного обеспечения предназначена для эксплуатации многочисленными группами пользователей в различных организациях и предприятиях, поэтому она должна обладать свойствами *гибкости, адаптируемости.* Эти требования обеспечиваются применением принципов открытости, машинной независимости обрабатывающих программ, унификации использования периферийного оборудования и т. д. По возможности должна достигаться *совместимость* программного обеспечения различных ЭВМ и систем обработки данных. Как правило, совместимость программ обеспечивается в ЭВМ одного семейства. Программная совместимость для различных семейств ЭВМ достигается на уровне языков программирования.

Требование *минимальности вмешательства человека* в процесс обработки информации на ЭВМ удовлетворяется путем автоматизации различных этапов вычислительного процесса. В частности, автоматическое распределение ресурсов повышает эффективность использования вычислительной системы. Программное обеспечение должно удовлетворять также требованиям *параметрической универсальности, функциональной избыточности* (наличие в системе нескольких программ, реализующих одну и ту же функцию), *функциональной избирательности* (возможность генерации системы программного обеспечения в соответствии с потребностями и возможностями конкретного пользователя).

Техническая документация на разработанные программные средства, являющаяся одним из важнейших элементов программного обеспечения, должна оформляться по единым требованиям стандартизации. К технической документации относятся графические и текстовые документы, определяющие назначение, состав и структуру созданного программного средства. В них должны содержаться сведения, необходимые для тестирования, приемки, обучения пользователей, эксплуатации и наращивания возможностей программ. Выпуск документации является трудоемким процессом, поэтому желательно его автоматизировать. Документация на программное обеспечение должна удовлетворять требованиям *единства* терминологии, номенклатуры и наименования документов, единой системы обозначений в документах, идентичности документов независимо от места их разработки. Кроме того, должны соблюдаться единые правила внесения изменений, учета и хранения документации. Детальность описания отдельных модулей программного обеспечения должна соответствовать уровню подготовки потенциальных пользователей (системного программиста, программиста-пользователя, оператора и т. д.).

§ 2.2. ОПЕРАЦИОННЫЕ СИСТЕМЫ И ИХ ОСНОВНЫЕ КОМПОНЕНТЫ

Операционная система - набор программ, обеспечивающий организацию вычислительного процесса на ЭВМ. Основные функции ОС следующие: увеличение пропускной способности ЭВМ (за счет организации непрерывной обработки потока заданий с автоматическим переходом от одного задания к другому и эффективного распределения ресурсов ЭВМ по нескольким задачам); уменьшение времени ожидания пользователями ответов от ЭВМ; упрощение работы разработчиков программных средств и сотрудников обслуживающего персонала ЭВМ (за счет предоставления им значительного количества языков программирования и разнообразных сервисных программ).

Виды операционных систем. Автоматическая смена объединенных в пакет заданий пользователей, называемая *пакетной обработкой,* увеличив полезную загрузку ЭВМ, лишила программистов непосредственного контакта в произвольные моменты времени со своими программами во время их нахождения в машине. Возникшая ситуация вызвала необходимость создания и эксплуатации многопультовых систем, обеспечивающих обслуживание одновременно нескольких программистов. Это так называемые *системы с разделением времени* или *системы коллективного пользования,* которые могут эффективно применяться только при наличии весьма развитых ОС.

Программы ОС ЕС постоянно (резидентно) занимают в оперативной памяти объем, установленный при генерации системы. Остальные части операционной системы по мере необходимости вызываются из внешней памяти на МД.

Компоненты ОС ЕС. Операционная система обеспечивает осуществление в вычислительной системе следующих процессов: пакетной обработки заданий; работы системы в режимах диалоговом и квантования времени; работы системы в реальном масштабе времени в составе многопроцессорных и многомашинных комплексов; связи оператора с системой; протоколирования хода выполнения вычислительных работ; обработки данных, поступающих по каналам связи; функционирования графических устройств ввода-вывода; использования широкого набора средств отладки и тестирования программ; планирования прохождения задач в соответствии с их приоритетами; ведения учета и контроля за использованием данных, программ и ресурсов ЭВМ.

Основные компоненты операционных систем - управляющая и обрабатывающие программы. У п р а в л я ю щ а я п р о г р а м м а управляет работой вычислительной системы, обеспечивая в первую очередь автоматическую смену заданий для поддержания непрерывного режима работы ЭВМ при переходе от одной программы к другой без вмешательства оператора. Управляющая программа определяет порядок выполнения обрабатывающих программ и обеспечивает необходимым набором услуг для их выполнения. Основные функции управляющей программы: ввод и планирование выполнения заданий, поступающих непрерывным потоком (управление заданиями); последовательное или приоритетное выполнение каждой работы (управление задачами); хранение, поиск и обслуживание данных независимо от их организации и способа хранения (управление данными).

В соответствии с этими функциями основными частями управляющей программы являются управление заданиями, супервизор, управление данными. Управляющие программы могут иметь различную конфигурацию, определяющуюся на этапе генерации ОС.

Основные составные части управляющей программы изображены на рис. 2.1. *Программа начальной загрузки* выполняет настройку оперативной памяти, загрузку ядра операционной системы и программы инициализации ядра.

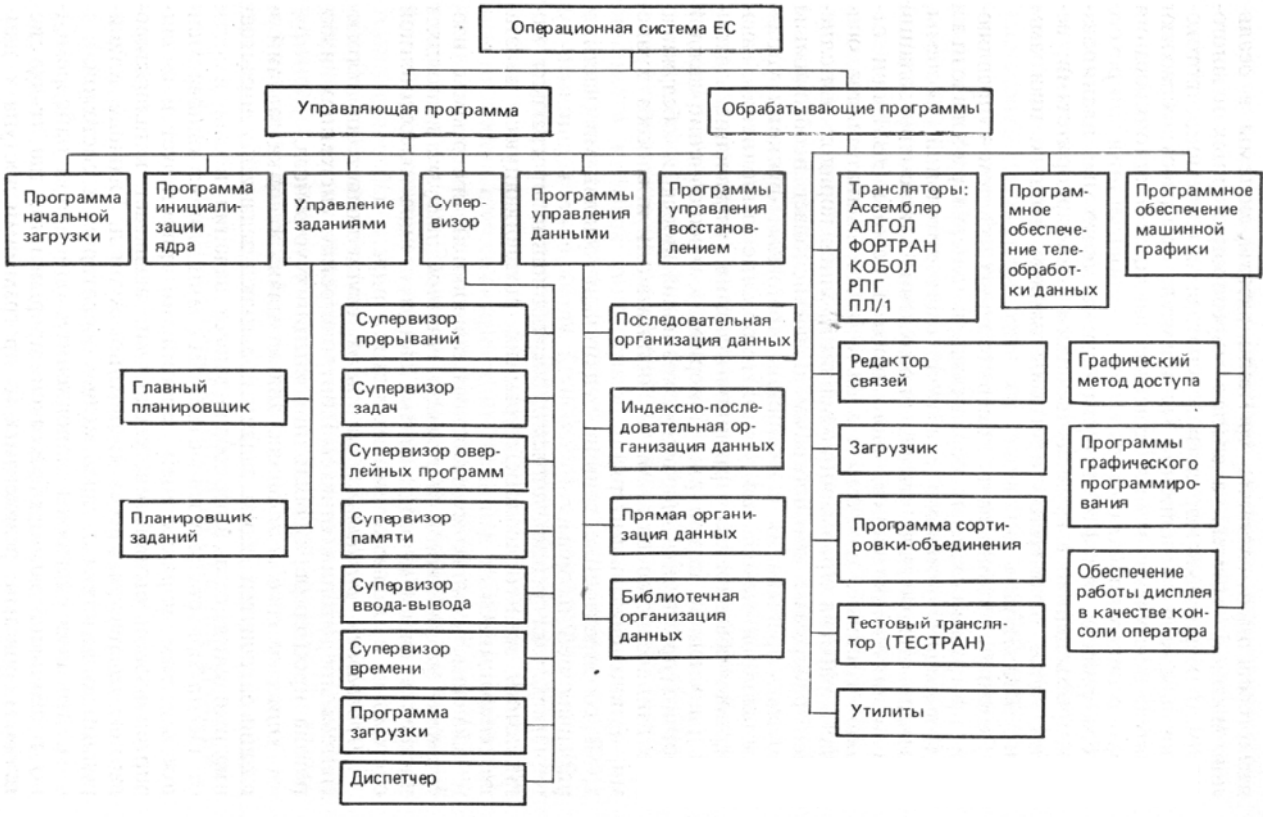


Рис. 2.1. Структура операционной системы ОС ЕС

Программа инициализации ядра осуществляет подготовку системных управляющих таблиц, настройку системных наборов данных, проверку состояния внешних устройств.

Управление заданиями состоит из планировщика заданий и главного планировщика. Планировщик заданий считывает входные потоки заданий; обрабатывает задания в зависимости от их приоритета и режима ОС ЕС; инициирует одновременное выполнение нескольких заданий; интерпретирует и анализирует языки управления заданиями; вызывает каталогизированные процедуры; ведет системный журнал. Главный планировщик осуществляет связь между оператором и системой.

Супервизор управляет выполнением одной или одновременно нескольких задач; обработкой прерываний; памятью; системными часами; резидентными программами доступа к данным; средствами защиты от несанкционированного доступа; совмещением работы каналов и процессора; работами программ с оверлейной структурой и системы в режимах квантования времени с пакетной обработкой; функционированием многопроцессорных комплексов; динамической загрузкой программ в оперативную память.

Супервизор состоит из программ, которые функционально можно объединить в группы, названные супервизорами прерываний, задач, оверлейных программ памяти, ввода-вывода, времени. Рассмотрим основные из этих программ. Супервизор прерываний обрабатывает шесть следующих типов прерываний: программных; при обращении к супервизору; внешних; ввода-вывода; повторного пуска; от схем контроля. Обработка включает следующие этапы: определения типа прерывания; сохранения информации, описывающей состояние вычислительной системы, для того чтобы ее можно было восстановить после обработки прерываний; передачи управления соответствующей программе обработки прерываний; определения (после обработки прерывания с помощью программы-диспетчера) задачи, которой передается управление. Супервизор оверлейных программ обеспечивает функционирование загрузочных модулей оверлейной структуры. Супервизор памяти распределяет оперативную память по запросам пользователей. Супервизор времени позволяет задавать и считывать текущие дату и время, организовывать интервалы квантования времени, в течение которых определение группы задач могут пользоваться центральным процессором.

Программы управления данными обеспечивают способы организации, идентификации, хранения, каталогизации и выборки обрабатываемых данных. Эти программы управляют вводом и выводом данных с последовательной, индексно-последовательной, библиотечной и прямой организацией, объединением записей в блоки и разделением блоков на записи, обработкой системных и пользовательских меток томов и наборов данных.

Программы управления восстановлением после сбоя обрабатывают прерывания от схем контроля машины, регистрируют сбои в процессоре, каналах и внешних устройствах, формируют записи о сбое в журнале, анализируют возможность завершения затронутой сбоем задачи и переводят систему в состояние ожидания, если завершение задачи невозможно.

Обрабатывающие системные программы под руководством управляющей программы непосредственно выполняют задания пользователей. Обрабатывающие программы сокращают усилия и время, затрачиваемые на написание, подготовку и выполнение проблемных программ пользователей. К ним относятся следующие программы:

1. Трансляторы, предназначенные для перевода текста исходных программ, записанных на алгоритмических языках высокого уровня, на машинный язык, а также для формирования листингов с текстами исходных программ и с диагностическими сообщениями о лексических, синтаксических и семантических ошибках в программе. В состав трансляторов ОС ЕС входят трансляторы с языков РПГ, Ассемблер, ФОРТРАН, ПЛ/1, КОБОЛ, АЛГОЛ.

2. Редактор связей, объединяющий отдельно оттраслированные модули в один готовый к выполнению модуль, резервирующий память для общих областей, создаваемых трансляторами, заменяющий, исключающий и перемещающий программные секции и формирующий дополнительную диагностическую информацию.

3. Загрузчик, осуществляющий редактирование и загрузку отредактированных, готовых к выполнению модулей.

4. Сортировки-объединения, объединяющие файлы, размещенные на внешних носителях (магнитных лентах, дисках, барабанах), с входными файлами, размещающие наборы данных в заданном порядке, сортирующие записи, мультитомный и мультифайловый ввод-вывод данных.

5. Тестовый транслятор (ТЕСТРАН), осуществляющий отладку программ на языке Ассемблер, редактирование и вывод на печать отладочной информации.

6. Утилиты, выполняющие различные функции, связанные с обслуживанием пользователей (разметка томов, обслуживание библиотек и пр.).

Программы телеобработки данных обеспечивают передачу данных по каналам связи (телефонным, телеграфным и пр.), что позволяет осуществить одновременную работу ЭВМ с несколькими удаленными АП. Программы телеобработки данных включают два телекоммуникационных метода доступа (базисный и с очередями), которые применяются при работе системы в режимах удаленной пакетной обработки, разделения времени, диалоговом, работы в реальном масштабе времени и т. д.

Программное обеспечение машинной графики включает графический метод доступа и осуществляет функционирование различных графических устройств (графопостроителей, графических дисплеев и т. д.).

§ 2.3. МУЛЬТИПРОГРАММИРОВАНИЕ И РАЗДЕЛЕНИЕ ВРЕМЕНИ

Все современные версии ОС ЕС работают в мультипрограммном режиме. В этом режиме программы управления задачами должны распределять ресурсы между одновременно выполняемыми задачами, следить за их использованием и освобождать ресурсы после окончания задачи. Если несколько задач дожидаются одного и того же устройства, управляющая программа организует очередь запросов. При работе в мультипрограммном режиме общее время обработки сокращается. Применение динамических структур для решения сложных задач позволяет различным сегментам программы совместно использовать системные ресурсы и таким образом оптимизировать их эксплуатацию. Мультипрограммный режим позволяет параллельно осуществлять несколько пунктов, принадлежащих различным заданиям. Выполняемый пункт может быть связан с последующими пунктами того же самого задания. Пункт задания не может передавать данные последующему пункту путем сохранения их в оперативной памяти. В условиях мультипрограммирования возможно независимое друг от друга выполнение заданий.

Режим мультипрограммирования с фиксированным числом задач (MFT). Этот режим обеспечивает одновременную работу фиксированного числа заданий (не более 15). Количество заданий определяется в зависимости от статического распределения памяти, которое задается оператором или устанавливается при генерации системы. В режиме мультипрограммирования с фиксированным числом задач память делится на системную и динамическую области. В *системной области* находятся программы режима MFT. *Динамическая область* подразделяется на 52 дискретные области, называемые *разделами.* Пятнадцать разделов предназначены для выполнения заданий пользователей, а остальные - для системных задач. Режим программирования с переменным числом задач бывает двух видов: без подзадач и с подзадачами.

Режим без подзадач характеризуется тем, что для каждой выполняемой программы пользователя образуется только одна подзадача. Внутри нее процесс выполнения нельзя распараллелить, и все модули осуществляются последовательно. Задачи, образованные в разных разделах для разных заданий, выполняются одновременно.

Режим с подзадачами характеризуется тем, что допускается распараллеливание программы. Это происходит за счет того, что любая задача (подзадача) может породить одну или несколько подзадач, которые могут обрабатывать данные одновременно друг с другом и с породившей их задачей, а также с задачами других разделов. Общее количество всех одновременно выполняемых задач и подзадач не должно превышать 255. В это число входят как задачи и подзадачи пользователей, так и системные задачи и подзадачи. Максимальное число задач и подзадач, исполняемых в рамках одного раздела, составляют 250, так как пять задач используются управляющей программой.

Режимы без подзадач и с подзадачами отличаются способом определения приоритетов задач. В режиме без подзадач приоритет присваивается разделам памяти, причем считается, что приоритет раздела тем выше, чем старше начальный адрес памяти, в которой расположен раздел. В режиме с подзадачами приоритет присваивается каждой задаче и подзадаче и имеет значение в диапазоне от 0 до 255. Приоритет подзадач можно изменить относительно порождающей задачи в момент их образования, а кроме того, приоритеты могут динамически изменяться в процессе работы.

Возможность образования подзадач в режиме мультипрограммирования с фиксированным числом задач задается при генерации операционной системы.

При мультипрограммном режиме с фиксированным числом задач необходимо уделять внимание способу, которым пользователь определяет различные типы заданий и направляет задания в соответствующие разделы. Например, задания, много использующие центральный процессор, могут быть направлены в разделы низкого приоритета, чтобы они не препятствовали выполнению заданий, которые мало загружают ЦП. Телекоммуникационные задания могут быть направлены в разделы высокого приоритета с целью уменьшения времени ответа системы пользователю терминала.

Режимы мультипрограммирования с переменным числом задач (MVT, SVS). Этот режим предоставляет те же преимущества, что и режим с фиксированным числом задач. Существенное различие между режимами заключается в способе использования памяти. Кроме того, режим мультипрограммирования с переменным числом задач предоставляет ряд дополнительных возможностей. В режиме MVT определенное количество основной памяти резервируется для управляющей программы (системная область). Остальная память (динамическая область) доступна программам пользователей. Она распределяется между разделами динамически в процессе выполнения заданий; это существенное отличие от режима MFT.

В режиме с переменным числом задач производится чтение одного или нескольких входных потоков и планирование заданий, классифицируемых по входным очередям, в соответствии с приоритетами. Каждое вызванное задание оперирует внутри зоны динамически выделяемых разделов. Одновременно может выполняться до 15 заданий. Пункты одного и того же задания исполняются последовательно, поскольку один пункт может зависеть от успешного завершения другого. Задача, соответствующая пункту задания, и ее подзадачи решаются независимо внутри одного раздела и используют один и тот же ключ защиты памяти. Для завершения пункта задания необходимо завершение всех его задач.

Средство свертывания заданий характерно только для режима мультипрограммирования с переменным числом задач.

Это средство позволяет осуществлять временное динамическое распространение конкретного задания за пределы первоначально отведенного ему раздела. Когда у некоторого задания появляется потребность в получении дополнительной памяти, управляющая программа с помощью средства свертывания заданий пытается предоставить память из нераспределенного участка. Если нераспределенного участка в памяти нет, то задание свертывается, т. е. переводится во вспомогательную память, так что его раздел может применяться первым заданием. Когда первое задание освобождает эту дополнительную память, она возвращается тому источнику, откуда была получена, т. е. либо в область нераспределенной памяти, либо заданию, переведенному во вспомогательную область памяти. Для этого свернутое задание вновь переводится в основную память (развертывается) и может выполняться дальше.

Управление задачами при мультипрограммировании. Основная управляющая информация по управлению задачами содержится в блоках управления задачами и связанных с ними блоках и элементах запроса.

Блок управления задачей (TCB) является основным блоком, создаваемым управляющей программой. В TCB содержатся адреса других управляющих блоков системы. Используя их, можно получить информацию о том, какие загрузочные модули были запрошены, какие наборы данных являются открытыми, какие устройства ввода-вывода назначены для данной задачи. Подлежащая выполнению программа может состоять из нескольких загрузочных модулей, которые обычно загружаются не все сразу, а запрашиваются по мере необходимости. Эта возможность требует введения для каждого загрузочного модуля специального управляющего блока запросов.

В режиме мультипрограммирования с переменным числом задач без подзадач одна задача может быть образована в каждом разделе динамической области памяти. Кроме того, в системной области памяти находятся задачи загрузки транзитной области, связи, главного планировщика. Для каждой задачи при ее образовании создается свой TCB. Все TCB связываются, согласно своим приоритетам, в очередь, которая начинается с TCB трех резидентных задач. Общее число задач не может превышать 52. Так как в любой момент времени в системе существует более одной задачи, в режиме MFT вводится понятие переключения задач. Относительный приоритет задач определяется очередью TCB. Управление часто передается от одной задачи к другой с более высоким приоритетом.

В режиме MFT с подзадачами число задач в одном разделе оперативной памяти, а следовательно, и во всей системе динамически изменяется. Все TCB пользовательских задач ставятся в очередь после TCB системных задач. Блоки управления задачей подзадач ставятся в очередь к TCB соответствующей задачи с учетом их приоритетов. Эти приоритеты могут быть изменены во время выполнения задачи с помощью специальной макрокоманды.

Блоки запросов (RB) создаются для описания системных и проблемных загрузочных модулей. Управляющая программа создает семь типов блоков RB. Тип создаваемого блока запросов зависит от описываемого загрузочного модуля. С одной задачей может быть связано несколько блоков RB, которые образуют очередь. Просматривая очередь блоков RB, можно определить: какие загрузочные модули были выполнены, по какой причине прекратилось выполнение каждого из них, какой из модулей является источником ошибок. В режиме MFT без подзадач система создает две очереди: RB активных программ и RB загруженных программ. В режиме MFT с подзадачами также создается очередь RB в общей области задания.

Очередь RB *активных программ* представляет собой цепочку блоков запросов соответствующих активных загрузочных модулей. Когда модуль выдает запрос на загрузку другого модуля, создается новый RB. Информация о выполняющемся модуле запоминается в соответствующем блоке RB. При этом меняется поле TCB, определяющее адрес RB текущего модуля, и управление передается вновь загруженному модулю. По окончании его работы RB исполненного модуля удаляется из очереди и обновляются соответствующие ссылки.

Очередь RB *загруженных программ* отличается от очереди RB активных программ тем, что RB этой очереди и соответствующие им загрузочные модули не удаляются автоматически. Для их удаления необходимо применение специальной макрокоманды, иначе они будут удалены лишь по окончании шага задания. Построение очереди RB загруженных программ зависит от режима работы управляющей программы ОС.

В режиме MFT без подзадач каждый RB содержит ссылки на предыдущий и последующий RB.

Очередь блоков RB *к общей области задания* образуется из RB, созданных для реентерабельных модулей в режиме MFT с подзадачами. Блоки запросов не удаляются автоматически из очереди к общей области задания, а остаются в ней до своего удаления посредством специальной макрокоманды. Поэтому реентерабельные загрузочные модули остаются в памяти и их может использовать любая задача или подзадача шага задания.

Рассмотренные аспекты управления задачами – общие для всех режимов работы управляющей программы. Однако имеются некоторые отличия в управлении задачами, зависящие от числа задач, которыми одновременно управляет ОС, и от распределения основной памяти.

Режимы обработки запросов пользователей. На базе мультипрограммных режимов работы управляющей программы операционная система позволяет осуществить такие основные режимы обработки запросов пользователей, как пакетная обработка, удаленная пакетная обработка, режимы разделения времени.

В режиме пакетной обработки задания обрабатываются непрерывно с автоматическим переходом от одного задания к другому. Задания после ввода в систему образуют входные очереди, размещаемые обычно на диске. Выбор заданий из очереди на обработку производится последовательно или на основе приоритетов. Результаты выполнения заданий записываются на диск и образуют выходные очереди. Из выходной очереди задания выводятся на системные устройства вывода (АЦПУ, перфораторы, магнитные ленты, дисплеи, устройства графического вывода и т. д.) либо последовательно, "либо на основе приоритетов. Допускается также непосредственный вывод на устройство системного вывода в процессе обработки задания.

В режиме удаленной пакетной обработки ввод заданий и вывод результатов производится на удаленный абонентский пункт, соединенный с ЭВМ посредством канала связи.

В режиме разделения времени в оперативной памяти выделяется один или несколько разделов памяти, каждый из которых предназначен для обслуживания одного или нескольких заданий. Любое из этих заданий получает управление на период времени, называемый *квантом.* При этом копии всех других заданий, выполняемых в режиме разделений времени, сохраняются во внешней памяти. После истечения кванта времени управление получает следующее квантируемое задание, а предыдущее вытесняется на внешнюю память. Величина кванта времени и признак принадлежности заданий к группе квантования устанавливаются при

генерации ОС. Всем заданиям, входящим в группу квантования, присваивается определенный приоритет.

На базе мультипрограммного режима с переменным числом задач может функционировать система разделения времени для обслуживания более 100 одновременно работающих пользователей терминалов.

Одновременно с программами режима разделения времени могут выполняться фоновые задания в режиме пакетной обработки данных. При этом программы пакетной обработки запускаются на исполнение лишь в том случае, если все программы режима разделения времени находятся в состоянии ожидания. Между пакетной обработкой и режимом разделения времени обеспечивается совместимость, позволяющая одну и ту же программу запускать как в режиме пакетной обработки, так и в режиме разделения времени.

§ 2.4. ГЕНЕРАЦИЯ ОПЕРАЦИОННОЙ СИСТЕМЫ

Сложные программные комплексы, например операционные системы, обычно делают универсальными, ориентированными на широкий класс различных применений. Каждый универсальный компонент системы программного обеспечения должен быть настроен на условия конкретного применения. Процесс подобной настройки программного комплекса называется *генерацией.*

Генерация операционной системы. Характерным примером настройки на условия конкретного применения является генерация ОС ЕС. Для генерации ОС ЕС необходимо уточнить условия функционирования вычислительной системы, т. е. выяснить общий объем и типы выполняемых работ, их приоритет и т. п.

Возможность выбора конфигурации ОС ЕС обеспечивается ее модульной структурой. В ОС одни и те же функции могут выполняться разными модулями. Например, один обладает высоким быстродействием, но нуждается в значительном объеме памяти, другой может поместиться в небольшом объеме памяти, но требует больших затрат машинного времени.

Выбранная конфигурация технических средств и средств ОС ЕС описывается с помощью специального языка (макрокоманд генерации), после чего осуществляется собственно генерация, которая является обычным заданием ОС ЕС, выполняемым под управлением ОС. Принцип генерации использован не только во всех операционных системах ЕС ЭВМ, но и в ряде универсальных ППП, расширяющих возможности ОС и имеющих сложную логическую структуру. Язык специальных макрокоманд генерации операционной системы представляет собой набор инструкций, порождающий пакет заданий для формирования конкретных вариантов ОС. Применение укрупненных макроопераций упрощает процесс описания условий обработки информации на каждой конкретной ЭВМ и автоматизирует компоновку подходящего конкретного варианта ОС. Генерация операционной системы может производиться не только на стартовой операционной системе, но на любой работающей ОС, сгенерированной ранее. Макрокоманды генерации, порядок проведения генерации и описание дополнительных возможностей приводятся в соответствующих руководствах по операционной системе. Сгенерированная операционная система размещается на одном или нескольких пакетах магнитных дисков, указанных при генерации.

Чтобы начать работу с операционной системой, необходимо загрузить ее ядро в оперативную память. *Ядром* операционной системы называется часть управляющей программы, загружаемая в фиксированную область основной памяти во время процедуры начальной загрузки, выполнение которой никогда не перекрывается другими частями ОС или программами пользователей. При загрузке ядра обязательно должен быть установлен пакет магнитных дисков, называемый *резидентным томом системы* и содержащий программу начальной загрузки, собственно ядро, системный каталог и другие компоненты сгенерированной операционной системы. Адрес данного устройства набирается при помощи переключателей на пульте ЭВМ, после чего нажимают кнопку начальной загрузки. После этого последовательно выполняются программа начальной загрузки и программа инициализации ядра ОС.

Программа начальной загрузки производит поиск на резидентном томе набора данных с именем SYS1. NUCLEUS, в котором записано ядро, и загружает его в оперативную память. Далее управление получает программа инициализации, которая находится в ядре операционной системы.

Программа инициализации выдает оператору специализированные запросы, на основе которых оператор может изменить значения некоторых системных параметров, список резидентных программ и данных. В список резидентных программ входят программы методов доступа, обработки обращений к супервизору, элементы библиотек и т. д. Расширение списка резидентных программ и данных увеличивает производительность системы за счет отсутствия времени, затрачиваемого на их поиск и загрузку из внешней памяти. Однако это вызывает увеличение объема используемой оперативной памяти. Во время инициализации операционной системы, обеспечивающей режим мультипрограммирования с фиксированным числом задач, оператор может переопределять количество разделов памяти, их размеры и классы заданий для каждого раздела.

По окончании начальной загрузки производится инициализация ядра операционной системы, в процессе которой подготавливаются и проверяются состояния системных таблиц и внешних устройств, настраиваются системные наборы данных и инициализируется основная память, используемая для выполнения задач пользователя и некоторых системных задач. После завершения загрузки системы и выдачи сообщения о ее готовности к работе управление передается программам "Планировщик заданий" и "Главный планировщик", которые обеспечивают управление потоком заданий, поступающих для обработки на ЭВМ.

Глава 3. УПРАВЛЯЮЩАЯ ПРОГРАММА ОПЕРАЦИОННОЙ СИСТЕМЫ ЕС ЭВМ

В главе описываются функции одной из основных программ системного программного обеспечения ЭВМ третьего поколения - управляющей программы. Изложение ведется применительно к управляющей программе, входящей в состав операционной системы ЕС ЭВМ. Рассматриваются следующие многопрограммные режимы функционирования: с фиксированным числом задач, с переменным числом задач и с виртуальной памятью. Такие режимы позволяют благодаря мультипрограммированию осуществлять разделение времени центрального процессора, пространства оперативной памяти и других ресурсов ЭВМ между несколькими обрабатывающими программами пользователей.

§ 3.1. СТРУКТУРА УПРАВЛЯЮЩЕЙ ПРОГРАММЫ

Управляющая программа представляет собой большой комплекс программ, составляющий основу операционной системы. Управляющая программа автоматизирует процесс управления вычислительной системой и реализует функции управления заданиями, задачами, данными.

Задание – основная единица работы, выполняемой вычислительной системой. Каждое задание описывается пользователем на специальном языке управления заданиями (ЯУЗ) (Job Control Language). Задание состоит из одной или нескольких логических частей, называемых *шагами задания* (пункт задания). *Шаг задания* – часть задания, связанная с выполнением одной обрабатывающей программы, т. е. это последовательность операторов ЯУЗ, подготавливающих ОС к исполнению программы. Задания, как правило, не зависят друг от друга и не влияют друг на друга. Следовательно, задание – это внешняя единица работы для ОС.

Программы управления заданиями планируют прохождение потока заданий через ОС и состоят из двух компонентов: главного планировщика и планировщика заданий. Главный планировщик обеспечивает двустороннюю связь между оператором ЭВМ и операционной системой. *Планировщик заданий* осуществляет ввод заданий в систему, вывод результатов заданий на внешние устройства, интерпретацию входного потока заданий, распределение устройств ввода-вывода, инициализацию и завершение выполнения шагов заданий (рис. 3.1).

Рис. 3.1. Схема связи пользователя с вычислительной системой

Программа управления заданиями создает для каждого шага задания задачу. *Задача* – внутренняя единица работы операционной системы. Задача может образовывать новые задачи – подзадачи, т. е. в рамках одного шага задания может быть образовано несколько задач.

Программы управления задачами осуществляют создание–уничтожение задач, распределение ресурсов вычислительной системы между задачами (диспетчерирование). Основу программных средств управления задачами представляет супервизор, который контролирует выполнение каждой задачи и включается в работу по сигналам прерывания. Основное назначение супервизора – динамическое распределение ресурсов ОС между задачами во время их выполнения. К ресурсам относятся время центрального процессора, оперативная память, другие системные ресурсы, за исключением устройств ввода-вывода.

Программы управления данными управляют устройствами ввода-вывода, хранят и извлекают данные из внешней памяти, используют каналы, предоставляют пользователю методы доступа к данным.

§ 3.2. РЕЖИМЫ РАБОТЫ УПРАВЛЯЮЩЕЙ ПРОГРАММЫ

Операционная система ЕС ЭВМ ориентирована на пакетную обработку заданий, при которой пользователь не имеет непосредственной связи с вычислительной системой. Задания сдаются оператору ЭВМ, формирующему входные пакеты заданий и организующему их выполнение (прохождение через ОС). Время ответа системы – время полного выполнения задания; обычно измеряется часами или минутами.

Для организации работы системы в режиме разделения времени, в котором пользователь получает все сообщения при исполнении задания и может вмешиваться в процесс обработки, необходимы дополнительные программные средства, такие, как монитор телеобработки данных.

При генерации ОС создается один из трех возможных вариантов управляющей программы, соответствующий одному из трех основных режимов работы ОС ЕС:

1) мультипрограммный режим с фиксированным числом задач (режим MFT);
2) мультипрограммный режим с переменным числом задач (режим MVT);
3) мультипрограммный режим с виртуальной памятью (режим SVS).

Режимы используют мультипрограммирование; это значит, что ОС обеспечивает одновременное нахождение в оперативной памяти нескольких программ пользователей и разделение времени процессора, пространства оперативной памяти и других ресурсов между активными задачами. Мультипрограммирование позволяет эффективно применять технические и программные ресурсы вычислительной системы. При выполнении каждой задачи возникают паузы, связанные с ожиданием завершения операции ввода-вывода, освобождения ресурса и т. п. В это время может исполняться задача другого задания. В любой момент времени управление принадлежит только одной программе.

Ядро ОС (управляющие программы) всегда находится в некоторой фиксированной области оперативной памяти. Остальная часть памяти (динамическая область) распределяется между активными задачами. Каждый из режимов работы управляющей программы связан с определенным способом распределения памяти между задачами (управления памятью).

Режим MFT. В этом режиме фиксируется количество, размер и размещение разделов, на которые делится динамическая область памяти. Каждый раздел предназначен для одного задания. Всего может быть создано не более 16 разделов, что связано с наличием 16 различных ключей защиты памяти. Из 16 разделов только 15 доступны для пользователя, один раздел отводится под системные задачи.

Средний размер раздела определяется количеством разделов (1...15) и размером оперативной памяти. Обычно режим MFT используется на ЭВМ с объемом ОП 128...512 К байт. Число и размеры разделов определяются при генерации ОС, но при необходимости могут быть изменены оператором ЭВМ.

Каждый раздел может иметь свой входной поток заданий, соответствующий заданным классам входной очереди заданий. Разделы не равноправны с точки зрения диспетчерского приоритета. Диспетчерский приоритет определяет порядок, в котором выполняемая в разделе задача получает ресурсы ОС. Наибольший приоритет имеет раздел P0, наименьший – Pn. Тем самым устанавливается определенное соответствие между классами заданий и их приоритетами.

Режим MFT с подзадачами отличается от стандартного режима MFT без подзадач возможностью обрабатывать в каждом разделе более одной задачи – задачи шага задания и порожденных ею подзадач. Общее число задач и подзадач не должно превышать 255. Подзадачи так же претендуют на ресурсы вычислительной системы, как и задачи. Приоритет подзадачи

может не совпадать с приоритетом породившей ее задачи. Эффективное использование ресурсов в режиме MFT зависит от того, насколько правильно распределены задания по классам.

Недостаток режима MFT – границы разделов нельзя менять при исполнении задания, поэтому часть операционной памяти в каждом разделе остается свободной, но не может быть использована.

Режим MVT. Основное отличие этого режима от режима MFT заключается в том, что границы раздела становятся подвижными. Это означает, что размер формируемого для задания раздела (зоны ОП) точно соответствует запрошенному объему памяти. В данный момент времени в памяти размещаются все задачи, которые могут в ней поместиться, но не более 15 проблемных задач. Память динамически распределяется не только между заданиями, но и между шагами заданий. После выполнения задания или его шага выделенная ему память освобождается.

Если пользователь определил необходимый размер памяти в операторе описания задания (оператор JOB), то указанный объем памяти будет выделен всем шагам задания. В противном случае каждому шагу выделяется столько памяти, сколько указано в операторе описания данного шага (оператор EXEC).

Стратегия выделения памяти задачам в режиме MVT часто приводит к нежелательной фрагментации памяти. Допустим, что в системе выполняется четыре задания (рис. 3.2,*а*) и задание *5,* требующее для исполнения 36 К байт ОП, ждет освобождения памяти. После завершения задания *1* (рис. 3.2,*б*) запуск задания *5* все еще невозможен, так как требуется непрерывный участок памяти размером 36 К байт. Задание *5* начнет исполняться только после завершения задания *2* (рис. 3.2,0, г), причем останется свободным участок ОП размером 86 К байт.

Использование режима MVT возможно только на ЭВМ с объемом оперативной памяти не менее 256 К байт. Создание подзадач является стандартной функцией управляющей программы в режиме MVT. Приоритет задания задается пользователем в карте JOB и может быть уменьшен при запуске задания в зависимости от класса входной очереди заданий.

Режим SVS. *Физическая память ЭВМ* - фиксированное число упорядоченных ячеек. Обращение к каждой ячейке осуществляется с помощью уникального номера, называемого адресом. Набор физических ячеек памяти постоянен и заранее определен для данной ЭВМ. *Математическая память* – набор логических имен в программе и математических адресов в командах оттранслированной программы. Для исполнения программы необходимо произвести отображение математических адресов на физические ячейки. Такое отображение выполняется средствами ОС. Операционная система осуществляет загрузку программы в один из непрерывных разделов физической памяти. Схематично отображение математической памяти на физическую показано на рис. 3.3. В системах, не применяющих виртуальную память, математический адрес

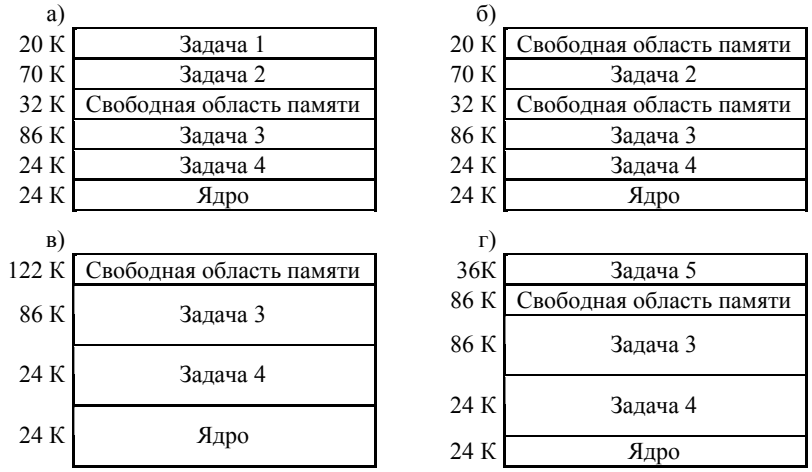


Рис. 3.2. Пример распределения памяти в режиме MVT

теряется, и при исполнении программы доступ происходит непосредственно к физическим ячейкам.

В системах с виртуальной памятью математический (виртуальный) адрес сохраняется во время исполнения. Это позволяет, во-первых, использовать для программы несмежные области физической памяти и, во-вторых, расширять объем математической памяти за пределы размера оперативной памяти путем применения дополнительной памяти на внешних устройствах прямого

доступа (магнитных дисках).

В ОС ЕС используется сегментно-страничная организация виртуальной памяти. Физическая память состоит из двух уровней: оперативной памяти и памяти на магнитных дисках. Математическая (виртуальная) память представляет собой последовательно адресуемое пространство, разделенное на сегменты, состоящие из страниц. Размер математической памяти ограничен только разрядностью адреса конкретной ЭВМ; для 24-разрядного адреса ЕС ЭВМ это составляет 16 М байт. Размер сегмента 64 К, размер страницы 2 К; следовательно, один сегмент насчитывает 32 страницы. Оперативная память делится на такие же страницы объемом 2048 байт (2 К), называемые блоками памяти.

Физический адрес оперативной памяти состоит из двух элементов: номера блока *s* и смещения *d* (относительного адреса) в блоке. Математический (виртуальный) адрес состоит из трех элементов: номера сегмента *s*; номера страницы *p*; смещения в странице *i* (индекса). Таким образом, смещение *d* в математическом сегменте представлено парой чисел (*p*, *i*). Как правило, размер оперативной памяти существенно меньше размера виртуальной памяти. Поэтому большая

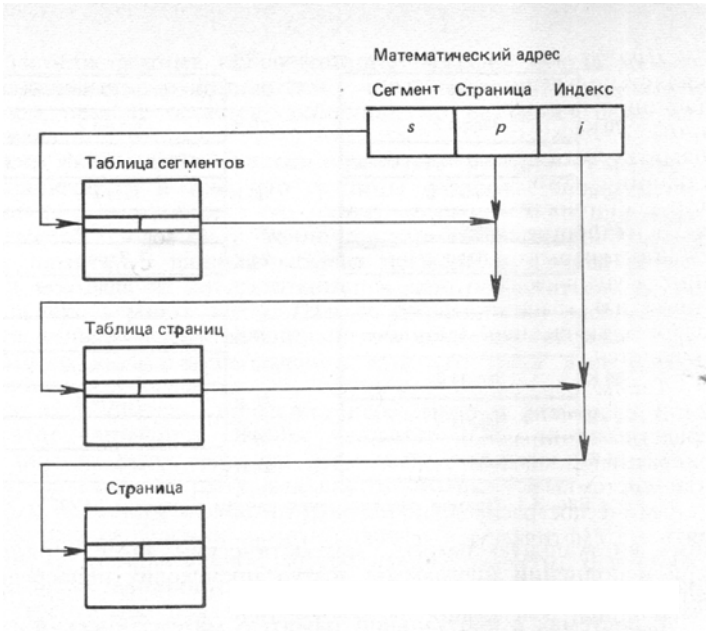


Рис. 3.3. Схема переадресации в системе SVS

часть виртуальных страниц расположена в виде страничного набора данных на диске. Для соотнесения математических и физических адресов используются таблицы сегментов и страниц.

Таблица сегментов содержит для каждого сегмента длину и адрес соответствующей таблицы страниц и индикатор доступности сегмента *I*. Если сегмент загружен в оперативную память, то он считается доступным (*I*=0).

Таблицы страниц строятся для каждого сегмента математической памяти. В таблице страниц для каждой математической страницы указан ее физический адрес, т. е. номер блока, содержащего данную страницу, а также индикатор доступности страницы.

Для выполнения любой команды, использующей обращение к оперативной памяти, необходимо обращаться к следующим элементам памяти: таблице сегментов; таблице страниц; странице. Для указанного трехэтапного процесса нужна аппаратная поддержка, так как в противном случае время доступа к оперативной памяти окажется слишком большим. *Аппаратные средства,* обеспечивающие режим SVS, существуют только на ЕС ЭВМ второй и третьей очереди. Основу аппаратных средств составляет блок динамической переадресации, осуществляющий трехэтапное преобразование адресов.

Программным средством управления виртуальной памятью является супервизор страниц, создающий и поддерживающий таблицы переадресации (таблицу сегментов и таблицы страниц), а также осуществления загрузку страниц в оперативную память. Если в команде содержится обращение к виртуальной странице, которая отсутствует в оперативной памяти (индикатор доступности страницы в таблице страниц равен 1), то происходит программное прерывание. Супервизор ОС обрабатывает это прерывание и передает управление супервизору страниц. Супервизор страниц организует подкачку требуемой страницы из страничного набора данных на

диске в оперативную память, после чего выполнение прерванной программы возобновляется.

Таким образом, режим SVS реализуется на ЕС ЭВМ Ряда 2 и 3 с помощью специальных программных и аппаратных средств. Минимально возможный для этого режима объем оперативной памяти 512 К. Функционирование вычислительной системы в режиме SVS позволяет эффективно проводить динамическое распределение памяти между задачами и расширять математическую память за пределы оперативной памяти. Работа в этом режиме целесообразна только для высокопроизводительных ЭВМ, так как выполнение программ значительно замедляется вследствие преобразования адресов и частых обращений к дискам. С точки зрения программиста режим SVS – это режим MVT с неограниченным количеством заданий и памятью до 16 М байт.

§ 3.3. СХЕМА ПРОХОЖДЕНИЯ ЗАДАНИЙ В ОПЕРАЦИОННОЙ СИСТЕМЕ

Прохождением заданий в вычислительной системе управляют программы планировщика заданий. Основными компонентами планировщика заданий являются следующие программы: системного ввода – чтение-интерпретация (RDR); инициализации/завершения задания – инициатор-терминатор (I/T); системного вывода (WTR).

Стадии прохождения заданий. Компоненты планировщика заданий отвечают трем стадиям прохождения заданий в ОС ЕС (рис. 3.4).

Первая стадия заключается в том, что программа системного ввода последовательно читает и интерпретирует операторы задания и помещает задание во входную очередь заданий (на томе прямого доступа).

Вторая стадия характеризуется тем, что программы инициализации/завершения организуют выполнение каждого

Рис. 3.4. Схема прохождения задания в ОС ЕС

шага задания и его завершение. По окончании выполнения задания выходная информация поступает в очередь выходных работ.

Третья стадия состоит в том, что программа системного ввода осуществляет вывод результатов исполнения задания из очереди выходных работ на внешнее устройство (АЦПУ).

Задание записывается на специальном языке управления заданиями. Обязательными являются следующие операторы описания: задания JOB; шага задания EXEC; данных DD. Каждое задание должно начинаться оператором JOB, в котором содержится информация о задании в целом. Оператор EXEC определяет шаг задания, указывает программу, которая должна быть выполнена на данном шаге. Операторы описания данных DD для данного шага задания должны располагаться после соответствующего оператора EXEC. В операторе DD содержатся информация о наборе данных и запрос на устройство ввода-вывода для этого набора данных.

Функции программы системного ввода. Для ввода заданий создается специальная задача системного ввода. Эта задача может обработать только один входной поток заданий, представляющий собой набор перфокарт описаний заданий или их образов на магнитных носителях.

Программа чтения организует покарточный ввод управляющих операторов задания.

Программа интерпретации, работающая параллельно с ней, анализирует синтаксис считанных операторов и формирует на их основе управляющую информацию в виде блоков управления файлом задания JFCB и таблиц: управления заданием JCT, управления шагом SCT, ввода-вывода для шага задания SIOT (рис. 3.5). После считывания всего задания эта

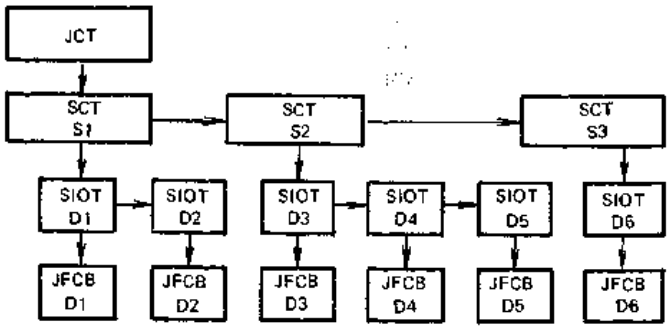


Рис. 3.5. Структура элемента очереди входных работ 64

управляющая информация помещается в *системную очередь ввода* (на том прямого доступа). Доступ ко всей информации, относящейся к одному заданию, осуществляется через таблицу управления заданием JCT. Эта таблица формируется на основе информации, содержащейся в операторе задания JOB. В таблице JCT есть указатель на список таблиц управления шагом задания SCT. С каждой такой таблицей связан список таблиц ввода-вывода для шага задания SIOT. В таблицах SIOT имеется указатель на соответствующий блок управления файлом заданий JFCB. Информация в каждую таблицу SCT поступает из соответствующего оператора шага задания EXEC, а в таблицу SIOT и блок JFCB - из соответствующего описания данных DD. Список таблиц SCT, связанные с ним списки таблиц SIOT и блоки JFCB образуют *элемент очереди входных работ* (входной очереди).

После чтения и интерпретации последнего оператора задания управление передается программе управления входной очередью заданий. Признаком того, что задание во входном потоке закончилось, является поступление следующего оператора JOB или пустого оператора. Программа управления входной очередью переписывает полученную управляющую информацию (рис. 3.5) во входную очередь, представляющую собой специально организованный набор данных (SYS.l SYSJOBQE) на системном диске. Задание помещается в очередь определенного класса задания, а в очереди заданий одного класса задания располагаются в соответствии с их приоритетами. Класс и приоритет задания указываются в операторе JOB (и соответственно в таблице JCT).

В системе может быть образовано несколько задач системного ввода, каждая для определенного потока заданий. По завершении чтения-интерпретации последнего в потоке задания задача системного ввода завершается.

Функции программы инициирования и завершения задания. Программа инициатора-терминатора выполняет следующие функции: выборку задания из входной очереди; планирование задания по шагам; подготовку шага задания; завершение шага задания и всего задания. Первые три функции осуществляются программами, называемыми *инициатором;* функция завершения выполняется программой *терминатор.*

Программа инициатора-терминатора работает с входной и выходной очередью заданий. Свободный инициатор просматривает входные очереди заданий в том порядке, в каком были заданы классы очередей при запуске инициатора. Как только будет обнаружена непустая очередь, инициатор формирует для первого стоящего в очереди задания задачу инициатора. Если входная очередь пуста, то в специальную область набора данных входной очереди помещается адрес программы инициатора. Задача системного ввода (RDR) при постановке задания в очередь производит вызов (запуск) инициатора с указанного адреса. Таким образом, связь задач первой и второй стадий прохождения заданий осуществляется через входную очередь заданий.

Программы инициатора, последовательно планирующие шаг задания, осуществляют следующее: анализируют требования шага задания на устройства ввода-вывода и назначают шагу необходимые устройства; организуют получение зоны памяти для задания; выдают инструкции оператору (через программу главного планировщика) об установке томов и проверяют их установку на соответствующих устройствах ввода-вывода; распределяют место на диске для наборов данных, которые должны быть помещены в выходную очередь; строят исходную информацию для супервизора, т. е. дополнительные блоки и таблицы, оформляющие шаг задания как задачу.

После выделения шагу задания всех необходимых ресурсов инициатор обращается к

специальной программе супервизора ATTACH, которая для данного шага образует задачу. После этого задача инициатора переводится в состояние ожидания до завершения созданной задачи. При завершении задачи (нормальном или аварийном) снимается состояние ожидания с задачи инициатора и в зону задания загружается терминатор.

П р о г р а м м а т е р м и н а т о р осуществляет следующее: диспозицию наборов данных для шага задания (сохранить, уничтожить, передать следующему шагу и т. д.); освобождение устройств ввода-вывода, выделенных шагу; формирование кода завершения шага задания; помещение управляющей информации в запись очереди выходных работ.

Шаг задания завершается аварийно по команде оператора CANCEL или при невозможности выполнения программы (например, при наличии ошибки в программе). Завершение шага задания характеризуется специальным кодом возврата, который сохраняется для проверки при инициации следующих шагов.

Если завершается последний шаг задания или задание снимается с обработки, терминатор выполняет следующее: уничтожает запись о задании из очереди входных работ; завершает формирование записи очереди выходных работ и помещает ее в набор данных SYS1. SYSJOBQE; очищает области оперативной памяти от управляющих блоков и таблиц данного задания; передает управление программе выборки для чтения следующего задания.

Для работы с входной и выходной очередями терминатор обращается к программе управления очередями. Эта программа исключает элемент входной очереди, связанный с завершенным заданием, и включает задание в выходную очередь. Элемент выходной очереди начинает формироваться на первом этапе чтения - интерпретации задания и завершается на втором при завершении задания.

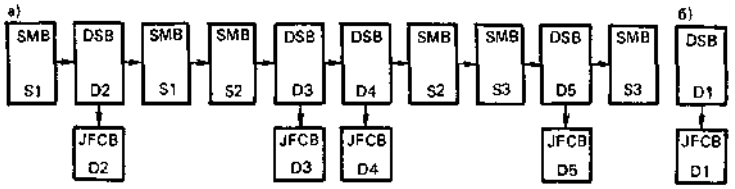


Рис. 3.6. Структура элементов выходных очередей классов А(а) и В(б)

Элемент выходной очереди состоит из списка блоков системных сообщений SMB и блоков наборов данных DSB для выходных наборов данных (рис. 3.6). Для каждого шага задания создается два системных блока для следующих сообщений программ: системного ввода и завершения шага задания. Между этими блоками могут размещаться блоки выходных наборов данных DSB, определенных в данном шаге задания. В каждом блоке DSB есть указатель на соответствующий блок управления файлом задания JFCB. Для одного задания может быть образовано несколько элементов, соответствующих выходным очередям различных классов. Класс системных сообщений указывается в операторе JOB в параметре MSGCLASS, класс выходного набора данных указывается в операторе DD в параметре SYSOUT и может не совпадать с классом системных сообщений.

Функции программы системного вывода. Программа системного вывода создается для управления выходными потоками данных. Эта программа организует вывод системных сообщений и выходных наборов данных из очереди выходных работ на устройства вывода (АЦПУ, НМЛ и др.). Каждая задача системного вывода обслуживает только одно устройство вывода, количество запускаемых программ в режимах MVT и SVS не ограничено. Одна программа системного вывода может обрабатывать до восьми классов выходных очередей.

Программа управления очередью отыскивает и выбирает элемент очереди выходных работ. Считанный элемент обрабатывается программой системного вывода в порядке следования блоков, в этом же порядке размещается выходная информация при выводе. По окончании вывода данных управление опять передается программе управления очередью. Она исключает из выходной очереди элемент завершенного задания и выбирает новый элемент. Если очередь выходных работ пуста, то задача системного вывода переводится в состояние ожидания.

Очереди. Эффективная работа и взаимодействие программ чтения-интерпретации, инициатора-терминатора и системного вывода осуществляется через специально организованный набор данных на томе прямого доступа SYS1. SYSJOBQE. Этот набор данных состоит из входных и выходных рабочих очередей.

Входная очередь данного класса характеризуется тем, что задания располагаются в порядке убывания приоритетов. Если последним поступило задание с высоким приоритетом, то указатели элементов очереди изменяются так, чтобы порядок приоритетов не был нарушен. Задания обрабатываются в порядке приоритетов, при равных приоритетах - в порядке их ввода (дисциплина FIFO).

Выходная очередь характеризуется тем, что ее элемент начинает формироваться при интерпретации задания и заканчивает при завершении задания. Выходные наборы данных заданий поступают в очередь того выходного класса, который указан в операторе DD. Наборы данных выходной очереди определенного класса выводятся на одно устройство вывода в порядке их поступления в очередь (FIFO). Одновременно может обслуживаться до 15 входных и до 36 выходных очередей.

Набор данных SYS1. SYSJOBQE состоит из областей управляющих записей и логических дорожек (рис. 3.7). *Область управляющих записей* содержит главную управляющую запись MQCR и по одной управляющей записи для каждой рабочей очереди. Записи строятся в виде 36-байтовых таблиц, в которые помещается информация об очередях.

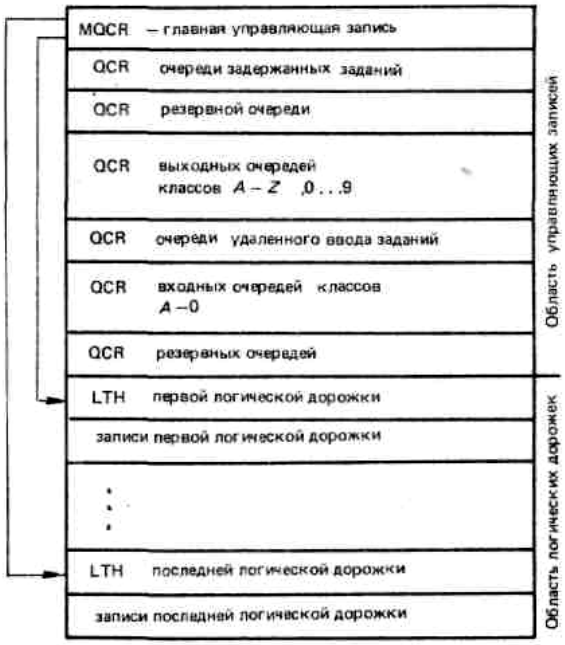


Рис. 3.7. Структура набора данных SYS1.SYSJOBQE

Область логических дорожек используется для размещения управляющих блоков и таблиц - элементов очередей входных и выходных работ. Каждая дорожка состоит из 20-байтового заголовка (LTH) и некоторого переменного числа 176-байтовых блоков, отведенных под записи. Количество блоков в одной логической дорожке задается при генерации ОС и может быть изменено при ее загрузке. Возможно задание до 255 блоков в дорожке. Все дорожки для одного задания сцеплены ссылочными адресами, что позволяет организовать последовательную выборку управляющих блоков задания.

§ 3.4. ЯЗЫК УПРАВЛЕНИЯ ЗАДАНИЯМИ В ОПЕРАЦИОННОЙ СИСТЕМЕ

Задания для ОС ЕС составляются на специальном языке управления заданиями. Существует девять операторов ЯУЗ. При записи операторов используются образы перфокарт, т. е. записи длиной 80 байт. Под поле оператора отводятся позиции с 1-й по 71-ю, позиция 72 называется колонкой *продолжения,* позиции 73...80 могут применяться для нумерации карт (поле идентификации).

Структура операторов. Поле оператора состоит из четырех полей, отделяемых друг от друга хотя бы одним пробелом (рис. 3.8).

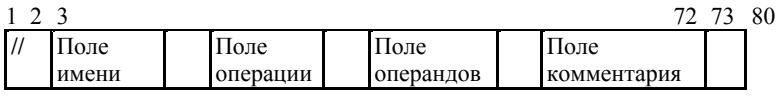


Рис. 3.8. Структура операторов ЯУЗ

Каждый оператор (кроме оператора конца набора данных) начинается с символа // в первых двух позициях, после него с третьей позиции без пробелов может стоять имя оператора. Пользователь может дать имя управляющему оператору для ссылки на него в другом управляющем операторе.

П о л е и м е н и оператора должно состоять не более чем из восьми символов, в числе которых могут быть буквы латинского алфавита, арабские цифры, а также специальные символы #, @, ¤. Первым символом должна быть буква.

П о л е о п е р а ц и и характеризуется тем, что в нем определяется тип управляющего оператора: JOB, EXEC, DD, PROC, PEND.

П о л е о п е р а н д о в содержит параметры, отделенные друг от друга запятыми. Если длина оператора превышает 71 символ, то его можно продолжить на следующей карте. На первой карте для любого оператора в позиции 72 кодируется пробел, а для оператора комментария – любой символ, отличный от пробела. Поле операндов должно прерываться на запятой. В первых двух позициях карты продолжения должны стоять символы //. Поле операндов может быть продолжено с 4-й позиции или со следующих, но не позже 16-й позиции, поле комментариев – с 3-й позиции.

Параметры в поле операндов делятся на позиционные и ключевые. *Позиционные параметры* должны стоять на определенном месте в списке операндов, например параметр «имя программиста» – в операторе JOB. *Ключевые параметры* можно представить в форме «ключевое слово»=«значение», например CLASS=A. Значение выбирается из некоторого набора значений данного ключевого параметра. Ключевые параметры могут следовать в любом порядке после всех позиционных параметров. Ряд возможных для данного типа оператора параметров может быть опущен. Если опускается позиционный параметр, то отделяющая его запятая сохраняется; если опускается ключевой параметр, то запятая не ставится. Например, в операторе //JOB1 JOB, ’ИВАНОВА’ опущен первый позиционный параметр.

П о л е к о м м е н т а р и е в может содержать любые сообщения, облегчающие чтение управляющих операторов.

Операторы ЯУЗ приведены в табл. 3.1. Параметры, указанные в квадратных скобках, являются необязательными. В задании первым должен стоять оператор задания JOB. Первым оператором для каждого шага задания должен быть оператор шага задания EXEC. За ним стоят операторы описания данных DD для данного шага. Оператор конца набора данных /* ставится после данных во входном потоке заданий. Пустой оператор // отмечает конец управляющих операторов. Операторы, стоящие после него, пропускаются при чтении-интерпретации вплоть до первого оператора JOB. Оператор команды используется оператором ЭВМ для связи с ОС ЕС. Операторы команды могут стоять во входном потоке или вводиться с пульта.

Таблица 3.1

Операторы	Содержимое полей			
	имени	операции	операндов	комментариев
Задания JOB	//Имя	JOB	Операнд	Комментарий
Шага задания EXEC	//Имя	EXEC	»	»
Описания данных DD	//Имя	DD	»	»
Процедуры	//Имя	PROC	»	»
Конца процедуры	//Имя	FEND	»	»
Команды	//	Команда оператора	Пробел	Пробел
Конца набора данных	/*	Пробел	»	Комментарий
Конца задания	//	»	»	Пробел
Комментария	//*	»	»	Комментарий

Управляющие операторы PROC и PEND применяются при составлении каталогизированных процедур и рассматриваются в гл. 5. Здесь описываются операторы JOB, EXEC, DD и их основные параметры.

Управляющий оператор задания JOB. Этот оператор стоит первым в задании и должен иметь

имя. Под этим именем задание обрабатывается планировщиком заданий. Если во входном потоке несколько заданий, то желательно, чтобы все задания имели уникальные имена. Поле операндов в операторе JOB может быть пустым, например задание с именем PROG1:

//PROGl JOB

Всего в поле операндов может быть 2 позиционных и 12 ключевых параметров (в режимах MVT, SVS).

П о з и ц и о н н ы м и п а р а м е т р а м и являются следующие: учетная информация; имя программиста.

Учетная информация является первым параметром, форма его записи устанавливается в соответствии с требованиями, принятыми на данном вычислительном центре (ВЦ). Формат параметра имеет вид ([учетный номер], [дополнительная учетная информация]), например

//PROG2 JOB (3781,3-27)

Если указывается только учетный номер, то круглые скобки опускаются, например

//PROG3 JOB 3781

Если дополнительная учетная информация содержит специальные знаки (за исключением дефиса), то этот параметр заключается в апострофы:

//PROG4 JOB (3781, ‘14/05/86‘,)

Учетная информация может содержать до 142 подпараметров, в число которых входят запятые. При переносе на следующую строку она прерывается на запятой:

```
//PROG5 JOB (3781, ‘14/05/86‘,
//      ‘ОТДЕЛ 9‘, ‘т.5-73‘)
```

Имя программиста должно стоять после учетной информации. Оно может содержать до 20 символов и записывается без апострофов в том случае, если включает только буквы латинского алфавита, например

//PROG6 JOB (841,5-48), IVANOV

В остальных случаях имя программиста заключается в апострофы:

//PROG7 JOB, ‘ИВАНОВ‘ //PROG8 JOB 763, ‘IVANOV T.P.‘

В первом примере учетная информация опущена, поэтому перед именем программиста стоит запятая. Во втором примере используются точки, поэтому имя программиста заключено в апострофы.

К л ю ч е в ы х п а р а м е т р о в – 1 2 , рассмотрим следующие из них:
CLASS, PRTY, MSGLEVEL, MSGCLACC, REGION, TIME, COND.

Класс задания опознается программой системного ввода RDR и задается с помощью параметра CLASS. В соответствии с классом задания производится постановка задания во входную очередь. Классов и соответственно входных очередей может быть 15. Значениями ключевого параметра CLASS являются буквы латинского алфавита от A до O. Обычно заданиям назначаются одинаковые классы в тех случаях, если у них примерно одинаковые требования к ресурсам ОС, например

//JOB JOB , PETROV, CLASS = J

Если параметр CLASS опущен, то по умолчанию CLASS=A.

Приоритет задания назначается с помощью параметра PRTY. Низший приоритет равен 0, высший -13. Во входной очереди одного класса задания ставятся и выполняются в порядке их приоритетов, например

//JOB2 JOB CLASS = C, PRTY = 7

Приоритет 13 используется в основном системными программистами. По умолчанию PRTY=0.

Уровень системных сообщений задается параметром MSGLEVEL (*message level*). Параметр MSGLEVEL имеет два подпараметра. Первый подпараметр принимает значения 0, 1, 2 и указывает планировщику заданий на то, какого уровня сообщения об управляющих операторах задания должны быть поставлены в выходную очередь. Если значение подпараметра 0, то на

печать выводится только оператор JOB, если 1 – выводятся все входные операторы задания и все операторы использованных в задании каталогизированных процедур, если 2 – выводятся только входные операторы задания.

Второй подпараметр принимает значения 0, 1 и указывает уровень сообщений о распределении устройств ввода-вывода. Если значение подпараметра 0, то на печать не выводятся сообщения о выделении устройств ввода-вывода шагам задания и о состоянии этих устройств после окончания шага задания, если значение 1 – на печать выводятся все сообщения о распределении устройств ввода-вывода. При аварийном завершении задания сообщения об устройствах выдаются всегда.
Общая форма параметра:

MSGLEVEL = (*n*1, *n*2)

Если параметр MSGLEVEL опущен, то используются значения, установленные при генерации системы. Можно опустить один из подпараметров, тогда для него берется установленное значение. Если опущен второй подпараметр, то первый можно писать без скобок, например

```
//JOB3 JOB MSGLEVEL = (2,0)
//JOB4 JOB MSGLEVEL=1
//JOB5 JOB MSGLEVEL=(, 1)
```

Выходной класс системных сообщений задает параметр MSGCLASS. Всего может быть 36 выходных очередей, каждая из которых обозначается определенным классом (A...O; 0...9). Элементы очередей одного класса выводятся на одно внешнее устройство, например

```
//JOB6 JOB MSGLEVEL = (2,0), MSGCLASS=J
```

Если параметр MSGCLASS опущен, то выходные сообщения ОС выводятся в выходную очередь класса А.

Запрос на выделение памяти под задание осуществляется с помощью параметра REGION, который используется только в режиме MVT. Если параметр REGION опущен, то заданию будет выделен минимальный объем памяти, равный размеру планировщика (около 60 К). Число килобайтов, указанных в REGION, должно быть четным, например

```
//JOB7 JOB , ‘КОВАЛЕВА‘, REGION=150 K
```

Время процессора, требуемое для выполнения задания, задается параметром TIME. Если этот параметр опущен, то задание снимается через 30 мин. Параметр TIME полезно вводить даже если для выполнения задания требуется мало времени, так как это позволяет избежать зацикливания при ошибке в программе. Общий вид параметра: TIME=(минуты, секунды), например:

– требуется 5 мин, секунды и скобки опущены:

```
//JOB8 JOB TIME = (10,30), REGION=70 K
//JOB9 JOB TTME=5
```

– необходимо 25 с (минуты опущены):

```
//JOB10 JOB TIME = (, 25)
```

Если в параметре TIME указать число 1440 (количество минут в сутках), то время выполнения становится неограниченным:

```
//JOB11 JOB TIME = 1440
```

Условие прекращения выполнения задания задается с помощью параметра COND. При завершении шага задания системой (или программистом) устанавливается код возврата, который может принимать значение от 0 до 4095. В системных программах используются следующие стандартные значения кодов возврата: 0 – ошибок нет; 4 – возможны ошибки (предостережение); 8 – обнаружены серьезные ошибки; 12 – обнаружены грубые ошибки и дальнейшее выполнение программы невозможно; 16 – обнаружены критические ошибки и нормальное выполнение программы невозможно; 20 – трансляция прекращена, так как обнаружены неустранимые ошибки ввода-вывода.

В параметре COND задаются подпараметры: код и условие. Условия бывают следующими: GT – больше; GE – больше или равно; EQ – равно; NE – не равно; LT – меньше; LE – меньше или

равно. По окончании каждого шага планировщик проверяет условие

«Код параметра» «Условие» «Код возврата»

Если условие выполняется, то решение задания прекращается. Использование параметра COND позволяет закончить исполнение задания, как только очередной шаг завершается аварийно. Параметр COND имеет следующий вид:

COND = ((код, условие), (код, условие), ...)

Максимально условий может быть восемь, например:

– задание завершается, если код завершения очередного шага больше семи:

//JOB12 JOB COND = (7,LT)

– задание завершается, если код завершения очередного шага больше восьми или равен четырем:

//JOB13 JOB COND = ((8,LT), (4, EQ))

Рассмотрим пример оператора JOB с несколькими параметрами:

```
//EXAMPLE1 JOB (221,38-64), ‘A.K. МОРОЗОВ‘, CLASS=D
//     PRTY = 10, MSGLEVEL = (1,1), MSGCLASS = 1, REGION = 200 K
//     TIME= (3,30), COND = (16,EQ)
```

где EXAMPLE1 – имя задания; (121,38-64) – учетная информация; А.К. Морозов – имя программиста; D – класс входной очереди; 10 – приоритет задания в очереди; 1 – выходной класс системных сообщений; 200 К – требуемая для задания память; (3,30) – требуемое время процессора (мин, с); 16 – условие завершения задания. На печать выводятся все операторы задания и каталогизированных процедур и все сообщения о распределениях устройств ввода-вывода.

Управляющий оператор шага задания EXEC. Этот оператор является первым для шага задания и содержит обращение к программе, которая должна выполняться на данном шаге, или к каталогизированной процедуре.

Имя шага не обязательно, но необходимо в случаях: ссылки на данный шаг задания, если впоследствии к этому шагу производится обращение; добавления оператора DD данного шага к шагу применяемой каталогизированной процедуры. Имя шага используется во многих системных сообщениях, выдаваемых оператору или пользователю. Его применение помогает установить, для какого шага задания выдается данное сообщение, например

//STEP1 EXEC PGM = SOLV

В операторе EXEC используются один позиционный и девять ключевых параметров.

Позиционным параметром является имя программы PGM или имя процедуры PROC, выбирается один из них.

Параметр PGM вызывает обрабатывающую программу, которая должна быть выполнена на данном шаге задания, например

– вызов программы пользователя

//STEP2 EXEC PGM = PROGRAM

– вызов программы IEYFORT компилятора с языка ФОРТРАН IV

//   EXEC PGM = IEYFORT

Параметр PROC применяется для обращения к каталогизированным процедурам или процедурам во входном потоке. Ключевое слово PROC указывать не обязательно, например

– обращение к каталогизированной процедуре FORTGC, описывающей шаг компиляции программы с языка ФОРТРАН IV

//STEP3 EXEC PROC=FORTGC

– обращение к каталогизированной процедуре PL1LFCLG, описывающей следующие шаги: компиляции программы с языка Пл/1; редактирования связей; исполнения

//STEP4 EXEC PL1LFCLG

Ключевые параметры не обязательны, здесь рассмотрим пять из них: ACCT, COND, DPRTY, RECTION, TIME.

Учетная информация для шага задания задается параметром ACCT. Общее число значений учетной информации не должно превышать 142. Правила записи значений такие же, как для параметра учетной информации оператора JOB, например

```
//EXEC PGM=ALPHA, ACCT=(253, '3/04/86')
//EXEC PGM=IEHDASDR, ACCT=348
```

Параметр COND позволяет обойти выполнение указанного шага задания в зависимости от кодов возвратов предыдущих шагов, а также установить, требуется ли исполнять данный шаг, если предыдущие завершились.

В первом случае используются такие же подпараметры (код, условие), как в параметре COND оператора JOB, или три подпараметра (код, условие, имя шага), если условие проверяется для какого-то одного шага, например

– шаг с именем STEP5 не исполняется (обходится) в том случае, если хотя бы один предыдущий шаг завершился с кодом возврата 8 или если шаг STEP завершился с кодом возврата, большим 8:

```
//STEPS EXEC PGM = BETA, COND = ((8, EQ), (8, LT, STEP3))
```

– шаг STEP7 обходится, если шаг STEP5 завершился с кодом возврата, большим 12:

```
//STEP7 EXEC PGM = GAMMA, COND = (12, LT, STEP5)
```

Количество условий в параметре COND не должно превышать восьми.

Во втором случае дополнительно применяются подпараметры EVEN (даже) или ONLY (только), которые могут стоять в любом месте; их общее количество не должно превышать восьми. В параметре COND может использоваться один из подпараметров, так как они определяются взаимоисключающие действия системы при аварийном завершении какого-либо предыдущего шага. Шаг аварийно заканчивается в том случае, если ОС не может его выполнить. Если указан параметр EVEN, то текущий шаг задания должен быть выполнен, даже если один или несколько предшествующих шагов закончились аварийно. Однако шаг задания обходится в случае, если в то же время удовлетворяется одна из заданных в параметре COND проверок кодов возврата. Если же указан подпараметр ONLY, то шаг задания выполняется только в случае аварийного завершения одного или нескольких предыдущих шагов. Если какая-либо из проверок кодов возврата, заданных в этом шаге, удовлетворяется, то шаг обходится, например:

– шаг задания STEPE исполняется в том случае, если одни или несколько предыдущих шагов завершились аварийно:

```
//STEPE EXEC PGM = PROG, COND=ONLY
```

– шаг задания STEPC обходится в том случае, если код возврата любого из предыдущих шагов меньше или равен 16, или если код возврата шага A больше или равен 20, или если все предыдущие шаги завершились нормально; если условия не выполняются и один из предыдущих шагов завершился аварийно, то шаг STEPC исполняется:

```
//STEPC EXEC PGM = PROG, COND = ((16, GE), (20, LE, A), ONLY)
```

– шаг задания STEPD обходится, если код возврата какого-либо предыдущего шага больше 8; в противном случае шаг STEPD выполняется, даже если один из предыдущих шагов завершился аварийно:

```
//STEPD EXEC PGM = MPROG, COND = (EVEN, (8, LT))
```

Если очередной шаг задания завершился аварийно, а в последующих операторах задания EXEC нет подпараметров ONLY или EVEN, то все задание завершается аварийно.

Параметр DPRTY назначает шагу задания текущий приоритет, определяющий приоритет использования ресурсов ОС во время выполнения шага задания. Этот параметр применяется для определения диспетчерского приоритета. Диспетчерский приоритет определяет порядок

обработки процессором задачи в условиях мультипрограммирования. Параметр DPRTY имеет два подпараметра (*n*1, *n*2), которые могут принимать значения от 0 до 15. Общая форма параметра: DPRTY = (*n*1, *n*2).

Диспетчерский приоритет вычисляется по формуле

$d=n1\cdot 16+n2.$

Если не задан первый параметр, то ОС присваивает шагу задания приоритет, указанный в операторе JOB. Если не задан второй параметр, то он принимается равным 11. Если не задан параметр DPRTY в операторе EXEC и параметр PRTY в операторе JOB, то диспетчерский приоритет для данного шага равен 11, например

//EXEC PGM = PROG, DPRTY = (3,2)

Изменение объема памяти для шага задания производится с помощью параметра REGION. Значение параметра должно быть четным, например

//STEP2 EXEC PROC=ALP, REGION=120 K

Время работы процессора для данного шага задается параметром TIME, аналогичным параметру TIME оператора JOB. Примеры:

//Al EXEC PGM = RPGC, REGION = 100 K, DPRTY=2, TIME=5

//A3 EXEC PGM = MAIN, COND = ONLY, T1ME=8, REGION=200 K

//A8 EXEC PROC=ABCD, TIME = (,30), DPRTY = (,5)

Управляющий оператор описания данных DD. В управляющих операторах DD описываются наборы данных, применяемые при выполнении шага задания, и периферийные устройства, необходимые для работы с наборами данных. Каждый оператор DD описывает один набор данных. Операторы DD для данного шага задания стоят сразу же после оператора EXEC, определяющего этот шаг.

Набор данных представляет собой поименованный, упорядоченный набор нескольких взаимосвязанных записей. Набор данных находится на каком-либо внешнем устройстве: перфокартах; перфоленте; магнитной ленте; магнитном диске. Для каждого устройства существует свой признак или особое состояние КОНЕЦ НАБОРА ДАННЫХ. Граница набора данных отслеживается и защищается ОС. При обработке наборов данных на перфокартах или магнитной ленте соответствующие устройства выделяются шагу задания до конца его выполнения, другие задания не имеют доступа к этим устройствам (монопольное использование устройств). Если набор данных находится на устройстве прямого доступа (магнитном диске), то при обращении к набору данных проверяются специальные управляющие таблицы. Если произошло обращение к чужому набору данных, то происходит прерывание и ОС сообщает об ошибке. Максимальное число операторов DD для одного шага задания 255.

С точки зрения ОС наборы данных подразделяются на входные, временные (промежуточные) и выходные.

Входной набор данных описывается оператором DD с именем SYSIN. Входной набор данных - набор данных, который должен быть обработан программой, указанной в операторе EXEC. Он передается шагу задания во входном потоке с какого-либо внешнего устройства. Наборы данных, находящиеся во входном потоке, вводятся с помощью программы чтения–интерпретации (RDR). Например, для шага трансляции с языка ФОРТРАН входным набором данных является составленная на этом языке программа. В конце выполнения шага задания входные наборы данных уничтожаются (DELETED) или сохраняются (KEPT).

Временный набор данных создается и уничтожается в одном и том же шаге задания. Временные наборы данных располагаются на магнитных лентах или дисках.

Выходные наборы данных подлежат выводу на внешние устройства. Они содержат результаты работы шага задания или системные сообщения. Для некоторых выходных наборов данных операторы DD имеют определенные имена: SYSPRINT – вывод на печать выходного набора данных; SYSABEND – вывод на печать содержимого памяти (ядра и раздела, в котором решалось задание) в случае аварийного завершения шага задания; SYSUDUMP – вывод на печать содержимого памяти для раздела, в котором выполнялось задание, в случае аварийного

завершения шага. В конце шага задания выходные наборы данных либо передаются следующему шагу задания (PASSED), либо выводятся (SYSOUT), либо сохраняются (KEPT).

Сообщения о типе наборов данных и их состоянии после выполнения шага входят в число системных сообщений. Уровень и класс выходных системных сообщений задаются в операторе JOB параметрами MSGLEVEL и MSGC1ASS.

Имя оператора DD идентифицирует данный оператор DD, и в последующих операторах DD в задании может быть ссылка на оператор DD по его имени. Если в шаге задания используются операторы DD с одинаковыми именами, то распределение периферийных устройств, памяти на них и обработка диспозиции выполняются для всех таких операторов, однако обращение всегда происходит к набору данных, описанному в первом операторе DD. Имя оператора DD опускается в тех случаях, когда оператор описывает набор данных, сцепленный с набором данных, описанным предыдущим оператором DD или когда данный оператор DD является вторым или третьим в группе операторов DD, описывающих индексно–последовательный набор данных.

Оператор DD имеет 4 позиционных и 19 ключевых параметров. Многие их этих параметров являются взаимоисключающими, т. е. не могут стоять в одном операторе DD. Рассмотрим основные позиционные параметры *, DATA, DUMMY и некоторые ключевые.

Параметры ввода данных через входной поток * или DATA указывают, что в задании непосредственно после оператора DD стоят данные для обрабатывающей программы данного шага. Параметр DATA применяется для ввода данных, начинающихся с //. Обычно эти данные содержат управляющие операторы ЯУЗ. В конце данных ставится оператор ограничения /*.

Если в данных нет карт, начинающихся с символов //, то применяется параметр *. Оператор ограничения не обязателен, так как признаком конца данных могут служить символы // следующего управляющего оператора или состояния КОНЕЦ ФАЙЛА на устройстве ввода. Например описание входного набора данных для шага компиляции с языка ФОРТРАН.

```
//FORT     EXEC PGM = IEYFORT, REGION=100 K
//SYSIN     DD     *
   «ИСХОДНЫЙ МОДУЛЬ НА ЯЗЫКЕ ФОРТРАН»
//SYSPRINT DD    SYSOUT=A
```

Если во входном потоке содержатся данные без // в первых двух позициях и перед ними отсутствует оператор DD * или DD DATA, то ОС автоматически вставит перед ними оператор //SYSIN DD *.

Позиционный параметр DUMMY позволяет организовать обход операций ввода-вывода для наборов данных. Если в операторе DD указан параметр DUMMY, то для данного оператора не будут выполняться распределение периферийных устройств и сама операция ввода-вывода, не будет обрабатываться диспозиция (параметр DISP), например

```
//DD1 DD DUMMY
```

Параметр DUMMY можно использовать при отладке программ.

Ключевой параметр SYSOUT указывает выходной класс для набора данных, например

```
//DD2 DD SYSOUT=B
```

Чтобы системные сообщения и выходной набор данных выдавались на одно выводное устройство, необходимо в параметре SYSOUT поставить тот же класс, что и в параметре MSGCLASS оператора JOB.

П е р и ф е р и й н ы е у с т р о й с т в а запрашиваются с помощью параметра UNIT, который может иметь до пяти подпараметров. Рассмотрим две формы параметра UNIT:

```
        адрес устройства
UNIT= { тип устройства }
       имя группы UNIT=AFF=имя DD
```

Подпараметр «адрес устройства» задает адрес периферийного устройства. Адрес формируется из номеров канала ввода-вывода, устройства управления и самого устройства.

Подпараметр «тип устройства» задает номер типа устройства. Например, номером типа устройств НМЛ ЕС-5010, ЕС-5016, ЕС-5017 является 5010, номером типа устройств НМД ЕС-5066

(100 М) - 5066.

Подпараметр «имя группы» указывает группу периферийных устройств, имеющих общее имя. Группа устройств определяется при генерации ОС. Например, накопители на магнитной ленте имеют групповое имя SYSSQ, накопители на сменных дисках – SYSSQ и SYSDA, алфавитно-цифровые печатающие устройства – SYSPRINT.

Подпараметр AFF означает, что запрашивается разделение периферийных устройств. Операционная система выделяет набору данных то же устройство, что было выделено набору данных в операторе DD с указанным именем.

Например:

– запрашивается одно из устройств ЕС-5010, ЕС-5016, ЕС-5017:

```
//DD1 DD UNIT=5010
```

– запрашивается один из накопителей на сменных дисках:

```
//DD2 DD UNIT=SYSDA
```

– запрашивается то же устройство, что и для набора данных, описанного в операторе DD с именем DD2:

```
//DD3 DD UNIT=AFF=DD2
```

Параметром VOLUME (VOL) задается информация о томе, на котором находится или будет находиться набор данных. Параметр имеет пять подпараметров; рассмотрим только один из них:

VOL = SER = (список серийных номеров).

Серийный номер тома – регистрационный номер тома (диска или ленты), который должен быть использован. Регистрационный номер тома может содержать от одного до шести буквенно-цифровых символов и дефис, например

```
//DD3 DD UNIT=5016, VOL=SER= 121262
//DD4 DD UNIT=SYSSQ, VOL=SER= (111121,A)
```

Параметр DSNAME идентифицирует набор данных. Некоторые его формы:

$$\left\{\begin{array}{l}\text{DSNAME}\\ \\ \text{DSN}\end{array}\right\} = \left\{\begin{array}{l}\text{имя набора данных}\\ *.\text{имя DD}\\ *.\text{имя тома}.\ \text{имя DD}\end{array}\right\}$$

Имя набора данных может быть простым или составным. Простое имя может содержать до восьми буквенно-цифровых символов. К буквенным относятся также специальные символы #, @, ¤, к цифровым - дефис. Первым символом должна быть буква. Составное имя содержит несколько простых имен, разделенных точками, и может иметь до 44 символов, например

```
//D1 DD DSNAME=A3684
//D2 DD DSN = BD-3584
//D3 DD DSN=AA1.C306.KLM
```

Использование вместо имени набора данных параметра «*. имя DD» означает, что имя набора данных определено в указанном операторе DD того же шага задания, а параметра «*. имя шага, имя DD» – другого предшествующего шага задания. Например:

– в операторе DD с именем D2 описан набор данных с именем ALPHA

```
//        EXEC PGM = PR1
//D1     DD     DSN=ALPHA, UNIT = SYSDA, VOL=SER= 111310
//D2     DD     DSN=*.D1
```

– в операторе DD именем DST2 описан набор данных с именем AST1

```
//ST1       EXEC PGM=AAA
//DST1     DD      DSN=AST1,...
.
```

.
.
//ST2 EXEC PGM = BBB
//DST2 DD DSN=*.ST1.DT1

Для временных наборов данных параметр DSN можно не задавать.

Параметр DISP (диспозиция) используется для определения состояния набора данных. Этот параметр включает три подпараметра.

Первый подпараметр описывает состояние набора данных, т. е. каковым он является: новым (NEW); старым (OLD), разделяемым несколькими заданиями (SHR); последовательно организованным, к которому надо добавить новые записи (MOD).

Второй подпараметр указывает, что нужно сделать с набором данных после нормального завершения шага задания. Он имеет следующие значения: DELETE – уничтожить; KEEP – сохранить; PASS – передать для использования в следующем шаге задания; CATLG - сохранить набор данных и его описание поместить в системный каталог; UNCATLG – сохранить, но запись о наборе данных в системном каталоге уничтожить.

Третий подпараметр определяет условную диспозицию, которая определяет, что нужно сделать при аварийном завершении шага задания. Значения третьего подпараметра следующие: DELETE, KEEP, CATLG, UNCATLG.

Операционная система автоматически сохраняет наборы данных, которые существовали до начала выполнения задания, и уничтожает наборы данных, не существовавшие до начала выполнения задания. Подпараметр NEW может быть опущен.

Параметры DISP и SYSOUT являются взаимно исключающими. Например:

– наборы данных *A* и *B* создаются в данном шаге задания, сохраняются при нормальном завершении шага и уничтожаются при аварийном:

//DD1 DD DSN=A, DISP = (NEW, KEEP) //DD2 DD DSN=B, DISP = (, KEEP)

– набор данных с именем C является разделяемым несколькими заданиями, используется только для чтения и сохраняется по завершении шага задания:

//DD3 DD DSN = C, DISP=SHR

– набор данных с именем D является модифицируемым, охраняется при нормальном завершении шага и уничтожается при аварийном завершении шага задания:

//DD4 DD DSN = D, DISP = (MOD,, DELETE)

§ 3.5. ГЛАВНЫЙ ПЛАНИРОВЩИК

Программы главного планировщика предназначены для осуществления связи оператора с операционной системой. Главный планировщик выполняет команды оператора, выдает оператору запросы и сообщения. Программы главного планировщика находятся в ядре операционной системы. Входным языком главного планировщика являются команды оператора. Функции главного планировщика - инициализация системы, обработка команд оператора.

Инициализация системы. Главный планировщик запускается программой инициализации ядра NIP. После того как программа закончит форматирование фиксированной области основной памяти, управление получают программы инициализации главного планировщика. Эти программы проверяют и инициализируют консольные коммуникации, инициализируют системный журнал и форматируют динамическую область памяти. Системный журнал ведется в двух наборах данных на томах прямого доступа, которые включаются в ОС при генерации системы. В системном журнале копируется информация с операторской консоли.

Обработка команд оператора. Команды оператора позволяют обращаться к управляющей программе, изменять ход операций, выполняемых ЭВМ, инициализировать или прекращать текущие операции. Можно выделить четыре типа команд оператора, обрабатываемых главным планировщиком.

Команды оператора первого типа воздействуют на статус системы, например команда VARY (или V) - изменить состояние устройств ввода-вывода в системе.

Команды оператора второго типа влияют на статус задания. Это следующие

команды: RESET (или E) – изменить характеристики задания (класс, приоритет задания) во входной или выходной очереди; HOLD (или H) – задержать выборку заданий из очереди; RELEASE (или A) – отменить действие команды HOLD; CANCEL (или C) – принудительно завершить (снять) задание и др.

Команды операторов третьего и четвертого типов осуществляют информационную связь с системой. Команды третьего типа реализуют запрос на информацию из системы. К ним относятся, например, следующие команды: DISPLAY (или D) – получить текущую информацию о времени и дате, о состояниях активных заданий, устройств ввода-вывода, системных очередей и др.; MONITOR (или MN) – обеспечить непрерывную выдачу на консоль текущей информации о работе планировщика задания, диспозиции наборов данных, установке и снятии томов и т. п. Обычно команду выдают сразу после загрузки ОС. Четвертый тип команд реализует ввод информации в систему. Это следующие команды: SET (или T) – сообщить параметры системы планировщику заданий; REPLY (или R) – ответить на запрос системы.

Команда оператора состоит из поля операции, содержащего мнемокод команды и поля параметров. *Мнемокод команды* может записываться полностью или одной буквой. *Параметры* разделяются запятыми. Между полем операции и полем параметров должен стоять хотя бы один пробел. Внутри полей пробелы недопустимы. Рассмотрим примеры команд оператора:

1) DISPLAY U, DASD – запрос сообщения о состоянии устройств прямого доступа;
2) D Q=(A, B, C) – запрос сообщения о количестве заданий во входной очереди в классах A, B, C;
3) SET DATE=87.145, CLOCK = 14.3800 - сообщение системе даты (1987 г., 145-й день года) и текущего времени суток (14 ч. 38 мин. 00 с);
4) *15IGF5091 REPLY DEVC OR‘NO‘ - запрос системы о номере устройства (DEVC) или о том, что оно не нужно (NO). Если устройство не нужно, оператор отвечает:R 15, ‘NO‘

После того как система приняла ответ, выдается следующее сообщение:

IEF600I REPLY TO 15 15 IS; NO

Оно означает, что ответ оператора ‘NO‘ принят.

§ 3.6. СИСТЕМА УПРАВЛЕНИЯ ЗАДАЧАМИ

Супервизором называется комплекс программ управления задачами. Низкооперативные ресурсы операционной системы (внешние устройства, наборы данных) распределяются на этапе подготовки заданий планировщиком заданий, а высокооперативные ресурсы распределяются между заданиями супервизором. К высокооперативным ресурсам относятся: время центрального процессора; участки динамической области основной памяти; таймер; системные программы, находящиеся в основной памяти. Супервизор также представляют программисту средства ввода собственных ресурсов и организации управления этими ресурсами.

Управление определенным типом ресурсов осуществляется отдельной программой. В состав супервизора входят следующие программы: супервизор прерываний; супервизор - диспетчер задач; супервизор задач; аппарат формальных ресурсов; супервизор основной памяти; супервизор виртуальной памяти; супервизор связей; супервизор времени. Рассмотрим функции этих программ.

Супервизор прерываний. Распределение времени ЦП между задачами, управление взаимодействием задач происходят через систему прерываний. Супервизор прерываний обрабатывает прерывания и передает управление после обработки прерывания. Программная обработка используется для четырех из пяти типов прерываний. Прерывания от схем контроля не обрабатываются супервизором, в этом случае управление аппаратно передается программе регистрации сбоев. Если программа отсутствует, то процессор переводится в состояние ожидания, из которого его можно вывести выполнением команды начальной загрузки.

Супервизор прерываний состоит из обработчика прерываний и программ обслуживания прерываний.

Обработчик прерываний выполняет следующие функции: сохраняет состояние прерывания программы (старого PSW) на момент прерывания для последующего возобновления работы программы; определяет тип прерывания; передает управление программам обслуживания прерываний; возвращает управление прерванной программе или готовой задаче с высшим

приоритетом.

П р о г р а м м ы о б с л у ж и в а н и я прерываний выполняют необходимые действия для прерывания данного типа. Рассмотрим типы прерываний и функции обслуживающих программ.

Прерывания по обращению к супервизору запланированы в программе для организации такого обращения. Программа пользователя может обращаться к той или иной программе супервизора с помощью специальных макрокоманд SVC. Некоторые из этих макрокоманд будут описаны при рассмотрении программ супервизора.

П р е р ы в а н и я о т в в о д а - в ы в о д а позволяют супервизору контролировать операции ввода-вывода, а также состояние каналов и устройств ввода-вывода. В старом PSW в качестве кода прерывания содержится адрес устройства ввода-вывода. Супервизор ввода-вывода осуществляет следующие функции: ставит запросы на ввод-вывод в очередь, если операции ввода-вывода не могут быть запущены немедленно; выводит запросы из очереди и запускает операции ввода-вывода; контролирует правильность выполнения операций ввода-вывода; планирует выполнение программ, обрабатывающих ошибки ввода-вывода. Супервизор ввода-вывода является программой управления данными.

Внешними прерываниями являются прерывания по таймеру; от кнопки «Прерывание» на пульте; от сигналов по линиям прямого управления в многопроцессорных системах. Прерывания первого типа обрабатываются супервизором времени, прерывания второго типа - главным планировщиком. Так как операционная система ЕС ЭВМ обслуживает однопроцессорные комплексы, то внешние прерывания третьего типа отсутствуют.

Программные прерывания возникают при некорректном применении команд или данных. Существует 15 типов программных прерываний. Стандартно программные прерывания не обрабатываются ОС, происходит только регистрация причины прерывания, и выполнение задачи аварийно завершается. Однако пользователь может написать свою программу обработки программных прерываний и указать ее обработчику программных прерываний с помощью макрокоманды SPIE.

Супервизор - диспетчер задач. *Диспетчер задач* анализирует и определяет, какая задача должна получить управление после обработки очередного прерывания. В памяти в области диспетчера в двух последовательных словах размещены два специальных поля, называемых NEW и OLD. Эти поля характеризуют состояние прерванной задачи соответственно после и до прерывания. Если состояние задачи в результате обработки прерывания не изменилось, т. е. NEW=OLD, то управление возвращается прерванной программе. Если состояние задачи изменилось, т. е. NEW ≠ OLD, то управление зависит от характера нового состояния. В том случае, когда задача перешла в состояние ожидания (NEW=0), диспетчер задач просматривает очередь блоков управления задачами TCB, определяет адрес TCB, готовой к выполнению задачи с высшим приоритетом, записывает адрес в поле NEW, приравнивает OLD=NEW и передает управление этой задаче.

Рис. 3.9. Схема цикла работы управляющей программы

На рис. 3.9 представлена схема цикла работы управляющей программы. Связь между отдельными блоками осуществляется через прерывания. В блоке *1* происходят чтение-интерпретация потока заданий, внесение заданий в очередь входных работ, запись данных во

входные наборы данных, инициализация некоторой задачи *A* и распределение ресурсов ввода-вывода для задачи *A*. В блоке *2* супервизор заносит задачу *A* в основную память и распределяет ресурсы между задачами. В блоке *3* задача *A* при обладании высшим приоритетом среди активных задач получает управление, запрашивает и обрабатывает данные и завершается. В блоке *4* супервизор интерпретирует запросы задачи *A*, передает запросы программам управления данными, возвращает управление задаче *A*. В блоке *5* программы управления данными; читают данные из входных наборов и записывают их в выходные наборы. В блоке *6* супервизор по завершении задачи *A* освобождает ресурсы задачи *A* и завершает задачу.

В блоке *7* программа управления заданиями (терминатор) освобождает выделенные устройства ввода-вывода, строит управляющие блоки для выходных данных, записывает выходные данные, т. е. осуществляется системный вывод.

Супервизор задач. Супервизор задач образует и управляет очередью задач в ЦП. Программы супервизора задач создают блоки управления задачами TCB и управляют ими. Новая задача образуется при обращении к супервизору задач с помощью макрокоманды ATTACH. Системные задачи, такие, как RDR, INIT, WTR, запускаются командой оператора START. Эта команда обрабатывается главным планировщиком, который и формирует макрокоманду ATTACH. Проблемная задача для шага задания образуется инициатором, формирующим макрокоманду ATTACH на основе оператора шага EXEC.

При выполнении макрокоманды ATTACH происходит прерывание по обращению к супервизору и управление передается программе супервизора задач ATTACH, которая в области системных очередей образует блок управления задачей TCB. Все блоки TCB выстраиваются в очереди к процессору по убыванию их диспетчерского приоритета. Системные и проблемные задачи стоят в одной очереди.

Граничный приоритет задачи шага задания определяет программа управления заданиями исходя из параметров PRTY, DPRTY, соответствующих операторов JOB и EXEC. Граничный приоритет является предельным максимальным значением для диспетчерского, и вначале диспетчерский приоритет устанавливается равным граничному. После запуска задачи диспетчерский приоритет может быть изменен с помощью макрокоманды CHAR. Значения граничного и диспетчерского приоритетов хранятся в блоке TCB.

В операционной системе в режимах MFT с подзадачами, MVT, SVS в зоне основной памяти, отведенной задаче, можно образовать дополнительные задачи, называемые *подзадачами.* Образованная при этом шаге задача называется *основной.* Подзадача образуется при выдаче программой основной задачи макрокоманды ATTACH.

При образовании дополнительных задач внутри шага задания управления требуется не одна, а несколько задач. Преимущество образования подзадач заключается в том, что, когда одна подзадача находится в состоянии ожидания, управление можно передать другой подзадаче данного шага, а не другого задания. В этом случае повышается загрузка вычислительной системы, ее производительность. Время выполнения задачи сокращается, так как в ней создаются параллельные процессы. Образование подзадач эффективно только тогда, когда можно
достичь значительного перекрывания во времени нескольких подзадач. Связи между подзадачами должен обеспечить пользователь. Однако при работе с подзадачами увеличивается расход времени, необходимого управляющей программе для образования дополнительных задач и управления ими. Подзадача претендует на управление и конкурирует с другими задачами за использование ресурсов ОС.

Любая подзадача данного уровня может образовать свои подзадачи последующего, более низкого уровня (рис. 3.10). Задачи *A1, B1, C1, A21, A22, B2, C2, A31, A32, C3* являются *подзадачами* задачи шага задания, а задачи *A21, A22, A3I, A32* - подзадачами задачи *A1*. Задача *A1* – порождающая задача для задач *A21* и *A22,* задача *A21* – порождающая для задач *A31* и *A32.* Число уровней подзадач не ограничено, но общее число задач в системе не должно превышать 255.

Подзадачи выполняются и завершаются асинхронно. Но так как каждая задача исполняет часть одного и того же шага задания, необходима координация выполнения задач. Если до окончания всех подзадач порождающая задача делает попытку завершиться, то эта задача и все ее подзадачи

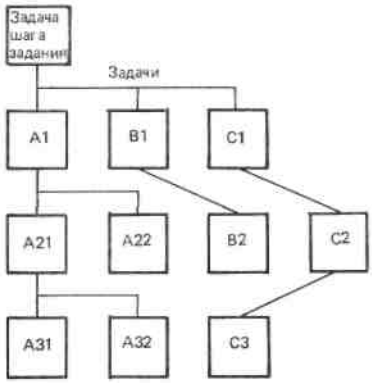


Рис. 3.10. Образование подзадач

завершаются аварийно. Нормально вывод завершенной подзадачи из системы осуществляется порождающей задачей с помощью макрокоманды DETACH. Если макрокоманда DETACH выдана до завершения подзадачи, то подзадача завершается аварийно. Основная задача шага задания является подзадачей инициатора. По завершении задачи шага задания инициатор удаляет блок TCB этой задачи из всех системных очередей. При образовании подзадачи ее граничный и диспетчерский приоритеты совпадают с соответствующими приоритетами порождающей задачи, однако с помощью специальных параметров макрокоманды ATTACH можно уменьшить граничный или увеличить диспетчерский приоритет подзадачи. В любом случае диспетчерский приоритет подзадачи не может превышать ее граничного приоритета.

При изменении приоритета задачи пользователь должен учитывать, что задача с высоким приоритетом получит управление ранее задачи с низким приоритетом. Задачам с большим числом операций ввода-вывода обычно присваивают более высокие приоритеты, чем другим задачам. Задачи, осуществляющие ввод-вывод, значительное время могут находиться в состоянии ожидания завершения операции ввода-вывода или освобождения внешнего устройства. В это время процессор может выполнять задачи с более низким приоритетом. Как только задача с высоким приоритетом перейдет из состояния ожидания в активное, она получает управление, а задачи с низким приоритетом переходят в состояние ожидания. Если порождающая подзадача должна ожидать завершения одной или нескольких подзадач, то при назначении приоритетов подзадачам следует стремиться к снижению общего времени ожидания.

На получение управления претендуют задачи из других заданий, поэтому первоначальная установка приоритетов подзадач данной задачи, выполненная без учета приоритетов других задач, может привести в определенных ситуациях к неверному исполнению шага задания. Поэтому в операционной системе имеется возможность изменять приоритет подзадач в процессе их исполнения. С помощью макрокоманды EXTRACT можно получить значения диспетчерского и граничного приоритетов из блока TCB и изменять эти значения макрокомандой CHAP. Макрокоманда CHAP изменяет диспетчерский приоритет активной задачи или ее подзадачи и граничный приоритет любой из активных подзадач.

При работе нескольких параллельных взаимосвязанных подзадач в рамках данного шага задания необходимо их синхронизировать. В операционной системе реализованы два пути синхронизации вычислительных процессов: прямой – через синхронизацию событий, косвенный – с помощью аппарата формальных ресурсов.

С и н х р о н и з а ц и я с о б ы т и й – основной способ синхронизации процессов, заключающийся в том, что один процесс ждет, пока другой процесс не сообщит ему о совершении определенного события. Когда это событие совершается, первый процесс переходит в активное состояние.

В качестве примера рассмотрим два процесса – две подзадачи [9]. Первая подзадача (производитель) осуществляет записи по одной за один раз и помещает их в буфер размером в одну запись. Вторая подзадача (потребитель) использует их по одной за один раз. Задачи должны быть синхронизированы, чтобы между любыми двумя последовательными записями в буфер выполнялась одна операция чтения, и наоборот. Эту синхронизацию можно осуществить следующим образом. Производитель пишет первую запись и совершает событие «запись», после чего переходит в состояние ожидания. Как только произошло событие «запись», выходит из состояния ожидания потребитель, читает запись из буфера и сообщает о событии «чтение». После

этого производитель переходит из состояния ожидания в активное состояние и описанный процесс повторяется.

Синхронизация событий осуществляется макрокомандами WAIT и POST. При обработке этих макрокоманд при прерывании по обращению к супервизору происходит обращение к одноименным программам WAIT и POST супервизора задач. Эти программы осуществляют взаимодействие между процессами с помощью блоков управления событиями ECB. Для каждого события строится блок ECB – четырехбайтовое поле основной памяти. Адрес блока идентифицирует событие. Два старших бита блока используются для обозначения ожидания (бит W) и завершения (бит P) события. Остальные биты заполняются информацией в зависимости от состояния битов W и P. В начальном состоянии биты W и P равны нулю.

В приведенном примере для события «запись» программа-производитель по завершении записи макрокомандой POST устанавливает в ECB бит P=1 и переходит в состояние ожидания. Программа-потребитель записи макрокомандой WAIT устанавливает в ECB бит W=l и проверяет бит P. Если P=1, то происходит чтение записи. Если P=0, то программа-потребитель переводится в состояние ожидания. Для поочередной работы программ производитель и потребитель необходим также блок ECB для события «чтение». Для этого события программа-потребитель после завершения чтения выдает макрокоманду POST, а программа-производитель после завершения записи выдает макрокоманду WAIT.

Указанные операции выполняются циклически. Процессы работают независимо, переходя в состояние ожидания по окончании событий «запись» (производитель) и «чтение» (потребитель) и включаясь в работу по окончании событий «чтение» (производитель) и «запись» (потребитель).

Аппарат формальных ресурсов. В операционной системе имеется возможность взаимодействия через использование одних и тех же ресурсов. К общим ресурсам, допускающим многократное применение различными задачами, относятся наборы данных, повторно используемые программы, область данных в основной памяти. В операционной системе эти ресурсы представляются формально, так как только пользователь знает их содержание. Супервизор управляет такими ресурсами без учета их содержания на основании формальных определений ресурсов и их текущего состояния. *Аппаратом формальных ресурсов* называются программы управления общими ресурсами.

Ресурсы могут применяться в двух режимах: совместном, когда допускается одновременное обращение к ресурсу из нескольких задач; монопольном, когда ресурс в данный момент времени может использоваться только одной задачей.

Проверку формальных ресурсов на доступность и управление формальными ресурсами производит программа супервизора задач ENQ/DEQ. Перед началом работы с ресурсом задача выдает макрокоманду ENQ, с помощью которой оформляется запрос на использование ресурса. Происходит прерывание по обращению к супервизору, супервизор ставит запрос в очередь к ресурсу и определяет текущее состояние ресурса. Если ресурс доступен, то управление возвращается задаче, выдавшей запрос, в противном случае задача переводится в состояние ожидания. Ресурс может быть доступен для задачи при выполнении одного из условий: 1) в системе нет других запросов на этот ресурс; 2) текущий и ранее выданные запросы допускают совместное использование ресурса.

По окончании работы с ресурсом следует выдать макрокоманду DEQ. Супервизор, получив управление в результате этой макрокоманды, выводит запрос из очереди и определяет, существуют ли еще запросы на данный ресурс. Если запросов нет, то управление возвращается программе, выдавшей макрокоманду DEQ. Если запросы имеются, то для последовательно используемых ресурсов супервизор выводит из состояния ожидания задачу, запрос которой стоит первым в очереди на ресурс. Для совместно применяемых ресурсов управление получает задача с наибольшим приоритетом, а не в порядке очереди запросов на данный ресурс.

Супервизор основной памяти. *Супервизор основной памяти* распределяет между задачами динамическую область основной памяти и область системных очередей. Управление динамической областью памяти распадается на уровни управления динамической областью и зоной.

Уровень управления динамической областью характеризуется тем, что на нем происходит поиск и выделение для задачи свободной зоны памяти. Программа управления заданиями (инициатор) выбирает из таблицы управления шагом задания SCB значение параметра

REGION и обращается к супервизору основной памяти с помощью макрокоманды GETMAIN. Супервизор основной памяти просматривает очередь свободных блоков до тех пор, пока не обнаружит блок, равный или больший требуемого размера. Если блок совпадает по размеру с требуемым, то он отводится под зону целиком, а соответствующий элемент очереди свободных блоков ликвидируется. Если блок превышает требуемый размер, то определяется длина оставшейся части блока и корректируется элемент очереди свободных блоков. Для новой зоны в области системных очередей строится элемент очереди разделов PQE, содержащий начальный адрес и длину зоны. Адрес элемента PQE помещается в блок управления задачей TCB. Зона освобождается терминатором с помощью макрокоманды обращения к супервизору основной памяти FREEMAIN. Освобожденный блок включается в очередь свободных блоков.

Уровень управления зоной связан с выделением участков зоны по запросу задачи, т. е. с динамическим распределением памяти в пределах зоны. Для получения или освобождения участка памяти задача должна выдать запрос супервизору. Запросы на основную память могут быть явными и неявными. Неявные запросы удовлетворяются, например, при динамической загрузке программ в память зоны. Использование макрокоманды GETMAIN является явным запросом на память, выделяемую активной задаче. Помимо программ пользователя супервизор и программы управления данными делают запросы на память для управляющих блоков. Область зоны освобождается с помощью макрокоманды FREEMAIN. Освобожденная область может применяться любой задачей данного шага задания. Число выделяемых задаче байтов всегда кратно восьми, т. е. области памяти выровнены на двойное слово. Задача может либо делать запросы на память по мере необходимости, либо сделать один большой запрос для удовлетворения всех требований на память. В последнем случае выделяется непрерывная область памяти и расход времени на работу супервизора памяти меньше. Однако значительно чаще необходимо динамически выделять память при исполнении задачи – в задачах с подзадачами, при организации структур с перекрыванием, реентерабельных программ (см. гл. б).

Динамическое распределение памяти внутри зоны приводит к фрагментации зоны, также как и динамическое распределение памяти между задачами (выделение зон) – к фрагментации основной памяти. Для снижения фрагментации памяти внутри зоны в режимах MVT и SVS используется аппарат подпулов. *Подпул* – множество блоков памяти, объединенных в один список. Логически подпул рассматривается как единая область основной памяти. Задача может использовать до 128 подпулов с номерами от 0 до 127. Подпулы создаются только в том случае, если в макрокоманде GETMAIN стоит параметр SP, задающий номер подпула. Подпул с номером 0 создается управляющей программой при инициализации главной задачи для шага задания. Подпул 0 может применяться совместно всеми задачами шага (главной задачей и ее подзадачами). Если в макрокоманде GETMAIN номер в параметре SP не указан, то выделяется память из подпула с номером 0.

Супервизор виртуальной памяти. Операционная система в режиме SVS, используя аппаратные средства динамической переадресации, обеспечивает виртуальную память. Функции программ управления заданиями в режимах MVT и SVS совпадают. *Супервизор виртуальной памяти* управляет обменом страниц в режиме виртуальной памяти. Программные средства ОС, обеспечивающие виртуальную память, состоят из супервизоров памяти и страниц. Динамическая область основной памяти в режиме SVS подразделяется на области V и R. В области V распределение памяти ведется сегментами, виртуальные адреса не совпадают с реальными и возможен страничный обмен. В области R распределение памяти ведется страницами, страничный обмен запрещен, все необходимые задачам страницы фиксируются в основной памяти, таблицы страниц и сегментов настраиваются так, чтобы виртуальные адреса совпадали с реальными. Область R создается для задач, для которых страничный обмен и, следовательно, режим виртуальной памяти нежелательны. Размер области R задается при генерации системы или при начальной загрузке.

Запрос на вид памяти осуществляется с помощью параметра ADDRSPC в операторах JOB и (или) EXEC задания. Значение ADDRSPC=VIRT указывает, что заданию или шагу задания необходима область виртуальной памяти V, значение ADDRSPC=REAL-что заданию или шагу задания необходима реальная область памяти R.

Супервизор памяти выделяет формируемый инициатором запрос на зону памяти с учетом параметра ADDRSPC. Учет свободных сегментов областей R и V ведется раздельно.

Главное отличие работы супервизора памяти в режиме SVS состоит в том, что программа, предназначенная для исполнения, загружается не в реальную память, а в виртуальную. Адреса программы преобразуются в виртуальные, и по окончании загрузки вся программа или большая ее часть располагается в специальном страничном наборе данных с именем SYS1.PAGE на томе прямого доступа. Одновременно для каждого сегмента строится таблица страниц. Страничный набор данных SYS1.PAGE создается на этапе начальной загрузки ОС.

Супервизор страниц управляет обменом страниц между основной памятью и набором данных SYS1.PAGE. Если при исполнении программы осуществляется обращение к виртуальной странице, отсутствующей в основной памяти, то происходит прерывание и организуется страничный обмен для загрузки отсутствующей страницы в память. Супервизор страниц выделяет странице блок основной памяти и организует ее загрузку в этот блок. Если свободного блока не окажется, то в памяти выбирается страница, которую можно вернуть в страничный набор данных, и по адресу этой страницы загружают необходимую.

Замещаются в соответствии с двумя правилами следующие страницы: 1) к которым дольше других не было обращения; 2) которые за время пребывания в основной памяти не изменились. Для быстрого получения такой информации применяются аппаратные средства, совмещенные с ключом защиты памяти. В ключе защиты дополнительно по сравнению с ЕС ЭВМ первой очереди введены биты обращения R и изменения C. В бит обращения помещается единица при каждом обращении к странице, в бит изменения – единица при каждой записи в какую-либо ячейку страницы. В соответствии с указанным правилом замещаются страницы со значениями битов R и C в указанном порядке: 00, 01, 10, 11. Если содержимое страницы не изменялось (бит C=0), то к обратному переносу страницы в страничный набор данных не прибегают, так как она там содержится.

Обмен страниц осуществляется следующим образом. Программы страничного обмена строят адрес блока для новой виртуальной страницы. Затем специальная программа подготавливает операции ввода-вывода страниц и обращается к супервизору ввода-вывода для запуска канальных программ. После завершения ввода-вывода эта программа снимает ожидание с прерванной из-за отсутствия страницы задачи.

Число операций обмена страниц и необходимое для них время зависят от числа активных задач в операционной системе и числа произведенных ими запросов. Если для страничного обмена требуется слишком много времени, то производительность вычислительной системы может сильно снизиться. Поэтому интенсивность обмена контролируется супервизором страниц.

При генерации операционной системы в режиме SVS задаются следующие параметры: оптимальное значение интенсивности страничного обмена; допустимое отклонение интенсивности обмена от оптимального; временной интервал регулирования интенсивности T_0. Интенсивность обмена – число перемещений страниц в единицу времени. Супервизор страниц запрашивает интервал T_0 макрокомандой STIMER и ведет счетчик замещения страниц. По истечении интервала T_0 происходит прерывание по таймеру и управление передается программе управления интенсивностью страничного обмена. Если интенсивность обмена за последний интервал T_0 превысила допустимую границу, то из рабочей смеси задача с низким приоритетом переводится в состояние ожидания. Если же интенсивность, обмена ниже допустимой границы, то снимается ожидание с одной приостановленной задачи. Таким образом, за интервал T_0 из рабочей смеси может выйти или поступить в нее одна задача. Эффективность управления интенсивностью обмена зависит от выбранных значений параметров. Этот выбор должен проводиться с учетом задач, решаемых на данной вычислительной системе.

Супервизор связей. Такой супервизор обеспечивает связь между программами, находящимися в основной памяти. Работа программы супервизора связей рассматривается в гл. 6.

Супервизор времени. *Супервизор времени* обеспечивает службу времени. Программы службы времени позволяют пользователю получить дату и время дня, заказать интервал времени. Организуется служба времени на базе аппаратного таймера.

Установку даты и времени дня производит оператор ЭВМ при загрузке ОС командой SET. Супервизор времени, используя аппаратный таймер, корректирует информацию о времени каждые 20 мс (при частоте сети 50 Гц). Запрос о дате и времени осуществляется с помощью макрокоманды TIME.

Временной интервал может быть установлен для любой задачи шага задания с помощью

макрокоманды STIMER. Время, оставшееся до истечения интервала, может быть проверено и сброшено макрокомандой TTIMER. Для режимов MVT и SVS каждая задача шага задания может быть обеспечена отдельным временным интервалом. Служба времени позволяет хронометрировать отдельные процессы и обеспечивает работу в реальном масштабе времени.

Квантование времени. Система квантования времени – дополнительное средство управления процессором, средство диспетчеризации. Задачам, принадлежащим к квантующейся группе, время процессора предоставляется циклически. Каждая задача по очереди получает определенный интервал (квант) времени процессора. В конце этого интервала активная задача прерывается и управление передается следующей задаче группы квантования. Снятая задача ставится в конец очереди.

При циклическом обслуживании приоритет задачи возрастает с увеличением времени ожидания с момента получения последнего кванта, а для новой задачи – с момента ее создания. Такая форма равного обслуживания неявно отдает предпочтение коротким процессам, так как они будут заканчиваться быстрее, чем длинные. В то же время длинные процессы не будут все время находиться в состоянии ожидания.

Значение кванта времени должно быть подобрано так, чтобы общее время ожидания и время, затраченное на переключение задач, не были слишком большими.

Управление очередью задач в системе квантования времени осуществляет диспетчер. Для задания кванта времени диспетчер обращается к супервизору времени. Все задачи группы квантования времени имеют один диспетчерский приоритет или диапазон диспетчерских приоритетов. Возможные приоритеты для задач, квантующих время, и величина кванта устанавливаются во время генерации ОС. Когда группа квантования времени становится самой приоритетной в очереди задач, все задачи из группы поочередно получают управление. Выполнение задач в группе квантования происходит до тех пор, пока все задачи группы не завершатся или не перейдут в состояние ожидания. Выполнение задач в группе квантования может закончиться, если в активное состояние перейдет задача с более высоким, чем у группы, приоритетом. Внутри группы квантования циклическое выделение времени происходит также с учетом приоритетов. Квантование времени чаще всего используется в режиме реального времени, при построении диалоговых систем.

Рестарт. Выполнение задания может завершиться аварийно вследствие ошибки в программе или сбоя операционной системы. Для заданий, требующих нескольких часов процессорного времени, необходимость повторного запуска с самого начала приводит к очень большим потерям времени.

Р е с т а р т позволяет повторить выполнение задания с любого шага задания или с некоторой контрольной точки программы. В ОС предусмотрены следующие типы рестарта: два автоматические – шага задания и с контрольной точки, два отложенные – шага задания и с контрольной точки.

Автоматический рестарт состоит в повторении задания сразу после аварийного завершения, *отложенный рестарт* – в повторении через некоторое время. Любой вид рестарта можно выполнить только с разрешения оператора.

К о н т р о л ь н а я т о ч к а создается в программе с помощью макрокоманды CHKPT. Информация о контрольной точке заносится в специальный набор данных контрольной точки. Контрольную точку следует создавать в тех местах программы, где имеется исчерпывающая информация о состоянии программы, чтобы можно было после аварийного завершения продолжить выполнение программы с контрольной точки. В одной программе можно создавать несколько контрольных точек.

Запрос на автоматический рестарт с начала шага задания или с контрольной точки выполняется с помощью параметра RD в операторах языка управления заданиями JOB или EXEC. Значениями параметра RD могут быть R; RNC; NC; NR. При задании R запрашивается автоматический рестарт с начала шага задания; RNC означает то же, что и R, но подавляется действие макрокоманды CHKPT, т. е. контрольная точка не создается.

При задании NC автоматический рестарт не запрашивается и подавляется действие макрокоманды CHKPT. При использовании значения NR автоматический рестарт не запрашивается, но разрешается установить контрольную точку с помощью макрокоманды CHKPT.

Запрос на выполнение отложенного рестарта производится с помощью параметра RESTART в

операторе JOB. Параметр RESTART добавляется программистом в повторно выполняемое задание, ранее закончившееся аварийно. При рестарте с определенного шага задания в параметре RESTART указывается имя шага задания, которое было установлено при работе макрокоманды CHKPT во время первого выполнения задания: RESTART = < имя шага>. При рестарте с контрольной точки указывается дополнительно идентификатор контрольной точки:

RESTART=<имя шага>, <идентификатор контрольной точки >

При этом в задание должен быть включен оператор DD с именем SYSCHK, описывающий набор данных контрольных точек.

Глава 4. СИСТЕМЫ ПРОГРАММИРОВАНИЯ И СИСТЕМНЫЕ ВСПОМОГАТЕЛЬНЫЕ ПРОГРАММЫ

В главе приводятся комплексы программ, составляющие систему программирования ЕС ЭВМ. Описываются назначение и характеристики основных языков программирования. Рассматриваются функции системной программы редактирования и объединения отдельных объектных модулей в готовые к использованию загрузочные модули, а также их логическая структура. Заключительная часть главы посвящена специальному набору вспомогательных программ, предоставляющих пользователям широкий набор сервисных функций обслуживания используемых массивов информации.

§ 4.1. ТРАНСЛЯТОРЫ

Систему программирования ОС ЕС образуют алгоритмические языки с трансляторами, редактор связей, загрузчик, средства отладки, некоторые системные библиотеки, принадлежащие трансляторам или содержащие часто используемые подпрограммы. Кроме того, к системе программирования относятся отдельные вспомогательные программы, предназначенные для обслуживания библиотек и других наборов данных.

Назначение и характеристики трансляторов. Трансляторы предназначены для преобразования программ, написанных на языках программирования, в программы на машинном языке. Программа, подготовленная на каком-либо языке программирования, называется *исходным модулем.* В качестве входной информации трансляторы применяют исходные модули и формируют в результате своей работы *объектные модули,* являющиеся входной информацией для редактора связей. Объектный модуль содержит текст программы на машинном языке и дополнительную информацию, обеспечивающую настройку модуля по месту его загрузки и объединение этого модуля с другими независимо оттранслированными модулями в единую программу.

Программа может состоять из одного или нескольких исходных модулей, которые могут быть написаны на одном или на разных языках программирования. Исходные модули помещаются во входной набор данных соответствующего транслятора.

Все трансляторы вырабатывают однотипный, с точки зрения дальнейшей обработки, объектный модуль и помещают его в выходной последовательный или библиотечный набор данных. Если полученный объектный модуль нужен только в текущем задании и его не предполагается сохранить для будущего использования, то в качестве выходного набора данных транслятора назначают временный последовательный набор данных или временную библиотеку. Вид выходного набора данных программист указывает в задании. В ходе трансляции по указанию программиста транслятор может формировать в отдельном выходном последовательном наборе данных листинг и диагностические сообщения для последующей выдачи на дисплей или на печать.

По функциональному назначению трансляторы разделяются на быстрые (отладочные) и медленные (оптимизирующие).

О т л а д о ч н ы й т р а н с л я т о р должен обеспечить высокую скорость обработки текста исходных программ с выявлением максимального количества ошибок. Для этого транслятора требования к качеству объектной программы невысокие.

О п т и м и з и р у ю щ и й т р а н с л я т о р при трансляции отлаженной программы должен обеспечивать минимальные время объектной программы и объем памяти, занимаемый программой. Для осуществления этих противоречивых требований можно применять следующие способы трансляции: 1) создание одного и того же транслятора в двух вариантах (отладочный и

оптимизирующий); 2) разработка параметрического транслятора, способного работать и в отладочном и в оптимизирующем режимах.

В системе программирования ОС ЕС используются оба способа. В частности, для ФОРТРАНа имеются два транслятора: отладочный (ФОРТРАН стандартный) и оптимизирующий (ФОРТРАН оптимизирующий). Транслятор с языка ПЛ/1 построен по параметрическому принципу. Указывая значение уровня оптимизации, можно управлять скоростью трансляции и качеством объектной программы.

Трансляторы ОС ЕС хранятся в общей системной библиотеке SYS1.LINKLIB в виде загрузочных модулей. Имя загрузочного модуля – это имя программы. Чтобы выполнить трансляцию, нужно указать это имя в параметре PGM оператора EXEC. Кроме такого *статического* вызова любой транслятор можно вызвать *динамически* во время выполнения проблемной программы (например, из программы на языке Ассемблер). Для этого в макрокоманде динамического вызова нужно указать имя транслятора и имена используемых транслятором наборов данных.

При увеличении объема оперативной памяти, выделяемой транслятору, время трансляции, как правило, сокращается за счет уменьшения числа обращений к внешней памяти.

Для выполнения часто повторяющихся в разных программах действий (распределение памяти для размещения данных, управление операциями ввода-вывода, вычисление элементарных функций и т. п.) трансляторы включают в объектный модуль обращения к стандартным библиотечным подпрограммам. Все трансляторы ОС ЕС имеют собственные библиотеки подпрограмм, в которых каждая подпрограмма, оформленная в виде загрузочного модуля, составляет отдельный раздел. Библиотечные подпрограммы объединяются с основным объектным модулем в единый загрузочный модуль на этапе редактирования.

У Ассемблера и транслятора РПГ нет библиотек стандартных подпрограмм, однако Ассемблер имеет библиотеку макроопределений с именем SYS1.MACLIB, которая содержит тексты макроопределений для системных макрокоманд.

Наборы данных трансляторов. Трансляторы используют входные, промежуточные и выходные наборы данных, имена которых необходимо указывать в операторах DD в задании на трансляцию.

Входные наборы данных трансляторов обязательно включают набор данных, который описывается оператором DD с именем SYSIN. Этот набор данных содержит исходный модуль, имеет последовательную организацию или является разделом библиотеки.

Ассемблер и трансляторы с алгоритмических языков КОБОЛ и ПЛ/1 могут также дополнительно применять входной библиотечный набор данных, описываемый оператором DD с именем SYSLIB. Для Ассемблера он определяет системную библиотеку SYS1.MACLIB, в которой содержатся определения системных макрокоманд и могут находиться тексты на языке Ассемблер, включаемые в программу оператором COPY. С библиотекой SYS1.MACLIB в задании иногда сцепляют личные библиотеки макроопределений и текстов на языке Ассемблер. Для транслятора с языка КОБОЛ набор данных, задаваемый оператором DD с именем SYSLIB, используется лишь в том случае, когда в исходном модуле имеются пункты COPY или утверждения INCLUDE, а для транслятора ПЛ/1 – когда в исходном модуле имеются утверждения % INCLUDE, в которых не указана используемая библиотека. Каждое макроопределение и каждый текст, копируемый из библиотеки, составляют один раздел библиотеки.

Промежуточные (рабочие) наборы данных, записываемые на магнитных лентах и дисках, применяются трансляторами при необходимости дополнительной внешней памяти. Они нужны всем трансляторам, кроме транслятора с фОРТРАНа, использующего большую область основной памяти. Транслятор с языка ПЛ/1 использует промежуточный набор данных на диске с именем SYSUT1 лишь в том случае, когда основной памяти недостаточно. Набор данных с именем SYSUT3 применяется в том случае, когда исходная программа на языке ПЛ/1 записана в 48-литерном алфавите или когда в исходной программе имеются утверждения % INCLUDE. В этих случаях перед началом трансляции исходный модуль обрабатывается специальными препроцессорами, которые записывают переработанный текст в промежуточный набор данных.

Выходные наборы данных трансляторов включают следующие наборы данных: на диске (или магнитной ленте), в который помещается объектный модуль, передаваемый редактору

связей; для выдачи объектного модуля на перфокарты; для выдачи листинга. Трансляторы помещают объектный модуль для редактора связей в набор данных, описываемый оператором DD с именем SYSLIN, а Ассемблер и транслятор с языка РПГ - в набор данных, описываемый оператором DD с именем SYSGO. Если исходный модуль транслируют с целью выявления лексических и синтаксических ошибок, то можно задать только набор данных DD с именем SYSPRINT.

Выходной набор данных листинга обычно ставится в одну из системных выходных очередей и, как правило, распечатывается. Форму листинга можно значительно изменить в зависимости от параметров, задаваемых транслятору. Эти параметры задают в операнде PARM оператора EXEC, вызывающем транслятор, и называют *опциями.*

Опции. Опции трансляторов задают ключевыми подпараметрами, поэтому они могут располагаться в произвольной последовательности. Различают два типа опций, задаваемых ключевыми словами и ключевыми словами со значениями.

Ключевыми словами задают опции, которые имеют два или три фиксированных значения. Большинство таких опций имеет утвердительную и отрицательную формы, например SOURCE - печатать исходную программу, NOSOURCE - не печатать исходную программу.

Ключевым словом со значением задаются опции, которым можно назначить произвольное значение из некоторого множества значений. Например, опция SIZE=56 К задает объем основной памяти, доступной транслятору ПЛ/1. В операционной системе ЕС ЭВМ стандартные значения опций устанавливаются при генерации. Это позволяет программисту не задавать специальные опции, если его устраивают стандартные значения.

Набор опций в разных трансляторах различен (от 4 опций в трансляторе РПГ до 25 опций в трансляторе ПЛ/1). Перечень основных опций трансляторов приведен в табл. 4.1.

Таблица 4.1.

Опции	Выполняемые действия
DECK	Выдача объектного модуля на перфокарты
NODECK	Отмена режима DECK
ID	Указание на включение порядковых номеров операторов исходной программы в объектный модуль
NOID	Отмена режима ID
LOAD	Вывод объектного модуля для его последующей обработки редактором связей
NOLOAD	Отмена режима LOAD
SOURCE	Печать исходной программы, написанной на языках ФОРТРАН или ПЛ/1, вместе с сообщениями транслятора
NOSOURCE	Отмена режима SOURCE. Выводятся операторы, в которых имеются ошибки и сообщения об ошибках
LIST	Печать исходной программы, написанной на языках АЛГОЛ, КОБОЛ И Ассемблер Для программы, написанной на языке ФОРТРАН, значение LIST определяет печать объектного модуля
NOLIST	Отмена режима LIST
LINECNI=N	Задание количества строк (N) на листе распечатки
MAP	Печать плана распределения памяти для элементов исходной программы, написанной на языке ФОРТРАН
NOMAP	Отмена режима MAP
XREF	Печать таблиц перекрестных ссылок для программ, написанных на языках ПЛ/1 и Ассемблер
NAME=имя	Определение имени основной ФОРТРАН - программы. Имя должно содержать не более восьми алфавитно-цифровых символов; имя идентифицирует выводимые распечатки и перфокарты. Если режим не указан, основной программе присваивается имя MAIN

Опции трансляторов разделяются на пять групп:

1) вывода, указывающие куда направить объектный модуль – редактору или на перфорацию;
2) ввода, определяющие особенности кодировки исходного модуля и формат карт (в ПЛ/1);
3) листинга, определяющие его состав;

4) управляющие, задающие режим трансляции и особенности формируемой объектной программы;

5) специальные, специфичные для каждого транслятора.

§ 4.2. СОСТАВ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ ОПЕРАЦИОННОЙ СИСТЕМЫ

Языки программирования – алгоритмические языки, предназначенные для описания совокупности инструкций, т. е. подлежащих обработке данных (информации) и алгоритмов (программ) их обработки с помощью ЭВМ, выполнение которых обеспечивает правильное решение требуемой задачи.

В состав трансляторов ОС ЕС входят трансляторы с таких языков программирования, как Ассемблер, ПЛ/1, ФОРТРАН, АЛГОЛ, РПГ.

Язык Ассемблер. *Ассемблер* – машинно-ориентированный язык, содержащий макросредства. *Машинно-ориентированным языком* называется язык программирования, отражающий структуру вычислительной машины или класса машин. Каждой команде машинно-ориентированного языка соответствует машинная команда, которая опознается и выполняется техническими средствами вычислительной машины. Язык Ассемблер – это компонент системного программирования, переводящий программу на автокоде в перемещаемую программу на машинном языке. Исходная программа на Ассемблере представляется в командах, отражающих внутреннюю структуру машины, и позволяет программисту ссылаться на внутренние регистры машины и адреса памяти, используя их символические обозначения. Помимо трансляции важной функцией Ассемблера является «сборка» или связывание программ из подпрограмм и программных модулей, а также необходимая коррекция их взаимных ссылок и связей. Современные версии этого языка обладают возможностями макрогенератора. Поэтому они называются *макроассемблерами.* Язык Ассемблер значительно облегчает процесс программирования по сравнению с программированием в машинных кодах.

Язык Ассемблер применяется в первую очередь системными программистами для составления, модификации и дополнения программ системного программного обеспечения. Ассемблер позволяет разрабатывать наиболее эффективные программы с минимальным временем счета и наиболее полно использовать все технические возможности ЕС ЭВМ.

Исходная программа на Ассемблере записывается в виде отдельных символических операторов. В Ассемблере применяются следующие типы операторов: машинные команды, команды Ассемблера, макрокоманды, команды генерации, комментарии.

Операторы машинных команд представляют собой символическую форму записи обычных машинных команд ЕС ЭВМ. Операторы команд Ассемблера определяют действия транслятора при переводе исходной программы на машинный язык.

Команды Ассемблера, в свою очередь, разделяются на следующие команды: определения; секционирования и соединения; управления. *Команды определения* предоставляют программисту различные способы определения констант и областей памяти, присваивают значения символическим именам, определяют базовые регистры. *Команды секционирования и соединения* позволяют делить программу на части, устанавливать связи между программными частями, которые будут транслироваться отдельно. *Команды управления* разрешают изменять содержание результатов трансляции, управлять вводом исходного модуля.

Операторы макрокоманд применяются для обращения к макроопределениям (ранее подготовленным наборам операторов на Ассемблере).

Операторы команд генерации вместе с макрокомандами составляют макросредства языка Ассемблера. С помощью команд генерации можно менять последовательность генерации машинных команд и команд Ассемблера, а также их содержание.

Операторы комментариев поясняют программу на Ассемблере и никакого влияния на содержание создаваемого объектного модуля (программы на машинном языке) не оказывают.

Ассемблирование, как правило, выполняется в два этапа.

На первом этапе осуществляется ввод исходной программы, включение макроопределений, указанных макрокомандами исходной программы, и копируемых текстов. Копируемый текст – это текст, составленный из операторов на Ассемблере, который включается в текст исходной

программы из библиотеки исходных модулей с помощью оператора COPY. Далее текст программы, расширенный макроопределениями и копируемыми текстами, обрабатывается согласно указаниям команд генерации; определяется порядок и содержание исходных операторов, которые поступят на обработку на втором этапе.

На втором этапе мнемонические коды операций заменяются машинными, данные, записанные на Ассемблере, переводятся в машинный формат. Для рабочих областей резервируются области памяти, исходным операторам присваиваются значения адресов памяти. Для правильного присваивания адресов памяти символическим именам транслятор Ассемблера каждому оператору присваивает значение счетчика адреса. Для первого оператора транслятор задает счетчику адреса некоторое первоначальное значение, затем счетчик адреса корректируется для каждого обработанного символического оператора. Адреса памяти в машинных командах обычно идентифицируются путем задания регистра базы и смещения.

В конце второго этапа выдаются результаты трансляции: объектный модуль, представляющий собой программу на машинном языке; распечатка результатов трансляции.

Объектный модуль записывается на магнитный диск или ленту или может быть выведен на перфокарты. *Распечатка результатов* трансляции содержит тексты объектного и исходного модулей и некоторые другие сведения, необходимые для анализа программы, например сообщения о синтаксических ошибках в исходной программе.

Язык программирования ПЛ/1. Название языка ПЛ/1 образовано из начальных букв слов PROGRAMMING LANGUAGE. Вначале предлагались названия ФОРТРАН VI, NPL и др. Наименование ФОРТРАН VI соответствует первоначальным попыткам создать расширенную версию языка ФОРТРАН, подмножеством которой являлись бы классические версии ФОРТРАН II и ФОРТРАН IV. Однако такая преемственность нецелесообразна. Предложение назвать новый язык ФОАЛБОЛом отражало цель синтеза языков ФОРТРАН, АЛГОЛ и КОБОЛ, хотя это не означало механического слияния автономных подмножеств. Обозначение NPL (NEW PROGRAMMING LANGUAGE) оказалось нежелательным из-за совпадения сокращения с первыми буквами названия NATIONAL PHISYCAL LABORATORY (в Великобритании). Наконец фирма «IBM» утвердила название ПЛ/1, которое официально считается именем, а не сокращением. В период 1963 - 1968 гг. фирма «IBM» организовала группу, разработавшую этот язык. К этому времени был накоплен опыт эксплуатации таких языков программирования, как ФОРТРАН, КОБОЛ, АЛГОЛ. Кроме того, достаточно широкое распространение получили большие вычислительные машины третьего поколения. Вычислительные машины IBM-360, обладая мощными операционными системами, предоставили программистам большие возможности, которые, однако, не подтверждались средствами существовавших в то время языков программирования. Необходимость разработки языка, использующего эти возможности в достаточно полном объеме, и обусловила появление языка ПЛ/1.

Как всякий язык программирования, ПЛ/1 в процессе разработки развивался и улучшался. Было опубликовано три версии ПЛ/1, пока язык не стабилизировался. Становление языка закончилось после завершения разработки первого компилятора уровня F для системы OS-360 в 1965 г., а широкое распространение языка началось в 1968 г., когда был создан улучшенный вариант компилятора. Опыт эксплуатации АЛГОЛа показал, что для завоевания признания новый язык должен быть прост и надежен при использовании на элементарном уровне, должен вызывать большой интерес теоретиков в своих более развитых формах.

Разработчики языка сформулировали следующие основные требования к языку ПЛ/1: 1) необходимость применения машинного кода должна возникать в крайних случаях; 2) независимость от характеристик вычислительной машины; 3) модульность.

В основе создания языка лежало положение, что в самом широком диапазоне приложений программирование основано на общих закономерностях и связано с общими проблемами. Поэтому ПЛ/1 охватывает максимально возможный диапазон средств программирования.

Первое требование во многом удовлетворено, поскольку практически все возможности, обеспечиваемые операционной системой, доступны на уровне языка.

Второе требование также выполнено, ПЛ/1 зависит от машины в гораздо меньшей степени, чем другие языки программирования. Однако полностью устранить зависимость от машины на сегодняшний день не представляется возможным.

Третье требование можно понимать в двух аспектах. Во-первых, в ПЛ/1 имеется

возможность выделения некоторых подмножеств языка для обучения и использования в конкретной сфере деятельности. Например, инженеру, применяющему численные алгоритмы, не нужно знать тонкости использования указателей и прямого доступа к данным, и, наоборот, разработчик системы управления предприятием должен хорошо знать средства редактирования при вводе-выводе, но менее подробно - вопросы точности при арифметических преобразованиях. Во-вторых, под модульностью также понимаются блочная структура языка, наличие внешних процедур, которые, будучи откомпилированными от отдельности, впоследствии могут быть объединены в одну программу.

Для реализации задач, написанных в языке ПЛ/1, существует программа-транслятор ПЛ/1. От того, насколько полно транслятор отражает средства языка, зависит отличие языка одной машины от другой, одной ОС от другой. В ЕС ЭВМ имеются два компилятора с языка ПЛ/1.

Программисту, работающему с языком ПЛ/1, следует учитывать, что транслятор, преобразующий текст программы на ПЛ/1 в программу, написанную на машинном языке, составляет далеко не оптимальный вариант. Опытный программист на Ассемблере может написать программу, содержащую в несколько раз меньшее число машинных команд, чем эквивалентная программа на ПЛ/1. Противовесом этому является скорость и удобство составления и отладки программ. Кроме того, качество компилятора оценивается также возможностью оптимизации составленной объектной программы, т. е. минимизации числа машинных команд. Компилятор ПЛ/1 ОС имеет три уровня оптимизации программ и делает доступным программисту управление оптимизацией.

Рассмотрим некоторые свойства языка ПЛ/1. Язык ПЛ/1- *универсальный язык* программирования, получивший широкое распространение в силу того, что объединяет лучшие черты таких языков, как АЛГОЛ, ФОРТРАН (для решения научных задач) и КОБОЛ (для решения экономических задач). Таким образом, он имеет средства для решения научных и экономических задач. Язык ПЛ/1 позволяет легко применять при написании программ принципы структурного программирования.

Язык ПЛ/1 – *язык высокого уровня.* Уровень языка определяется, с одной стороны, тем, насколько удобен язык для записи алгоритма, т. е. насколько средства языка близко приближаются к математической формулировке задачи; с другой стороны, степенью обобщения понятий. Конкретно это проявляется в том, сколько машинных команд порождается для оператора языка.

Язык ПЛ/1 – *машинно-независимый язык.* Влияние ЭВМ проявляется в способах адресации и обмена с внешними устройствами. Сделать язык полностью машинно-независимым означало бы заведомо отказаться от каких-то возможностей ЭВМ, может быть, нужных и полезных, потому что язык не будет иметь средств для использования этих особенностей. Язык ПЛ/1 зависим от ЭВМ в меньшей мере, чем другие языки программирования, поскольку эта зависимость проявляется только в способе представления данных и в аппарате записеориентированного ввода-вывода, где учитываются характеристики конкретных внешних устройств.

Язык ПЛ/1 имеет *средства организации* в рамках одной программы *параллельного выполнения нескольких процессов,* называемых подзадачами. Мультизадачность позволяет программисту более эффективно использовать машинное время, например запускать параллельную подзадачу в момент, когда основная задача ожидает окончания операции ввода-вывода. Кроме того, программист получает средства моделирования параллельных процессов, поскольку ПЛ/1 обладает возможностями синхронизации выполнения задач. Кроме того, имеются средства, позволяющие формировать данные со сложной организацией - массивы и структуры. Язык ПЛ/1 предоставляет программисту средства управления памятью, позволяет запрашивать и освобождать память, управлять размещением в памяти специальных переменных. Это дает возможность использовать в программе сложно организованные структуры данных, такие, как «списки», «магазины», «деревья».

Язык ПЛ/1 располагает *средствами,* обеспечивающими почти полное *использование системы прерываний* и обработку ситуаций, возникающих при различных прерываниях. Развитая система прерываний – важное свойство современных вычислительных машин, позволяющее при некоторых ситуациях прерывать ход процесса и запускать другой процесс. Система прерываний обеспечивает совмещение операций ввода-вывода с работой процессора и возможность программной обработки прерываний.

Язык ПЛ/1 предлагает программисту *средства ввода-вывода,* позволяющие не учитывать особенностей конкретных внешних устройств, интерпретируя данные на внешнем носителе как непрерывный поток символов. Кроме того, имеются средства, обеспечивающие полный учет особенностей внешних устройств при вводе-выводе. Другими словами, средства языка отражают почти все методы доступа, имеющиеся в ОС ЕС. Язык ПЛ/1 совместно с транслятором ПЛ/1 образует *одну из систем программирования* ОС ЕС. Транслятор ПЛ/1 уровня F представляет собой сложную программу, построенную по оверлейному принципу. Трансляция осуществляется за несколько просмотров текста. Для комплектации программ имеется обширная библиотека подпрограмм, реализующих встроенные функции, операции ввода-вывода и преобразования данных.

Язык ПЛ/1 позволяет осуществить *принцип структурного программирования,* т. е. систематического применения типичных групп операторов для написания ясных и легкообозримых программ. Для структурного программирования характерен принципиальный отказ от операторов перехода GO TO. Анализ статистики показывает, что количество ошибок в программах пропорционально количеству операций передачи управления, а не общему количеству операторов или команд. Язык ФОРТРАН не приспособлен для структурного программирования, а АЛГОЛ в этом отношении практически равноценен языку ПЛ/1.

Язык ПЛ/1 *сложен для освоения* в полном объеме, причем в большей степени, чем такие языки, как ФОРТРАН и АЛГОЛ. Например, четыре версии языка используют 331 ключевое слово (включая имена встроенных функций, но не считая слов-омонимов, сокращений и модификаций с символом %), тогда как язык Ассемблер имеет 158 элементарных команд. Расширение языка ПЛ/1 может идти путем добавления новых ключевых слов и операторов.

Язык программирования ФОРТРАН. Это алгоритмический язык, ориентированный на решение научно-технических и инженерных задач. Первая версия появилась в 1956 г. в США. В дальнейшем он был существенно развит, в ряде стран разработаны стандарты ФОРТРАНа. Язык ФОРТРАН широко распространен, так как прост в изучении и позволяет эффективно применять ресурсы вычислительной машины.

Система программирования включает языки ФОРТРАН IV, Базисный ФОРТРАН, трансляторы и библиотеки стандартных программ. Алгоритмический язык Базисный ФОРТРАН, реализованный в ДОС ЕС, является подмножеством языка ФОРТРАН IV. Программы, написанные на Базисном ФОРТРАНе, можно обрабатывать транслятором ФОРТРАН IV. Операционная система ОС ЕС содержит следующие трансляторы с языка ФОРТРАН: ФОРТРАН IV уровней G и H, ФОРТРАН 77.

Объектами языка ФОРТРАН является целые и вещественные числа, числовые переменные. Выражения конструируются с помощью операций арифметических, определяющих четыре действия арифметики и возведение в степень, логических (И, ИЛИ, НЕ) и операций отношения, а также круглых скобок. Основные операторы ФОРТРАНа: присваивания, условного и безусловного переходов, цикла, вызова подпрограмм, ввода и вывода. Последние включают средства, позволяющие экономно задавать внешнее представление информации, и осуществляют при вводе и выводе необходимое редактирование данных согласно заданному спецификатору формата. Имеются также средства для управления размещением информации на печатной странице. Аппарат подпрограмм ФОРТРАНа представляет возможность эффективно комплектовать независимо составленные и оттранслированные части программ, что особенно удобно при программировании больших по объему задач. Программы библиотек ФОРТРАНа предназначены для вычисления наиболее распространенных математических функций и выполнения служебных действий, часто применяемых в программах. Многие средства ФОРТРАНа использованы в языке ПЛ/1.

Язык программирования АЛГОЛ. Это алгоритмический язык высокого уровня, ориентированный на решение задач математического характера. Язык был разработан группой специалистов различных стран в 1960 г. Идея динамического распределения памяти и развитый аппарат использования процедур, заложенные в АЛГОЛе, обеспечили возможность гибкой настройки процедур по параметрам. Язык АЛГОЛ-60 – первый из языков программирования, синтаксис которого был строго описан в виде нормальных форм Бэкуса, что послужило толчком к дальнейшему развитию формальных языков.

Программа на языке АЛГОЛ-60 представляет собой последовательность описаний величин и

действий над ними. В языке используются следующие типы величин: простые переменные, представляющие целые, действительные числа и логические величины; массивы; метки; переключатели и процедуры. Программа на языке АЛГОЛ-60 разделяется на отдельные участки – блоки, которые могут быть смежными или вложенными друг в друга. Описания величин имеют силу только в том блоке, в котором они указываются в программе, однако с помощью специальных средств можно указать, что значение величины должно сохраняться и после выхода из блока. Помимо описаний блоки содержат также операторы, определяющие действия над данными. К таким операторам относятся операторы присваивания, условного перехода, цикла, обращения к процедуре. В языке АЛГОЛ-60 допускается настройка параметров процедуры по значению (присваиваются значения фактических параметров соответствующим формальным параметрам) и по наименованию (указываются адреса фактических параметров).

Первая группа вариантов языка АЛГОЛ называется эталонной и содержит фиксированный набор основных символов и только их видом отличается от других вариантов.

Вторая группа вариантов называется *языками публикаций; в* этих языках можно пользоваться любыми знаками в качестве основных символов, если использование этих знаков облегчает применение языка. При этом необходимо соблюдать четкое соответствие с эталонным языком.

Третья группа вариантов языка называется *конкретными представлениями;* в них основным символам эталонного языка соответствуют знаки или сочетания знаков, являющиеся знаками входного алфавита конкретной ЭВМ.

В СССР АЛГОЛ-60 для ЭВМ второго поколения получил значительное распространение. Однако для ЭВМ следующих поколений наметилась устойчивая тенденция перехода при решении научно-технических и инженерных задач от АЛГОЛа к более простому ФОРТРАНу. Алгоритмический язык АЛГОЛ оказал большое влияние на развитие идей в области программирования и явился базовым языком для создания многих других языков программирования.

Язык программирования РПГ. Это язык программирования (генератор отчетов), ориентированный на решение задач по обработке экономической информации, включающий в себя выполнение следующих процедур: организации, хранения, корректировки и обновления файлов; сортировки данных; составления и печати различных бухгалтерских отчетов, ведомостей, таблиц, сводок и т. п. Язык РПГ позволяет производить несложные вычисления над входными данными, формировать отчеты и выдавать их на печать. Этот язык представляет собой достаточно характерный пример проблемно-ориентированного языка.

Программа на РПГ представляет собой совокупность бланков, каждый из которых отражает одну из сторон процесса получения отчета, т. е. описания входных и выходных данных, вычислений, используемых файлов. Пользователь может суммировать указанные поля просматриваемых записей, производить над исходными данными операции арифметические, сравнения, пересылки, проверки, проверки состояния, поиска по таблице. Имеются операторы условного и безусловного перехода внутри программы, а также возможность передачи управления внешним программам и осуществление связи с ними.

§ 4.3. РЕДАКТОР СВЯЗЕЙ, ОБЪЕКТНЫЕ И ЗАГРУЗОЧНЫЕ МОДУЛИ

Редактор связей - системная обрабатывающая программа, редактирующая и объединяющая объектные модули, полученные в результате работы транслятора, в единые загрузочные, готовые к выполнению программные модули. Загрузочный модуль может быть помещен ОС в основную память и выполнен. Загрузочный модуль компонуется из одной или нескольких программных секций - перемещаемых частей программы. Каждый объектный модуль из входного потока редактора связей может иметь ссылки к программным секциям в других модулях. Такие ссылки называются *внешними* и реализуются с помощью адресных констант. Символ, указываемый внешней ссылкой, называется *внешним именем.* Он должен быть или именем программной секции, или именем точки входа в программную секцию. Установленное Редактором связей соответствие между внешними ссылками и внешними именами называется *разрешением ссылки.* Внешние имена называются *внешними символами.*

Структура объектных и загрузочных модулей. Объектные и загрузочные модули имеют следующую логическую структуру:

1) управляющие словари, элементы которых вырабатываются транслятором во время обработки

внешних символов, адресных констант или программных секций. Каждый транслятор обычно вырабатывает два вида управляющих словарей: словарь внешних символов (ESD) и словарь перемещаемых символов (RLD). Информация управляющих словарей используется для разрешения перекрестных ссылок между программными секциями разных модулей и для переадресации адресных констант;

2) текст (TXT), представляющий собой инструкции и данные программы;

3) запись конца модуля (END).

С л о в а р ь в н е ш н и х с и м в о л о в содержит по одному элементу для каждого внешнего символа, определенного внутри модуля или к которому имеется ссылка, для каждой внешней ссылки, псевдорегистра (внешняя фиктивная секция), имени входа, поименованных или непоименованных программных секций и общих областей. Эта информация необходима Редактору связей для объединения модулей.

К имени точки входа, к поименованной программной секции или к псевдорегистру можно обращаться из любой другой программной секции или отдельно обработанного модуля, а к непоименованной программной секции – нельзя. Каждый элемент идентифицирует символ или ссылку и задает их относительный адрес внутри модуля, если он известен. Элементы ESD классифицируются следующим образом:

внешняя ссылка – символ, определенный как внешнее имя в другом отдельно обработанном модуле, к которому обращается данный обрабатываемый модуль. В элементе ESD указывается символ, адрес неизвестен;

слабая внешняя ссылка – специальный тип внешней ссылки, которая не может быть разрешена посредством автоматического вызова библиотеки, если отсутствует обычная внешняя ссылка к этому символу. В элементе словаря указывается символ, адрес неизвестен;

имя входа – имя внутри программной секции, определяющее точку входа. Элемент словаря внешних символов указывает символ и его адрес и идентифицирует программную секцию, к которой принадлежит этот символ;

имя программной секции – символическое имя. Элемент словаря содержит символ, длину программной секции и ее адрес. В этом случае под адресом понимается начало программной секции, т. е. адрес ее первого байта в программе;

поименованная или именованная общая область – программная секция, используемая для резервирования области основной памяти, к которой могут обращаться другие модули. В элементе словаря внешних символов указываются имя и длина области. Если область не поименована, то поле имени содержит пробелы;

частный код – непоименованная программная секция. В элементе словаря указывается длина программной секции и адрес начала, в поле имени помещаются пробелы;

псевдорегистр – ячейка в динамически выделенной основной памяти, которая может применяться реентерабельными программами и служит указателем к динамически распределенной памяти. Таким образом, память может не резервироваться в загрузочном модуле, а выделяться во время выполнения.

Во время обработки входных модулей Редактор связей находит для каждой внешней ссылки соответствующий внешний символ.

С л о в а р ь п е р е м е щ а е м ы х с и м в о л о в содержит по одному элементу для каждой перемещаемой адресной константы, которая должна быть модифицирована перед выполнением модуля.

В элементе словаря указываются адрес адресной константы внутри секции и внешний символ, на который ссылается адресная константа. Адресные константы содержатся в объектных модулях. Внешний символ определяется в словаре внешних символов в другом модуле или в другой секции того же модуля. Редактор связей обрабатывает адресные константы для формирования загрузочного модуля. Кроме того, при выборке загрузочного модуля для формирования загрузочного модуля. Кроме того, при выборке загрузочного модуля для выполнения адресные константы «настраиваются» по месту основной памяти.

У п р а в л я ю щ и е с л о в а р и выходного модуля составляются из управляющих словарей входных модулей. Управляющие словари загрузочного модуля состоят из объединенного словаря внешних символов CESD и RLD.

Управляющие предложения Редактора связей. Эти предложения позволяют наряду с опциями формировать требования, которым должны удовлетворять полученные в результате редактирования загрузочные модули.

Рассмотрим основные управляющие предложения Редактора связей

П р е д л о ж е н и е N A M E определяет имя загрузочного модуля (имя раздела библиотеки), созданного из модулей с помощью операторов, предшествующих данному оператору. Это предложение записывается в виде

NAME имя-модуля [(R)]

где имя-модуля определяет имя раздела библиотеки, содержащего созданный загрузочный модуль; R - символ, означающий, что созданный загрузочный модуль должен заменить существующий модуль с тем же именем в библиотеке выходных модулей, определенной оператором DD с именем SYSLMOD.

При создании нескольких загрузочных модулей предложение NAME должно быть написано столько раз, сколько загрузочных модулей нужно создать. Предложения NAME должны быть так размещены, чтобы управляющие предложения Редактора, относящиеся к одному загрузочному модулю, располагались над соответствующим предложением NAME.

Предложение ALLIAS позволяет присваивать дополнительные имена (до 16 шт.) программному модулю; записывается в виде

ALLIAS имя$_1$ [,имя$_2$]...[,имя$_{16}$]

Это предложение должно быть записано перед соответствующим оператором NAME, если создается несколько загрузочных модулей. При создании загрузочного модуля предложение ALLIAS может быть записано перед, после или между редактируемыми модулями.

П р е д л о ж е н и е E N T R Y указывает первый оператор, который должен выполняться в загрузочном модуле после редактирования. Это предложение записывается в виде

ENTRY «внешнее-имя»

где «внешнее-имя» может быть именем программной секции или входным именем в программе.

Этот оператор может быть записан в любом месте потока управляющих предложений Редактора связей, если создается один загрузочный модуль. При создании нескольких загрузочных модулей это предложение должно быть записано /перед предложением NAME, определяющим заданный модуль.

П р е д л о ж е н и е C H A N G E позволяет Редактору связей заменять имена программных секций, внешние имена и ссылки, не производя новой трансляции. Это предложение /записывается в виде

CHANGE стар. - имя$_1$ (нов. - имя$_1$),
стар. -имя$_2$ (нов. -имя$_2$), ...,

где стар. – имя$_k$ - внешняя ссылка в редактируемом модуле, заменяемая на внешнюю ссылку нов. - имя$_k$. В одном предложении можно осуществить несколько замен, к=1, 2, *..1*

П р е д л о ж е н и е R E P L A C E позволяет осуществить замену одних программных секций другими, а также удаление входных имен. Это предложение записывается в виде

REPLACE имя - сек - 1$_1${ [(имя - сек - 2$_1$)] }
вход-имя$_1$
имя - сек - 1$_2$ {[(имя - сек - 2$_2$)]}
вход-имя$_2$, ...,

где имя – сек1$_k$ обозначает имя заменяемой или удаляемой программной секции, к=1, 2, ...;

вход – имя$_k$ определяет удаляемое входное имя.

Если указано имя – сек - 2$_k$, то программная секция с этим именем заменяет программную секцию с именем имя – сек - 1$_k$. При отсутствии имя – сек - 2$_k$ программная секция с именем имя – сек - l$_k$. удаляется. Последовательность удаляемых внешних имен или заменяемых программных секций можно продолжать необходимое число раз.

П р е д л о ж е н и я I N S E R T O V E R L A Y при своем чередовании позволяют задать оверлейную структуру загрузочного модуля. Оверлейная структура загрузочного модуля дает возможность за счет выполнения его по частям экономить оперативную память.

Предложение INSERT записывается в виде

INSERT имя-сек$_1$ [, имя - сек$_2$...],

где имя – сек$_k$ - имя программной секции, помещаемой из входной последовательности в требуемое место оверлейной структуры.

В одном предложении INSERT может быть записано несколько имен программных секций. С помощью предложения

OVERLAY имя [(REGION)]

определяется имя загружаемого сегмента. В сегмент входят все программные секции, перечисленные в предложении INSERT. Идентификатор «имя» назначается программистом произвольно по правилам написания имен программных секций. При указании значения REGION для сегмента создается новая область [20].

Предложения INCLUDE и LIBRARY применяются для описания источников ввода информации в Редактор связей и будут описаны ниже.

Наборы данных Редактора связей. Редактор связей использует следующие наборы данных: основного ввода, библиотеку автовызова, дополнительного ввода, промежуточный, выходную библиотеку загрузочных модулей и диагностический выходной. Наборы данных основного ввода, промежуточный, выходной диагностический, а также выходная библиотека загрузочных модулей входят в число обязательных наборов данных Редактора связей.

Набор данных основного ввода, имеющий DD-имя SYSLIN, содержит объектные модули, являющиеся результатом трансляции, и управляющие предложения Редактора связей. Основной ввод может быть последовательным набором данных, разделом библиотечного набора данных или цепочкой последовательных наборов данных и разделов библиотечных наборов данных. Этот набор может располагаться на любом устройстве (устройстве чтения с перфокарт, магнитных ленте, диске, барабане). Если набор данных библиотечный, то в параметре DSNAME оператора DD необходимо указать имена библиотеки и раздела. Загрузочные модули в наборе данных основного ввода находиться не могут.

Комбинация наборов данных с одинаковой или различной организацией формируется с помощью сцепления операторов DD с SYSLIN.

Библиотеки автовызова используются для подключения программных модулей, не найденных в основном вводе. Запрашиваемое внешнее имя отыскивается в оглавлении вызванной библиотеки. При обнаружении требуемого имени внешняя ссылка считается разрешенной, если оно было описано в модуле как имя программной секции или дополнительное имя. В противном случае раздел подключается к загрузочному модулю и ссылка считается неразрешенной.

Обычно библиотеки автовызова требуются в том случае, когда Редактор связей обрабатывает объектные модули, полученные трансляторами АЛГОЛ-60, ФОРТРАН, КОБОЛ и ПЛ/1. Библиотеки автовызова находятся на дисках и могут содержать либо объектные модули, либо объектные модули и управляющие предложения Редактора связей, либо загрузочные модули. Смешивать в этом наборе данных загрузочные модули с объектными модулями и управляющими предложениями нельзя. Это ограничение объясняется различием формата записей, из которых состоят загрузочный и объектный модули.

Автоматически вызываемая библиотека описывается в DD со специальным именем SYSLIB. Допускается сцепление наборов данных. Часто это свойство применяется при вызове каталогизированных процедур для обработки программных модулей, написанных на алгоритмических языках. Так, например, процедура FORTGCLG содержит оператор DD с именем SYSLIB, который описывает системную библиотеку объектных модулей языка ФОРТРАН SYS1.FORTLIB. Поэтому при вызове этой процедуры и необходимости подключения личной библиотеки автоматического вызова следует написать:

```
//LKED. SYSLIB DD DSNAME=SYSl.FORTLIB, ... DD DSN=AA1
```

В этом примере к системной библиотеке автовызова подключается личная библиотека с именем AA1, содержащая объектные модули.

Средство сцепления наборов данных имеет ограниченное применение, так как накладывает на наборы достаточно жесткие ограничения, поэтому Редактор связей допускает возможность описания автовызова с помощью предложения LIBRARY.

LIBRARY имя-DD (имя-раз$_1$ [, имя-раз$_2$, ...]),
(вн - ссылка$_1$ [, вн-ссылка$_2$, ...]),

* (вн-ссылка$_1$ [, вн-ссылка$_2$, ...]),

где имя – DD – имя оператора DD, описывающего библиотеку; имя – раз$_k$ – основное или альтернативное имя, применяемое для разрешения внешних ссылок; k=1, 2, ...–внешняя ссылка, которая не может быть разрешена средствами автовызова на данном шаге редактирования. При указании символа * перед перечнем внешних ссылок вводится запрет на разрешение внешних ссылок средствами автовызова на всех шагах редактирования.

Предложение LIBRARY в отличие от опции NCAL Редактора связей позволяет осуществить запрет только на перечисленные модули.

Наборы данных дополнительного ввода используются для разрешения тех внешних ссылок, которые не были разрешены Редактором связей при обработке данных основного ввода. Такими ссылками, например, являются ссылки к стандартным программам библиотеки языка ФОРТРАН.

Дополнительный ввод может содержать объектные и загрузочные модули, а также управляющие операторы Редактора связей. Объектные модули и управляющие операторы могут размещаться в последовательных и библиотечных наборах данных, загрузочные модули – только в библиотечных наборах данных. Данные дополнительного ввода могут быть включены 116
в загрузочный модуль из библиотеки, определенной оператором DD с именем SYSLIB или из библиотеки, указанной в управляющем предложении Редактора связей LIBRARY. Кроме того, данные дополнительного ввода могут быть включены в загрузочный модуль по управляющему предложению Редактора связей INCLUDE. Такими данными в этом случае могут быть последовательные и библиотечные наборы данных.

Средства дополнительного ввода Редактора связей позволяют упростить описание основного ввода при нескольких вводимых наборах данных, так как средство сцепления операторов DD не всегда удобно.

Дополнительный ввод с помощью управляющего предложения Редактора связей INCLUDE задается следующим образом:

INCLUDE имя - DD$_1$ [(имя - раз$_{1,1}$ [, имя - раз$_{1,2}$)] ...
[имя - DD$_2$ [(имя - раз$_{2,1}$[, имя - раз$_{2,2}$],...)],

где имя – DD$_k$ – имя оператора DD, описывающего последовательный или библиотечный набор данных; имя - раз$_k$ - основное или дополнительное имя раздела, определенное в k-м операторе DD.

Если при редактировании объектных модулей из основного и дополнительного ввода останутся неразрешенные внешние ссылки, то Редактор связей автоматически просматривает указанные ему библиотеки и пытается обработать неразрешенные внешние ссылки. Если из основного и дополнительного ввода будут прочитаны несколько программных секций с одинаковыми именами, то в выходной модуль включается только первая прочитанная программная секция, остальные игнорируются.

Промежуточный набор данных с именем SYSUT1 применяется для внутренних нужд Редактора связей. Этот набор должен быть последовательным. Он всегда располагается на устройстве прямого доступа (обычно на магнитном диске).

Редактор связей выдает два вида выходной информации: загрузочные модули и диагностическую информацию.

Библиотека загрузочных модулей с именем SYSLMOD всегда создается на магнитном диске. Она может быть постоянной или временной. Имя библиотеки указывается в параметре DSNAME оператора DD; имя раздела можно не указывать. Имя раздела, являющееся именем загрузочного модуля, можно указать в управляющем предложении Редактора связей NAME. Если имя раздела не назначено ни в операторе DD, ни в предложении NAME, то Редактор связей назначает временное имя.

Диагностический выходной набор данных с именем SYSPRINT является последовательным и предназначен для листинга Редактора связей.

Диагностическая информация включает сообщения об ошибках и предупреждающие сообщения, а также сведения о расположении модуля и распечатку таблицы перекрестных ссылок. Распечатка распределения памяти содержит информацию о программных секциях, входящих в загрузочный модуль. Редактор связей выдает следующие сведения о программной секции: имя, относительный начальный адрес, длину, номер сегмента для программ с перекрытием, имена и

адреса точек входа. Для Редактора связей каждая программная единица (основная программа или подпрограмма) и каждая общая область в языке ФОРТРАН являются программной секцией. Программные секции, вызываемые из наборов данных дополнительного ввода, отмечаются звездочкой, которая следует за именем секции, В таблицу перекрестных ссылок входят следующие элементы: адрес слова, содержащего внешнее имя; внешнее имя; имя программной секции, включающей ссылку к внешнему имени; номер сегмента, куда входит программная секция с внешним именем (если имеются перекрытия). Неразрешенные внешние имена отмечаются текстом ¤ UNRESOLVED. Таблица перекрестных ссылок следует за распечаткой распределения памяти.

Все опции Редактора связей, кроме опции, указывающей размер области памяти, задаются ключевыми словами. Для этих опций, в отличие от опций транслятора существует только одна форма - утвердительная или отрицательная. Противоположная форма принимается по умолчанию.

Перечень основных опций Редактора связей приведен в табл. 4.2.

Редактор связей имеет 16 опций, разделяемых на две группы: управляющих и листинга. *Управляющие опции* приписывают модулю атрибуты, записываемые в оглавлении библиотеки загрузочных модулей вместе с именем модуля, определяют особенности структуры модуля и задают режим редактирования. *Опции листинга* определяют состав листинга. Некоторые опции несовместимы, например такие, как OVLY и SCTR, XREF и MAP, XREF и NE, MAP и NE. Другие опции должны использоваться совместно, например опция XCAL применима только вместе с опцией OVLY.

Таблица 4.2

Наименование опций	Выполняемые действия
LIST	Передача в системную печать обработанных высказываний Редактора связей
MAP	Запрос на печать плана выработанного загрузочного модуля
XREE	Печать плана и списка внешних перекрестных ссылок
LET	Присвоение загрузочному модулю статуса «выполнимый», несмотря на наличие некоторых специально оговоренных ошибок. При сборке программы из модулей не все внешние ссылки были разрешены Редактором связей, но программист знает, что в данном варианте выполнения нет обращений к неподключенным модулям
NCAL	Запрет на разрешение внешних ссылок из набора данных дополнительного ввода (SYSLIB); модуль помечается как выполнимый
OVLY	Определение оверлейной структуры загрузочного модуля, т.е. структуры с перекрытием
RENT	Пометка модуля как реентерабельного, т.е. имеется возможность применения одной копии модуля одновременно несколькими программами
REUS	Пометка модуля как повторно используемого, т.е. модуль обладает свойством самовосстановления перед повторным использованием
REFR	Объявление модуля рефрежерабельным, т.е. модуль не меняет своей логики работы, если в процессе выполнения его прервать, затем загрузить потому же адресу памяти и передать управление в точку прерывания
TEST	Проверка модуля программой ТЕСТРАН

§ 4.4. ЗАГРУЗЧИК

При обработке программ простой структуры, как правило, все возможности Редактора связей не используются. Нередко бывает так, что сразу после редактирования необходимо выполнить полученный загрузочный модуль. В этом случае промежуточная запись загрузочного модуля в библиотеку на магнитном диске, последующая его загрузка в оперативную память, а также повторная корректировка адресных констант программной выборки оказываются излишними операциями. Кроме того, вынужденное разделение редактирования и выполнения на два пункта

задания приводит к дополнительным затратам времени в системе на работу управляющей программы. Для уменьшения затрат машинного времени в тех случаях, когда исполнение программы осуществляется непосредственно после ее редактирования, обычно вместо Редактора связей применяется Загрузчик.

Назначение программы «Загрузчик». *Загрузчик* - системная обрабатывающая программа, объединяющая основные функции Редактора связей и программы выборки в одном пункте задания. Загрузчик помещает находящиеся в его входном наборе данных объектные и загрузочные модули в оперативную память, объединяет их в единую программу, корректирует перемещаемые адресные константы с учетом фактического адреса загрузки и передает управление в точку входа созданной программы. Если после редактирования в созданной программе остались неразрешенные внешние ссылки, то для их разрешения в мультипрограммных режимах Загрузчик может организовать поиск недостающих модулей в общей области резидентного пакета дисков. Далее поиск может быть продолжен в библиотеках автовызова. Загрузчик не может обрабатывать управляющие предложения Редактора связей и некоторые опции и не может создавать оверлейные программы. Обработанная программа загружается в оперативную память и тут же выполняется. Загрузочные модули при этом в библиотеку не записываются.

Загрузчик постоянно хранится в библиотеке SYS1.LINKLIB, содержащей все системные обрабатывающие программы. В библиотеке Загрузчик занимает практически вдвое меньший объем оперативной памяти, чем Редактор связей. Вызов Загрузчика осуществляется указанием его имени IEWDRGO в параметре PGM оператора EXEC или в макрокоманде динамического вызова.

Управление работой Загрузчика осуществляется посредством опций и параметров, которые записываются в параметре PARM оператора EXEC, вызывающего Загрузчик. Запись производится следующим образом:

PARM = ’список опций загрузчика/список параметров.

Элементы обоих списков разделяют запятыми. Знак «косая черта» всегда предшествует списку параметров объектной программы, даже если опции Загрузчика отсутствуют. Этот знак не применяется только в том случае, когда нет списка параметров. Список опций Загрузчика приведен в табл. 4.3.

Загрузчик использует следующие наборы данных: входной SYSLIN; библиотеки автовызова SYSLIB; диагностический выход SYSOUT.

Входной набор данных является обязательным и может содержать объектные или загрузочные модули. Объектные модули могут находиться в последовательных и библиотечных

Таблица 4.3

Наименования опций	Выполняемые действия
MAP/NOMAP	Печать списка внешних имен
CALL/NOCALL	Использование библиотеки автовызова
RES/NORES	Просмотр общей области резидента для разрешения внешних ссылок до просмотра библиотек автовызова; RES автоматически устанавливает CALL
LET/NOLET	Выполнение объектной программы даже при обнаружении в ходе ее формирования ошибок уровня 2 (код возврата 8)
PRINT/NOPRINT	Печать диагностических сообщений
SIZE-размер	Указание размера области памяти, которую Загрузчик может использовать для буферов, таблиц и размещения обрабатываемой программы (по умолчанию SIZE-100K)
EP-имя	Указание точки входа объектной программы

наборах данных, а загрузочные модули – только в библиотечных. Для библиотечных наборов данных нужно указать имена библиотеки и раздела. Для включения во входной набор данных нескольких модулей применяют сцепленные операторы DD.

Набор данных библиотеки автовызова необходим в том случае, если используются опции CALL или RES. Этот набор данных может состоять из нескольких сцепленных библиотек, однако все библиотеки должны содержать только объектные или только

загрузочные модули.

Диагностический выходной набор данных Загрузчика необходим в том случае, если применяется опция PRINT (или MAP). Этот набор данных должен быть обязательно последовательным. Он предназначен для листинга Загрузчика.

В пункте задания, в котором выполняется программа «Загрузчик», должны быть также определены все наборы данных объектной программы.

§ 4.5. СИСТЕМНЫЕ УТИЛИТЫ

В состав ОС ЕС входят вспомогательные обрабатывающие программы, называемые *утилитами,* позволяющие автоматизировать процедуру создания и ведения системных и пользовательских наборов данных. Одна и та же утилита может реализовывать различные функции, которые указываются посредством управляющих операндов этой утилиты.

Каждую утилиту можно вызвать или с помощью оператора EXEC, или с другой программы макрокомандами LINK или ATTACH.

При вызове утилиты требуемые тома или наборы данных описываются с помощью операторов DD. Задания для утилит определяются с помощью функциональных операторов, задаваемых во входном потоке в виде

```
//SYSIN DD *
<набор функциональных операторов>
/*
```

В отдельных случаях набор функций, выполняемых утилитами, доопределяется в параметре PARM оператора EXEC.

Функциональные операторы утилит имеют стандартный формат:

ИМЯ ОПЕРАЦИЯ $ОПЕРАНД_1$ [, $ОПЕРАНД_2$, ..., $ОПЕРАНД_N$] КОММЕНТАРИИ.

Поле ИМЯ может содержать произвольное символическое имя. Использование имени для всех программ не обязательно, за исключением программы IEHINITT. Имя записывается начиная с первой позиции бланка кодирования для всех утилит, кроме утилиты IEBUPDTE, в которой имя указывается начиная с третьей позиции бланка кодирования. В первых двух позициях в этом случае записывается последовательность символов /. Имя должно содержать от одного до восьми алфавитно-цифровых символов или специальных знаков, первым символом не должна быть цифра.

Поле ОПЕРАЦИЯ необходимо в любом случае. При отсутствии имени оно может начинаться со второй или четвертой колонки. От имени поле ОПЕРАЦИЯ отделяется пробелом.

Поле ОПЕРАНД 1 содержит ключевое слово и переменное значение, называемое *параметром.* Поля операндов разделяются запятыми.

Поле КОММЕНТАРИИ необязательное и следует после одного или нескольких пробелов. В этом поле программист может поместить требуемую информацию. Квадратные скобки выделяют операнды, которые не являются обязательными в операторе.

Утилиты находятся в системной библиотеке SYS1.LINKLIB и могут быть вызваны по своим именам. Принадлежность утилит к группам обслуживания системных наборов данных или пользовательских наборов данных определяется по первым трем символам в названии этих программ. Утилиты, названия которых начинаются первыми символами IEH, осуществляют обслуживание системных наборов данных. Символы IEB определяют утилиты, предназначенные для операций над пользовательскими наборами данных.

Группы утилит. По функциональным возможностям утилиты разбиты на несколько групп.

Утилиты IEHMOVE и IEBCOPY осуществляют копирование наборов данных.

Утилита IEHMOVE копирует наборы данных с последовательной, библиотечной и прямой организацией.

Программа IEBCOPY копирует только библиотечные наборы данных, кроме того, производит дополнительные операции: уплотнения библиотечных наборов данных, т. е. производится удаление всех свободных мест внутри набора данных; повторного создания наборов данных в случае необходимости увеличения отводимого под набор данных места на диске; создания одного набора данных из нескольких или всех разделов, принадлежащих другим наборам данных, и т. д.

Утилиты IEBGENER, IEBPTPCH, IEBUPDTE, IEHIOSUP преобразуют, выводят на печать и перфорируют наборы данных.

Программа IEBGENER позволяет получать копии последовательных наборов данных, преобразовывать последовательный набор данных в библиотечный и копировать записи из последовательного набора в библиотечный. При этом одна или несколько записей последовательного набора данных образует разделы библиотечного набора.

Эта программа также осуществляет редактирование как отдельных записей, так и полного набора. Редактирование может заключаться в замене символьной информации, преобразовании форматов представления десятичных данных, переводе кода КОИ-8 в ДКОИ.

Входной набор данных описывается DD-предложением с именем SYSUT1, выходной – SYSUT2, набор данных для протокола – SYSPRINT, набор данных с управляющими операторами – SYSIN. Если производится только копирование набора данных (без преобразования записей), то управляющие операторы утилиты не нужны. Однако присутствие оператора DD с именем SYSIN обязательно.

Программа IEBPTPCH позволяет выдавать на печать и перфорацию последовательные или библиотечные наборы данных. Имеется возможность выбирать требуемые разделы или записи наборов данных для случаев, когда нет необходимости выдавать на печать или перфорацию весь набор данных. На печать или перфорацию данные выдаются в шестнадцатеричном или алфавитно-цифровом представлении. Для выполнения этой утилиты необходимы операторы DD, аналогичные тем, что применяются в утилите IEBGENER.

Наиболее часто используемыми управляющими операторами этой утилиты являются PRINT, PUNCH, TITLE, RECORD, MEMBER.

Оператор PRINT (оператор PUNCH) указывает на то, что данные должны быть распечатаны (отперфорированы). Он является обязательным и должен быть первым среди управляющих операторов утилиты. Оператор PRINT (оператор PUNCH) имеет следующие основные операнды:

TYPORG - PO - входной набор данных имеет библиотечную организацию (если TYPORG отсутствует, подразумевается последовательная организация);

MAXNAME - n - возможное число имен разделов во входном наборе данных;

MAXFLDS - n - общее число операндов FIELD последующих операторов RECORD;

MAXLINE - n - число строк на странице (по умолчанию n = 60).

Оператор TITLE необязателен и предназначен для печати заголовка. Этот оператор имеет обязательный операнд ITEM, имеющий следующий вид:

ITEM = (заголовок, позиция).

Заголовок может быть длиной до 40 символов и печататься с указанной позиции или с первой, если позиция не задана.

Оператор RECORD при необходимости задает информацию для редактирования каждой записи. Если оператор RECORD не задан, то весь входной набор данных обрабатывается без редактирования.

Утилита IEBUPDTE предназначена для создания и обновления символьных библиотек. При этом можно вносить изменения в разделы библиотек или в последовательные наборы данных. Эта утилита имеет возможности, аналогичные утилитам IEBGENER, IEBCOPY: преобразования библиотечных наборов данных в последовательные и наоборот; создания копий, замены разделов или наборов данных. Рассматриваемая утилита имеет также большой набор функций по включению, замене и перенумерации записей библиотечных или последовательных наборов данных.

Для выполнения утилиты IEBUPDTE необходимы операторы DD, аналогичные тем, которые используются в утилите IEBGENER. Если при вызове утилиты указан параметр PARM-NEW, то это значит, что исходная информация задается только через управляющий набор данных (//SYSIN DD...) и оператор DD с именем SYSUT1 не нужен. Все управляющие операторы утилиты имеют в позициях 1 и 2 символы ./, в остальном они аналогичны управляющим операторам других утилит.

Оператор ADD указывает на то, что раздел или набор данных добавляется в выходной набор данных, а оператор CHANGE – на то, что набор данных или раздел модифицируется. Основные операнды этих операторов следующие:

NAME – имя – указывает имя раздела;

LIST – ALL - помещает обновленный набор данных или раздел в протокол утилиты (набор данных, описанный оператором //SYSPRINT DD...), при отсутствии операнда в протокол помещаются только модифицированные записи;

UPDATE = INPLACE – используется только с оператором CHANGE и указывает, что модифицированный набор данных будет располагаться на том же месте, что и входной набор.

При этом оператор //SYSUT2 DD не нужен. Операнд можно применять только если количество записей после модификации набора не больше исходного.

Оператор NUMBER используется для добавления записей и их нумерации. Его основные операнды следующие:

SEQl = n – задается номер первой записи для обработки;
SEQ2 = n –указывается номер последней записи для обработки;
SEQ1=ALL – обрабатывается весь набор данных или раздел;
NEW1 = n – задается первый номер, присваиваемый перенумерованным записям;
INCR=n – указывается шаг нумерации;
INSERT = YES- добавляются в набор новые записи, содержащие пробелы в поле нумерации.

Оператор DELETE предназначен для удаления записей, при этом применяются только операнды SEQ1 и SEQ2.

Оператор ENDUP означает конец управляющего набора данных и не имеет операндов.

Утилита IEHIOSUP предназначена для операций над системной библиотекой SYS1. SVCLIB. Она позволяет модифицировать адреса в этой библиотеке или включить в SYS1. SVCLIB новые программные модули. Эта библиотека в ОС ЕС способствует реализации функций супервизора и поэтому включает много различных встроенных таблиц, содержащих абсолютные адреса загружаемых в тех или иных случаях модулей. Утилита IEHIOSUP используется при генерации операционной системы.

У т и л и т ы IEHPROGM, IEHLIST предназначены для обслуживания системных наборов данных.

Программа IEHPROGM служит для модификации каталога и логической структуры томов прямого доступа. При модификации предусматриваются операции: каталогизации набора данных; исключения из каталога набора данных; создания нового индекса в каталоге; исключения существующего индекса из каталога; присвоения псевдонима старшему индексу; удаления ранее присвоенного псевдонима; расширения каталога на другие тома прямого доступа; устранения расширения каталога; создания индекса группы поколений данных.

Последовательные, библиотечные, индексно-последовательные наборы данных, наборы данных с прямой организацией, а также отдельные разделы библиотек могут быть удалены с томов данных. Программа позволяет переименовать наборы данных и разделы библиотек.

Программа IEHLIST позволяет распечатывать содержание каталога, оглавления тома прямого доступа и библиотечного набора данных. Исходными данными для этой утилиты являются оператор DD, в котором описывается требуемый том и при необходимости имя набора данных, а также набор управляющих операторов, вводимых во входном потоке после карты //SYSIN DD *.

Результат выполнения программы - распечатка требуемой информации. В этом же наборе данных печатаются сообщения об имеющихся ошибках.

Программа IEHLIST вырабатывает код возврата, с помощью которого классифицируются результаты выполнения программы. Код ОО означает успешное завершение, 08 - возникновение ошибки, в результате которой запрос игнорируется, продолжение обработки по остальным управляющим предложениям утилиты, 16 - некорректируемую ошибку, возникшую во время чтения требуемых наборов данных. Выполнение задания прекращается.

При составлении пакета задания для вызова утилиты в соответствии с требованиями, изложенными в задании на исполнение лабораторной работы, формируется оператор JOB.
Вызов утилиты осуществляется с помощью предложения

```
//EXEC    PGM = IEHLIST
```

Далее следуют операторы DD, определяющие тома требуемых наборов данных:

```
// любое - имя - 1    DD UNIT = ,VOL = SER = ,DISP = OLD
```

ИЛИ

```
// любое - имя - 2    DD UNIT = ,VOL=SER = ,DISP = OLD
```

Первый оператор применяется для указания постоянно смонтированных томов. Любое имя 1 определяет имя DD в соответствии с синтаксисом языка управления заданиями, которое должно заканчиваться единицей. Значения параметров UNIT и VOL формируется по обычным правилам.

Второй оператор задается в случае монтируемых томов. В этом операторе имя должно заканчиваться двойкой, значения параметров определяются так же, как для первого оператора DD.

Для выдачи результатов в системную печать класса А применяется оператор

```
//SYSPRINT    DD SYSOUT=A 126
```

Задания для утилиты определяются с помощью управляющих операторов утилиты.

Для распечаток каталога тома используется оператор, формат которого следующий:

```
[имя] LISTCTLG [VOL = устройство = серийный,] [NODE = имя]
```

Здесь квадратные скобки служат признаком того, что значение в них можно опустить; имя идентифицирует запрос программиста на выполнение работы; операнд VOL=устройство=серийный задает номер тома, на котором требуется распечатать каталог (если операнд опущен, то выводится на печать системный каталог); операнд NODE = имя определяет распечатку подкаталога, исключающего наборы данных, в состав имен которых входит указанное имя (если операнд опущен, то печатается весь каталог).

Для распечаток оглавления тома применяется оператор, формат которого следующий:

```
[имя]    LISTVTOC [VOL = устройство = серийный]
                  DSNAME = (имя1[, имя2]...),]
                  [DATE = DDYYY,]
                  [DUMP,] [FORMAT]
```

Здесь имя идентифицирует запрос программиста; операнд VOL=устройство = серийный определяет том, оглавление которого должно быть распечатано (если операнд опущен, то это значит, что предполагается резидентный том); операнд [DSNAME= (имя1[,имя2]...),] задает имена наборов данных, которые должны быть распечатаны в оглавлении тома (если операнд опущен, то печатаются все имена, входящие в оглавление тома); операнд DATE=DDDYY указывает на то, что каждый набор данных, срок хранения которых истек до указанной даты, должен быть отмечен в листинге символом *; DDD – порядковый номер дня года, YY – последние две цифры года (если операнд опущен, то просроченные наборы данных не отмечаются); DUMP и FORMAT – запрашивается для печати результатов соответственно неотредактированный и отредактированный формат. Если операнды DUMP и FORMAT опущены, то для выдачи результатов используется частично отредактированный формат.

Для распечатки оглавления библиотеки применяется оператор LISTPDS, формат которого следующий:

```
[имя]    LISTPDS [VOL=устройство = серийный,]
                  DSNAME=(имя1[,имя2]...)
                  [FORMAT,] [DUMP]
```

Назначения операндов VOL =...,FORMAT и DUMP, а также идентификатора «имя» аналогичны использованию в операторах LISTCTLG и LISTVTOC. Операнд DSNAME=(имя1[,имя2]...) специфицирует полные составные имена библиотечных наборов данных, оглавления которых должны быть распечатаны.

У т и л и т ы IEBCOMPR, IEBDG, IEBEDIT, IEBISAM применяются для осуществления операций над наборами данных.

Программа IEBCOMPR сравнивает наборы данных между собой с последовательной или библиотечной организацией. Сравниваемые наборы данных могут находиться на разных носителях и иметь типы записей фиксированной, переменной или неопределенной длины. Коэффициент блокирования записей тоже может быть различным.

Программа IEBDG позволяет создавать тестовые наборы данных с последовательной, индексно-последовательной и библиотечной организацией. Записи в создаваемых тестовых наборах данных можно формировать по стандартным образцам или заполнять собственной тестовой информацией. Перед занесением записи в тестовый набор данных допускается ее проверка и модификация,

Программа IEBEDIT применяется для автоматической переформировки входного потока заданий с учетом условий, описанных в этой программе, и возможных аварий во время

выполнения этого потока заданий.

Программа IEBISAM используется для создания копии индексно-последовательного набора данных на другом томе. При создании такой копии происходит реорганизация набора – освобождается область переполнения, переобозначаются индексы. В результате создается аналогичный индексно-последовательный набор данных, но с лучшими эксплуатационными характеристиками. Эта программа также может преобразовывать индексно-последовательный и последовательный набор данных, размещенный на диске или магнитной ленте.

Набор данных, размещенный на томе прямого доступа, используется для вспомогательных целей. Размещение индексно-последовательного набора данных на магнитной ленте позволяет перевозить информацию на различные вычислительные центры в компактном виде. Выгрузка и восстановление индексно-последовательного набора данных вместе с реорганизацией при необходимости осуществляются этой же утилитой. Важной является возможность распечаток записей индексно-последовательного набора данных на АЦПУ. Для работы утилиты управляющие операторы не нужны.

При использовании утилиты оператор EXEC записывается следующим образом:

```
//EXEC   PGM = IEBISAM
```

Функции утилиты задаются параметром PARM оператора EXEC, который принимает следующие значения:

PARM=COPY – операция копирования; при копировании индексно-последовательного набора данных с одного тома на другой записи, предназначенные для удаления, автоматически удаляются, а записи, содержащиеся в области переполнения, заносятся в основную область копии;

PARM=UNLOAD – операция создания последовательной копии, помещаемой на том магнитной ленты или прямого доступа. Эта копия состоит из записей длиной 80 байт и содержит данные и управляющую информацию; записи, предназначенные для удаления, не копируются;

PARM=LOAD – операция создания индексно-последовательного набора данных из последовательного, полученного в результате выполнения операции UNLOAD, при этом записи, содержащиеся в области переполнения исходного индексно-последовательыюго набора, заносятся в основную область; PARM = PRINT[,N] – операция печати; если N не указано, содержимое каждой записи перед печатью преобразуется в шестнадцатеричное представление; если N указано, то преобразование не производится.

Операторы DD с именем SYSUT1, SYSUT2, SYSPRINT имеют тот же смысл, что и в утилите IEBGENER.

У т и л и т а IEHDASDR относится к группе утилит, связанных с обслуживанием томов, наиболее часто применяемых для хранения информации (пакетов, дисков, магнитных лент), а также с тиражированием информации, хранящейся на этих томах.

Утилита IEHDASDR осуществляет инициализацию тома прямого доступа с заменой дефектных дорожек на альтернативные; замену регистрационного номера тома; создание копии тома прямого доступа (дампа) на томе прямого доступа или магнитной ленте; выгрузку информации, расположенной на магнитной ленте, содержащей копию тома прямого доступа, на том прямого доступа; распечатку на АЦПУ содержимого тома или группы дорожек.

Исходной информацией для выполнения утилиты являются набор данных и при необходимости текст программы IPL (программы первоначальной загрузки). Утилита использует оператор EXEC в следующем виде:

```
//EXEC   PGM = IEHDASDR [,PARM = (список подпараметров)]
```

Список подпараметров может содержать следующие компоненты: LINECNT = XX, указывающий число строк на странице при печати сообщений (если он опущен, число строк на странице равно 58); N=n-максимальное число одновременно выполняющихся операций разметки томов (если этот подпараметр опущен, то число одновременно выполняющихся операций равно числу последовательных одинаковых управляющих операторов во входном потоке).

В описании шага задания применяются следующие операторы DD:

```
//SYSPRINT DD - определяющий последовательный набор данных для сообщений;
//имя тома DD - описывающий размечаемый том;
//SYSIN DD - определяющий набор данных, содержащий управляющие операторы утилиты.
```

Для управления функциями утилиты используются следующие управляющие операторы:

ANALIZE, FORMAT, LABEL, GETALT, IPLTXT.

Оператор ANALIZE выполняет следующие действия: размечает каждую дорожку тома, записывая ее собственный адрес и нулевую запись RO (остальная часть дорожки стирается); анализирует дорожку перед разметкой (если дорожка оказалась дефектной, то ей назначается альтернативная дорожка); строит записи IPL (R1 и R2) на нулевой дорожке тома; строит метку тома (запись R3) на нулевой дорожке и записывает в нее при необходимости информацию о владельце тома; строит оглавление тома (VTOC), размер и местоположение которого определяются программистом. Кроме того, при разметке тома, который будет в дальнейшем применяться как резидентный, на правую дорожку необходимо записать программу IPL. Оператор ANALIZE имеет следующие обязательные операнды: TODD, VTOC, EXTENT.

Операнд TODD записывается в виде

$$\text{TODD}=\left\{\begin{array}{l}(\text{CUU},...)\\ \text{имя DD},...\end{array}\right\}$$

Этот операнд используется для впервые производимой разметки тома и указывает адрес канала и номер устройства, содержащего размещаемый том.

Оператор TODD=(имя DD,...) используется при повторной разметке тома, указывает имя оператора DD, описывающего размечаемый том.

Операнд VTOC имеет вид

VTOC = xxxxx

и указывает десятичный относительный адрес дорожки, представляющий первую дорожку, с которой начинается оглавление тома.

Операнд EXTENT записывается в виде

EXTENT=xxxxx

и указывает длину VTOC в дорожках.

Необязательные параметры следующие:

NEWVOLID=серийный номер обозначает серийный номер тома (обязателен при первичной разметке тома);

IPLDD = имя DD – имя оператора, описывающего набор данных, содержащий программу;

PASSES=0 указывает на то, что осуществляется так называемая быстрая инициализация тома, пропускающая анализ и разметку дорожек, записывающая только Rl, R2 и R3 нулевой дорожки, VTOC и текст IPL, если требуется. Быстрая инициализация применяется при повторной разметке томов;

OWNERID указывает информацию о владельце тома, помещаемую в метку тома (запись R3 нулевой дорожки).

Оператор FORMAT отличается от оператора ANALIZE тем, что не выполняются операции анализа дорожек и назначения альтернативных дорожек, поэтому его нельзя применять для первой разметки тома. Этот оператор имеет те же операнды, что и ANALIZE, кроме операнда PASSES=0.

Оператор LABEL позволяет изменить серийный номер размеченного тома и по желанию записать информацию о владельце тома в метку тома. Оператор имеет два обязательных операнда

$$\text{TODD}=\left\{\begin{array}{l}\text{имя}\\ \text{имя DD}\end{array}\right\},\ \text{NEWVOLID=серийный номер}$$

и один необязательный OWNERID=имя. Смысл операндов тот же, что и в операторе ANALIZE.

Оператор GETALT назначает указанным дорожкам тома независимо от того, являются они дефектными или нет, альтернативные дорожки и применяется только для уже размеченных томов. Этот оператор имеет два обязательных операнда TODD=имя DD и TRACK =ccccхххх, указывающих в шестнадцатеричной системе счисления номер цилиндра (cccc) и дорожки (hhhh), для которой требуется альтернативная дорожка.

Оператор IPLTXT применяется для отделения операторов ANALIZE и FORMAT от текста программы IPL, если она включена во входной поток.

Программа IEHINITT выполняет подготовку инициализации магнитной ленты к работе с операционной системой со стандартными метками. Эта подготовка заключается в записи

стандартной метки тома и фиктивной начальной метки первого набора данных и ленточного маркера.

Оператор EXEC записывается для этой утилиты следующим образом:

```
//EXEC PGM = IEHINITT [,PARM = LINECNT = nn]
```

Подпараметр LINECNT=nn указывает на то, что на странице печатается от 60 до 90 строк. Утилита использует управляющий оператор INITT. Его имя обязательно кодируется и должно совпадать с именами оператора DD шага задания. Таким образом осуществляется связь между соответствующим устройством и управляющим оператором.

Оператор INITT имеет один обязательный операнд SER=xxxxxx, определяющий серийный номер тома. Если размечается несколько томов, то им присваиваются последовательные номера. Необязательные операнды оператора INITT следующие:

OWNER=имя задает информацию (длиной до 10 символов) о владельце;

NUMBTAPE=nn указывает число размечаемых томов (от 1 до 255). При отсутствии операнда размечается один том;

DISP=REWIND указывает на то, что после разметки ленту нужно перемотать;

DISP=UNLOAD указывает на то, что после разметки ленту надо разгрузить. При отсутствии параметра лента разгружается.

Глава 5. ОРГАНИЗАЦИЯ ПРОЦЕДУР ОПЕРАЦИОННЫХ СИСТЕМ

В главе приводятся функции языка процедур, служащего для сокращения количества используемых управляющих операторов в процессе составления задания на обработку информации. Рассматривается использование символических параметров, а также специфика замещения и добавления параметров в процедурах. Описывается применение процедур при организации вычислительного процесса, а также процедур трансляции, редактирования связей, запуска на исполнение, используемых при составлении программ на алгоритмических языках.

§ 5.1. ПРОЦЕДУРЫ И ИСПОЛЬЗОВАНИЕ СИМВОЛИЧЕСКИХ ПАРАМЕТРОВ

Процедура - заранее составленная последовательность операторов PROC, EXEC, DD, PEND языка управления заданиями, которые предназначены для сокращения количества необходимых управляющих операторов при составлении задания.

Процедуры. Процедура может вызываться в задании с помощью одного оператора EXEC или планироваться как отдельное задание с помощью команды оператора ОС ЕС START. С этой точки зрения процедуры условно разделяют на два класса. Командой START вызываются следующие процедуры: системного ввода RDR, инициатора INIT, системного вывода WTR.

К о м а н д о й E X E C вызываются процедуры, использующиеся в задании.

Первым оператором в процедуре является оператор процедуры PROC со следующим форматом:

```
//имя процедуры PROC символические параметры комментарий
```

Далее следуют операторы EXEC и DD. Каждый оператор EXEC и соответствующие ему операторы DD представляют шаг процедуры. Оператор EXEC определяют программу, которая должна выполняться на этом шаге, а операторы DD – данные для этого шага.

Процедуры могут быть *каталогизированы,* т. е. записаны в специальную библиотеку процедур SYS1.PROCLIB, или находиться во входном потоке заданий. В каталогизированной процедуре оператор PROC не обязателен, так как имя процедуры указано в каталоге. Каталогизированные процедуры не могут содержать следующие операторы: EXEC, обращающийся к другим каталогизированным процедурам; DD с именем JOBLIB; DD с параметрами * или DATA.

Процедура во входном потоке заданий ставится сразу за операторами JOB и DD с именем JOBLIB. Она должна отделяться от других операторов задания оператором конца процедуры PEND, имеющим следующий формат:

```
// имя процедуры    PEND    комментарий
```

Обычно задания, содержащие большое число часто применяемых операторов, помещают в библиотеку процедур. С целью отладки процедуры ставят, как правило, во входной поток. Проверенные процедуры записываются в библиотеку процедур с помощью утилиты IEBUPDTE. Для обращения к процедуре в операторе EXEC используется ее имя:

```
// EXEC    <имя процедуры >
```

ИЛИ

```
//EXEC   PROC=<имя процедуры>
```

При вызове процедуры в задании может потребоваться ее временная модификация, т. е. изменение некоторых ее параметров. Осуществление указанного изменения возможно путем введения в процедуру символических параметров либо замещением и добавлением операторов DD в шаге задания, вызывающем процедуру.

Использование символических параметров. Символические параметры позволяют модифицировать процедуру заданием определенных значений этих параметров в операторе EXEC, вызывающем процедуру. Символические параметры могут ставиться в полях операндов, операторов процедуры PROC, EXEC, DD. Символический параметр должен начинаться знаком & (амперсанд), после которого может быть от 1 до 7 буквенно-цифровых символов, включая такие символы, как ¤, #, @. При этом первым символом не может быть цифра. При вызове процедуры оператором EXEC символические параметры в процедуре заменяются на заданные в этом операторе значения. Присваивание значений производится путем записи в поле операндов оператора EXEC выражений вида

```
< символический параметр > = < значение >
```

Если среди символов значения параметра имеются спецсимволы, то значение необходимо указать в апострофах. Символические параметры в операторах процедуры можно записывать подряд без запятой:

```
// NAME   DD    DSN=&DA&DB,...
```

Пример 5.1. Рассмотрим следующую процедуру:

```
// IEBGENER PROC
//            EXEC PCM = IEBGENER
// SYSIN        DD      DUMMY
// SYSPRINT   DD      DUMMY
// SYSUT1       DD      DISP = SHR, UNIT=SYSDA, VOL=SER=&V, DSN=&D
// SYSUT2       DD      DISP = SHR, UNIT=SYSDA, VOL=SER=&VT, DSN = &DT
```

Процедура с именем IEBGENER описывает вызов утилиты IEBGENER, которая в данном случае переписывает набор данных, заданный оператором DD с именем SYSUT1, в набор данных, описанный в операторе DD с именем SYSUT2. Имена наборов данных заданы символическими параметрами &D и &DT, серийные номера томов - символическими параметрами &V и &VT. При вызове процедуры в операторе EXEC указываются конкретные значения имен наборов данных RCS.OLD и RCS.NEW и серийные номера томов UD001 и UD002:

```
// EXEC IEBGENER, D = ’RCS.OLD,’V = UD001, DT=’RCS.NEW,’
//         VT=UD002
```

В ряде случаев в конце символического параметра необходимо ставить особый ограничитель в виде точки. Точка требуется в тех случаях, если после символического параметра стоит некоторая постоянная информация; если следующим символом является буква, цифра, левая скобка, точка; если символический параметр позиционный, и за ним следуют другие параметры. При интерпретации задания точка рассматривается как ограничитель и не включается в состав символического параметра.

Пример 5.2. Задан фрагмент процедуры:

```
// PROC1    PROC
//              EXEC ...
// Dl           DD     DSN = & A. BC, ...
```

В процедуре PROC1 введен символический параметр &A для задания начальной части имени набора данных. Конец имени набора данных (BC) изменяться не будет. В следующих операторах EXEC вызывается указанная процедура, причем в первом операторе набор данных процедуры получает имя AAABC, а во втором - BBBBC:

```
// EXEC     PROC1, A=AAA
// EXEC      PROC1, A = BBB
```

Пример 5.3. Рассмотрим процедуру

```
// PROC2   PROC
// D2       DD    &POSIT.DSN = PROGRAM, DISP = OLD
```

В процедуре PROC2 в операторе DD с именем D2 позиционный параметр задан как символический. После него следуют другие параметры, поэтому он отделяется от них точкой, а не запятой. Чтобы при замене символического параметра его значением не произошло ошибки, запятая включается в значение параметра:

```
// EXEC     PROC2, POSIT='DUMMY',
```

В этом случае оператор DD в процедуре PROC2 после обработки программой RDR будет иметь вид

```
// D2       DD    DUMMY, DSM=PROGRAM, DISP = OLD
```

Значения символических параметров могут быть заданы и в операторе PROC. Эти значения будут стандартными, т. е. использоваться по умолчанию при вызове процедуры. При этом указанные стандартные значения могут быть заменены на другие в операторе вызова процедуры EXEC.

Пример 5.4. Дана процедура

```
// PROCA   PROC   RG = 100 K, U = SYSDA
           EXEC   PGM = SUBR, REGION = &RG
// DA1      DD       DSN=&D,UNTT=&U, DISP = OLD, VOL=SER=&V
```

В операторе PROC процедуры PROCA заданы стандартные значения символических параметров RG и U. Первый параметр определяет необходимую область памяти -100 К, второй - устройство прямого доступа SYSDA. Стандартные значения параметра D, задающего имя набора данных, и параметра V, определяющего серийный номер тома, не заданы. При вызове процедуры значения символических параметров D, U, V задаются следующим образом:

```
// EXEC PROCA,D= RCS.SOUR,      U = 322, V = BOSH
```

В процедуру передаются имя набора данных, описанного в операторе DA1 - RCS.SOUR, серийный номер тома BOSH и устройство - 322. Значение параметра U = 322 заменяет стандартное значение U = SYSDA, заданное в операторе PROC. Так как при вызове процедуры символический параметр RG умалчивается, то берется его стандартное значение RG = 100 K.

Символический параметр можно аннулировать. Для этого в операторе EXEC, вызывающем процедуру, или в операторе PROC указывают этот параметр и знак равенства, например

```
// EXEC PROC=PRl, ABC=
```

В этом случае символический параметр &ABC удаляется из всех шагов процедуры PR1,

Пример 5.5. Рассмотрим процедуру

```
// PROCB    PROC   U=5067, V=UD001, D =
//          EXEC   PGM=PROGB
// DD1        DD       DSN = &D,DISP = SHR,UNTT=&U,VOL = SER=&V
```

В процедуре PROCB введенные символические параметры U и V получают соответственно стандартные значения: устройство - накопитель на магнитном диске 5067; серийный номер тома - UD001. Символический параметр D для задания имени набора данных аннулируется. Это позволяет указать в первом операторе процедуры PROC все символические параметры, что облегчает работу с процедурой.

```
// EXEC      PROCB, D= 'TEST.ONE.MY', V = , U =
```

В операторе вызова процедуры задается значение параметра D и аннулируются параметры V и U. Такая запись правомерна только для каталогизированных наборов данных, так как если параметры < устройство > и < серийный номер тома> не указаны, программа управления заданиями ищет набор данных с заданным именем в системном каталоге. Если набора данных в каталоге нет, то это приведет к снятию задания с обработки.

§ 5.2. ЗАМЕЩЕНИЕ И ДОБАВЛЕНИЕ ПАРАМЕТРОВ ОПЕРАТОРОВ В ПРОЦЕДУРАХ

В тех случаях, когда символические параметры, необходимые для модификации процедуры, не определены, в задании можно заменить или добавить параметры, стоящие в операторах EXEC процедуры, а также заменить или добавить операторы DD.

Изменение и добавление параметров в операторах EXEC процедуры. Изменение и добавление параметров производится в операторе EXEC, вызывающем данную процедуру, для чего после ключевого слова параметра через разделитель (точку) ставится имя шага процедуры. Сначала ставятся все замещающие параметры для первого шага процедуры, затем для второго шага и т. д.

Пример 5.6. Рассмотрим следующую процедуру:

```
// PROC3   PROC P =
// ST2       EXEC PGM = PGM2,COND = &C
// ST3       EXEC PGM = PGM3,PARM =  &P
```

В процедуре PROC на шаге ST2 определен символический параметр C для параметра COND, на шаге ST3 - символический параметр P для параметра PARM. Параметр PARM применяется для передачи управляющей информации выполняемой программе, в данном случае программе PGM3. В операторе вызова процедуры указываются значения этих параметров:

```
// EXEC      PROC3, ST2.C = EVEN, ST3, P=  'Q1, Q2'
```

В этом случае шаг процедуры ST2 выполняется даже если один (или более) предшествующий шаг завершился аварийно (параметр COND = EVEN), для программы PGM3 шага ST3 передается параметр PARM = 'Q1, Q2'.

Если во всех операторах EXEC процедуры нужно заменить параметры одной совокупностью значений, то в операторе EXEC вызова процедуры ставятся ключевые слова без имени шага. В том случае, когда параметра нет в каком-либо операторе EXEC процедуры, он добавляется. Параметр PARM применяется только к первому шагу процедуры, а параметр TIME – к процедуре в целом.

Пример 5.7. Разберем процедуру

```
// PROC4   PROC
// ST1       EXEC PGM = PROG1, PARM = &P
// ST2       EXEC PGM = PROG2, COND=&C
// ST3       EXEC PGM = PROG3, REGION = &R
```

Вызов процедуры может иметь следующий вид:

```
// EXEC     PROC4, P= 'AA.BB1' C=  '(8,GE),' R=128 K
```

Параметр P применяется только к шагу ST1, параметры R и C - ко всем шагам. Таким образом, при вызове процедуры заданы значения PARM= AA.BB для шага ST1 и условие выполнения всех шагов задания в зависимости от завершения предыдущих шагов: COND=(8,LE); определена максимальная величина основной памяти REGION = 128 K для всех шагов. Условие осуществления всех шагов означает, что любой шаг задания обходится, если какой-либо предыдущий шаг завершился с кодом возврата, большим восьми.

Замещение и добавление параметров в операторах DD процедуры. Для замещения и добавления параметров в задании после оператора вызова процедуры EXEC необходимо включить дополнительные операторы DD. В поле имени таких операторов DD должно стоять следующее:

<имя шага процедуры > . <имя оператора DD >

Для замещения оператора DD указываются имя шага, в котором стоит оператор, и его имя. Для добавления оператора DD задается имя шага, к которому он должен быть добавлен, и имя, отличное от других имен DD в этом шаге.

Пример 5.8. В процедуре компиляции с языка ФОРТРАН и редактирования связей FORTGCL на шаге LKED определяется временный набор данных для загрузочного модуля:

```
// LKED       EXEC   PGM = IEWL,...

// SYSLMOD DD DSNAME=&GOSET(MAIN), DISP = (NEW, PASS),
//                UNIT=SYSDA, SPACE=(1024,(20,10,1), RLSE), DSB = BLKSIZE=1024
```

Для записи загрузочного модуля в личную библиотеку необходимо заместить оператор DD с именем SYSLMOD:

```
// EXEC    FORTGCL...
// LKED.SYSLMOD    D DSN=MYLIB (TEST), DISP = SHR,
//        VOL=SER=UD001, SPACE=
```

При вызове процедуры в операторе DD с именем SYSMOD замещаются параметры <имя набора данных> (DSN), <диспозиция> (DISP), добавляется параметр < серийный номер тома > (VOL), отменяется параметр SPACE (чтобы не сработал параметр RLSE).

Пример 5.9. При вызове процедуры компиляции с языка ФОРТРАН FORTGC необходимо добавить оператор DD, описывающий набор данных во входном потоке, представляющий собой исходный модуль на ФОРТРАНе:

```
// EXEC   FORTGC
// SYSIN DD *
<исходный модуль >
/*
```

В задании на шаге вызова процедуры сначала должны стоять все замещающие операторы DD, а затем добавляющиеся операторы. Замещающие операторы DD должны стоять в том же порядке, что и соответствующие операторы в процедуре.

Часто в процедуре применяется параметр DDNAME, позволяющий отложить полное описание

набора данных до тех пор, пока не встретится оператор DD с именем, указанным в параметре DDNAME. В большинстве случаев это бывает необходимо тогда, когда процедура использует данные во входном потоке.

Пример 5.10. Рассмотрим процедуру с отложенным описанием набора данных:

```
// PROC5    PROC
// ST1      EXEC   PGM=AAA
// Dl       DD       DDNAME = INPUT
```

В шаге ST1 процедуры PROC5 используется набор данных, описание которого отложено до появления оператора DD с именем INPUT. В задании этот набор данных определяется как набор во входном потоке:

```
// EXEC   PROC = PROC5
// ST1.INPUT DD *
        <данные >
/*
```

Операторы каталогизированной процедуры включаются в выходной листинг в том случае, если в параметре MSGLEVEL оператора JOB первый параметр равен единице. В отличие от операторов задания операторы процедуры в первых двух позициях помечаются символами XX, если оператор не замещен, и символами X /, если некоторые параметры были замещены. В операторах комментария в первых трех позициях ставятся символы ***.

Аналогично, незамещенные операторы процедуры, стоящей во входном потоке, в первых двух позициях помечаются символами + +, замещенные операторы - символами + /. Символы *** появляются в первых трех позициях операторов комментария.

§ 5.3. ПЛАНИРОВАНИЕ СИСТЕМНЫХ ЗАДАЧ

Процедуры - универсальное средство работы для различных групп пользователей ЭВМ. Они применяются операторами ЭВМ и системными программистами, а также прикладными программистами, т. е. пользователями ЭВМ. Процедуры, которые можно вызывать с операторской консоли командой START, должны быть каталогизированы, т. е. записаны в библиотеку SYS1.PROCLIB. Командой START вызываются процедуры системных задач. Формат записи команды START(S) аналогичен оператору EXEC вызова процедуры из задания:

```
S <имя процедуры>, <параметры>
```

Использование параметров было рассмотрено в § 5.1 на примере оператора EXEC. При вызове системных процедур широко применяется прием умолчания. Общее число параметров может быть большим (до 30), при вызове процедуры обычно указывается не более трех параметров.

Системные задачи ввода заданий, инициализации задач, вывода выходных наборов данных запускаются на последней стадии загрузки операционной системы; запуск осуществляется автоматически или по команде оператора.

Запуск задачи системного ввода осуществляется вызовом процедуры системного ввода RDR. В качестве параметров можно указать адрес или тип периферийного устройства (параметр U), регистрационный номер тома (параметр V), имя набора данных (параметр D), номер файла и тип меток магнитной ленты (параметр L), например

```
S RDS. S,  U =
S RDS.S, U=5010, L=5, V=AAD01, D = CX
```

Если параметры не заданы, то ввод заданий осуществляется с устройства ввода с перфокарт, адрес которого задан по умолчанию. Если в заданиях применяется специальная библиотека процедур, то ее имя нужно указать в параметре DP, имеющем следующий вид:

```
DP= <имя библиотеки процедур>
```

При вызове процедуры RDR можно задать следующие параметры для обрабатываемых заданий: уровень системных сообщений; класс задания; приоритет задания; предельное время выполнения шага задания; объем оперативной памяти и др. По умолчанию перечисленные параметры имеют соответственно следующие значения: 0,1; A; 01; 30 мин, 50 К.

Запуск задачи инициатора осуществляется процедурой INIT. При этом можно задать классы заданий, обслуживаемых инициатором в параметре CL. По умолчанию инициатор

будет обслуживать задания класса A. Например, запуск задачи инициатора для заданий классов A, B, C имеет вид

```
S INIT, CL=ABC
```

Запуск задачи системного вывода выходных потоков запускает процедуру задачи системного вывода WR. При вызове можно указать выходной класс CL и адрес периферийного устройства для вывода U.

```
S WR, CL=B, U=00F
```

По умолчанию выводится выходной поток класса A на устройство вывода, определенное в процедуре.

§ 5.4. ПРОЦЕДУРЫ ТРАНСЛЯЦИИ, РЕДАКТИРОВАНИЯ СВЯЗЕЙ, ИСПОЛНЕНИЯ ПРОГРАММ ПОЛЬЗОВАТЕЛЯ

Для получения результатов работы программы, написанной на языке высокого уровня, необходимо написать задание, состоящее из следующих шагов: компиляции исходного модуля; редактирования связей полученного при компиляции объектного модуля; исполнения загрузочного модуля. На каждом шаге необходимо указать выполняемую программу, определить условия исполнения шага, описать применяемые при обработке наборы данных. Специальные каталогизированные процедуры для описания одного или нескольких шагов значительно облегчают пользователю процесс составления заданий. Ниже приводятся тексты некоторых каталогизированных процедур для языка ФОРТРАН и примеры их вызова.

Процедуры трансляции. Часто при начальной отладке программы необходима только стадия компиляции. В этом случае используется процедура FORTGC, где G – уровень компилятора, C – компиляция.

Пример 5.11. Каталогизированная процедура компиляции имеет вид

```
//FORT    EXEC PGM = IEYFORT, REGION=100 K
//SYSPRINT      DD   SYSOUT=A
//SYSPUNCH      DD    SYSOUT=B
//SYSLIN    DD DSNAME = &LOADSET, DISP = (MOD,PASS), UNIT=SYSSQ.
//SPACE=(80,(200, 100), RLSE), DCB = BLKSIZE=80
```

Процедура состоит из одного шага FORT, вызываемая на этом шаге программа - компилятор с языка ФОРТРАН IEYFORT. Определены выходные классы для устройства печати (A) и для устройства вывода на перфокарты (B). Оператор DD с именем SYSLIN описывает временный набор данных с именем &LOADSET для вывода объектного модуля, построенного компилятором. Этот набор данных передается последующим шагам (диспозиция PASS). Если набор данных нужно сохранить, например записать в библиотеку объектных модулей, то оператор DD с именем SYSLIN замещается. При вызове процедуры добавляется оператор DD, описывающий набор данных исходного модуля:

```
// EXEC   FORTGC // FORT. SYSIN DD*
   <исходный модуль на языке ФОРТРАН >
/*
```

Операторы DD с именами SYSPRINT.SYSPUNCH могут быть заменены. Например, для вывода выходной информации на экран дисплея при работе с системами диалоговой отладки нужно изменить выходной класс задания и выходного набора данных:

```
// JOB1 JOB (23-17), IVANOV, MSGCLASS=J
// EXEC FORTGC
// FORT.SYSPRINT DD SYSOUT=J
// FORT.SYSIN DD *
   < исходный модуль>
/ *
```

Процедуры трансляции и редактирования связей. Эти каталогизированные процедуры отличаются от процедур трансляции наличием второго шага редактирования связей.

Пример 5.12. Рассмотрим каталогизированную процедуру компиляции и редактирования связей FORTGCL. Первый шаг процедуры совпадает с процедурой FORTGC. На втором шаге LKED вызывается программа редактора связей LEWL:

```
// LKED EXEC PGM=IEWL, REGION = 100 K, PARM = (XREF,LET,
//              LIST), COND = (4,LT,FORT)
```

```
// SYSLIB DD DSNAME = SYS1.FORTLIB, DISP = SHR
// SYSLMOD DD DSNAME=&GOSET(MAIN), DISP=(NEW,PASS),
// UNIT=SYSDA, SPACE = (1024,(20,10,1),RLSE)
// SYSPRINT DD SYSOUT=A
// SYSUT1 DD DSNAME = &SYSUT1, UNIT=SYSDA,
// SPACE = (1024,(20,10), RLSE)
// SYSLIN DD DSNAME=&LOADSET, DISP=(OLD, DELETE)
```

В операторе EXEC с именем LKED заданы управляющие параметры для Редактора связей: XREF означает, что выводится таблица перекрестных ссылок; LET - объектный модуль считается выполнимым даже если он содержит ошибки; LIST - в распечатку включаются управляющие операторы Редактора связей. Шаг LKED обходится, если шаг компиляции FORT завершился с кодом возврата, большим четырех, т. е. были существенные ошибки при компиляции. Оператор DD с именем SYSLIB описывает библиотеку загрузочных модулей в языке ФОРТРАН, откуда могут быть взяты модули, вызываемые редактируемым объектным модулем. Полученный в результате редактирования загрузочный модуль помещается во временный набор данных на диске, описанный в операторе DD с именем SYSLMOD. Этот набор данных передается следующему шагу. Набор данных, описанный в операторе DD с именем SYSUT1, необходим для работы Редактора связей.

Процедуры трансляции, редактирования связей и исполнения. Эти процедуры состоят из трех шагов. Первые два шага аналогичны описанным в предыдущем примере.

Пример 5.13. Рассмотрим каталогизированную процедуру компиляции, редактирования связей и выполнения FORTGCLG. На третьем шаге процедуры исполняется загрузочный модуль, полученный в результате работы Редактора связей и описанный в шаге LKED в операторе DD с именем SYSLMOD:

```
// GO EXEC    PGM = * . LKED. SYSLMOD,
//              COND = ((4,LT,FORT), (4,LT,LKED))
// FT05F001     DD DDNAME=SYSIN
// FT06F001     DD SYSOUT=A
// FT07F001     DD SYSOUT=B
```

Шаг GO выполняется в том случае, если оба кода возврата предыдущих шагов меньше четырех. В процедуре определены стандартные ссылочные номера для наборов данных, используемых в операторах ввода-вывода языка ФОРТРАН. Номер 05 применяется для ввода данных с перфокарт. Соответствующий оператор DD имеет имя FT05F001. Описание этого входного набора данных отложено. Пользователь должен описать его в задании как данные во втором потоке в операторе DD с именем GO.SYSIN. Номер 06 используется для вывода результатов работы программы на печатающее устройство (выходной класс A), а номер 07-для вывода результатов на перфокарты (выходной класс B).

Если программист применяет нестандартные ссылочные номера, то он должен заменить соответствующий оператор DD процедуры. Выходные классы также могут быть заменены.

§ 5.5. ТЕХНОЛОГИЯ ИСПОЛЬЗОВАНИЯ ПРОЦЕДУР ПРИ ОРГАНИЗАЦИИ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА

Информацию о всех выполняемых в данный момент времени процедурах можно получить по команде оператора ЭВМ DISPLAY, для чего указывается имя задания или имя процедуры, запущенной с операторской консоли.

При исполнении некоторых процедур, запущенных с помощью команды START, необходимо вводить данные или задавать значения параметров. В этом случае данные вводятся по запросу ОС с той же консоли, с которой была вызвана процедура.

Выполнение процедуры можно прекратить с помощью команды оператора CANCEL, в которой указывается имя задания, если процедура была вызвана оператором EXEC, или имя процедуры. Если при исполнении процедуры были сообщения об ошибках, ее можно исполнить повторно, исправив ошибки.

При вызове процедуры обычно необходимо задать объекты, с которыми процедура работает. *Под объектами* понимаются пакеты магнитных дисков и магнитные ленты, наборы данных, внешние устройства и т. д. Каждый объект характеризуется группой параметров, значения которых нужно задать при вызове процедуры. Для каждой процедуры существуют основные и дополнительные объекты. Как правило, основные объекты необходимо определять при вызове, а дополнительные связаны со способом выполнения процедуры или редко используемыми возможностями. Обычно при вызове параметры дополнительных объектов не задают, а используют из значения по умолчанию.

Объекты одного типа, например пакеты магнитных дисков, новые наборы данных, характеризуются одной группой параметров. Чтобы различать эти параметры друг от друга, к именам параметров добавляется определенный суффикс. Например, суффикс W означает рабочий

набор данных, пакет магнитных дисков; P – набор данных для распечатки.

Имеются процедуры, позволяющие получать различные инструкции о вызове и об использовании процедур. Список всех процедур на консоль можно получить вызовом процедуры GVHT или HELP без параметров. Инструкцию по вызову всех процедур с выдачей их на печать можно получить, вызвав процедуры PRHT или LSHT без параметров. Для получения информации о вызове какой-либо одной процедуры применяются процедуры GVHT или HELP:

S { GVHT / HELP }, A=<имя процедуры>

Для каждой процедуры выдается следующая информация: имя процедуры; ее значение; группа пользователей, для которых предназначена процедура; дополнительные имена процедуры; описание основных объектов, с которыми работает процедура. Для получения более подробной информации необходимо при вызове инструкции указать параметр J=Y.

Пояснения в инструкции могут быть как на английском, так и на русском языках. Чтобы пояснения были на русском языке, при вызове инструкции нужно указать параметр LG=R или LG=P, а чтобы пояснения были на английском языке – параметр LG=A или LG = E.

§ 5.6. ВЫПОЛНЕНИЕ СИСТЕМНЫХ УТИЛИТ ПО КОМАНДЕ ОПЕРАТОРА

Работа по обслуживанию наборов данных и томов осуществляется с помощью утилит – специальных программ обслуживания. Вызов утилит удобно производить с помощью системных процедур как оператору ЭВМ с помощью команды START, так и пользователю с помощью оператора EXEC. Особенно часто действия по обслуживанию томов и наборов данных выполняется оператор ЭВМ, поэтому изложение будет вестись для команды оператора START.

Процедуры обслуживания томов. Получение информации о пакете магнитных дисков и о наличии свободной памяти происходит с помощью процедур GVDK или LSDK. *Процедура* GVDK используется для вывода информации на консоль оператора, а *процедура* LSDK – для вывода информации в выходной поток. Необходимо указать параметр V, задающий регистрационный номер тома. Если этот параметр не указан, то выдается информация о томе, заданном при настройке процедурного средства работы. Можно в одной команде получать информацию о двух или трех пакетах сразу. Тогда применяются параметры V1, V2, V3, имеющие следующий вид:

S GVDK, V1=V1536, V2 = 1248B, V3 = W1611

В процедуре LSDK можно использовать дополнительный параметр T=F, тогда выдается информация о месте расположения на томе каждого набора данных и о количестве в нем свободной памяти.

Изменение регистрационного номера тома можно провести не затрагивая информации на томе с помощью *процедуры* RENMDKON для накопителя на магнитном диске, находящегося в оперативном состоянии, и процедуры RENMDK для автономного состояния устройства.

При вызове *процедуры* RENMDKON задаются следующие параметры: V – старый регистрационный номер тома; NM – новый регистрационный номер тома. Используя эту процедуру, следует учитывать, что временные наборы данных, находящиеся на томе, могут стать недоступными для ОС.

Для *процедуры* RENMDK задаются следующие параметры: A – адрес накопителя, на котором установлен пакет дисков; NM – новый регистрационный номер тома пакета. Чтобы в одной команде переименовать сразу несколько пакетов, нужно в параметрах задавать значения списками.

Уничтожение наборов данных на пакете магнитных дисков осуществляется с помощью *процедуры* DLDK с параметром V, задающим регистрационный номер тома. При указании регистрационного номера тома может быть допущена ошибка и, как следствие, уничтожена информация на другом пакете магнитных дисков, поэтому перед уничтожением наборов данных утилита DLDK выдает на консоль запрос о номере тома. Если оператор подтверждает правильность указанного номера, то утилита выполняется, в противном случае – завершается аварийно.

Копирование пакетов магнитных дисков производится утилитой

CPDKDK. При вызове соответствующей процедуры задают следующие параметры: V – регистрационный номер копируемого пакета; VT – регистрационный номер тома, на который копируются наборы данных.

Если на пакете, на который производится копирование, нет нужной информации и его тип совпадает с типом копируемого пакета, то следует вызывать процедуру CPDK, время выполнения которой существенно меньше. При этом указываются те же параметры V и VT. Пакет, на который производится копирование, может получить тот же регистрационный номер, что и копируемый пакет, если в команде вызова процедуры указать параметр T=YES или T=Y; в противном случае пакет сохраняет свой регистрационный номер.

Сохранение образа пакета магнитных дисков на магнитных лентах позволяет в любое время восстановить информацию на другом пакете магнитных дисков. Утилита SVDKNL используется для магнитных лент без меток, а утилита SVDK – для магнитных лент со стандартными метками. В первом случае указываются следующие параметры: V – регистрационный номер сохраняемого тома; AT – адрес накопителя на магнитной ленте, а во втором случае V – регистрационный номер сохраняемого тома; VT – регистрационный номер тома магнитной ленты. Например, S SVDK,V=S1536,VT=T1417.

Для размещения образа пакета магнитных дисков на магнитной ленте необходимо два файла. Номер файла на магнитной ленте задается параметром LT. Если этот параметр опущен, то применяются номера 01 и 02. Если образ пакета магнитных дисков не помещается на одной магнитной ленте, то операционная система выдает запрос об установке дополнительных магнитных лент.

Если на магнитной ленте нужно сохранить образ не всего пакета магнитных дисков, а только группы дорожек пакета, то при вызове процедур SVDK и SVDKNL дополнительно указываются следующие параметры: CYL – номер цилиндра; TRK – номер дорожки на цилиндре для указания начальной дорожки в группе; CYLE и TRK – номера цилиндра и дорожки для указания конечной дорожки в группе. Если параметры CYL и TRK не заданы, то сохранение начинается с первой дорожки пакета. Если не заданы параметры CYLE и TRKE, то сохранение производится от заданной начальной дорожки до конца пакета. Номера цилиндров и дорожек задаются в шестнадцатеричном виде.

Восстановление пакета магнитных дисков по образу, сохраненному на магнитных лентах, осуществляется *процедурой* RSTDKNL, в которой указываются следующие параметры: V – регистрационный номер тома, на котором происходит восстановление; AT – адрес накопителя на магнитной ленте. Для восстановления пакета магнитных дисков с ленты со стандартными метками используется *процедура* RSTDK. со следующими параметрами; V – регистрационный номер тома, на котором происходит восстановление; LT – номер файла на магнитной ленте. Аналогично процедуре SVDK параметр LT можно опускать, если образ пакета размещен в первых двух файлах магнитной ленты.

Если при сохранении образ пакета был переписан на несколько магнитных лент, то при восстановлении их нужно последовательно установить на устройство по запросу ОС. Чтобы пакет, на который производится восстановление, получил тот же регистрационный номер тома, что и восстанавливаемый пакет, необходимо задать параметр T=YES или T=Y. Если восстанавливаются отдельные группы дорожек, то для сохранения наборов данных, которые уже имеются на пакете магнитных дисков, необходимо задать параметр TP=N.

Процедуры обслуживания наборов данных. Перемещение наборов данных с одного тома на другой происходит с помощью *процедуры* MVFL; для библиотечного набора данных применяется *процедура* MVLB. Для этого необходимо указать следующие параметры: D – имя набора данных; V – регистрационный номер тома, с которого перемещается набор данных; VT – регистрационный номер тома, на который перемещается набор данных. На томе, на который набор данных перемещается, должно быть достаточно свободной памяти и не должно быть набора данных с тем же именем. Можно переименовать перемещенный набор данных, указав дополнительно параметр NM (имя набора данных).

При перемещении набор данных удаляется со старого тома. Чтобы он не удался, вызываются процедуры копирования набора данных. Копирование последовательного набора данных выполняется процедурой CPSEL, а библиотечного набора данных – процедурой CPSLB. В этих процедурах используются те же параметры D, V и VT. Для указания нового имени

скопированного набора данных необходимо задать параметр DT. Уже созданные наборы данных также можно применять для копий, указав в параметре DT имя набора данных для копии.

Для создания наборов данных для копий нужно использовать следующие *процедуры:* CRFL – для последовательного набора данных, CRLB – для библиотечного набора данных, CRXS – для индексно-последовательных наборов данных. При вызове этих процедур задаются следующие параметры: D – имя набора данных; V – регистрационный номер тома; S – размер внешней памяти.

Переименование наборов данных необходимо иногда потому, что все имена наборов данных, находящихся на одном пакете магнитных дисков, должны быть уникальными. Для переименования применяется *процедура* RENMDS, при вызове которой указываются следующие параметры;

D – старое имя набора данных; V – регистрационный номер тома, на котором размещен набор данных; NM – новое имя набора данных. В параметрах D и NM значения могут задаваться списками, если в одной команде переименовывается сразу несколько наборов данных.

Освобождение памяти в наборах данных, т.е. сжатие библиотеки, производят с помощью процедуры CPRS, в которой задаются следующие параметры: D – имя набора данных; V – регистрационный номер тома. Если в наборе данных значительный объем памяти остался свободным и расширение набора данных происходить не будет, то целесообразно присоединить свободную память набора данных к общей свободной памяти магнитных дисков. Для последовательных наборов данных освобождение памяти производится с помощью *процедуры* RLSPFL, для библиотечного набора данных – с помощью *процедуры* RLSPLB. При этом указываются уже известные параметры D и V.

Каталогизация набора данных, т. е. запись в системный каталог, осуществляется *процедурой* CGDS. При вызове этой процедуры задаются следующие параметры: D – имя набора данных; V – регистрационный номер тома, на котором он расположен.

Набор можно исключить из каталога с помощью *процедуры* UNCGDS, имеющей параметр D (имя набора данных). В процедуре UNCGDS можно применять список имен.

Создание библиотечных наборов данных для программ необходимо для хранения программ. Программы хранятся в трех видах библиотек: библиотеках исходных модулей, в которых хранятся тексты программ на языках высокого уровня; библиотеках объектных модулей, в которых хранятся полученные в результате компиляции объектные модули; библиотеках загрузочных модулей, в которых хранятся готовые к исполнению загрузочные модули.

Создание библиотеки исходных модулей выполняется *процедурой* CRLB, библиотеки объектных модулей – *процедурой* CROLB, библиотеки загрузочных модулей – *процедурой* CRSLB. При вызове указанных процедур задаются следующие параметры: V; D; S – размер основной области памяти, выделяемой библиотеке; SM – размер дополнительной области памяти; SP – размер оглавления библиотеки.

Размер оглавления определяется в зависимости от предполагаемого количества разделов в библиотеке. Он рассчитывается исходя из того, что в одном элементе оглавления библиотек исходных и объектных модулей размещается информация о 16 разделах, а в одном элементе оглавления библиотеки загрузочных модулей – информация о четырех разделах.

Переименование наборов и разделов библиотечных наборов данных производится с помощью *процедуры* RENUMDS со следующими параметрами: V – регистрационный номер тома; D – старое имя набора данных; NM – новое имя набора данных. Значения для параметров D и NM можно задавать списками, если нужно переименовать несколько наборов данных, находящихся на одном пакете магнитных дисков, например

```
S RENUMDS, V=AB1601, D = ‘(AN,BN,CN)‘, NM = ‘(A,B,C)‘
```

Чтобы переименовать раздел библиотечного набора данных, используется *процедура* RENMB, в которой указываются следующие параметры: V – регистрационный номер тома; D – имя библиотеки; M – старое имя раздела; NM – новое имя раздела. Параметры M и NM могут также задаваться списками.

Удаление наборов данных и разделов библиотечных наборов данных выполняется *процедурой* DLDS, в которой указываются параметры D и V. Параметр

имени набора данных D можно задавать списком значений, т. е. можно удалить несколько наборов данных, находящихся на одном пакете магнитных дисков. Удаление разделов библиотек выполняется вызовом *процедуры* DLMB. При этом должны быть указаны следующие параметры: D; V; M – имя удаляемого раздела или список имен удаляемых разделов.

Глава 6. РАЗРАБОТКА ПРОГРАММ СЛОЖНОЙ СТРУКТУРЫ

В главе описываются принципы разработки сложных программ, в том числе установленные между программными модулями соглашения о связи по информации и управлению, а также правила использования общих регистров, структуры загрузочных модулей. Рассматриваются процедуры поиска и вызова требуемых в процессе работы программы загрузочных модулей, а также правила возврата управления между загрузочными модулями.

§ 6.1. СОГЛАШЕНИЯ О МЕЖМОДУЛЬНЫХ СВЯЗЯХ И СТРУКТУРЫ ЗАГРУЗОЧНЫХ МОДУЛЕЙ

Достаточно сложную программу удобно оформлять в виде отдельных модулей. В такой структуре модуль вызывает один или несколько других модулей, которые тоже могут вызывать один или несколько модулей. В языках высокого уровня модули называются *подпрограммами.*

Межмодульные связи. Когда один модуль вызывает другой, то он передает ему необходимые данные через параметры и сохраняет информацию о своем состоянии, необходимую для своего возобновления. При этом используются определенные процедуры и соглашения о межмодульных связях. Соглашения о межмодульных связях не надо путать со «связыванием», выполняемым Редактором связей для разрешения межмодульных символических ссылок. Речь пойдет о связывании модулей, происходящем во время их исполнения.

Каждая программа, получающая управление при выполнении шага задания, первоначально является вызываемой, так как ее вызывает управляющая программа ОС. Во время исполнения эта программа может передать управление другой программе и т. д. Требования о связях программ одинаковы для всех видов программ и способов передачи управления.

Рассмотрим соглашения о связях на примере передачи управления из управляющей программы. При передаче управления из управляющей программы должно сохраняться содержимое регистров общего назначения центрального процессора с номерами 0, 1, 13, 14, 15. В регистре 15 находится адрес точки входа в вызываемую программу. Адрес можно использовать в качестве значения базы. Управляющая программа в основной памяти выделяет область размером 18 слов, применяемую для сохранения содержимого регистров. Содержимое каждого регистра запоминается в строго определенном месте области сохранения. Следовательно, первым действием вызванной программы должна быть запись в область сохранения содержимого регистров. Адрес области сохранения содержится в регистре 13.

Область сохранения имеет строго определенную структуру: первое слово используется программами, написанными на языке ПЛ/1; второе слово хранит адрес следующей области сохранения. Адрес предыдущей области сохранения записывается вызывающей программой, а адрес следующей области сохранения – текущей программой, если она вызывает другую программу. В следующих словах хранится содержимое регистров 14, 15, 0...12.

Для сохранения содержимого общих регистров можно применять команду групповой записи в память STM. Например, команда STM 14, 12, 12(13) выгружает в память содержимое всех регистров, кроме регистра 13. Для упрощения программирования при запоминании нужных регистров можно применять макрокоманду SAVE.

Пример 6.1. Сохранение всех регистров, кроме регистра 13, осуществляется следующим образом:

```
PROG SAVE   (14, 12)
     USLNG PROG, 15
```

Использование макрокоманды SAVE возможно только в точке входа программы, так как ее адрес должен находиться в регистре 15.

Если исполняемая программа содержит системные макрокоманды (кроме SAVE, RETURN, GETMAIN) или обращение к другой программе, то она является вызывающей программой. Вызывающая программа должна обеспечить создание области сохранения для вызываемой

программы.

Адрес области сохранения данного модуля можно помещать либо в регистре 13, либо в области программы. Однако если создается новая область сохранения, то необходимо запомнить этот адрес во втором слове новой области сохранения. Адрес новой области сохранения записывается в третье слово старой области сохранения и затем помещается в регистр 13.

Связь областей сохранения (старой и новой) может быть нереентерабельной или реентерабельной. *Реентерабельность* – возможность повторного входа, т. е. обеспечивается вход в программу до окончания ее работы при предыдущем входе в нее.

Пример 6.2. Нереентерабельную связь областей сохранения можно осуществить следующим образом:

```
PROG   SAVE   (14, 12)
       USING  PROG,  12
       LR          12, 15
       LA          1, SAVEAREA DS 18F
       ST          13, 4(1)
       ST          1, 8(13)
       LM          13, 1, 8(13)
```

Пример 6.3. Реентерабельную связь областей сохранения можно выполнить следующим набором команд:

```
PROG SAVE      (14, 12)
USING       PROG, 12
LR          12, 15
GETMAIN R,LV = 72
ST          13, 4(1)
ST          1, 8(13)
LM          13, 1, 8(13)
```

Регистры связи. Управляющая программа операционной системы использует в качестве регистров связи регистры 0, 1, 13, 14 и 15 и не изменяет содержимого регистров 2...12. В программах пользователя для межмодульных связей также рекомендуется применять регистры 0, 1, 13...15.

Передача параметров вызываемой или вызывающей программ осуществляется с помощью регистров 0 и 1. Системные макрокоманды загружают параметры управляющей и вызываемой программ в эти регистры. В других случаях в регистр 1 загружается адрес списка параметров.

В регистре 13 содержится адрес области сохранения для программ, вызываемой текущей программой. При запросах текущей программы, сделанных с помощью системных макрокоманд к управляющей программе, управляющая программа может использовать эту область сохранения. Вызываемая программа также может пользоваться этой областью с помощью макрокоманды SAVE.

В регистре 14 содержится адрес точки в управляющей программе, в которую должна передать управление вызванная программа после завершения своей работы. Если обрабатывающая программа вызвала другую программу, то в регистре 14 хранится адрес точки возврата в вызывающую программу. В большинство макрокоманд включена команда, загружающая в регистр 14 адрес следующей команды программы. Управление вызывающей программе из вызываемой можно передать командой безусловного перехода по регистру 14, имеющей следующий вид: BR 14.

При передаче управления из управляющей программы в регистр 15 загружается адрес точки входа в программу. При передаче управления из этой программы в другую в регистр 15 загружается адрес точки входа в вызываемую программу. При возврате управления в вызывающую программу в регистре 15 может содержаться код возврата. Если в программе применяются системные макрокоманды, то при их исполнении в регистр 15 загружается адрес списка параметров, которые нужно передать управляющей программе.

Передача параметров из поля PARM оператора EXEC. Для передачи параметров из поля PARM оператора EXEC в вызываемую программу управляющая программа использует регистр 1. В этот регистр загружается адрес слова в выделенной программе области основной памяти. В старшем разряде слова должна быть единица. В трех младших разрядах слова хранится адрес двухбайтового поля, с которого начинается список передаваемых параметров. Собственно, в двух байтах содержится количество байтов информационного поля PARM (поле длины), за ними

следует поле PARM, размер которого ограничен 100 байт.

Структуры загрузочных модулей. Структура загрузочных модулей может быть простой, оверлейной, динамической.

Модуль простой структуры – единственный загрузочный модуль, подготовленный Редактором связей. Он целиком загружается в память и во время своего выполнения не передает управления другим загрузочным модулям. По времени исполнения это наиболее эффективная структура, однако она может потребовать слишком большого объема оперативной памяти. В таких случаях следует пользоваться более сложными структурами.

Оверлейная структура представляет собой один загрузочный модуль, подготовленный Редактором связей, который не загружается в память целиком. Модуль оверлейной структуры разделен на сегменты, которые последовательно загружаются управляющей программой в одну и ту же выделенную область основной памяти. Таким образом, экономится память, но необходимо дополнительное время на загрузку сегментов из библиотеки.

Модуль динамической структуры загружается в память целиком, но при исполнении он может передавать управление другим загрузочным модулям. Вызванные модули могут иметь как простую, так и оверлейную или динамическую структуру.

По времени выполнения динамическая структура менее эффективна, чем оверлейная. При исполнении модуля динамической структуры может потребоваться найти целый ряд загрузочных модулей из разных библиотек и загрузить их в память, в то время как в оверлейной структуре работа идет с сегментами одного модуля, находящегося в библиотеке. Динамические структуры нужно применять только для сложных программ, использующих те или иные модули в зависимости от вводимых данных.

§ 6.2. ПЕРЕДАЧА УПРАВЛЕНИЯ В МОДУЛЕ ПРОСТОЙ СТРУКТУРЫ

Термин «простая структура» относится к структуре загрузочного модуля при его выполнении. В то же время сам модуль может быть получен Редактором связей из совокупности подпрограмм. Рассмотрим связи между подпрограммами.

Передача управления без возврата. В некоторых случаях при передаче управления в подпрограмму не требуется возврат управления в вызывающую программу. Например, инициализирующая программа может устанавливать значения, подготавливать буферы и передавать управление дальше. При этом следует учитывать, что какая-либо из последующих вызванных программ будет осуществлять возврат к первоначальной программе.

Адрес точки возврата в вызывающую программу содержится в регистре 14. Его нужно передать из вызванной программы в ту программную секцию, которая произведет возврат передачи управления. Необходимо также восстановить регистры 2...12.

Адрес точки входа в вызванную программу необходимо загрузить в регистр 15, так же как это делается управляющей программой. Для передачи параметров в подпрограмму испрограммой. Для передачи параметров в подпрограмму используется регистр 1, в который загружается адрес списка параметров. Список параметров состоит из последовательно расположенных слов. В трех младших байтах каждого слова содержится адрес передаваемого в подпрограмму параметра.

После загрузки требуемых регистров в регистр 13 необходимо загрузить адрес области сохранения, используемой текущей программой. Так как управление назад не передается, образование второй области сохранения не обязательно. Для передачи управления внутри одного загрузочного модуля в регистр 15 загружается адрес соответствующей точки входа и производится передача управления по этому адресу.

Передача управления с возвратом. В регистр 15 загружается адрес точки входа в подпрограмму, в регистр 1 – адрес списка передаваемых параметров, а в регистр 14 – адрес точки возврата в вызывающую программу. По адресу точки возврата должно быть передано управление об окончании работы вызванной программы. Обычно в регистр 14 загружается адрес следующей команды текущей программы.

Для вызываемой программы необходимо создать новую область сохранения и занести адрес этой области в регистр 13. Однако это не означает, что областей сохранения будет столько же, сколько вызываемых программ. После возврата управления в вызывающую программу память, отведенную под область сохранения для вызванной программы, можно использовать под новую область сохранения.

Передача управления может осуществляться двумя способами. Первый аналогичен способу передачи управления без возврата. Во втором способе указывается макрокоманда CALL, в которой задаются точка входа и список передаваемых параметров. При выполнении макрокоманды CALL осуществляется следующее: адрес точки входа загружается в регистр 15; для указанных параметров формируются адресные константы типа A; адрес первой адресной константы загружается в регистр 1; адрес команды, следующей за макрокомандой CALL, загружается в регистр 14 для осуществления возврата; командой перехода с возвратом управление передается в указанную точку.

Пример 6.4. Рассмотрим вызов подпрограммы с помощью макрокоманды CALL:

```
CALL SUBR, (A, B, C, D), VL = 1
   ... SUBR...
```

Здесь SUBR – название вызываемой программной секции; A, B, C, D - передаваемые параметры. При указании параметра VL в старшем разряде последней адресной константы типа A устанавливается единица, что указывает на конец списка.

Если название точки входа не совпадает с названием вызываемой секции, то ее нужно определить в начале вызываемой секции оператором ENTRY.

Для возврата управления вызывающей программе рекомендуется следовать стандартным правилам, применяемым при возврате управления в управляющую программу. Не должно изменяться содержание регистров 2...14. В регистре 15 может находиться код возврата, характеризующий результат выполнения вызванной программы. Нулевой код возврата должен соответствовать нормальному завершению, значения кодов возврата должны быть кратны четырем. Операционная система может пользоваться значением кода возврата для управления выполнением последующими шагами задания. Смысл каждого значения кода возврата определяется пользователем.

Для облегчения программирования возврата в вызываемую программу можно использовать макрокоманду RETURN. Перед исполнением макрокоманды RETURN должно быть восстановлено содержимое регистра 13-адрес старой области сохранения. При выполнении макрокоманды RETURN происходит следующее: восстановление регистров 2...14; установление кода возврата в регистре 15; переход по адресу, указанному в регистре 14.

Рассмотрим выполнение одного загрузочного модуля, которому было передано управление управляющей программой. При передаче управления внутри одного загрузочного модуля рекомендуется использовать стандартные системные соглашения. По окончании работы модуля управление возвращается управляющей программе в точку возврата, адрес которой был передан первой программе модуля в регистре 14. Управляющая программа (инициатор-терминатор) терминирует задачу данного шага задания и передает управление задаче следующего шага. Если в операторе JOB или в последующих операторах EXEC указан параметр COND, то программа управления заданиями сравнивает код возврата, полученный в регистре 15, со значением параметра COND и определяет необходимость обхода последующих шагов.

§ 6.3. ПЕРЕДАЧА УПРАВЛЕНИЯ В МОДУЛЕ ДИНАМИЧЕСКОЙ СТРУКТУРЫ

Загрузка в основную память и выполнение загрузочного модуля, название которого указано в операторе EXEC, производятся управляющей программой автоматически.

Вызов загрузочного модуля. Если из модуля во время его исполнения нужно обратиться к другому загрузочному модулю, то для этого выдаются макрокоманды LOAD, LINK, ATTACH или XCTL. Запрашиваемый загрузочный модуль может находиться в одном из библиотечных наборов данных общей библиотеки, библиотеки задания, личной библиотеки.

О б щ а я б и б л и о т е к а всегда доступна для всех шагов задания.

Б и б л и о т е к а з а д а н и я образуется следующим образом: в задании за оператором JOB помещается оператор JOBLIB DD. Эта библиотека будет доступна любому шагу задания. Для логической связи библиотеки с программой управляющая программа строит необходимый блок управления данными DCB и выдает макрокоманду OPEN.

Л и ч н у ю б и б л и о т е к у можно вызвать только из того шага задания, в котором она определена. Для определения личной библиотеки в шаг задания включается описывающий ее оператор DD. Пользователь должен обеспечить построение блока управления данными и выдать

макрокоманду OPEN.

В операционной системе, работающей в режиме MVT, в общей области основной памяти могут находиться часто используемые реентерабельные загрузочные модули из общей библиотеки и загрузочные модули управления данными. Копии остальных модулей, запрашиваемых на шаге задания, загружаются в выделенную шагу область памяти. Дальнейшее изложение будет проводиться для режима MVT.

Поиск загрузочного модуля. Порядок поиска загрузочного модуля в общей области, в области задания и в библиотеках зависит от операндов, указанных в макрокомандах вызова загрузочного модуля. Это следующие операнды: EP, EPLOC, DE и DCB. Для определения имени точки входа в загрузочный модуль задается один из трех операндов: EP, EPLOC, DE. Операнд DCB не обязателен, он применяется для задания адреса блока DCB для личной библиотеки. Операнд DCB можно опустить или указать в нем нулевой адрес. Тогда будут использоваться блоки управления данными общей библиотеки и библиотеки заданий. Имя точки входа является именем раздела в библиотеке.

Если применяется операнд EP или EPLOC, то задается только имя точки входа. Порядок поиска модуля следующий: копия в области зоны памяти задания; модуль в библиотеке задания, если библиотека существует; копия в общей области памяти; модуль в общей библиотеке.

Если используется операнд DE, то время, затрачиваемое на поиск копии загрузочного модуля, существенно сокращается. Для этого перед макрокомандой вызова загрузочного модуля необходимо создать в основной памяти копию элемента оглавления библиотеки для искомого загрузочного модуля с помощью макрокоманды BLDL. Одной макрокомандой BLDL можно строить копии сразу нескольких элементов оглавления библиотеки. В макрокоманде BLDL следует указать имена загрузочных модулей и адрес блока DCB для библиотеки.

Макрокоманда LOAD. Эта макрокоманда применяется для загрузки копии загрузочного модуля в область задания в оперативной памяти. Если до этого копия вызываемого модуля была загружена в общую область, то копирование в область задания не производится. Макрокоманду LOAD применяют обычно для реентерабельных или последовательно используемых модулей, так как модуль, помещенный в область памяти задания макрокомандой LOAD, остается в ней до тех пор, пока не будет выдана макрокоманда DELETE.

Для реентерабельных модулей управляющая программа устанавливает счетчик обращений для копии. Каждый раз при выдаче макрокоманды LOAD для данного модуля счетчик увеличивается на единицу, при выдаче макрокоманды DELETE -уменьшается на единицу. Если счетчик обращений для копии равен нулю, то область памяти, содержащая копию, освобождается и применение копии становится невозможным.

Если при выдаче макрокоманды LOAD или DELETE загрузочный модуль уже находится в общей области памяти, то при обработке макрокоманд передается код возврата, соответствующий успешному их выполнению.

Макрокоманда LOAD используется для загрузки копии модуля в область памяти задания, макрокоманда CALL - для передачи управления этому модулю. Соглашения о связях при передаче управления между загрузочными модулями такие же, как и при передаче управления между программами внутри одного загрузочного модуля. Возврат управления осуществляется в вызывающий модуль.

Перед передачей управления загрузочному модулю необходимо проверить тип загрузочного модуля и текущее состояние копий. Если вызываемый модуль реентерабельный, то для исполнения шага задания требуется только одна копия и ее текущее состояние определять не нужно.

Для последовательно используемого модуля перед входом в него следует проверить, свободен ли он. Если на шаге задания образуется только одна задача, то логика вызывающей программы должна быть построена таким образом, чтобы вторичное применение копии происходило только после завершения ее первого использования. Если на шаге задания образуется несколько задач (подзадач), то требуется соблюсти больше условий при вызове последовательно используемых модулей. Во-первых, в любой задаче не должно быть вторичного использования модулей до их освобождения. Во-вторых, в один и тот же момент несколько задач не должны обращаться к одной и той же копии загрузочного модуля. Для соблюдения второго условия применяется макрокоманда ENQ. Если загрузочный модуль не является повторно используемым, то каждую

копию можно применять только один раз; при вызове модуля нужно получать новую копию.

Возврат управления между загрузочными модулями производится таким же образом, как и возврат управления в подпрограммах внутри одного загрузочного модуля. Возврат можно осуществить с помощью команды перехода или макрокоманды RETURN.

Макрокоманда LINK. Для передачи управления с возвратом применяются макрокоманды LOAD, CALL, DELETE. Для этой же цели можно использовать одну макрокоманду LINK, обеспечивающую загрузку копии модуля, передачу ему управления и возврат в вызывающий модуль. Существенным отличием макрокоманды LINK является то, что данные действия выполняются управляющей программой ОС, т. е. при выдаче макрокоманды LINK происходит обращение к управляющей программе через команду SVC. Управляющая программа осуществляет поиск копии загрузочного модуля и передачу управления в указанную в макрокоманде LINK точку входа.

Действие макрокоманды LINK аналогично действию макрокоманды CALL в модуле простой структуры. Вызываемый загрузочный модуль должен восстановить содержимое регистров 2...12 из своей области сохранения перед возвратом управления. В регистр 13 помещается адрес области сохранения для вызываемого загрузочного модуля, в регистр 1 – адрес списка адресных параметров для вызываемого загрузочного модуля.

Однако при использовании макрокоманды LINK адреса точки входа (регистр 15) и точки возврата (регистр 14) устанавливает не вызывающая, а управляющая программа. Информация в точке входа макрокомандой LINK из вызывающего модуля передается управляющей программе. Управляющая программа осуществляет поиск в библиотеках загрузочных модулей и помещает адрес точки входа в регистр 15. Адрес, получаемый вызванным загрузочным модулем в регистре 14, не является адресом следующей после макрокоманды LINK команды вызывающего загрузочного модуля. Управляющая программа сохраняет адрес возврата к вызывающему модулю в своей области сохранения, а в регистр 14 помещает адрес возврата в подпрограмму внутри управляющей программы.

Передача управления без возврата. Если при передаче управления из одного модуля в другой не требуется возврата в первый модуль, то для вызова модуля можно применять макрокоманду XCTL. Эта макрокоманда помимо передачи управления в вызванный модуль сообщает управляющей программе о завершении вызывающего модуля, содержащего макрокоманду XCTL. Перед вызовом необходимо восстановить содержимое регистров из старой области сохранения, так как управление в вызывающий модуль возвращать не нужно.

В макрокоманде XCTL передается адрес списка параметров (в регистре 1). В этом случае список параметров должен находиться вне области памяти текущего загрузочного модуля, содержащего макрокоманду XCTL, так как его копия может быть уничтожена прежде, чем вызванный загрузочный модуль будет использовать эти параметры. Способ передачи управления в макрокоманде XCTL аналогичен способу в макрокоманде LINK.

Глава 7. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВВОДА-ВЫВОДА

В главе приводится порядок обмена информацией между оперативной памятью и внешними устройствами. Рассматриваются функции центрального процессора, каналов и контроллеров, выполняемые ими в процессе данного обмена. Описывается роль аппарата прерываний ввода-вывода для синхронизации работы каналов и центрального процессора. Обмен данными между терминалами и ЭВМ осуществляется под управлением программ телеобработки данных. В качестве программного обеспечения систем телеобработки рассматриваются базисный и общий телекоммуникационные методы доступа, управляющие телемониторы КАМА и ОБЬ, диалоговые программные средства, автоматизирующие работу системных и прикладных программистов.

§ 7.1. СТРУКТУРА СИСТЕМЫ ВВОДА-ВЫВОДА

Операцией ввода-вывода называется процесс обмена информацией между оперативной памятью и внешними устройствами. К внешним относятся следующие устройства: единичных записей (перфокарточные устройства ввода-вывода, печатающие устройства), внешние запоминающие (накопители на магнитных дисках и лентах), телеобработки данных и др.

Контроллеры и каналы ввода-вывода. Внешние устройства работают под управлением *контроллеров,* т. е. устройств управления. Связь устройств управления с центральным процессором осуществляется через каналы ввода-вывода. Контроллер расшифровывает команды

канала, полученные через интерфейс ввода-вывода, и обеспечивает выполнение конкретных операций на управляемом внешнем устройстве. Контроллер конструируется в виде отдельной стойки управления или как единое целое с внешним устройством.

Каналы ввода-вывода представляют собой специализированные ЭВМ, управляющие обменом информацией между внешними устройствами и оперативной памятью. Наличие каналов позволяет процессору выполнять операции по обработке данных одновременно с исполнением операций ввода-вывода. Управление каналом производится с помощью команд канала, которые хранятся в оперативной памяти и составляют программу канала.

Центральный процессор управляет работой каналов с помощью команд ввода-вывода. Эти команды позволяют проверить состояние канала (команда TCH) и внешних устройств (команда TIO), начать (команда SIO) и прекратить выполнение программы канала (команда НЮ). Для связи ЦП и каналов используются две фиксированные ячейки оперативной памяти и аппарат прерываний от ввода-вывода.

В первой фиксированной ячейке по адресу 72 содержится полное адресное слово канала CAW. При исполнении команды ввода-вывода SIO (начать ввод-вывод) адресное слово канала передается в канал. Программа канала начинает выполняться с адреса, указанного в CAW. При этом обеспечивается защита памяти (ключ защиты памяти, к которой получает доступ канал, должен совпадать с ключом, указанным в CAW).

Все команды ввода-вывода имеют единый формат, представленный на рис. 7.1. Здесь КОП - код операции, определяющий конкретную команду канала. Из содержимого общего регистра B1 и смещения D1 формируется исполнительный адрес, который интерпретируется как адрес устройства, занимающий младший байт, и адрес канала, находящийся в старшем байте. Таким образом, возможна адресация до 256 каналов (до 16 в ЭВМ Ряд 1) и до 256 устройств на каждом канале. Внешние устройства, подключенные к общему устройству управления, например накопители на магнитных дисках, адресуются последовательными числами, начинающимися с числа, кратного 16, и различающимися не более чем на 15. Например, OCO,OC1,...,OCE,OCF-адреса дисплеев локального дисплейного комплекса на нулевом канале.

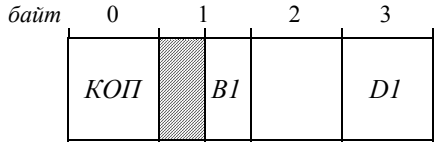


Рис. 7.1. Формат команды ввода-вывода

Во вторую фиксированную ячейку, применяемую в операциях ввода-вывода, по адресу 64 заносится двойное слово состояния канала CSW. Это слово полностью или частично заносится в оперативную память при исключительных ситуациях выполнения операции ввода-вывода: завершении программы канала, ошибках в программе или обмене данными и др. Ключ в CSW переносится из адресного слова канала в начале выполнения программы канала. Адрес команды канала и остаточный счетчик указывает, насколько продвинулось исполнение программы канала к моменту занесения CSW в память.

Байты состояния устройства и канала. Байт состояния устройства (БСУ) и байт состояния канала (БСК) описывают состояния системы ввода-вывода на момент записи CSW. Значения битов 0...7 фиксированы для всех внешних устройств и в байте состояния устройства имеют следующие значения:

0– внимание. На устройстве (обычно устройстве телеобработки) нажата специальная кнопка;

1– модификатор состояния. Устанавливается при использовании некоторых внешних устройств для изменения последовательности выполнения команд канала;

2– контроллер закончил. Устройство управления закончило свою часть операции ввода-вывода;

3– занято. Внешнее устройство или контроллер заняты выполнением предыдущей операции ввода-вывода или в них хранится условие прерывания;

4– канал закончил. Канал закончил передачу данных между внешним устройством и оперативной памятью;

5– устройство закончило. Внешнее – устройство закончило операцию ввода-вывода;

6– ошибка в устройстве. При операции ввода-вывода в устройстве возникла сбойная ситуация. В этом случае с помощью специальной команды канала можно уточнить состояние устройства;

7– особый случай. Необычная, но допустимая ситуация в устройстве, например считывание ленточной марки накопителем на магнитной ленте.

Значения битов в байте состояния канала следующие:

0– программно-управляемое прерывание. Канал выбрал команду канала со специальным признаком;

1– неправильная длина. Физическая длина записи не совпадает с длиной, запрошенной в команде канала;

2– ошибка в программе. Обнаружены ошибки в CAW или командах канала;

3– нарушение защиты. Ключ защиты памяти не совпадает с ключом в CAW;

4– ошибка в данных. Нарушена правильная четность данных, передаваемых через канал, или обнаружена машинная ошибка;

5– ошибка в управлении. Обнаружен неправильный сигнал управления;

6– ошибка в интерфейсе. Сбой в устройстве управления или внешнем устройстве;

7– ошибка в цепочке. Канал не успел расшифровать следующую команду канала до исполнения предыдущей.

При операции ввода-вывода канал исполняет программу канала, которая хранится в оперативной памяти и адрес начала которой указан в адресном слове канала.

Многие устройства управляются простейшими программами канала, состоящими из одной–двух команд. Однако сложные и разветвленные программы канала используются при выполнении операций ввода-вывода на устройствах прямого доступа и телеобработки.

Работа каналов и центрального процессора синхронизируется с помощью аппарата прерываний от ввода-вывода. Прерывания осуществляются по стандартной для ЕС ЭВМ схеме: текущее слово состояния программы PSW запоминается в фиксированной ячейке памяти по адресу 56, в него помещается адрес канала и устройства, вызвавшего прерывания, затем новое PSW выбирается из ячейки с адресом 120, передавая тем самым управление программе-обработчику прерываний от ввода-вывода. Прерывания от ввода-вывода могут маскироваться (задерживаться) с помощью битов маски системы в PSW. При этом сигнал прерывания хранится в канале, пока соответствующий бит маски системы не станет равным единице. Тем самым предотвращается перекрытие прерываний, например возникновение прерывания от ввода-вывода во время работы программы-обработчика прерываний от ввода-вывода. Наличие в канале необработанного сигнала прерывания обнаруживается с помощью команд ввода-вывода SIO и TIO (опросить ввод-вывод).

Причины возникновения прерываний ввода-вывода. Основные причины, по которым возникает прерывание от ввода-вывода:

– окончание работы канала (нормальное или аварийное завершение исполнения программы канала). Запоминается полное CSW, бит 4 БСУ равен единице. Анализ битов БСУ и БСК позволяет определить характер завершения операции ввода-вывода;

– программно-управляемое прерывание. Канал выбрал CCW со взведенным битом 4 байта флажков. Запоминается CSW с байтом состояния устройства, равным нулю. Такие прерывания могут использоваться для дополнительной синхронизации действий канала и ЦП, например для модификации программы канала в процессе выполнения;

– прерывание от устройства, возникающее в том случае, когда устройство переходит в состояние готовности или при нажатии специальных кнопок на устройстве (см. значение бита 0 БСУ). Запоминается CSW, в котором все поля, кроме БСУ и БСК, равны нулю. Этот тип прерываний от ввода-вывода активно применяется при программировании для локальных устройств телеобработки.

§ 7.2. ОБЕСПЕЧЕНИЕ ВВОДА-ВЫВОДА В ОПЕРАЦИОННОЙ СИСТЕМЕ

Проблемная программа, работающая в среде ОС ЕС, непосредственно выполнением операций ввода-вывода не занимается. Это связано с необходимостью обеспечения разделения доступа к данным (в связи с чем команды ввода-вывода являются привилегированными, т. е. требующими для своего исполнения состояния «Супервизор» в PSW) и с большим объемом сложного однообразного программирования операций ввода-вывода. Реализация операций ввода-вывода в ОС ЕС осуществляется по запросам от проблемных программ к операционной системе.

Супервизор ввода-вывода. Совокупность программ системы, инициализирующих и завершающих программы канала, обрабатывающих прерывания от ввода-вывода, исправляющих

ошибки ввода-вывода, отслеживающих состояния внешних устройств и т. п., называется *супервизором ввода-вывода.*

Супервизор ввода-вывода – составная часть класса программ операционной системы, объединяемых под названием «управление данными». Программы управления данными обеспечивают логический интерфейс проблемной программы с внешними устройствами. Рассмотрим основные этапы обработки данных.

Подготовка к обработке данных (OPEN) осуществляется следующим образом. Программы управления данными проверяют санкционированность доступа к данным и настраивают системные управляющие блоки и таблицы для работы с данными на внешних устройствах.

Выполнение операций ввода-вывода включают следующие этапы. По запросам проблемной программы управление данными строит программы канала, инициирует их выполнение супервизором ввода-вывода, определяет характер завершения операций ввода-вывода, оповещает проблемную программу об этом окончании и т. п.

Окончание обработки данных (CLOSE) заключается в том, что система управления вводом-выводом логически отключается от проблемной программы.

Методы доступа. Вся деятельность программ управления данными базируется на фундаментальном для ОС ЕС понятии метода доступа. *Метод доступа* – совокупность программного способа доступа к данным и метода организации данных. В ОС ЕС используются следующие методы доступа: базисные; с очередями; EXCP.

Базисный метод доступа характеризуется тем, что отпадает необходимость в написании канальных программ, управляющих блоков, при этом программист должен в своей программе организовать управление буферами, объединение данных в блоки, выделение их из блоков и т. д.

Базисный последовательный метод доступа BSAM характеризуется следующим: последовательной организацией данных, означающей последовательное следование записей данных, при этом порядковый номер записи является ее единственным поисковым признаком; программным способом доступа к последовательно организованным данным; независимостью от устройства. Последовательно организованные данные могут размещаться на любых устройствах единичных записей (перфокарточных, перфоленточных, печатающих) и устройствах внешней памяти (магнитных дисках и лентах). Однако в проблемной программе специфика внешнего устройства не отражается и одну и ту же программу можно использовать для обработки данных на различных устройствах.

Методы доступа с очередями позволяют обеспечить обмен данными на уровне логических записей. Все операции, связанные с буферизацией данных, объединением их в блоки и выделением из блоков, управлением очередями, выполняют программные средства методов доступа.

Несмотря на наличие развитых методов доступа ОС ЕС, в некоторых случаях возникает необходимость непосредственного составления программы канала в проблемной программе. Это происходит при повышении эффективности обработки данных со стандартной для ОС ЕС организацией; при организации данных некоторым нестандартным способом и обслуживании нестандартных внешних устройств.

Метод доступа EXCP (Execute Channel Program) можно применять во всех случаях. При использовании этого метода доступа в проблемной программе формируется программа канала, которая предоставляется для выполнения супервизору ввода-вывода. Обработка завершения канальной программы осуществляется супервизором и может быть продолжена и расширена проблемной программой. Все исключительные ситуации, возникающие при работе программы канала, могут быть обработаны специальными программами-аппендиксами, представляющими собой расширение супервизора ввода-вывода и включающимися в систему системным программистом. Рассмотрим основные этапы реализации операций ввода-вывода при использовании метода доступа EXCP:

1) в проблемной программе формируется блок управления данными DCB, в котором указываются тип метода доступа, организация данных, имя оператора DD, способы обработки ошибок, имена программ-аппендиксов;

2) запускается функция управления данными OPEN, при этом по указанному блоку DCB

проверяется санкционированность доступа к данным, загружаются требуемые программы-аппендиксы и строится защищенное от модификации проблемной программой расширение блока DCB. Таким расширением является блок экстентов DEB, в котором указываются границы внешней памяти на магнитном диске, адрес внешнего устройства, ключ защиты памяти;

3) в проблемной программе формируется программа канала, составленная из любых команд канала, допустимых для данного внешнего устройства;

4) проблемная программа формирует и передает супервизору ввода-вывода блок ввода-вывода IOB, в котором указаны адреса канальной программы и блока DCB. После этого проблемная программа должна дождаться завершения операции ввода-вывода и проверить ее результаты. Для этого в блок IOB копируется слово состояния канала CSW;

5) выполняется системная функция CLOSE, разрушающая связь проблемной программы с супервизором ввода-вывода (уничтожается блок DEB, удаляются программы-аппендиксы).

Метод доступа EXCP делает практически невозможной независимость от устройства и требует от программиста достаточных знаний о функционировании конкретных устройств ввода-вывода. Тем не менее этот метод часто применяется для программирования операций ввода-вывода, особенно для таких сложных в обслуживании устройств, как устройства прямого доступа и телеобработки.

§ 7.3. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ ТЕЛЕОБРАБОТКИ ДАННЫХ

Рассмотрим некоторые вопросы программирования для устройств телеобработки, относящихся к диалоговым внешним устройствам. Обеспечение диалога базируется на синхронизации действий пользователя терминала и ЭВМ, основанной на обработке сигналов внимания. При локальном подключении терминалов к ЭВМ (непосредственно к интерфейсу ввода-вывода) нажатие специальных клавиш на терминале приводит к возникновению прерываний от ввода-вывода (см. § 7.1). Эти прерывания могут интерпретироваться проблемной программой как признак готовности пользователя, т. е. разрешение ЭВМ производить операции ввода-вывода на терминале. Операции (писать, читать) достаточно стандартны и особенных трудностей в программировании не представляют.

Значительно сложнее ситуация при удаленном подключении терминалов к ЭВМ, т. е. через аппаратуру передачи данных. В этом случае нажатие клавиши «Внимание» не вызывает прерывания в ЭВМ, а только переводит терминал в некоторое специальное состояние. Для обнаружения этого состояния, а значит, и сигнала «Внимание» проблемная программа должна вести непрерывный опрос терминала с помощью специальных команд канала на аппаратуре передачи данных. Операции чтения и записи также осуществляются с помощью достаточно сложных канальных программ. При этом возникает много ошибок ввода-вывода на терминалах и в аппаратуре передачи данных. Предоставляемые операционной системой телекоммуникационные методы доступа не всегда могут удовлетворительно обработать и исправить возникающие ошибки, что заставляет обращаться к методу доступа EXCP. В этом случае ориентированность программы на конкретный тип терминала позволяет наладить и более корректное исправление ошибок ввода-вывода.

Под программным обеспечением систем телеобработки данных ОС ЕС обычно понимают методы теледоступа, телемониторы (например, ОБЬ, КАМА и др.) и функциональные прикладные программы, ориентированные на решение конкретных терминальных задач. Для обеспечения обмена информацией между ЭВМ и удаленными терминальными станциями ОС ЕС предоставляет базисный телекоммуникационный метод доступа (БТМД) и в качестве метода доступа с очередями - общий телекоммуникационный метод доступа (ОТМД).

Базисный телекоммуникационный метод доступа. Этот метод предназначен для реализации функций обмена данными между ЭВМ и удаленными станциями в стартстопном и в синхронном режимах передачи. При этом используются коммутируемые или некоммутируемые каналы связи [18].

Программные средства БТМД обеспечивают выполнение следующих основных функций: идентификации системы телеобработки; ее активизации и деактивизации; перекодировки передаваемых данных; организации буферных пулов; управления каналами связи; неавтономного тестирования и исправления обнаруженных ошибок.

Во время генерации операционной системы необходимо описать технические средства системы

телеобработки данных (типы мультиплексоров передачи данных (МПД), каналов передачи данных, удаленных станций), что обеспечит при работе с описанными устройствами применение нужных модулей БТМД.

Если используемые каналы связи имеют сходные характеристики, то целесообразно объединить их в группы, обслуживаемые одним и тем же модулем БТМД. Группы каналов связи задаются во время генерации ОС при помощи макрокоманды UNITNAME или на этапе выполнения при помощи оператора DD. При объединении каналов в группы должны соблюдаться следующие условия: синхронные и стартстопные каналы не могут находиться в одной группе; все каналы группы должны иметь одинаковую конфигурацию (например, для стартстопного режима они должны быть или коммутируемыми или некоммутируемыми многоточечными, либо коммутируемыми одноточечными).

Операционная система строит для каждого набора данных блок управления данными DCB, в котором содержатся все сведения о наборе данных. Блок управления данными строится путем задания соответствующей макрокоманды DCB. Для получения доступа к набору данных необходимо открыть блок DCB при помощи макрокоманды OPEN, при этом в блок DCB заносится дополнительная информация из оператора DD, описывающего набор данных, и метки набора данных. Соответствие между набором данных, используемым в программе, и оператором DD, описывающим этот набор данных, устанавливается при помощи оператора DCB. В системах телеобработки данных блок DCB строится для каждой группы каналов связи. Макрокоманда DCB позволяет определять тип канала связи, вид буферизации; в ней задаются процедуры исправления ошибок.

Для идентификации абонента каждой удаленной станции должен быть присвоен уникальный номер или идентификатор. Все идентификаторы станций заносятся в специальные абонентские списки. Для некоммутируемых каналов связи строятся списки опроса и списки выборки. Для задания абонентского списка используется макрокоманда DFTRMLST. При неудачном открытии набора данных предусмотрена возможность повторного открытия набора с помощью макрокоманды LOPEN. После завершения работы с группой каналов связи необходимо выдать макрокоманду CLOSE.

При передаче данных по каналам связи возникает необходимость в создании буферов. *Буфер* – область оперативной памяти, временно резервируемая для хранения сообщений. В БТМД предусмотрены программный и динамический способы организации буферизации.

При п р о г р а м м н о й б у ф е р и з а ц и и перед каждой операцией чтения необходимо запросить необходимое количество буферов, которые после завершения операции по приему сообщения необходимо освободить.

При д и н а м и ч е с к о й б у ф е р и з а ц и и буферы выбираются из буферного пула автоматически. Буферный пул может строиться либо с помощью макрокоманд GETMAIN, BUILD и DCB, либо путем использования макрокоманд GETPOOL и DCB. Для получения буфера из буферного пула перед выполнением операции чтения-записи применяется макрокоманда REQBUF, а для его освобождения после выполнения указанных операций - макрокоманда RELBUF.

В ОС ЕС для передачи информации по каналам связи применяются семибитовый код КОИ-7 и пятибитовый телеграфный код МТК-2. Поэтому необходимо преобразование информации из кода передачи во внутренний код ЭВМ ДКОИ. В БТМД для такого преобразования используются макрокоманды ASMTRTAB и TRANSLATE. С помощью макрокоманды ASMTRTAB задается тип таблицы перекодировки, соответствующей применяемым удаленным станциям. При употреблении нестандартных кодов в БТМД предусмотрена возможность организации новой таблицы. Посредством макрокоманды TRANSLATE осуществляется перевод из кода ЭВМ в код передачи. Для задания и приема сообщений от удаленных станций используются макрокоманды WRITE и READ.

При передаче сообщений посредством макрокоманды WRITE выполняются следующие операции: установить соединение и передать блок сообщения; передать следующий блок сообщения; передать блок сообщения и закончить передачу. При приеме сообщений от терминальных станций при помощи макрокоманды READ можно осуществить следующее: установить соединение и читать первый блок сообщения; передать подтверждение и читать следующий блок сообщения; передать отрицание приема, указывающее на необходимость чтения

блока сообщения. Задание различных типов макрокоманд WRITE и READ вызывает выполнение различных канальных программ. Для идентификации установления соединения и окончания процесса передачи используются специальные служебные символы «кто там» (КТМ) и «конец передачи» (КП).

Общий телекоммуникационный метод доступа. Этот метод обладает большим количеством функциональных возможностей, предоставляемых пользователям. В ОТМД наборами данных, применяемыми обрабатывающими терминальными программами, являются очереди сообщений, получаемых или передаваемых терминальными станциями.

Управление сообщениями в ОТМД обеспечивает программа управления сообщениями (ПУС). Основные функции этой программы: активизация и отключение каналов связи; получение сообщений от терминалов; динамическая буферизация сообщений; приоритетное управление сообщениями; редактирование поступающих и выходных сообщений; постановка сообщений и ответов в соответствующие очереди; формирование сообщений об ошибках и т. д.

Рабочие варианты программы управления сообщениями создаются (генерируются) в зависимости от конкретных конфигураций системы телеобработки данных, имеющихся в вычислительных центрах. Для создания ПУС существует набор специальных макрокоманд ОТМД.

Сообщение, передаваемое средствами ОТМД, состоит из заголовка и текста. Заголовок содержит служебную информацию, необходимую для правильной его обработки (приоритет, тип очереди и т. д.).

В ОТМД имеются следующие способы организации очередей сообщений:

1) очереди на однократно используемом накопителе на магнитных дисках заполняются до тех пор, пока имеется доступная память, после чего выдается сообщение о переполнении и очередь необходимо переформировать. Сообщение хранится в очереди до тех пор, пока оно не будет выбрано;

2) очереди на повторно используемых магнитных дисках организованы таким образом, что после заполнения доступной области памяти запись поступающих сообщений производится в начале очереди на место сообщений, прибывших в очередь первыми;

3) очереди в оперативной памяти (ОП) позволяют резко повысить оперативность передачи информации, однако экономически нецелесообразно обеспечение хранения сообщений в течение длительного промежутка времени в связи с использованием больших объемов оперативной памяти;

4) очереди в оперативной памяти с регулярной перезаписью сообщений на однократно используемый МД;

5) очереди в оперативной памяти с перезаписью на повторно используемый МД.

Обычно ПУС состоит из пяти секций: активизации и деактивизации, определения наборов данных, управления абонентскими пунктами и каналами связи, программ выходов пользователей, обработчика сообщений.

Секция активизации ПУС включает следующие макрокоманды: INTRO, OPEN, READY. Макрокоманда INTRO – первая макрокоманда ПУС, ей передается управление от операционной системы. В макрокоманде INTRO указывается имя ПУС, определяются размеры буферов, обеспечивается включение средств отладки. Макрокоманда OPEN инициализирует наборы данных очередей сообщений. Макрокоманда READY – последняя макрокоманда секции активизации ПУС.

Если в системе телеобработки данных применяются прикладные программы, то для завершения работы ПУС используется макрокоманда MCPCLOSE. Если прикладных терминальных программ нет, то для завершения работы ПУС выдается команда оператора SYSCLOSE.

Секция определения наборов данных состоит из макрокоманд, описывающих наборы данных групп каналов связи и очередей сообщений.

Секция управления абонентскими пунктами и каналами связи обеспечивает создание таблиц, необходимых ПУС для управления терминальными станциями. Основу ПУС составляет обработчик сообщений (МН), являющийся набором программ, определяющих вид обработки сообщений при помощи анализа управляющей информации, содержащейся в их заголовках. Также МН обеспечивает редактирование входных и выходных сообщений, проверку кодов передачи, отправку сообщений и т. д.

Секция программ выходов пользователей является средством предоставления пользователю возможности написания собственной терминальной программы, получающей управление от ПУС.

Секция обработки сообщений, поступающих от терминалов или посылаемых им, включает прикладные терминальные программы, выполняемые асинхронно по отношению к ПУС. При этом ПУС имеет более высокий приоритет. Если требуются различные виды обработки сообщений, то для каждого из них должна быть написана своя прикладная терминальная программа, управление которой будет передаваться после анализа заголовка сообщения.

Общий телекоммуникационный метод доступа предоставляет пользователям значительно большее количество возможностей, чем базисный метод. Однако при этом требуются расход значительных объемов оперативной памяти (до 200 К байт) и перегенерация ПУС в случае изменения конфигурации системы телеобработки данных.

Для управления разветвленными терминальными сетями и организации систем межмашинного обмена данными между удаленными вычислительными комплексами более целесообразно по сравнению с БТМД и ОТМД использование телемониторов. Описанию телемониторов КАМА и ОБЬ, посвящены следующие параграфы.

§ 7.4. ПАКЕТ ПРИКЛАДНЫХ ПРОГРАММ КАМА

Пакет прикладных программ (ППП) телемонитор КАМА предназначен для расширения возможностей ОС ЕС в части телеобработки данных. Этот пакет является аналогом пакета CICS, разработанного фирмой «IBM». При написании прикладных программ, предназначенных для работы под управлением ППП КАМА, программисту предоставляется набор макрокоманд, обеспечивающих взаимодействие с файлами данных и терминалами, а также обработку аварийных ситуаций. Пользователь может использовать для написания своих программ языки Ассемблер, ПЛ/1 или КОБОЛ. При разработке прикладных программ в среде ППП КАМА запрещается применять средства обращения к управляющей программе ОС ЕС.

Основные компоненты системы КАМА. Система КАМА построена по модульному принципу, поэтому для настройки пакета на конкретную конфигурацию технических средств и для обеспечения требуемых функциональных возможностей производится генерация системы. Единицей работы в системе КАМА является транзакция. Информация, необходимая для работы функциональных средств системы КАМА, содержится в управляющих таблицах. Программные средства системы КАМА делятся на управляющие и сервисные.

Управляющие программные средства обеспечивают управление задачами, памятью, программами, терминалами, файлами, транзитными данными, программными прерываниями, временем, временной памятью.

Основные сервисные средства системы КАМА следующие: средства подключения и отключения операторов, главного терминала, сбора статистики, обработки аварийных ситуаций, завершения работы системы, трассировки, пакетной обработки транзакций, динамического открытия и закрытия наборов данных.

Рассмотрим взаимодействие основных компонентов системы КАМА. Оперативная память, выделяемая системе, делится на следующие области: статическую, предназначенную для размещения управляющих программ и таблиц, а также резидентных модулей системы; динамическую - для размещения областей ввода-вывода и прикладных программ; область памяти, выделяемую для программ операционной системы. Загрузка управляющих программ и таблиц осуществляется программой инициализации системы. Для запуска задачи необходимо ввести с терминала код транзакции. Программа управления терминалами опрашивает терминалы и устанавливает адрес терминала, с которого был произведен ввод информации. Далее управление передается программе управления задачами, выделяется память, вычисляется приоритет задачи и она помещается в цепочку задач. После этого из библиотеки программ выбирается требуемая прикладная программа, которая загружается в оперативную память. После загрузки прикладной программы в оперативную память ей передается управление и она выполняется до тех пор, пока в ней не встретится макрокоманда обращения к функциональным средствам системы.

Операции ввода-вывода в прикладных программах выполняются средствами системы КАМА и требуют использования специальных макрокоманд. В состав макрокоманд, предоставляемых пользователям системой КАМА, входят макрокоманды управления памятью, задачами,

терминалами и связью.

§ 7.5. МУЛЬТИТЕРМИНАЛЬНАЯ СИСТЕМА РАСПРЕДЕЛЕННОЙ ОБРАБОТКИ ДАННЫХ ОБЬ

Система ОБЬ – универсальная программная система, предназначенная для решения широкого класса задач в мультитерминальном интерактивном режиме с распределенной средой хранения и обработки информации. Это могут быть различные задачи, решаемые в АСУ с использованием средств теледоступа, и задачи по реализации информационно–справочных и информационно–поисковых систем и автоматизированных рабочих мест программистов.

Система ОБЬ принадлежит к классу управляющих программ. Объекты ее управления следующие: обрабатывающие программы; устройства документирования информации; терминалы; сопряженные ЭВМ. Кроме того, имеются средства управления системой в процессе ее работы, позволяющие изменять конфигурацию обслуживаемых периферийных устройств и состав обрабатывающих программ без перезапуска системы. Обеспечивается накопление статистической информации о работе системы. Объектом управления системы ОБЬ является также вспомогательный набор данных на томе прямого доступа, так называемый файл работ.

Обрабатывающие программы. Такие программы разрабатываются пользователями системы, и их назначение может быть произвольным. Система ОБЬ обеспечивает динамический вызов программ с терминалов; мультитерминальный доступ к обрабатывающим программам; передачу обрабатывающим программам входной информации, которая может быть подготовлена с помощью средств управления терминалами системы ОБЬ и записана в файл работ в виде запросов пользователей; прием выходной информации от обрабатывающих программ и запись ее в файл работ системы в виде ответов пользователям терминалов; запись произвольной текстовой информации от обрабатывающих программ на терминалы пользователей; чтение обрабатывающими программами информации с терминалов пользователей; передачу данных программам по стандартным соглашениям связи через поле PARM оператора EXEC; отключение терминала от системы ОБЬ и передачу его под непосредственное управление обрабатывающей программы с указанием ей всех параметров, необходимых для взаимодействия с ним (имя оператора DD, описывающего терминал, списки опроса и выборки и т. д.).

Система ОБЬ позволяет применять в обрабатывающих программах любые средства ОС ЕС. Программы могут быть написаны на языке Ассемблер или любом из языков высокого уровня. Взаимодействие между системой ОБЬ и обрабатывающими программами реализуется следующим образом: по стандартным соглашениям передачи информации через поле PARM оператора EXEC; операторами ввода-вывода языков программирования (например, GET/PUT – на языке ПЛ/1, READ /WRITE – на ФОРТРАНе), причем нет ограничений на использование спецификаторов форматов в этих операторах; с помощью оператора CALL.

Обрабатывающие программы могут работать в зоне оперативной памяти системы ОБЬ и выполняться как самостоятельные задания ОС. В течение одного сеанса работы системы программа может запускаться в зоне системы и вне ее. Режимами выполнения программы управляет оператор. Для пользователя обращение к программе, работающей в зоне памяти системы ОБЬ, не отличается от вызова этой программы, выполняющейся как задание ОС.

Для управления терминалами используется метод доступа EXCP. Диалог, который пользователь терминала ведет с системой, позволяет ему осуществлять следующее: формировать и корректировать данные, которые могут содержать произвольную текстовую информацию и оформляются в виде запросов и ответов (данные хранятся в файле работ системы и используются для передачи обрабатывающим программам, пользователям терминалов или распечатываются на устройствах документирования); управлять вызовом обрабатывающих программ, документированием информации на печатающих устройствах; получать информацию о доступных ему ресурсах; посылать сообщения пользователям терминалов или оператору системы; управлять отключением терминала от системы ОБЬ с возможностью возврата его в систему; управлять форматом отображения информации на терминале.

Устройства документирования информации. Такие устройства, обслуживаемые системой ОБЬ, предназначены для распечатки произвольных текстовых данных; оформленных в виде запросов или ответов. Процедуру документирования информации инициализирует пользователь терминала с помощью команды, в которой указываются адрес устройства (принтера) и

необходимый запрос или ответ.

Система ОБЬ может функционировать на любой модели ЕС ЭВМ, кроме ЕС-1010, и требует для своей работы хотя бы один накопитель на магнитных дисках любого типа для размещения библиотеки загрузочных модулей системы и файла работ.

В комплекс технических средств, обслуживаемых системой, входят следующие устройства:

ЕС -7906, ЕС-7920 (локальный и удаленный варианты), ЕС-7910 (MERA), ЕС-7970, ЕС-8501 (АП-1, коммутируемый и некоммутируемый), ЕС-8504 (АП-4), ЕС-8534 (ТАП-34, коммутируемый и некоммутируемый), ЕС-8562 (АП-62 с дисплеями VTS-56100 и ИЗОТ-7925), ЕС-8564 (АП-64), ЕС-8566 (ЕС-ТЕЛ 4.2), СМ-1604 (НРБ), СМ-1611 (Куба), СМ-7209 (ПНР), VDT-52100 (ВНР), Телетайпы Т-63 (ГДР), ЕС-8575 М (АП-75);

любые устройства сопряжения ЕС ЭВМ, функционирующие по синхронному байториентированному способу передачи информации по выделенным одноточечным каналам связи (СНХ1);

любые устройства, пригодные для документирования информации, обслуживаемые базисным последовательным методом доступа (BSAM);

любые устройства, обеспечивающие накопление статистических данных, поддерживаемые последовательным методом доступа с очередями (QSAM).

В телемониторе ОБЬ обеспечиваются режимы межмашинного взаимодействия ЕС ЭВМ с персональным ЭВМ типа «Ро-ботрон-1715», «Искра-226», ЕС-1840 и IBM PC.

Терминалы. Обрабатывающие программы должны храниться в библиотеках загрузочных модулей, доступных системе в процессе ее работы. Такими библиотеками могут быть любые библиотеки, подключенные пользователями или операторами к системе во время работы; сцепленные библиотеки шага задания; сцепленные библиотеки задания; библиотека SYS1.LINKLIB или любая из библиотек, перечисленных в разделе LNKLSTOO этой библиотеки.

Системе ОБЬ через поле PARM оператора EXEC или на этапе инициализации указывается адрес (суффикс) списка обрабатывающих программ. В этом списке, подготавливаемом системным программистом, находятся имена программ и их характеристики (пакетная, терминальная, доступная для удаленного использования и т. д.). На основании информации из этого списка система обеспечивает пользователей терминалов соответствующим доступом к этим программам. Во время работы системы оператор имеет возможность изменять состав и характеристики обрабатывающих программ. При этом накладываются следующие ограничения: нельзя одновременно использовать больше обрабатывающих программ, чем первоначально было указано в списке; нельзя подключать новую обрабатывающую программу, если ее нет в библиотеках загрузочных модулей.

Состав обрабатывающих программ, доступных пользователю терминала на данный момент времени, отображается в виде таблиц ресурсов (локальных и удаленных). Содержимое этих таблиц выводится пользователю в ответ на команды получения информации о состоянии ресурсов. Система ОБЬ поддерживает достоверность этих таблиц на любой момент времени. Наряду с обрабатывающими программами в таблицах указываются терминалы, принтеры, удаленные ЭВМ, операторы.

Обрабатывающие программы составляются пользователями системы или персоналом, ответственным за ее эксплуатацию. Эти программы могут решать различные задачи (СУБД, терминальные программы, сервисные программы и т. д.). Система ОБЬ обеспечивает мультитерминальный доступ к программе и передачу информации между пользователями терминалов и программами.

Пользователи терминалов взаимодействуют с системой ОБЬ с помощью диалога. Диалог включает в себя следующие понятия: листинг; фрейм; диагностика; команда; режим.

Л и с т и н г представляет собой область, в которой хранится информация о диалоге с пользователем терминала за какой-то промежуток времени. Листинг реализуется в виде буфера памяти, в который записываются введенные с терминала команды, сообщения диалога, определенные сообщения межпроцессного взаимодействия. Размер листинга (буфера памяти) зависит от типа терминала. Для АП-1 листинг не обеспечивается программными средствами, так как вся предыстория диалога сохраняется на рулоне бумаги пишущей машинки.

Ф р е й м – область, через которую происходит обмен данными с пользователем терминала. Данные могут быть следующими: тексты запросов, ответов или писем во время их создания,

просмотра или корректировки; справочная информация о состоянии данных пользователя; таблицы локальных и удаленных ресурсов системы. Фрейм реализуется в виде буфера памяти, который выделяется в момент перехода в режим фрейма и освобождается в момент выхода из этого режима. Управление фреймом, т. е. изменение его содержимого, осуществляется с помощью подкоманд команды, которая перевела терминал в режим фрейма.

Диагностика представляет собой сообщение, предназначенное для разрешения пользователю терминала ввода определенных команд или подкоманд. Пользователь не имеет права вводить команды или подкоманды до тех пор, пока диагностика не появится на терминале. На печатающих терминалах диагностика может отображаться специальным символом, например #. Пример диагностики:

RCS001I ВВОДИТЕ КОМАНДУ

Команда вводится пользователем терминала (после появления диагностики) в соответствующее поле. Начало поля отмечается курсором (для дисплеев) или положением каретки (для печатающих терминалов).

Режимы работы пользователя терминала системы ОБЬ следующие: листинг, фрейм; терминальная обрабатывающая программа.

Режим листинга – основной режим работы пользователя. В этом режиме отображаются листинг и диагностика, после появления которой можно вводить команды. Переход в режимы фрейма и терминальной обрабатывающей программы производится из режима листинга с помощью команд.

В режиме фрейма пользователь может вводить только подкоманды той команды, которая осуществила перевод терминала в режим фрейма. Возврат терминала в режим листинга выполняется с помощью специальной подкоманды.

В режиме терминальной обрабатывающей программы пользователь работает по алгоритму обрабатывающей программы. После окончания работы с программой происходит возврат терминала системе и перевод его в режим листинга. Пользователь терминала имеет возможность перевести терминал в режим терминальной обрабатывающей программы без вызова программы. В этом случае терминал становится доступным для распределения другим программам, работающим параллельно с системой ОБЬ и ОС ЕС. Пользователь может запросить возврат терминала системе через заданное время.

В режимах фрейма и терминальной обрабатывающей программы система запоминает все сообщения, адресованные пользователю. Листинг отображается на терминале в момент перехода терминала в режим листинга.

Накопление статистических данных о работе системы ОБЬ осуществляется процессами блока управления сбором статистики. Статистическими данными являются команды оператора; наиболее важные сообщения, выдаваемые системой (динамическая реконфигурация, состояние процессов, рестарт системы и др.); внутренние управляющие сообщения, по которым можно отследить работу процессов (терминалов, программ, ЭВМ и др.).

В мультитерминальной системе ОБЬ обеспечено сопряжение с системами интерпретации неформальных запросов ТЕЛЕСПРАВКА, NATURAL, с помощью которых может быть реализовано эффективное взаимодействие с базами данных, поддерживаемыми СУБД СИОД, СЕДАН и другие. Также под управлением системы ОБЬ работают компоненты дисплейной отладки KDO и редактирования текстов JESSY, программы проверки и оперативного восстановления данных на дисках ПДТ и лентах ПЛТ; система оперативного информирования ЛИСТЕР, система выборочного просмотра линейных массивов ТЕРМЕС и ряд других проблемно-ориентированных программ.

По сравнению с монитором телеобработки данных КАМА преимущество системы ОБЬ заключается в ее способности изменять конфигурацию обслуживаемого периферийного оборудования как автоматически во время работы (по состоянию готовности устройств), так и принудительно (по командам оператора). Помимо этого телемонитор ОБЬ допускает взаимодействие пользователей терминалов с различными программами, предназначенными для работы в среде ОС ЕС. Каждая прикладная программа может быть загружена как в зону оперативной памяти, отведенной телемонитору, так и запускаться как самостоятельное задание ОС ЕС.

§ 7.6. ДИАЛОГОВЫЕ ПРОГРАММНЫЕ СРЕДСТВА АВТОМАТИЗИРОВАННОГО РАБОЧЕГО МЕСТА ПРОГРАММИСТА

Для повышения эффективности труда программистов при подготовке и отладке программ используются подключенные к системе ОБЬ в числе прочих компонент дисплейной отладки программ (КДО) и универсальное программное средство проверки дискового тома (ПДТ).

Компонент дисплейной отладки программ. Такой компонент служит для организации рабочего места программиста и предоставляет пользователям возможности одновременного обслуживания расширенного спектра терминалов, поддерживаемых системой ОБЬ в режиме разделения времени и терминалов типа пишущей машинки и обеспечения запуска заданий на исполнение без добавления в ОС программы SVC; чтения выходного листинга задания и т. д.

Основное назначение КДО – обеспечить повышение эффективности труда программиста в процессе подготовки и отладки программ за счет замены трудоемкой работы по подготовке данных на перфокартах на более оперативную и наглядную работу на основе алфавитно-цифровых дисплеев; использования более доступных и простых средств ведения библиотек, чем аналогичные утилиты операционной системы; наличия команд–подсказок; применения принципа выбора команд из предлагаемого «меню» и большого количества диагностических сообщений, позволяющих быстро освоить работу с КДО.

Компонент дисплейной отладки предоставляет пользователям возможность на экране алфавитно-цифрового дисплея просматривать оглавление тома прямого доступа, исключать, переименовывать наборы данных с любой организацией. Пользователям доступны характеристики тома (количество свободной памяти на томе и в оглавлении тома, количество дефектных дорожек и т. д.) и наборов данных (организация, формат и размер блоков, количество и распределение экстентов, количество свободного места, признаки защищенности набора данных и другие).

При работе с оглавлением произвольных библиотек пользователь может просматривать оглавление, исключать, переименовывать разделы, назначать разделу альтернативное имя, просматривать данные пользователя, включенные в элемент оглавления.

При работе с разделами символических библиотек пользователь может просматривать раздел на экране, листая его вперед и назад, постранично и построчно, а также производить поиск нужной страницы путем задания признака-метки, номера или контекста. Избранные строки раздела можно корректировать, замещать, удалять, вставлять. Строки раздела можно перенумеровывать.

При создании нового раздела в него можно включать данные, вводимые с экрана и копируемые из разделов других библиотек. Разделы можно выводить на печать или на перфокарты. Задание, представленное разделом, может быть запущено для выполнения. Выходной листинг задания может быть выведен на экран дисплея.

Компонент дисплейной отладки функционирует под управлением системы ОБЬ на любой модели ЕС ЭВМ, кроме ЕС-1010; КДО работает со всеми терминалами, обслуживаемыми системой ОБЬ. Постоянно требуемая оперативная память не превышает 40 К.

Максимальное количество терминалов, одновременно обслуживаемых КДО, равно 16. Использование возможностей вывода на печать, перфокарты или магнитные ленты требует наличия соответствующих устройств. Для наглядности и простоты ввода и вывода информации экран дисплея разбивается при работе системы КДО на три части: первую (верхнюю) строку, служащую для вывода диагностических сообщений; вторую строку – для вывода команд КДО; остальные строки – для вывода команд, доступных в данный момент, с указанием формата команд и с комментарием к ним, а также для вывода необходимых пользователю страниц томов прямого доступа, страниц оглавления библиотек, страниц разделов библиотек. Эти строки служат также для ввода данных при создании, исправлении или обновлении разделов.

Входной информацией являются команды КДО, библиотечные и последовательные наборы данных, выходной информацией – библиотечные и последовательные наборы данных, диагностические сообщения.

Управление работой КДО производится с помощью команд, вводимых с терминалов и выполняемых в режиме интерпретации. Команды КДО связаны между собой по иерархическому принципу. В каждый момент времени для выполнения доступны только команды выбранного

уровня иерархии («меню»).

Программное средство проверки дискового тома. Основное назначение универсального средства ПДТ - осуществление в интерактивном режиме при помощи видеотерминалов анализа информации, записанной на магнитные диски. Универсальность программы заключается в ее полной независимости от структур данных, расположенных на дисковом томе. Программа позволяет читать любые записи на диске, подготовленном для работы в ОС ЕС. Допускаются чтение произвольной записи (включая запись RO) по ее дисковому адресу, отображение и обновление полей ключа и данных. Программа проверки дискового тома включает вспомогательные средства сканирования тома прямого доступа; последовательный просмотр записей по возрастанию или убыванию дисковых адресов; поиск метки в оглавлении тома по имени набора данных; непосредственный выход на начало первого экстента набора данных по его имени; непосредственный выход на начало раздела библиотечного набора данных по имени раздела. С помощью программы ПДТ можно перезаписывать данные и назначать альтернативные дорожки для дефектных.

Программное средство ПДТ функционирует как под управлением системы ОБЬ, так и автономно на любой модели ЕС ЭВМ, кроме ЕС-1010. Постоянно требуемая оперативная память не превышает 40 К. Отображение информации осуществляется на экраны дисплеев всех типов, поддерживаемых телемонитором ОБЬ.

Входной информацией ПДТ являются команды ПДТ и любые записи на диске, выходной информацией – любые записи на диске и диагностические сообщения. Просмотр записей на экране дисплея может производиться по символьному, шестнадцатеричному и командному форматам. На экран дисплея можно вызывать произвольные фрагменты записи или анализировать информацию кадрами.

Глава 8. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МИКРОПРОЦЕССОРНЫХ ЭВМ

В главе рассматриваются вопросы программирования на языке Ассемблер для персональных ЭВМ, совместных с IBM PC. Помимо IBM PC в этот класс компьютеров входят ЕС-1840, ЕС-1841, «Искра»-1030», «Правец-16», «Роботрон-1910» и др. Эти ЭВМ работают под управлением операционных систем типа MS DOS. Приводится логическая структура персональных ЭВМ указанного класса. Описание MS DOS содержит представление о командном процессоре, ядре операционной системы и базовой системе ввода-вывода. Рассматриваются команды языка Ассемблер, способы адресации данных, сегментная организация памяти.

§ 8.1. МИКРОПРОЦЕССОРЫ В ПРОЕКТИРОВАНИИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

В начале 70-х годов микропроцессоры, интерфейсные микросхемы и полупроводниковая память произвели революцию в проектировании вычислительных систем. Это объясняется программируемостью микропроцессоров, а также тем, что однокристальный микропроцессор по своим возможностям эквивалентен сотням микросхем с малой и средней степенью интеграции. Микропроцессоры применяют во все большем числе вычислительных систем, реализованных ранее на так называемой «жесткой логике», и эта тенденция сохраняется. Причины этой тенденции станут понятными при рассмотрении основных критериев, учитываемых при проектировании вычислительных систем: стоимости, гибкости, надежности, времени проектирования, быстродействия.

Стоимость. Кроме стоимости собственно микросхем на единичную стоимость системы влияют факторы приобретения, хранения и контроля микросхем, оплаты монтажа, пайки и внешнего оформления, расходы на приобретение печатных плат, блоков питания, корпусов и т. д. Расходы на приобретение микросхем обычно не превышают 10 % общей стоимости системы, однако единичная стоимость пропорциональна числу микросхем, а не их внутренней сложности. Следовательно, экономичнее применять более дорогие БИС и СБИС, если они заменяют достаточное число микросхем с малой и средней степенью интеграции.

Гибкость. После запуска системы в производство могут потребоваться ее модификации или усовершенствования, что объясняется выявлением по результатам эксплуатации просчетов проектирования или необходимостью введения новых функций. Если система реализована на основе «жесткой логики», ее придется заново проектировать, модифицировать и проверять, на что затрачивается много времени и средств. С другой стороны, благодаря программируемости микропроцессора, изменения касаются в основном управляющей программы, а не аппаратных

средств.

Надежность. Интенсивность отказов системы пропорциональна числу микросхем, а поэтому надежность системы увеличивается по мере роста сложности микросхем и уменьшения их числа. Кроме того, сокращение числа микросхем ведет и уменьшению межсоединений с соответствующим упрощением решения проблем отказов, помех и синхронизации.

Время проектирования. Процесс традиционного проектирования систем на основе «жесткой логики», последовательный, т. е. следующий этап невозможно начинать до завершения предыдущего. Проектирование же микросхемы, т. е. разработки аппаратных и программных средств, можно проводить параллельно.

Быстродействие. Основное преимущество системы на основе «жесткой логики» перед микросистемой заключается в том, что она намного быстрее реагирует на входные воздействия. Однако во многих случаях высокого быстродействия не требуется и определяющим фактором оказывается гибкость программируемой системы. Кроме того, производительность новых микропроцессоров увеличивается и расхождение в быстродействии сокращается.

Эра микропроцессоров началась с тех пор, когда технология позволила реализовать в одной микросхеме все необходимые функции центрального процессора. Совершенствование микропроцессоров шло параллельно развитию микроэлектронной технологии, позволяющей размещать на кристалле все большее количество логических схем. По мере усложнения микропроцессора растет и его стоимость, но она все же намного меньше стоимости эквивалентной системы, реализованной на микросхемах с меньшими функциональными возможностями. Кроме уменьшения числа микросхем, необходимых для реализации данной функции, сокращается и общее число контактов, и расходы на монтаж.

По мере перехода технологии от БИС к СБИС микропроцессоры стали иметь не 4 линии данных и 12 линий адреса (4 из которых были к тому же линиями данных), а 16 линий данных и 20...24 независимые линии адреса. Первые микропроцессоры подходили только для калькуляторов и простых контроллеров, а современные микропроцессоры допускают адресацию памяти до 24 М байт и имеют средства мультипрограммирования и организации мультипроцессорных систем. Вместе с новыми микропроцессорами выпускаются разнообразные микросхемы, предназначенные для реализации памяти, интерфейсов и управления шиной.

Рассмотрим для иллюстрации развитие базового семейства микропроцессоров фирмы «INTEL» (США). Это семейство началось с первого 4-битового микропроцессора 4004 и семейства 4000 и пришло к 16-битовому микропроцессору 8086 с рассчитанным на него многочисленными вспомогательными микросхемами. Микропроцессоры 8008, 8080 и 8085 относятся к классу 8-битовых, причем каждый следующий из них оказывается более сложным и гибким. Микропроцессор 8088 представляет собой 8-битовый вариант микропроцессора 8086; он имеет меньше линий данных, но сохраняет все функциональные возможности микропроцессора 8086. Микропроцессоры 80186 |и 80286 – однокристальные расширения микропроцессора 8086, обладающие более высокими быстродействием и дополнительными вычислительными возможностями, которые особенно важны при проектировании сложных микросистем. Производительность семейства микропроцессоров фирмы «INTEL за прошедшие годы увеличилась в тысячи раз. В нашей стране выпускаются аналоги, например микропроцессору 8080 соответствует микропроцессор КР580ИК80А, микропроцессору 8086 – микропроцессор К1810ВМ86.

§ 8.2. ПРОГРАММНЫЕ СРЕДСТВА МИКРОПРОЦЕССОРНЫХ ЭВМ

Набор программных средств микропроцессорных систем имеет структуру, сходную со структурой набора программных средств для ЭВМ третьего поколения (ЕС ЭВМ и СМ ЭВМ). Программные средства микрокомпьютеров подразделяются на системные и прикладные.

Системные программные средства. Эти средства (или системное программное обеспечение) образуют программы, необходимые для создания, подготовки и выполнения других программ.

Операционная система является ядром системного программного обеспечения и представляет собой совокупность системных программ, обеспечивающих взаимодействие пользователя с ЭВМ и эффективное использование его ресурсов. Наиболее важная часть операционной системы – резидентный монитор, постоянно находящийся в памяти компьютера. Резидентный монитор

должен воспринимать приказы пользователей и инициировать выполнение операционной системой соответствующих действий.

В операционную систему входят также драйверы ввода-вывода, предназначенные для выполнение операций ввода-вывода, и процедуры управления файлами, т. е. наборами данных, хранящимися во внешней памяти. Когда прикладной или системной программе требуется обращение к устройству ввода-вывода, она не выполняет эту операцию сама, а запрашивает операционную систему, которая вызывает драйвер ввода-вывода. При таком подходе обеспечивается лучшее управление ЭВМ и отпадает необходимость введения подпрограммы ввода-вывода в пользовательские программы. Процедуры управления файлами применяются вместе с драйверами ввода-вывода внешней памяти для доступа, копирования и других операций с файлами.

В системные программные средства входят также трансляторы с языков высокого уровня, Ассемблер, редакторы текстов и программы, помогающие в разработке других программ. Имеется три уровня программирования на языках машинном, Ассемблер, высокого уровня.

Программы на машинном языке – программы, которые компьютер может непосредственно «понимать» и выполнять.

Программы на языке Ассемблер характеризуются тем, что команды более или менее однозначно соответствуют машинным командам, но написаны в виде символьных цепочек, более понятных человеку.

Команды на языке высокого уровня намного ближе к естественному языку (как правило, английскому) и структурированы так, что близко соответствуют мышлению программиста. Программу на языке Ассемблер или на языке высокого уровня необходимо перевести на машинный язык, для чего применяются программы, называемые *трансляторами.* Транслятор программ, написанных на языке Ассемблер, называется *Ассемблером,* а на языке высокого уровня – *компилятором* или *интерпретатором.*

Редактор текста – программа для ввода или модификации текста (букв, цифр, знаков пунктуации и др.), который необходимо запомнить или который уже хранится во внешней памяти. Текст может представлять собой программу на языке Ассемблер или языке высокого уровня, набор данных или отчет. Редакторы текста применяются для различных целей, но программисты используют их, как правило, для создания программ.

При подготовке программ к выполнению требуются еще две системные программы. Часто возникает ситуация, когда одну и ту же задачу должны выполнять несколько программ. Поэтому в большинстве ОС предусматривается возможность образования наборов подпрограмм, называемых библиотеками, которые можно подключать к любой системной или пользовательской программам. Обычно имеется системная библиотека широкого назначения, но пользователи могут создавать и личные библиотеки. Вспомогательная программа, объединяющая выполняемую программу с библиотекой и другими ранее транслированными подпрограммами, называется Редактором связей или компоновщиком. Служебная программа, предназначенная для размещения выполняемой программы в памяти, называется Загрузчиком. Иногда функции редактирования связей и загрузки реализует одна программа.

Прикладные программные средства включают в себя программы, разработанные пользователями в их попытках применить компьютер для решения своих задач.

§ 8.3. АРХИТЕКТУРА МИКРОПРОЦЕССОРА 8086

Рассмотрим 16-битовый микропроцессор 8086 фирмы «INTEL» и его отечественный аналог – микропроцессор К1810ВМ86, в настоящее время наиболее часто используемый в персональных ЭВМ, выпускаемых нашей промышленностью (ЕС-1840, ЕС-1841, «Искра-1030», НЕЙРОН и др.).

Шестнадцатибитовый микропроцессор 8086 содержит на кристалле около 29 тыс. транзисторов и производится по высококачественной МОП–технологии. Производительность его значительно выше 8-битового предшественника – микропроцессора 8086. Число линий адреса увеличено с 16 до 20, что позволяет адресовать память 1 М байт вместо 64 К байт. Увеличение емкости памяти обеспечивает переход к мультипрограммированию, поэтому в микропроцессоре 8086 предусмотрено несколько мультипрограммных возможностей. В микропроцессор 8086 встроены некоторые средства, упрощающие реализацию микропроцессорных систем, что позволяет

применять его с другими процессорами, например с процессором числовых данных 8087 и процессором ввода-вывода 8089.

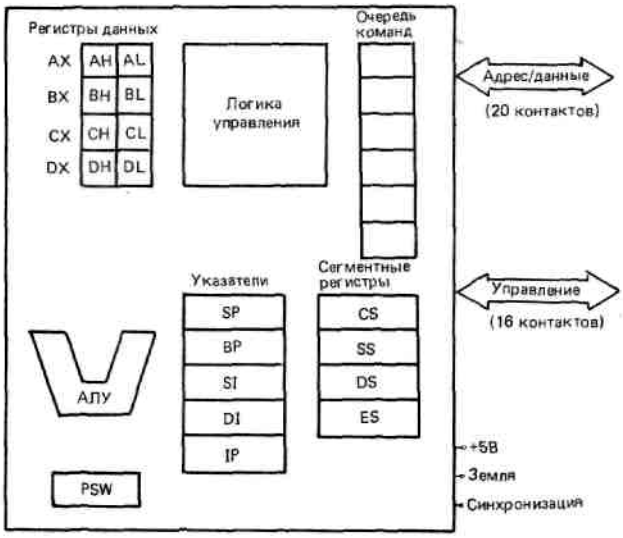


Рис. 8.1. Внутренняя архитектура микропроцессора 8086

Архитектура микропроцессора 8086. Внутренняя архитектура микропроцессора 8086 представлена на рис. 8.1. За исключением регистра команд, которым фактически является 6-байтовая очередь, в соответствии с выполняемыми функциями рабочие регистры разделены на три группы: регистров данных, представляющих собой набор арифметических регистров; указателей, содержащих базовые и индексные регистры, а также программный счетчик и указатель стека; сегментных регистров, в состав которых входят специальные базовые регистры. Все регистры имеют длину 16 бит.

В группу регистров данных входят регистры AX, BX, CX и DX, предназначенные для хранения операндов и результатов операций и допускающие адресацию не только целых регистров, но и их младшей и старшей половин. Например, допускается использование двух байтов в регистре AX вместе, а также указывание отдельных байтов AL (младший) и AH (старший). Регистры кроме арифметических выполняют следующие специальные функции: BX служит базовым регистром в вычислениях адреса; CX в некоторых командах выступает неявным счетчиком; DX в некоторых операциях ввода-вывода содержит адрес порта ввода-вывода.

Указательная и индексная группы представлены регистрами IP, SP, BP, SI и DI. Указатель команды IP и регистр SP фактически являются программным счетчиком и указателем стека (специальной области памяти), но полные адреса команды и стека образуются суммированием содержимого этих регистров и сегментных регистров CS и SS. Регистр BP - базовый при обращении к стеку и может применяться с другими регистрами и (или) смещением, являющимся частью команды. Регистры SI и DI предназначены для индексирования. Их можно использовать самостоятельно, но они часто комбинируют с регистрами BX и BP и (или) смещением. За исключением регистра IP, любой из указателей может хранить операнд, но допускает обращения только к 16-битовому регистру.

Для обеспечения гибкой базовой адресации и индексирования адрес данных формируют путем сложения содержимого регистров BX или BP, содержимого регистров SI или DI и смещения. Результат вычислений адреса называется *эффективным адресом* EA или *смещением.* В руководствах фирмы «INTEL» термин «эффективный адрес» применяется в контексте машинного языка, а термин «смещение» (offset) – в контексте языка Ассемблер. Слово «смещение» (displacement) означает величину, которая прибавляется к содержимому регистра(ов) для образования EA. Окончательный адрес данных определяется EA и соответствующим сегментным регистром DS, ES или SS.

Сегментную группу образуют регистры CS, SS, DS и ES. Участвующие в формировании адреса регистры BX, IP, SP, BP, SI и DI имеют длину всего 16 бит, поэтому эффективный адрес имеет такую же длину. С другой стороны, выдаваемый на шину адреса физический адрес должен содержать 20 бит. Дополнительные четыре бита образуются сложением эффективного адреса с содержимым одного из сегментных регистров (рис. 8.2). Перед сложением

к содержимому сегментного регистра справа добавляются четыре нуля, что дает 20-битовый результат. Если, например, (CS) = 123A и (IP)=341B, следующая команда будет выбираться по адресу 157BB. Здесь круглые скобки подразумевают слово «содержимое»; все адреса даются в шестнадцатеричной системе.

Применение сегментных регистров, по существу, разделяет пространство памяти на перекрывающиеся сегменты, каждый из которых имеет размер 64 К байт и начинается на 16-байтовой границе (называемой границей параграфа), т. е. начинается с адреса, кратного 16.

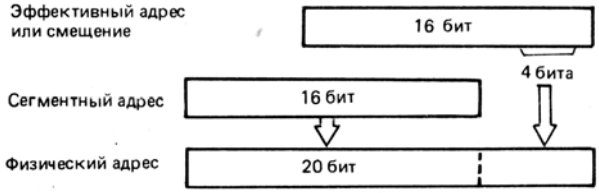


Рис. 8.2. Формирование физического адреса

Далее содержимое сегментного регистра будем называть *сегментным адресом,* а сегментный адрес, умноженный на 16, – *начальным физическим сегментным адресом* или начальным сегментным адресом.

Благодаря наличию сегментных регистров обеспечивается следующее: емкость памяти может доходить до 1 М байт, хотя команды оперируют 16-битовыми адресами; секции кода, данных и стека могут иметь длину более 64 К байт при использовании нескольких сегментов кода, данных или стека; упрощается применение отдельных областей памяти для программы, ее данных и стека; при каждом выполнении программы она сама и(или) ее данные могут размещаться в различных областях памяти.

Слово состояния микропроцессора. Слово состояния процессора PSW микропроцессора 8086 содержит 16 бит, семь из них используются. Флажки микропроцессора 8086 (рис. 8.3)

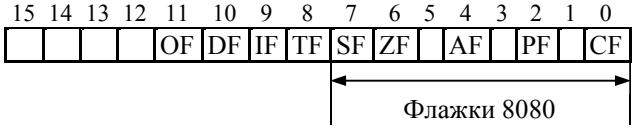


Рис. 8.3. Слово состояния микропроцессора 8086

разделяются на условные, отражающие результат предыдущей операции АЛУ, и управляющие, от которых зависит выполнение специальных функций. Младший байт в PSW соответствует 8-битовому PSW микропроцессора 8080 и содержит все флажки условий, кроме OF.

Ф л а ж к а м и у с л о в и й являются следующие флажки: знака, нуля, паритета, переноса, вспомогательного переноса, переполнения.

Флажок знака SF равен старшему биту результата. В дополнительном коде старший бит отрицательных чисел содержит 1, у положительных чисел он равен 0; флажок SF показывает знак предыдущего результата.

Флажок нуля ZF устанавливается в 1 при получении нулевого результата и сбрасывается в 0, если результат отличается от нуля.

Флажок паритета PF устанавливается в 1, если младшие 8 битов результата содержат четное число единиц, в противном случае сбрасывается в 0.

Флажок переноса CF при сложении (вычитании) устанавливается в 1 в том случае, если возникает перенос (заем) из старшего бита. Другие команды также воздействуют на этот флажок.

Флажок вспомогательного переноса AF устанавливается в 1 в том случае, если при сложении (вычитании) возникает перенос (заем) из третьего бита. Флажок предназначен только для двоично-десятичной арифметики и непосредственно не может быть проверен командами условных переходов.

Флажок переполнения OF устанавливается в 1 в том случае, если возникает переполнение, т. е. получение результата вне допустимого диапазона. При сложении этот флажок устанавливается тогда, когда имеется перенос в старший бит и нет переноса из старшего бита, или наоборот. При вычитании он устанавливается тогда, когда возникает заем из старшего бита, а заем в старший бит

отсутствует, или наоборот.

Пусть, например, предыдущая команда производила следующее сложение:

```
  0010  0011  0100  0101
+ 0011  0010  0001  1001
-------------------------
  0101  0101  0101  1110
```

Тогда после выполнения команды получают следующие состояния флажков: SF=0, ZF=0, PF=0, CF=0, AF=0, OF=0.

Если в предыдущей команде выполнялось сложение

```
  0101  0100  0011  1001
+ 0100  0101  0110  1010
-------------------------
  1001  1001  1010  0011
```

то флажки принимают следующие состояния: SF=1, ZF=0, PF=1, CF=0, AF=l, OF=l.

Флажки управления микропроцессора 8086 следующие: направления, разрешения прерываний, прослеживания.

Флажок направления DF применяется в командах манипуляций цепочками. Если он сброшен, цепочка обрабатывается с первого элемента, имеющего наименьший адрес, в противном случае цепочка обрабатывается от наибольшего адреса к наименьшему.

Флажок разрешения прерываний IF позволяет микропроцессору распознавать маскируемые прерывания, в противном случае эти прерывания игнорируются.

Флажок прослеживания (трассировки) TF обеспечивает после выполнения каждой команды генерацию внутреннего прерывания.

Формат команды. Команда разделяется на группы битов или поля, причем поле кода операции показывает на то, что должен делать компьютер, остальные поля, называемые операндами, идентифицируют требуемую команде информацию (рис. 8.4). Операнд может содержать данное, часть адреса данного, косвенный указатель данного или другую информацию, относящуюся к обрабатываемым командой данным.

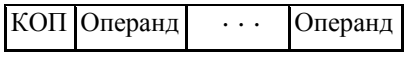


Рис. 8.4, Общий формат команды

Команды могут содержать несколько операндов, но чем больше операндов и чем они длиннее, тем больше места занимает команда в памяти и тем больше времени требуется для передачи ее в микропроцессор. Для минимизации общего числа битов в команде большинство команд, особенно в 16-битовых компьютерах, имеют один или два операнда, причем минимум один из операндов в двухоперандной команде является регистром. Так как пространства памяти и ввода-вывода относительно велики, адреса памяти и ввода-вывода требуют сравнительно много битов, а из-за ограниченного числа регистров для определения регистров требуются всего несколько битов. Следовательно, для экономии длины команды следует максимально пользоваться регистрами. Ограничение двумя операндами, конечно, уменьшает гибкость многих команд, но в действительности излишняя гибкость не нужна.

Режимы адресации. Способ определения операнда называется *режимом адресации.* Режимы адресации микропроцессора 8086 разделяются на режимы адресации данных и адресации переходов.

Режимы адресации данных графически представлены на рис. 8.5; рассмотрим их подробнее.

Непосредственный режим (рис. 8.5,*а*) характеризуется тем, что данное длиной 8 или 16 бит является частью команды.

Прямому режиму (рис. 8.5,*б*) свойственно то, что 16-битовый эффективный адрес данного является частью команды. Эффективный адрес EA суммируется с умножением на 16 с содержимым соответствующего сегментного регистра.

Регистровый режим (рис. 8.5,в) обладает тем свойством, что данное содержится в определяемом командой регистре; 16-битовый операнд может находиться в регистрах AX, BX, CX, DX, SI, DI, SP или BP, а 8-битовый-в регистрах AL, AH, BL, BH, CL, CH или DL, DH.

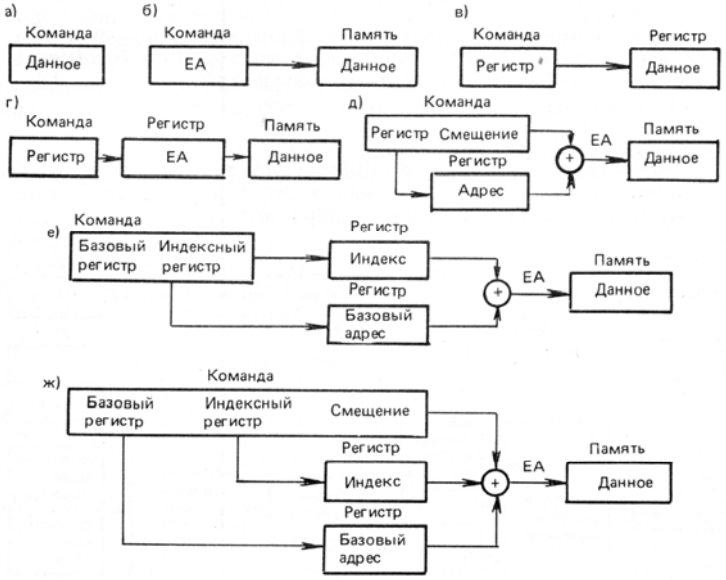


Рис. 8.5. Режимы адресации данных:

непосредственный (*а*); прямой (*б*); регистровый (*в*); регистровый косвенный (*г*); регистровый относительный (*д*); базовый индексный (*е*); относительный базовый индексный (*ж*)

В регистровом косвенном режиме (рис. 8.5,*г*) эффективный адрес данного находится в базовом регистре BX или индексном регистре, определяемых командой

$$EA = \left\{ \begin{matrix} (BX) \\ (SI) \\ (DI) \end{matrix} \right\}$$

В регистровом относительном режиме (рис. 8.5,*д*) эффективный адрес равен сумме 8- или 16-битового смещения и содержимого базового или индексного регистров:

$$EA = \left\{ \begin{matrix} BX \\ BP \\ SI \\ DI \end{matrix} \right\} + \left\{ \begin{matrix} 8-\text{битовое смещение} \\ 16\text{-битовое смещение} \end{matrix} \right\}$$

Базовый индексный режим (рис. 8.5,*е*) характеризуется тем, что эффективный адрес равен сумме содержимого базового и индексного регистров, определяемых командой

$$EA = \left\{ \begin{matrix} (BX) \\ (BP) \end{matrix} \right\} + \left\{ \begin{matrix} (SI) \\ (DI) \end{matrix} \right\}$$

Относительному базовому индексному режиму (рис. 8.5,*ж*) свойственно то, что эффективный адрес равен сумме 8- или 16-битового смещения и базового-индексного адреса:

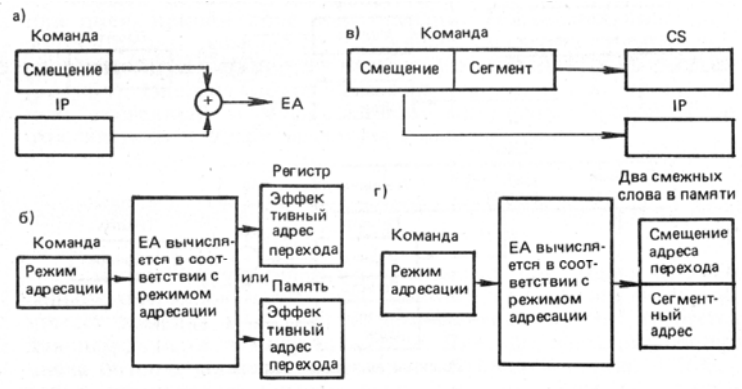


Рис. 8.6. Режимы адресации переходов:

внутрисегментный прямой (*а*); внутрисегментный косвенный (*б*); межсегментный прямой (*в*); межсегментный косвенный (*г*)

$$EA = \left\{ \begin{matrix} (BX) \\ (BP) \end{matrix} \right\} + \left\{ \begin{matrix} (SI) \\ (DI) \end{matrix} \right\} + \left\{ \begin{matrix} 8-\text{битовое смещение} \\ 16-\text{битовое смещение} \end{matrix} \right\}$$

Р е ж и м ы а д р е с а ц и и п е р е х о д о в графически определены на рис. 8.6; рассмотрим их подробнее.

Внутрисегментный прямой режим (рис. 8.6,*а*) характеризуется тем, что эффективный адрес перехода равен сумме 8-или 16-битового смещения и текущего содержимого IP. Когда смещение имеет длину 8 бит, этот режим называется *коротким переходом.* Внутрисегментную прямую адресацию в большинстве случаев называют относительной адресацией, так как смещение вычисляется относительно IP. Этот режим допустим в условных и безусловных переходах, но в команде условного перехода может быть только смещение длиной 8 бит.

Внутрисегментному косвенному режиму (рис. 8.6,*б*) свойственно то, что эффективный адрес перехода есть содержимое регистра или ячейки памяти, которые указываются в любом режиме (кроме непосредственного) адресации данных. Содержимое IP заменяется эффективным адресом перехода. Данный режим допустим только в командах безусловного перехода.

Межсегментный прямой режим (рис. 8.6,*в*) заключается в том, что заменяет содержимое IP одной частью команды, а содержимое CS – другой частью команды. Данный режим адресации предназначен для обеспечения перехода из одного сегмента кода в другой.

Межсегментный косвенный режим (рис. 8.6,г) характеризуется тем, что заменяет содержимое IP и CS содержимым двух

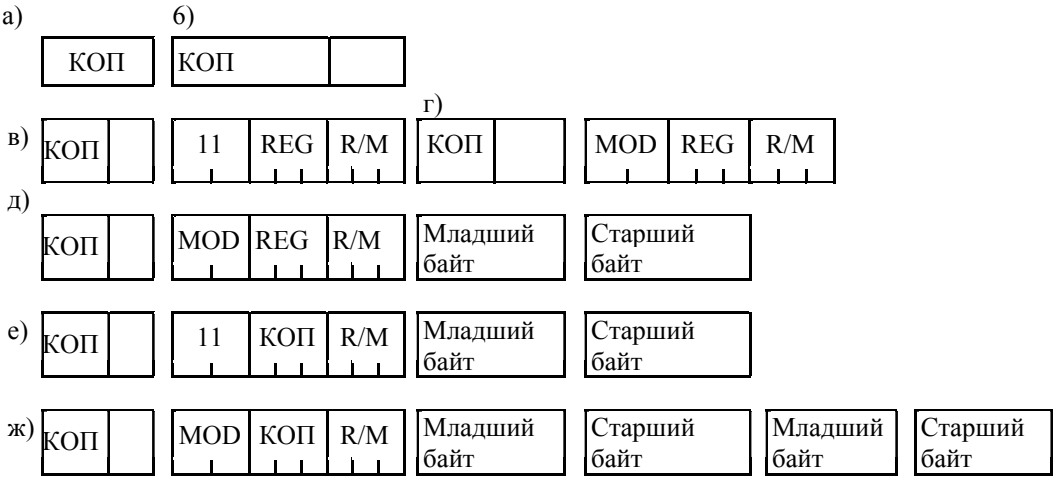


Рис. 8.7. Примеры форматов команд микропроцессора 8086:
однобайтовая команда, неявный операнд (*а*); однобайтовая команда, регистровый режим (*б*); регистр - регистр (*в*); регистр в память без смещения (*г*); регистр в память со смещением (*д*); непосредственный операнд в регистр (*е*); непосредственный операнд в память с 16-битовым смещением (*ж*)

смежных слов из памяти, которые определяются в любом режиме адресации данных, кроме непосредственного и регистрового.

Физический адрес перехода равен сумме нового содержимого IP и содержимого CS, умноженного на 16_{10}. Межсегментный переход может быть только безусловным.

Несколько типичных форматов команд микропроцессора 8086 приведены на рис. 8.7. Длина команд варьируется от 1 до 6 байт. Длина смещений и непосредственных данных может быть 8 или 16 бит в зависимости от команды. На рис. 8.7 приняты следующие обозначения: REG – регистр; MOD – режим; R/M – регистр или память; DISP – смещение; DATA – непосредственные данные.

В первых одном или двух байтах команды находятся код операции и указание режима адресации. После них может не быть ни одного дополнительного байта или находиться следующее: двухбайтовый эффективный адрес (только для прямой адресации), одно- или двухбайтовое смещение; одно- или двухбайтовый непосредственный операнд; одно- или двухбайтовое смещение с последующим одно- или двухбайтовым непосредственным операндом; двухбайтовое смещение и двухбайтовый сегментный адрес (только для прямой межсегментной адресации).

Применение одной из этих возможностей определяется кодом операции и режимом адресации. Если длина смещения или непосредственного операнда составляет 2 байт, первым всегда следует

младший байт.

Обычно код операции занимает первый байт команды, но в некоторых командах в первом же байте указывается регистр, а в нескольких других командах 3 бит кода операции находятся во втором байте. В большинстве кодов операций имеются такие однобитовые индикаторы, как W, D, S, V, Z.

Бит W имеется в коде операции в случае, если команда может оперировать байтом и словом. Этот бит определяет операцию с байтом (W=0) или словом (W=l).

Бит D содержится в двухоперандных командах (за исключением команд с непосредственным операндом и цепочечных команд). Одним из операндов должен быть регистр, определяемый полем REG. В таких командах бит D показывает, чем является регистр - операндом-источником (D=0) или операндом-получателем (D = l).

Бит S характеризуется следующим: 8-битовое число в дополнительном коде можно расширить до 16-битового в дополнительном коде, если сделать все биты старшего байта равными старшему биту младшего байта. Такая операция называется *расширением знака.* Бит S появляется вместе с битом W в командах сложения, вычитания и сравнения с непосредственным операндом и расшифровывается следующим образом:

8-битовая операция – S:W=00;

16-битовая операция с 16-битовым непосредственным операндом – S:W=01;

16-битовая операция с 8-битовым операндом, который расширяется со знаком до 16 бит,– S:W=11.

При небольших числах последний вариант допускает использование однобайтового непосредственного операнда.

Бит V применяется в командах сдвигов для определения числа сдвигов.

Бит Z используется в команде REP, которая применяется в операциях с цепочками символов.

Сегментный регистр определяется двумя битами, а любой другой регистр - тремя. Если на код операции и режим адресации отводится 2 байт, то второй байт имеет одну из двух форм, изображенных на рис 8.7. Одна из них, где присутствует код операции (КОП), предназначена для однооперандных команд (или двухоперандных, в которых один из операндов неявно определяется кодом операции). Вторая форма характерна для двухоперандных команд, причем поле REG определяет регистр, который в зависимости от бита D является операндом-источником или операндом-получателем.

Операнд, указываемый полями MOD и R/M, определяется в соответствии с табл. 8.1.

Если MOD ≠ 11, то эффективный адрес вычисляется по табл. 8.1. Отметим, что MOD=00 означает отсутствие смещения, за исключением случая, когда R/M = 110, который обозначает прямую адресацию. Комбинация MOD=01 означает, что третий байт команды содержит 8-битовое смещение, которое до вычисления эффективного адреса автоматически расширяется со знаком до 16 бит. Если MOD=10, то третий и четвертый байты команды содержат 16-битовое смещение. В том случае, если MOD = 11, операндом является регистр, адрес которого определяется полем R/M. В табл. 8.1 также показаны сегментные регистры, используемые в каждой из комбинаций полей MOD и R/M. Эффективный адрес операнда в памяти определяется полями MOD и R/M, но 20-битовый физический адрес равен сумме эффективного адреса и содержимого сегментного регистра, умноженного на 16. В режимах адресации с привлечением регистра BP с эффективным адресом суммируется содержимое сегментного регистра SS, в остальных режимах адресации участвует регистр DS.

Чтобы изменить используемые в соответствии с табл. 8.1 сегментные регистры, предусмотрена специальная однобайтовая команда, называемая *префиксом замены сегмента*; ее формат

0 0 1 REG 1 1 0

Если команде предшествует префикс замены сегмента, то при обращении к данным в процессе ее выполнения участвует сегментный регистр REG из префикса. Для приведенных в табл. 8.1 режимов адресации регистр DS можно заменить на CS, SS и ES, а регистр SS при участии в адресации регистра BP - на DS, CS и ES. Замену нельзя производить в следующих случаях: при вычислении адреса следующей выполняемой команды в качестве сегментного регистра всегда применяется CS; при участии в адресации регистра SP сегментным регистром всегда служит SS; в цепочечных командах в качестве сегментного регистра операнда-получателя всегда используется ES.

Таблица 8.1

Значения поля **R/M**	Эффективный адрес и адрес сегментного регистра при различных значениях поля				
	00	**01**	**10**	**11**	
				W=0	W = 1
000	(BX)+(SI) DS	(BX) + (SI) + (D8) DS	(BX) + (S1) + (D16) DS	AL	AX
001	(BX) + (DI) DS	(BX) + (DI) + (D8) DS	(BX) + (DI) + (D16) DS	CL	CX
010	(BP) + (SI) SS	(BP) + (SI) + (D8) SS	(BP) + (SI) + (D16) SS	DL	DX
011	(BP) + (DI) SS	(BP) + (DI) + (D8) SS	(BP) + (DI) + (D16) SS	BL	BX
100	(SI) DS	(SI)+(D8) DS	(SI) + (D16) DS	AH	SP
101	(DI) DS	(DI) + (D8) DS	(DI) + (D16) DS	CH	BP
110	D16 DS	(BP) + (D8) SS	(BP) + (D16) SS	DH	SI
111	(BX) DS	(BX) + (D8) DS	(BX) + (D16) DS	BH	DI

Примечание. 8-битовое смещение (расширяется со знаком) обозначено D8, а 16-битовое смещение - D16.

§ 8.4. ЯЗЫК АССЕМБЛЕР

За исключением программы на машинном языке, состоящей из комбинаций нулей и единиц, которые непосредственно дешифрирует компьютер, программа на языке Ассемблер является простейшей. В языке Ассемблер имеются операторы двух типов: команды, транслируемые в машинные команды; директивы, служащие указаниями ассемблеру во время процесса ассемблирования, не транслируемые в машинные команды.

Команды языка. Каждая ассемблерная команда порождает только одну машинную команду, которую проще писать благодаря тому, что аббревиатуры, называемые *мнемониками,* показывают тип команды, а символьные цепочки, называемые *идентификаторами,* представляют собой адреса и, возможно, числа. Типичная ассемблерная команда

```
ADD AX, COST
```

прибавляет содержимое слова памяти, ассоциируемое с идентификатором COST, к регистру AX. Аббревиатура ADD является мнемоникой команды. Директива

```
COST DW ?
```

заставляет Ассемблер зарезервировать слово (2 байт) и ассоциировать с ним идентификатор COST, но не порождает машинной команды.

Поскольку Ассемблер оказывается просто транслирующей программой, формат и синтаксис команд и директив определяется не компьютером, а тем, как написан Ассемблер. Здесь рассматривается Ассемблер ASM-86, разработанный фирмой «INTEL». Основные его характеристики аналогичны для других Ассемблеров.

Общий формат ассемблерной команды имеет следующий вид:

МЕТКА: МНЕМОНИКА ОПЕРАНД, ОПЕРАНД; КОММЕНТАРИЙ

Пробелы вводятся произвольно, но минимально один пробел должен быть в местах, где его отсутствие ведет к неоднозначности (например, между мнемоникой и первым операндом). Пробелы не допускаются в мнемониках и идентификаторах, а в цепочках-константах и в комментариях они должны вводиться специальными символами. Метка – идентификатор присваиваемый первому байту команды, у которой она появляется. Наличие метки в команде не обязательно, но если она есть, то становится символическим именем, применяемым в командах переходов для передачи управления отмеченной команде. При отсутствии метки двоеточия быть не должно. Во всех командах необходимо наличие мнемоник, которые допускается модифицировать префиксами, вводимыми по мере необходимости. Наличие операндов зависит от команды, в одних командах нет операндов, в других требуется один операнд, а в третьих – два

операнда. Два операнда разделяются запятой. Поле комментария предназначено для пояснения программы и может содержать любую комбинацию символов. Оно не обязательно, и при отсутствии комментария точка с запятой не нужна. Комментарием может быть целая строка, в этом случае первым символом в строке должна быть точка с запятой. Команду допускается переносить на следующие строки, помещая в начале каждой строки-продолжения символ амперсанда.

Ассемблерная команда должна иметь операнд для каждого операнда машинной команды, и обозначение каждого операнда должно идентифицировать режим его адресации. При двух операндах первым указывается операнд-получатель, вторым – операнд-источник.

Типичная ассемблерная команда, прибавляющая содержимое памяти к регистру AX:

BR: ADD AX,COUNT[BX]; прибавить к AX элемент массива COUNT.

Здесь BR – метка, обеспечивающая возможность перехода к этой команде; ADD – мнемоника; AX – операнд-получатель; COUNT[BX] – операнд-источник.

В ассемблерной команде используются такие режимы адресации как регистровый относительный и регистровый. Когда операндом является слово в памяти, то младший байт слова имеет меньший (младший) адрес, а старший – больший. Если, например, идентификатор COST обозначает операнд-слово, то COST ассоциируется с младшим байтом, а (COST+1) – со старшим байтом.

В табл. 8.2 показаны форматы операндов для различных режимов адресации.

Таблица 8.2

Режимы адресации данных и переходов	Форматы операндов	Примеры
Непосредственный	Выражение-константа	10110B 529 0A9H
Прямой	Переменная+выражение-константа	AB CHT-5
Регистровый	Регистр	AX
Регистровый косвенный	[Регистр]	[BX]
Регистровый относительный	Переменная [Регистр ± выражение-константа] или [Регистр ± выражение-константа]	CM [BX-1] [BP + 20H]
Базовый индексный	[Базовый регистр] [Индексный регистр]	[BP] [DI]
Относительный базовый индексный	Переменная [Базовый регистр ± выражение-константа] [Индексный регистр ± выражение-константа] или [Базовый регистр ± выражение-константа + Индексный регистр ± выражение константа]	E[BX] [SI-2] [BP + 2 + SI]
Внутрисегментный прямой	Метка + выражение-константа	M10 + 7
Внутрисегментный косвенный	Как и адресация данных, кроме непосредственной	
Межсегментный прямой	Метка ± выражение-константа	EXT
Межсегментный косвенный	Как и адресация данных, кроме непосредственной или регистровой	

Примечание. Во всех режимах, кроме непосредственного, «выражение-константа» не является обязательным.

Под переменной понимается идентификатор, который ассоциируется с первым байтом данного. Выражение – это конкатенация (сцепление) имен, называемых термами. Выражение-константа представляет собой выражение, которое можно вычислить с получением числа. В качестве термов применяются идентификаторы переменных, а также следующие конструкции: константы, константы-цепочки, арифметические и логические операторы, подвыражения, имена.

Константа – число, система счисления которого указывается такими суффиксами, как B - двоичное, D – десятичное, O – восьмеричное, H – шестнадцатеричное. По умолчанию принимается десятичная система. Первая цифра в шестнадцатеричном числе должна быть 0...9; если старшая цифра оказывается буквой (A...F), то перед ней вводится нуль.

Константа-цепочка - это символьная цепочка, заключенная в апострофы.
Арифметические операторы следующие: +, -, *, /.

Логические операторы следующие: AND, OR, NOT, XOR (исключающие ИЛИ). Логические операции выполняются посредством представления операндов в двоичной форме и обработки соответствующих пар разрядов.

Подвыражение - выражение, являющееся частью другого выражения и ограниченное от порождающего выражения круглыми скобками.

Имя – идентификатор, представляющий собой константу, константу-цепочку или выражение.

Старшинство операторов такое же, как в языках высокого уровня. Идентификатор, будь он меткой, переменной или именем, представляет собой цепочку длиной до 31 символа (она может быть и более длинной, но Ассемблер распознает только первые 31 символ). Символами могут быть буквы, цифры, вопросительные знаки, значки амперсанда и подчеркивания, но первым символом не должна быть цифра. Идентификаторами не могут быть ключевые слова языка Ассемблер, например мнемоники команд, имена регистров и специальные операторы. Несколько форматов команд ADD представлено ниже:

```
ADD   AX,   BX              ; (BX) прибавляется к AX
ADD   BX,   621             ; прибавить 621 к BX
ADD   [BX][DI],   DX        ; прибавить (DX) к слову памяти,
                            ; смещение которой равно (BX) + (DI)
ADD   AL, NUM[BP+11]        ; прибавить к AL содержимое байта,
                            ; смещение которого равно сумме смещения NUM, (BP) и 11
```

Директивы языка. Команды языка Ассемблер транслируются в машинные и в программах на языках высокого уровня соответствуют исполняемым операторам. Но в этих программах должны быть и неисполняемые операторы, предназначенные для инициализации значений, резервирования памяти, назначения имен константам, формирования структур данных и окончания компилирования. Аналогичным образом в ассемблерных программах должны присутствовать директивы, выполняющие такие же функции. В работе микропроцессора 8086 большое значение имеют сегментные регистры, поэтому в Ассемблере имеются директивы, показывающие предполагаемое содержимое сегментных регистров при различных обстоятельствах в ходе ассемблирования. Так как программирование на языке Ассемблер довольно близко к фактическим действиям компьютера, то имеются директивы, предоставляющие программисту большую свободу в точном размещении данных и сегментации программы, чем при программировании на языке высокого уровня.

Напомним, что физический адрес формируется посредством сложения смещения и увеличенного в 16 раз сегментного адреса, содержащегося в сегментном регистре. Одна из задач Ассемблера при трансляции команд в машинный код – назначение смещений меткам и переменным. Ассемблер должен также передавать Редактору связей (через объектные модули) всю информацию, необходимую для объединения различных сегментов и модулей в законченную программу.

Чтобы назначить смещения переменным и меткам, Ассемблер должен знать точную структуру каждого сегмента. Сегменты данных, дополнительных данных и стека обычно имеют следующую структуру:

Имя сегмента SEGMENT

«Директивы определения, памяти, распределения и выравнивания»

Имя сегмента ENDS

Структура сегмента кода следующая:

Имя сегмента SEGMENT

«Команды и относящиеся к ним директивы»

Имя сегмента ENDS

Именем сегмента может быть любой допустимый идентификатор. Директивы и команды, находящиеся между директивами SEGMENT и ENDS, считаются содержащимися в сегменте. Полная структура программы, имеющей два сегмента данных и один сегмент кода, такова:

DATA-SEGl SEGMENT

«Директивы»

```
DATA-SEGl ENDS
DATA-SEG2 SEGMENT
        «Директивы»
DATA-SEG2 ENDS
CODE-SEG2   SEGMENT
START: : «Команды и директивы»
CODE-SEG   ENDS
END START
```

Кроме конструкции сегментов Ассемблер в ходе трансляции команд должен знать точное соответствие между сегментами и сегментными регистрами. Знание соответствия позволяет проконтролировать определенные виды ошибок и несовместимостей, например определена ли переменная в нужном сегменте данных или сегменте стека. Назначение сегментов сегментным регистрам осуществляется директивами

```
ASSUME Назначение,..., назначение
```

Здесь каждое назначение имеет следующий формат:

Имя сегментного регистра: Имя сегмента

Оператор

```
ASSUME CS:CODE-SEG, DS:DATA-SEG1, ES:DATA-SEG2
```

информирует Ассемблер о необходимости считать, что сегментный адрес CODE-SEG находится в CS, DATA-SEG1 в DS и DATA-SEG2 в ES. Директива ASSUME не загружает сегментные адреса в соответствующие сегментные регистры. Для всех сегментных регистров, кроме CS, который обычно загружается командой межсегментного перехода при инициировании сегмента кода, загрузка должна производиться явными передачами, обычно командами MOV. Директива ASSUME дает как бы «обещание» Ассемблеру и должна сопровождаться командами MOV, выполняющими это «обещание».

В соответствии со структурой программы сегмент кода обычно начинается следующим образом:

```
CODE-SEG SEGMENT
            ASSUME    CS:CODE-SEG, DS:DATA-SEG1, ES:DATA-SEG2
START:   MOV          AX, DATA-SEG1
            MOV       DS, AX
            MOV       AX, DATA-SEG2
            MOV       ES, AX
```

Поскольку команда MOV не может передавать непосредственный операнд в сегментный регистр, для каждого регистра требуются две команды. Имена DATA-SEG1 и DATA-SEG2 - имена сегментов, а не переменные; поэтому Ассемблер транслирует первую и третью команды MOV в команды, у которых операнды-источники являются непосредственными значениями. Операнды-источники должны быть сегментными адресами, но так как размещение сегментов в памяти осуществляет Редактор связей, секции данных этих команд фактически формирует Редактор связей.

Один и тот же сегмент допускается назначать двум сегментным регистрам, например директива

```
ASSUME CS:CODE-SEG, DS:CODE-SEG
```

назначает один и тот же сегмент регистрам CS и DS. Операторы ASSUME можно использовать также для переназначения сегментных регистров. Если в приведенном выше примере в программе появляются операторы

```
ASSUME DS:DATA-SEG3
MOV     AX, DATA-SEG3
MOV     DS, AX
```

то Ассемблер при трансляции последующих операторов будет использовать DATA-SEG3 вместо DATA-SEG1, но назначения для остальных сегментных регистров останутся неизменными.

Когда Ассемблер встречает в операнде команды переменную, он обычно предполагает сегментный регистр по умолчанию в соответствии с режимом адресации (см. табл. 8.1). Однако, если регистром по умолчанию принимается DS, а переменная не находится в сегменте, назначенном этому регистру, Ассемблер просматривает сегменты, назначенные регистрам CS, ES

и SS. Если переменная найдена, Ассемблер вводит перед командой, содержащей переменную, префикс замены сегмента на CS, ES и SS. При выполнении команды однобайтовый префикс заставит ее использовать вместо регистра DS регистры CS, ES или SS.

Сегментные регистры по умолчанию разрешается заменять явно, помещая перед переменной их имена с последующим двоеточием. Команда

```
ADD     AX, ES:ALT
```

заставит Ассемблер ввести перед машинным кодом команды префикс замены сегмента 26H. В этом случае Ассемблеру не нужно вначале отыскивать сегмент, назначенный регистру DS. Если имя сегментного регистра явно не появляется во всех операндах, которые его используют, оно должно фигурировать в операторе ASSUME, в противном случае возникает ошибка. Операторы инициализации данных и резервирования памяти имеют следующий формат:

```
ПЕРЕМЕННАЯ МНЕМОНИКА ОПЕРАНД, ...,
ОПЕРАНД; КОММЕНТАРИЙ
```

Здесь переменная не обязательна, но при ее наличии ей назначается смещение первого байта, резервируемого директивой. В отличие от метки переменная должна заканчиваться пробелом, а не двоеточием. Мнемоника определяет длину каждого операнда: DB (определить байт) – каждый операнд-данное занимает один байт; DW (определить слово) – каждый операнд-данное занимает одно слово, причем его младшая половина находится в первом байте, а старшая – во втором байте; DD (определить двойное слово) – каждый операнд-данное имеет длину в два слова, первым из которых следует младшее слово.

Операнды показывают инициализируемые данные или объем резервируемого пространства. Комментарии поясняют инициализацию данных или распределение, но могут отсутствовать. После переменной и мнемоники должны быть одни или несколько пробелов; кроме этого условия размещение пробелов произвольно (конечно, их нельзя вводить в мнемоники, идентификаторы и т. д.).

Директивы DB, DW и DD применяются для размещения данных в конкретных ячейках или просто для резервирования пространства без инициализации. Кроме того, директивы DW и DD используются для формирования смещений или полных адресов меток и переменных.

Для инициализации данных операнд должен быть константой, выражением, вычисление которого дает константу, или цепочкой. Например, директивы

```
DATA-BYTE DB     10, 4, 10H
DATA-WORD DW     100, 100H, 4373
DATA-DW     DD     3*20, OEEECH
```

вызывают инициализацию данных, представленную на рис. 8.8.

Символьную цепочку можно инициализировать, используя в качестве операнда цепочку-константу. Операторы

```
MEM   DB   ‘A‘, ‘B‘, ‘C‘
MEM   DB   ‘ABC‘
```

эквивалентны, т. е. вызывают одинаковую инициализацию памяти. Первый символ цепочки помещается в первый байт, второй - во второй и т. д. Директивы DW и DD также можно

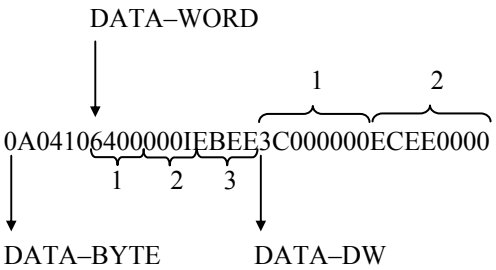


Рис. 8.8. Типичная инициализация данных

использовать для инициализации символьных цепочек, но для этого их применяют довольно редко, так как байты меняются местами. Например, директива DB ‘AB‘ помещает ‘A‘ в первый байт и ‘B‘ во второй, а директива DW ‘AB‘ помещает ‘B‘ в первый байт и ‘A‘ во второй.

Когда операндом служит вопросительный знак, инициализация не производится, но резервируется соответствующее пространство (1 байт в директиве DB, 2 байт в директиве DW и 4 байт в операторе DD). Операторы

```
ABC   DB    0, ?, ?, ?, 1
DEF   DW    ?, 52, ?
```

Генерируют последовательность байтов, показанную на рис. 8.9. Здесь XX - байт, резервируемый без инициализации.

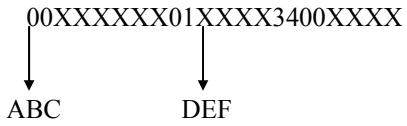


Рис. 8.9. Применение вопросительного знака в качестве операнда

Инициализация путем перечисления содержимого отдельных ячеек оказывается удовлетворительной при заполнении небольшого числа ячеек, но при наличии нескольких операндов такой подход будет громоздким. Поэтому Ассемблер допускает применение оператора дублирования DUP, аналогично коэффициентам дублирования в операторах инициализации языков высокого уровня. Несколько операндов или наборов операндов разрешается заменять следующей конструкцией:

```
ВЫРАЖЕНИЕ DUP (ОПЕРАНД, ..., ОПЕРАНД)
```

Здесь выражение вычисляет положительное целое число, которое определяет число повторений набора операндов. Операторы

```
ARRAY1   DB    2DUP (0, 1, 2, ?)
ARRAY2   DB    100DUP (?)
```

вызывают инициализацию и распределение, показанное на рис. 8.10. Первый оператор резервирует 8 байт, а второй -100 байт.

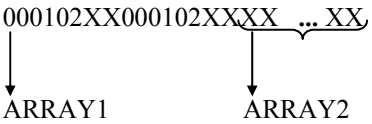


Рис. 8.10. Применение оператора дублирования

Допускается вложение операторов DUP, например оператор

```
ARRAY3   DB    2DUP (0, 2 DUP(1,2), 0,3)
```

вызывает инициализацию данных:

```
00 01 02 01 02 00 03 00 010201020003
```

Возможно также инициализировать смещение или полный адрес переменной или метки, указывая в поле операнда операторов DW или DD выражение, которое вычисляет смещение или адрес, т. е. имеет вид

```
ПЕРЕМЕННАЯ  ±  ВЫРАЖЕНИЕ-КОНСТАНТА
или
МЕТКА  ±  ВЫРАЖЕНИЕ-КОНСТАНТА
```

Если операнд находится в операторе DW, то запоминается только смещение, а если в операторе DD, запоминаются смещение и сегментный адрес, причем смещение располагается в первом слове. Оператор DB нельзя применять для запоминания адреса, так как он резервирует всего 1 байт.

Переменная, ассоциируемая с оператором определения памяти, представляет собой смещение первого байта первого элемента данных в текущем сегменте и имеет атрибут типа показывающего длину каждого элемента данных в операторе. Длина равна 1 байт в операторе DB, 2 байт в операторе DW и 4 байт в операторе DD. Выражение

```
ПЕРЕМЕННАЯ  ±  ВЫРАЖЕНИЕ-КОНСТАНТА
```

придает тот же атрибут типа, что и у переменной. Ассемблер использует атрибут типа для

определения того, оперирует ли машинная команда словом или байтом, т. е. следует ли устанавливать или сбрасывать бит W в команде. Например, в командах

```
        MOV   OPER1, O
        MOV   OPER2, O
        :
OPER1   DB    ?
OPER2   DW    ?
```

бит W=0 в первой команде и W=l во второй команде.

Так как все переменные типизируются в соответствии с оператором, в котором они появляются, Ассемблер может обнаружить ошибки кодирования, контролируя, чтобы оба операнда в двухоперандной команде имели один и тот же атрибут типа. Например, команды MOV

```
MOV   OPER1, AX
MOV   OPER2, AL
:
OPER1   DB    ?
OPER2   DW    ?
```

Ассемблер не транслирует, так как они содержат операнды разных типов.

Допускается замена неявного типа, придаваемого операнду, для этого предназначен атрибутный оператор PTR со следующим форматом:

```
ТИП PTR ПЕРЕМЕННАЯ  ±  ВЫРАЖЕНИЕ-КОНСТАНТА
```

Здесь типом может быть BYTE, WORD, DWORD (двойное слово). В приведенном примере команды

```
MOV   WORD PTR OPER1, AX
MOV   BYTE PTR OPER2, AL
```

будут оперировать словом и байтом (замена первого байта слова OPER2) и не вызовут сообщения об ошибке.

Поскольку неудобно пользоваться оператором PTR для частого обращения к переменным, имеющим как бы два различных типа, предусмотрен способ ассоциирования ячейки с двумя различными переменными. Директива LABEL с форматом

```
ПЕРЕМЕННАЯ LABEL ТИП
```

вызывает типизирование переменной и назначения ей текущего смещения. Директивы

```
BYTE-ARRAY LABEL   BYTE
WORD-ARRAY DW      50DUP(?)
```

ассоциируют BYTE-ARRAY и WORD-RRAY с одной и той же ячейкой – первым байтом 100-байтовой области. Команда

```
MOV   WORD-ARRAY, O
```

загружает 0 в первый и второй байты области, а команда

```
MOV   BYTE-ARRAY, O
```

загружает 0 только в первый байт.

Если выражение появляется в программе несколько раз, удобно присвоить ему имя, которым затем и пользоваться при обращении. Замена коротким именем длинного выражения позволит воспользоваться содержательным именем, которое проще запомнить. Оператор, который назначает имя выражению, имеет формат

```
ИМЯ ВЫРАЖЕНИЯ EQU ВЫРАЖЕНИЕ
```

Здесь именем выражения может быть любой допустимый идентификатор, а выражение может иметь формат любого допустимого операнда, быть любым выражением, вычисление которого дает константу (тогда имя выражения будет именем константы), или быть любой допустимой мнемоникой. Команда MOV в последовательности

```
IND-CHAR   EQU   CHAR-ARRAY[SI + 10]
           :
           MOV   AL, IND-CHAR
```

эквивалентна команде

```
MOV   AL,   CHAR-ARRAY[SI + 10]
```

Если выражение в операторе EQU содержит идентификатор (переменную, метку, имя константы и др.), то либо он должен представлять собой все выражение, либо должен определяться в программе до оператора EQU. Оператор

```
AB      EQU   DATA-ONE
```

допустим независимо от того, где в программе определяется DATA-ONE, а оператор

```
AB      EQU   DATA-ONE+2
```

вызовет ошибку, если DATA-ONE ранее не определено. Поэтому рекомендуется помещать все определения данных до операторов EQU. Однако оператор EQU следует указать до того, как применить его назначения, поэтому целесообразный порядок заключается в том, чтобы первыми разместить операторы определения данных, затем операторы EQU, а после них команды и связанные с ними директивы.

Кроме того, имена используются вместо выражений для более простой модификации программы. Приведем фрагмент программы управления адаптером асинхронной связи 8250 персональной ЭВМ IBM PC:

```
SERIAL     EQU       03F8H              ; Базовый адрес порта ввода-вывода адаптера
;          Вывод символа в канал
           MOV       DX,SERIAL + 5      ; Загрузить в DX адрес регистра состояния
SEND:      IN        AL, DX             ; Читать регистр состояния в AL
           TEST      AL, 20H            ; Проверить бит состояния буфера передачи
           JZ        SEND               ; Если бит равен 0, то передавать нельзя
           MOV       AL, 'A'            ; Записать символ A в AL
           MOV       DX, SERIAL         ; Загрузить в DX базовый адрес адаптера
           OUT       DX, AL             ; Вывести символ в канал связи
```

В случае другого подключения адаптера достаточно изменить один оператор в программе, например

```
SERIAL    EQU     02F8H
```

В противном случае (если адреса портов ввода-вывода явно указаны в операндах) потребуется изменять группу машинных команд.

В языке Ассемблер предусмотрены две директивы, предназначенные для выравнивания. Директива EVEN превращает адрес следующего байта в четное число. Микропроцессор 8086 обращается к словам быстрее, если они начинаются с четных адресов. Следовательно, перед резервированием массива слов целесообразно поместить директиву EVEN. Однако в микропроцессоре 8088, в котором обращение к операнду - слову требует два цикла шины независимо от адреса операнда, выравнивание по четным адресам не дает преимуществ.

Директива

```
ORG   ВЫРАЖЕНИЕ-КОНСТАНТА
```

заставляет следующий байт быть *h-байтом* в сегменте, где A – значение выражения-константы.

В языках Ассемблера обычно предусматривается несколько операторов, которые не нужны для правильного ассемблирования программы, но которые облегчают действия программиста, обеспечивая гибкость, улучшая читабельность и заставляя сам Ассемблер подставлять некоторые числа. В этот класс попадают следующие атрибутные операторы, возвращающие значение, относящееся к находящимся в операндах выражениям LENGTH, SIZE, OFFSET, SEG.

Оператор LENGTH возвращает число единиц, назначенных переменной, например

```
FEES     DW    100 DUP(0)
MOV CX, LENGTH FEES
MOV CX, 100
```

Две последние команды эквивалентны, но целесообразнее применять оператор LENGTH, так как если директиву DW изменить на резервирование 150 слов, то первую команду модифицировать не надо, а вторую - надо.

Оператор SIZE аналогичен оператору LENGTH, но вместо числа единиц он возвращает число

байтов. Предполагая наличие предыдущей директивы DW, команда

```
MOV   CX, SIZE FEES
```

вызовет загрузку в регистр CX числа 200.

Оператор OFFSET возвращает смещение метки или переменной. Команда

```
MOV   BX, OFFSET OPER-ONE
```

заставляет смещение OPER-ONE ассемблироваться как непосредственный операнд, а во время выполнения смещение OPER-ONE загружается в регистр BX.

Оператор SEG вызывает введение как непосредственного операнда сегментного адреса переменной или метки (хотя фактическое введение осуществляет редактор связей). Если DATA-SEG ассоциирован с блоком памяти, начинающимся по адресу 05000, и OPER1 находится в DATA_SEG, то команда

```
MOV   BX, SEG OPER1
```

загружает в BX значение 05000.

§ 8.5. ОПЕРАЦИОННЫЕ СИСТЕМЫ ПЕРСОНАЛЬНЫХ ЭВМ

Операционные системы образуют самый нижний уровень информационной оболочки аппаратных средств персональных компьютеров. Операционная система выполняет следующие главные функции: поддерживает работу всех прикладных и системных программ, обеспечивая их взаимодействие с аппаратурой; позволяет пользователям осуществлять общее управление машиной. Первая задача ОС состоит в том, чтобы обеспечить взаимодействие программ с внешними устройствами и друг с другом, распределение оперативной памяти, выявление различных событий, возникающих в процессе работы, соответствующее реагирование на них и др. Для общего управления машиной используется командный язык, с помощью которого человек может выполнять такие операции, как разметка дисков, копирование файлов, запуск программ, установка режимов работы внешних устройств.

В разных моделях персональных электронных вычислительных машин (ПЭВМ) применяются операционные системы с разной архитектурой и разными возможностями; для их хранения и работы необходимы различные ресурсы оперативной памяти; не одинаков и предоставляемый пользователям сервис для разработки программ и их использования.

Типы операционных систем. Число типов операционных систем для персонального компьютера в настоящее время невелико - не более нескольких десятков, но их роль чрезвычайно важна, поскольку именно они обеспечивают работу всех остальных программ на данной машине. На персональных компьютерах некоторых типов, предназначенных в основном для игр или учебных целей, ОС как типовые отсутствуют. Эти машины обычно используют лишь один язык программирования, например БЭЙСИК, ЛОГО или ФОРТ, а функции ОС в этом случае выполняются специальными операторами языка.

На более мощных ПЭВМ, где необходимо поддерживать работу нескольких языков программирования и прикладных программ, без отдельной ОС уже нельзя обойтись. В настоящее время широко распространены и фактически стандартизированы несколько «семейств» ОС, ориентированных на определенные классы машин: CP/M, MSX, UNIX, MS – DOS. Остальное программное обеспечение можно разделить на большие группы, связанные с перечисленными семействами.

О п е р а ц и о н н а я с и с т е м а типа CP/M была разработана впервые для персональных компьютеров в 1974 г. на основе 8-разрядных микропроцессоров, а позднее стала применяться и на 16-разрядных машинах. Она положила начало новому классу операционных систем. В рамках CP/M и ее более поздних модификаций (MP/M, CP/M-86 и др.) создано большое число программ, включая трансляторы языков БЕЙСИК, ПАСКАЛЬ, СИ, ФОРТРАН, КОБОЛ, ЛИСП, АДА, а также текстовые и табличные процессоры, системы управления базами данных, графические пакеты и прочие проблемно-ориентированные программы. Популярность этой системы обусловлена ее предельной простотой, компактностью и возможностью быстрой настройкой на разные конфигурации персональных компьютеров.

О п е р а ц и о н н а я с и с т е м а типа MSX была предложена в 1984 - 1985 гг. для дешевых 8-разрядных ПЭВМ, ориентированных на применение в школах. Эта система соединяет свойства

CP/M и MS - DOS и в настоящее время находит все большее распространение, в том числе в нашей стране.

О п е р а ц и о н н а я с и с т е м а типа UNIX предназначена в основном для эффективной поддержки процесса разработки программного обеспечения. Здесь имеется развитая файловая система, обеспечивается программирование доступа по всем типам внешних устройств, пользователям предоставляется очень мощный командный язык. В состав системы входит множество сервисных программ (утилит), рассчитанных на выполнение разнообразных функций, потребность в которых систематически возникает при разработке программного обеспечения. В этих системах заложена возможность одновременной работы нескольких терминалов, хотя для практической ее реализации нужна более мощная аппаратная поддержка, чем в массовых персональных компьютерах. Системы этого типа требуют значительных ресурсов, не всегда доступных на дешевых компьютерах, в то время как их мощность часто является избыточной с точки зрения большинства пользователей, занятых не столько разработкой программ, сколько решением своих профессиональных задач. Поэтому применение операционных систем типа UNIX на современных 16-разрядных ПЭВМ пока ограничено.

О п е р а ц и о н н а я с и с т е м а типа MS – DOS появилась в 1981 г. одновременно с машинами типа IBM PC. Фирма «IBM» первоначально объявила о возможности использования на своих персональных компьютерах трех операционных систем: CP/M-86, MS – DOS и UCSDp-system, однако вскоре система MS – DOS стала доминирующей. К настоящему времени разработано несколько базовых версий этой ОС.

Принятие MS – DOS в качестве главной операционной системы для персональных компьютеров, имеющих широкое распространение, послужило стимулом для многих программистов к созданию многочисленных инструментальных систем и прикладных программ. В результате MS – DOS приобрела статус фактического стандарта ОС для 16-разрядных персональных компьютеров. Она обеспечивает возможность организации многоуровневых каталогов, подключение дополнительных драйверов внешних устройств, работу со всеми последовательными устройствами как с файлами, развитый командный язык, запуск фоновых задач одновременно с диалоговой работой пользователя и т. д.

Средства, предоставляемые MS – DOS, позволяют, с одной стороны, формировать удобную операционную обстановку для разработки программного обеспечения; с другой – создавать автоматизированные рабочие места с простыми средствами доступа пользователей к прикладным пакетам и программам.

Часто возникает вопрос о соотношении систем MS – DOS и PC – DOS. Система MS – DOS разработана фирмой «Microsoft» и является ее собственностью, а система PC – DOS – собственностью фирмы «IBM». Хотя PC – DOS создана на основе MS – DOS и по всем основным функциям эти две системы совершенно идентичны, формально их следует считать разными системами. Разница между ними проявляется в модуле, который называется *базовой системой ввода-вывода* (BIOS). Часть этого модуля, зависящая от конкретной архитектуры аппаратных средств, заносится в постоянное запоминающее устройство и, таким образом, связывает конкретную ОС с конкретным типом ПЭВМ. Система MS – DOS часто устанавливается на персональных компьютерах, совместимых с IBM PC; с этой целью фирма–производитель таких компьютеров должна приобрести лицензию у фирмы «Microsoft». Что касается операционной системы PC – DOS, то она имеется только на машинах фирмы IBM. В СССР разработаны системы типа MS - DOS для разных типов отечественных ПЭВМ (например, ДОС-16).

Операционные системы типа MS – DOS устанавливаются на многих ПЭВМ, используемых в нашей стране, в частности на ПЭВМ ЕС-1840, ЕС-1841, «Искра-1030». Кроме того, операционная система MSX, применяемая на школьных компьютерах, по набору команд во многом совместима с MS – DOS. Поэтому рассмотрим подробнее структуру этой операционной системы.

Структура MS – DOS. Основная функция DOS – обеспечить файловую систему и операционную среду для программ. Файловая система представляет собой набор программных средств записи и считывания данных с дискеты или жесткого диска. Если все прикладные программы для записи данных используют DOS, то они могут коллективно пользоваться этими данными. Кроме того, в каждом случае при разработке приложений не надо будет заново переписывать файловую систему.

На каждой стороне дискеты (дискеты могут быть односторонними и двусторонними),

обслуживаемой DOS, может храниться от 160 до 360 К байт информации, а на фиксированном диске – более 1 млн. байт. Очевидным образом возникает задача ведения архива. При таких объемах информации необходим способ полного упорядочения хранимых данных. Физическое распределение данных на поверхности дискеты – это забота системы.

Основной единицей хранения данных является файл. *Файл* – совокупность данных, интерпретируемых некоторым образом. Владелец или создатель файла присваивает ему имя. Это имя может быть использовано при любых ссылках на эти данные, для того чтобы обеспечить к ним доступ. Ссылка на данные не требует никаких указаний в программе на то, где они физически располагаются. Записи – это то, что составляет файл. Размер и содержание записей определяются программистом. Операционная система DOS не проверяет формат записей, а просто помещает их в файл. Содержимое байтов, составляющих запись, определяется программистом.

Рассмотрим программу на языке Ассемблер как пример файла. У программы есть имя, и это имя станет именем соответствующего файла. Файл состоит из записей, каждая из которых представляет собой один оператор языка Ассемблера. Формат любой записи ни о чем не говорит DOS, он понятен только Ассемблеру. Отдельные части одной записи соответствуют полям языка Ассемблер. Для ОС не важно, как записи разбиваются на поля. Это дело прикладной программы, в данном случае Ассемблера.

Каждое сформированное DOS имя файла состоит из двух частей. Первая часть имени файла имеет длину от 1 до 8 символов. Эта часть определяется пользователем и соответствует присвоенному им имени файла. Вторая (необязательная) часть имени, называемая расширением, имеет длину от 1 до 3 символов. Эта часть, определяющая тип файла, обычно задается прикладной программой. Имя и его расширение разделяются точкой. Например, в имени файла COMMAND.COM именем является COMMAND, а расширением – COM. С ассемблированием связаны один входной файл и от одного до трех выходных файлов. Расширением входного ассемблерного файла является ASM, а выходных файлов: объектного – OBJ, листинга – LST, файла перекрестных ссылок – CRF. Во многих случаях в прикладных программах требуется, чтобы у имен файлов были определенные расширения.

Так как на любой дискете может поместиться большое количество информации, то использовать всю дискету для записи только одного файла было бы неэкономно. Операционная система позволяет хранить на фиксированном диске или на дискете одновременно несколько файлов.

На магнитном носителе может храниться более одного файла благодаря тому, что система организует каталог содержимого диска. Этот справочник напоминает оглавление книги. В нем перечислены все файлы, имеющиеся на диске или на дискете. Кроме имени файла операционная система DOS помещает в каталог и другую необходимую и полезную информацию. Прежде всего в нем имеются указатели, необходимые для определения фактического местоположения данных на магнитном носителе. Каталог удобен тем, что имеет временные ярлыки для каждого файла. Когда какая-нибудь программа создает либо обновляет файл, ОС производит запись соответствующих даты и времени. Это удобно в тех случаях, когда существует множество копий некоторой информации и требуется самая последняя версия.

Организация каталога на диске решает задачу хранения нескольких файлов на одном диске или дискете. В каждый момент времени DOS может обращаться только к одному дисководу. Если в системе имеется больше одного дисковода с гибким или жестким диском, то нужно сообщить ОС, на каком из них расположен файл. В этом случае наименование дисковода указывается в качестве префикса имени файла. Например, у файла C0MMAND.COM в дисководе A уточненное полное имя будет ACOMMAND.COM.

Наряду с файловой системой операционная система DOS обеспечивает операционную среду для прикладных программ. Первый компонент DOS, с которым сталкивается пользователь, - командный процессор, который берет на себя обработку вводимых пользователем команд и запуск прикладных программ.

В первый момент после включения питания «интеллектуальность» ПЭВМ типа IBM PC невысока. В постоянной памяти машины хранятся программы тестирования компонентов системы POST (Power-On Self-Test – самопроверка при включении) и установка в начальное состояние устройств ввода-вывода. Остальная часть записанной в ПЗУ базовой системы ввода-вывода BIOS предоставляет программисту, работающему с языком Ассемблер, набор средств, помогающих ему

обращаться к аппаратным средствам, не беспокоясь о том, как технически они реализованы. Но этого недостаточно, чтобы обеспечить среду для выполнения серьезных прикладных программ. Эту функцию выполняет DOS. После установки ЭВМ в начальное состояние программа POST выполняет загрузку ОС с диска или дискеты в оперативную память. Этот процесс называется *загрузкой системы.* Операционная система DOS загружает минимальную программу, необходимую для загрузки остальной части DOS. По окончании процесса загрузки на дисплей выводятся название ОС и указание на авторские права. В названии также указан номер соответствующей версии операционной системы. Иногда этот номер очень важен, так как каждая новая версия означает дополнительные функциональные возможности ОС.

После операции самозагрузки система готова к приему команд от оператора, за исключением случая, о котором будет сказано ниже. На этом этапе управление передается Командному процессору. Система произвела загрузку в память Командного процессора, файловой системы и других служебных программ, и все они готовы выполнять свои функции. Передача управления командному процессору определяется им следующим запросом к оператору:

A >.

Этот запрос содержит двоякий смысл. Символ > означает готовность командного процессора к приему команды. Префикс A указывает на выбранный по умолчанию дисковод с гибким диском. Так как DOS может обрабатывать файлы, расположенные только на каком-то одном диске, то при одновременной работе с несколькими дисками или дискетами пользователь должен указать, к какому из дисководов следует обратиться системе. В DOS дисководы обозначаются буквами латинского алфавита. Дисковод - логическое понятие, он определяет информационный канал для потока данных между программами и дисковыми файлами. Например, в ЕС-1840 имеется два накопителя на гибких дисках (НГМД). Операционная система DOS ставит им в соответствие четыре дисковода (A, B, C, D). Дисковод A связан с верхней стороной дискеты первого НГМД, B – с верхней стороной дискеты второго НГМД, C – с нижней стороной дискеты первого НГМД и D – с нижней стороной дискеты второго НГМД. Обычно файлы, используемые DOS, считываются с дисковода, выбранного системой по умолчанию, если только оператор не изменит этот порядок. Для считывания файла с выбранного по умолчанию дисковода системе требуется только имя этого файла. Для считывания файла с любого другого дисковода нужно кроме имени файла указать DOS наименование дисковода.

Только в ответ на команды, поступающие от пользователя, DOS выполняет какие-то действия. Все команды, относящиеся к ОС, вводятся в ответ на запрос с ее стороны, индуцируемый символом >. Пользователь вводит имя нужной ему команды, после чего поступившая заявка обрабатывается Командным процессором. Командный процессор обрабатывает эту заявку в зависимости от команды, которую ввел пользователь. Имеются встроенные, или резидентные, команды, которые всегда доступны, и команды нерезидентные или транзитные. Для выполнения последних должен существовать определенный файл на диске.

Встроенные команды обеспечивают поддержку файловой системы. Они сделаны резидентными в DOS, поскольку часто используются при работе с данными, хранящимися на дисках. После того как пользователь ввел команду, интерпретатор командных строк передает управление соответствующей программе ОС. Программа реализует свою функцию и возвращает управление DOS. В табл. 8.3 приведены резидентные команды дисковой операционной системы.

Таблица 8.3

Команды	Действия
COPY	Копирование файла с одного места на другое
DATE	Вывод-корректировка текущей даты
DIR	Вывод каталога дискеты
ERASE	Удаление файла с дискеты
PAUSE	Перевод системы в режим ожидания до нажатия клавиши
REM	Ввод комментария
RENAME	Переименование файла на дискете
TIME	Вывод-корректировка текущего времени
TYPE	Вывод содержимого файла

Если пользователь введет нерезидентную команду, то Командный процессор попытается

загрузить ее с диска или дискеты. В этом случае он выступает в роли загрузчика программы. Предполагая, что имя файла совпадает с именем команды, интерпретатор просматривает каталог указанного дисковода в поисках этого файла, а когда находит - загружает его в память. Затем Интерпретатор передает управление загруженной программе, которая теперь может реализовать свои функции.

Не каждый файл может быть загружен с помощью командного процессора. Тип файла должен быть либо COM, либо EXE, что соответствует файлу типа «команда» или выполняемому файлу, являющемуся продуктом выполнения операций транслирования и редактирования связей. Отсюда возникает возможность написания собственной системной команды. Если написали, оттранслировали и скомпоновали программу на языке Ассемблера, а затем оставили эту программу на дискете, то ее можно загружать и выполнять так же, как любую другую программу DOS. Именно это позволяет писать программы, которые будут выполняться под управлением DOS.

Существует различие между файлами типа COM и EXE. Они имеют различные структуры, и управление им передается по-разному. Хотя обычно после этапа редактирования связей получаются файлы типа EXE, существуют некоторые причины для использования и файлов типа COM. Главное различие между ними связано с форматом записи соответствующего объектного файла. Оба типа файлов являются программами, записанными на машинном языке. Программа, записанная в файле типа COM, может сразу выполняться. Операционная система может непосредственно загрузить его в память машины. После этого DOS передает управление в сегмент памяти, отведенный для команд, в точку со смещением 100H. Файл типа EXE непосредственно выполнен быть не может. У соответствующего объектного файла имеется заголовок. В нем содержится информация, сгенерированная Редактором связей. Наиболее важная ее часть относится к информации, связанной с перемещением. В то время как у файла типа COM перемещаем один сегмент команд, у файла типа EXE могут быть перемещены многие различные сегменты. Это ограничивает максимальный размер файла типа COM до 64 К байт. Файл типа EXE может содержать ряд сегментов, динамически перемещающихся в пределах программной области.

Рассмотрим пример вызова программы. Хорошей иллюстрацией здесь может служить язык Ассемблер. Чтобы вызвать Ассемблер, нужно ввести следующую команду: A > ASM

Каталог дискеты содержит файл с именем ASM.EXE, это и есть Ассемблер. После ввода команды ASM командный процессор просматривает дискету в дисководе A (выбранном по умолчанию). Найдя файл с именем ASM.EXE, он загружает и передает управление Ассемблеру. Теперь вычислительная система находится под управлением Ассемблера. При благополучном завершении трансляции Ассемблер вернет управление Командному процессору. Файл, содержащий Ассемблер, - это файл типа EXE, поэтому он может быть загружен Командным процессором.

Если Ассемблер находится на дискете, соответствующий дисководу B, то пользователь может к нему обратиться следующим образом:

```
A > B:ASM
```

Для того чтобы оттранслировать файл, расположенный в дисководе B, с помощью Ассемблера, который находится на дискете в дисководе C, нужно ввести следующую команду:

```
A > C:ASM B:FILE.ASM
```

Можно поступить и так:

```
A > B
B > C:ASM FILE.ASM
```

Первая команда принуждает ОС сменить дисковод, выбираемый по умолчанию. После этого запрос со стороны системы меняется на B. Приведенная в этом примере команда по своему действию полностью идентична команде из предыдущего примера.

Интерпретатор командных строк может работать с файлом, называемым файлом с пакетом команд, с расширением имени BAT. Этот тип файла совершенно отличен от файлов типов COM и EXE. Файл типа BAT не содержит выполняемого машинного кода, а состоит из нетранслированных команд, интерпретируемых Командным процессором. Файл типа BAT заменяет процедуру ввода команд с клавиатуры, так как они содержатся в соответствующем

файле. После того как система закончила обработку пакетного файла, она обращается за следующей командой к клавиатуре. Эти особенности делают файл типа BAT удобным средством выполнения повторяющихся заданий. После того как такой файл уже создан, единственная команда обращения к нему заменяет ввод всех содержащихся в нем команд.

В системе допускается специальный файл AUTOEXEC.BAT. Если такой файл имеется в дисководе A, то сразу после своей загрузки DOS обращается к нему, передавая управление командам, составляющим пакет команд этого файла, что позволяет автоматически загружать с диска нужную пользователю программу.

Интерпретатор командных строк обеспечивает средства, необходимые для того, чтобы программа начала выполняться. Система уже во время выполнения программы обеспечивает доступ к файловой системе с помощью механизма функций DOS. Этот механизм доступен программам, написанным на языке Ассемблер, и реализован посредством программного прерывания.

Механизм прерываний. Этот механизм является важной составной частью микропроцессора 8086, обеспечивающей эффективное переключение программных процессов и реакцию вычислительной системы на разнообразные события. Прерывания бывают аппаратные и программные, которые идентифицируются номером, находящимся в диапазоне от 0 до 255. В случае аппаратного прерывания номер выдает контроллер прерываний (микросхема), в случае программного прерывания его генерирует машинная команда. Как только микропроцессор получает номер прерывания, он должен передать управление соответствующему обработчику прерываний. Первые 1024 байт памяти зарезервированы для векторов прерываний. Каждому из 256 возможных прерываний отводится 4-байтовая область. Прерывание «0» имеет 4 байт по адресам от 0 до 3, прерывание «1» – от адреса 4 до адреса 7 и т. д. Каждая 4-байтовая ячейка содержит указатель на соответствующий обработчик прерываний. Первые 2 байт содержат смещение адреса обработчика прерываний, последние 2 байт - сегмент. Для задания значения этого поля может использоваться директива Ассемблера DD (резервирование двойного слова). Алгоритм передачи управления обработчику аппаратного и программного прерываний одинаков и соответствует вызову программы типа FAR, за исключением того, что дополнительно в стеке сохраняется значение регистра флагов F микропроцессора. В оперативную память по адресу (SS)x10H+(SP) записывается регистр флагов F, (SP) = (SP)-2, и по новому адресу записывается значение сегментного регистра CS, опять (SP) = (SP)-2, и запоминается (IP). Таким образом, в текущий стек заносятся регистры F, CS, IP.

Далее, код прерывания умножается на четыре, полученное число интерпретируется как смещение от начала оперативной памяти, из которой загружаются новые значения регистров IP и CS. Тем самым микропроцессор начнет выполнять команды требуемого обработчика прерываний. Обработчик прерываний должен обеспечить сохранение значений регистров микропроцессора на момент прерывания и их восстановление на момент возврата управления прерванному процессу.

Программа использует функции DOS посредством программного прерывания. Благодаря этому программа может вызывать соответствующую служебную подпрограмму, не зная ее адреса. Код нужного прерывания задается программистом в машинной команде. Во время инициализации DOS векторы прерываний для функций системы определяются таким образом, чтобы они указывали на соответствующие подпрограммы. Следовательно, по мере получения других версий DOC нет необходимости вносить изменения в программы. В табл. 8.4 приводятся коды прерываний, относящиеся к DOS.

Таблица 8.4

Прерывания	Действия
20H	Завершение программы
21H	Запрос к файловой системе и операции DOS
22H	Завершение задачи
23H	Реакция на нажатие клавиш УПР-СТОП
24H	Обработка неустранимой ошибки системы
25H	Абсолютное чтение с диска
26H	Абсолютная запись на диск
27H	Программа завершена, но остается резидентной

Компоненты DOS. Некоторые прерывания фактически предназначены для пользовательских

программ. Прерывания 22H, 23H и 24H являются указателями на подпрограммы, которые могут быть в программе пользователя.

Прерывания с кодами 25H и 26H связывают между собой две части системы. С точки зрения своей файловой структуры DOS фактически состоит из четырех компонентов.

Блок начальной загрузки расположен на дорожке О, секторе 1, стороне 0 любого гибкого диска, размеченного для работы в DOS. На жестком диске блок начальной загрузки расположен на первом секторе (сектор 1, головка 0) первого цилиндра раздела DOS.

Модуль взаимодействия с базовой системой ввода-вывода BIOS расположен в файле IBMBIO.COM и обеспечивает обращение к подпрограммам BIOS.

Собственно DOS расположен в файле IBMDOS.COM и обеспечивает взаимодействие с прикладными программами. Состоит из программ файловой системы и программ блочного обмена с дисками и других встроенных операций, доступных программам пользователя.

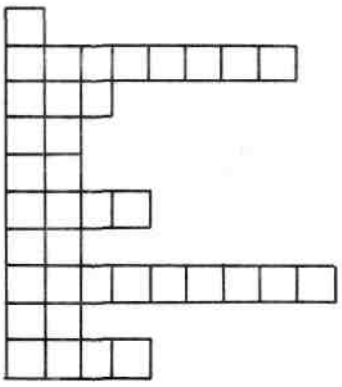
Командный процессор расположен в файле C0MMAND.COM.

Упомянутые два прерывания используются в том случае, когда компонент DOS, относящийся к файловой системе, обращается к другому компоненту –BIO. Хотя эти прерывания и могут использоваться программистом, их основное назначение – разделить две части DOS. Прерывание 25H осуществляет чтение, а прерывание 26H – запись информации с абсолютной адресацией диска. На этом уровне обеспечивается доступ к определенным участкам диска, а не записям в файле.

Прерывания 20H и 27H обеспечивают механизм возврата управления к ОС после выполнения программы. Прерывание 20H соответствует нормальному завершению работы программы. Прерывание 27H интересно тем, что, хотя оно связано с завершением программы, занимаемая программой область памяти не возвращается обратно в распоряжение системы. Все содержимое данной области сохраняется неизменным до тех пор, пока не будет выключено питание или система не будет установлена в начальное состояние. Это удобно в тех случаях, когда нужно ввести специальную обработку прерываний или аналогичную функцию, которая должна сохраняться как часть системного программного обеспечения.

Прерывание 21H является прерыванием, с помощью которого производится обращение к основным функциям DOS. Это прерывание обеспечивает доступ к системе ввода-вывода. Выбор функции в программе осуществляется с помощью записи в регистр AH нужного значения перед выполнением прерывания 21H, Например, в DOS 2.10 насчитывается 60 различных функций, доступных программисту.

Прежде чем передать управление прикладной программе, DOS формирует блок управления файлом FCB (File Control Block), который является существенным элементом файловой системы и участвует во всех файловых операциях.



Номер дисковода
Имя файла
Тип файла
Текущий блок
Длина записи
Длина файла
Дата
Резерв
Относительный номер записи в блоке
Номер записи при произвольном доступе

Рис. 8.11. Блок управления файлом

Блок управления файлом обеспечивает связь пользовательской программы с функциями DOS. При любой файловой операции происходит обращение к блоку FCB. На рис. 8.11 показан состав стандартного блока FCB. Имеется модификация блока, называемая расширенным блоком FCB, которая применяется в специальных случаях, когда нужно «скрыть» файл. Скрытый файл защищен от записи; это значит, что программа не может модифицировать содержимое этого файла, не изменив предварительно его блока FCB. Скрытый файл отсутствует в листинге каталога. Скрыть файл – это один из простейших способов защиты файла от неумелого пользователя.

Например, IBMBIO.COM и IBMDOS.COM – это скрытые файлы.

Поля данных блока FCB охватывают все атрибуты файла. Номер дисковода, имя и тип файла составляют идентификатор файла. Размер файла и дата являются атрибутами файла, которые находятся в каталоге диска. Оставшиеся поля – текущий номер блока, длина записи и номер записи при произвольном доступе – служат для определения местоположения внутри файла при операциях чтения и записи. Длина записи указывает на число байтов в назначаемой пользователем записи. Так как все операции чтения и записи в файл начинаются с границы записи, то длина записи в файл начинаются с границы записи, то длина записи определяет количество данных, обрабатываемых во время каждой из этих операций.

Существует два способа вычисления текущей записи при обращении к файлу. При первом, последовательном, способе записи обрабатываются по порядку. При этом текущий номер блока и относительный номер записи определяют запись, которая будет обрабатываться следующей. По мере того как программа выполняет операции чтения или записи, DOS увеличивает на единицу относительный номер записи, чтобы он указывал на следующую запись. Выполнение последовательных операций идет в направлении от начала до конца этого файла.

Программа может обращаться к файлу, используя также и произвольный доступ. Произвольный доступ в данном случае не означает, что запись выбирается случайным образом. Наоборот, это значит, что программа может выбрать в качестве следующей любую запись в файле. Иначе такой файл называют файлом с прямым доступом, так как у программы имеется непосредственный доступ к любой записи.

Программа может обращаться к файлу любым из двух указанных способов. При последовательных операциях DOS автоматически изменяет в блоке FCB значение поля относительного номера записи. При произвольном доступе к файлу номер записи должен определяться программой. В файловой системе файл с произвольным доступом является аналогом вектора. Точно так же как для обращения к любому элементу вектора программа должна задать значение соответствующего индекса, так и при обращении к записи в файле с произвольным доступом номер записи должен определяться программой.

ЗАКЛЮЧЕНИЕ

В настоящее время вычислительная техника становится все более мощной, разнообразной, дешевой и компактной. Это позволяет применять ее для автоматизации трудоемких процессов сбора и переработки информации независимо от масштабов и характера производственного процесса.

Важным фактором, определившим прогресс в области эксплуатации вычислительной техники, явилась организация коллективного использования нескольких сопряженных при помощи аппаратуры передачи данных ЭВМ, что позволило существенно увеличить количество пользователей и число предоставляемых им услуг, а также более рационально расходовать вычислительные мощности. Методы теледоступа позволили по-новому подойти к решению проблемы взаимодействия человека и ЭВМ при принятии управленческих решений в условиях функционирования систем автоматизированной переработки данных. В свое время создание ЭВМ Единой Системы Ряда 1 и Ряда 2 позволило в значительной степени решить задачу оснащения отраслей народного хозяйства СССР, а также стран – членов СЭВ средствами вычислительной техники. Сейчас работы по созданию более совершенных моделей ЕС ЭВМ интенсивно продолжаются в плане улучшения технических параметров, увеличения производительности, расширения сфер применения, создания новых программно-технических средств, уменьшения стоимостных характеристик, разработки развитых средств и методов диагностики и контроля и т.д.

СПИСОК ЛИТЕРАТУРЫ

1.*Глушков В.М.* Основы безбумажной информатики. - М.: Наука, 1982.

2.*Дел Рио Салседа Б.* АСУ реального времени на базе ЕС ЭВМ. - М.: Машиностроение, 1983.

3.*Данильченко И.А., Мясников В.А., Четвериков В.Н.* - Автоматизированные системы управления предприятиями. - М.: Машиностроение, 1984.

4.*Дьяконов В.Ю., Калинчев И.А., Китов В.А.* Прикладное программирование в системе ОБЬ. - В кн.: Прикладная информатика. Вып. 1 (10). - М.: Финансы и статистика, 1986.

5.*Дьяконов В.Ю., Калинчев И.А., Китов В.А.* Принцип мобильности программного обеспечения мультитерминальных систем распределенной обработки данных// Программирование. 1984. № 2.

6.*Дьяконов В.Ю., Калинчев И.А., Китов В.А.* Метод проверки и оперативного восстановления данных// Программирование. 1982. № 6.

7.*Донован Дж.* Системное программирование. - М.: Мир, 1976.

8.*Девис У.* Операционные системы. - М.: Мир, 1980.

9.*Кейлингерт П.* Элементы операционных систем. - М.: Мир, 1985.

10.*Кетков Ю.Л., Максименко В.С., Рябов АН.* Введение в систему программирования на языке Ассемблера ЕС ЭВМ. - М.: Наука, 1982.

11.*Лебедев В.Н., Соколов А.П.* Введение в систему программирования ОС ЕС. - М.: Финансы и статистика, 1985.

12.*Лихачева Г.Н., Медведев В.Д.* Операционные системы. - М.: Статистика, 1980.

13.Принципы работы системы IBM/370/Пер. с англ.; Под ред. *Л.Д. Райкова* - М.: Мир, 1978.

14.Пул Л. Работа на персональном компьютере. - М.: Мир, 1986.

15. Справочник системного программиста по операционной системе ОС ЕС/Под ред. *Л.Д. Райкова.* - М.: Финансы и статистика, 1982.

16.Технические средства АСУ: Справочник. В 2 т./Под ред. *Г.Б. Кезлинга.* - Л.: Машиностроение, 1986.

17.*Хаузер Д., Хирт Дж., Хоукинс Б.* Операционная система MS-DOS. - М.: Финансы и статистика, 1987.

18.*Хромов В.И., Ульянов С.А.* Введение в программирование для систем телеобработки данных. - М.: Финансы и статистика, 1982.

19.*Шураков В.В., Алферова З.В., Лихачева Г.Н.* Программное обеспечение ЭВМ.-М.: Статистика, 1979.

20.*Руденко Ю.М., Самедова М.А.* Организация решения задач в операционных системах ОС ЕС и ДОС СМ. - М.: МИРЭА 1984.

21.*Лю Ю-Чжен, Гибсон Г.* Микропроцессоры семейства 8086/8088. Архитектура, программирование микрокомпьютерных систем. - М.: Радио и связь, 1987.

22.*Брэдли Д.* Программирование на языке Ассемблера для персональной ЭВМ фирмы IBM. - М.: Радио и связь, 1988.

23.*Шнайдер А.* Язык Ассемблера для персонального компьютера фирмы IBM. М.: Мир, 1988.

24. *Ги К.* Введение в локальные вычислительные сети. - М.: Радио и связь, 1986.

Учебное издание

Дьяконов Владимир Юрьевич,
Китов Владимир Анатольевич,
Калинчев Игорь Алексеевич

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Заведующая редакцией *И.И. Хрусталева.* Редактор *Т.Д. Жарова.* Мл. редакторы *Г.Г. Бучина, Е.В. Судьенкова.* Художник *В.В. Гарбузов.* Художественный редактор *В.Е. Мешалкин.* Технический редактор *Л.А. Овчинникова.* Корректор *Г.И. Кострикова.* Оператор *Н.В. Мясникова*

ИБ № 8267

Изд.№ СТД-634. Сдано в набор 24.10.89. Подп. в печать 12.04.90. Т- 08907.
Формат 60x88$^{1}/_{16}$ Бум. офсет. №2. Гарнитура Таймс. Печать офсетная.
Объем 13,72 усл. печ. л. 13,97 усл.кр.-отт. 14,30 уч.-изд.л.
Тираж 30 000 экз. Зак. №493. Цена 55 коп.
Издательство «Высшая школа», 101430, Москва, ГСП-4, Неглинная ул., д. 29/14.

Набрано на персональном компьютере издательства.

Московская типография № 8 при Государственном комитете СССР по печати. 101898, Москва, Центр, Хохловский пер., 7.

В 1991 Г. ИЗДАТЕЛЬСТВОМ
БУДЕТ ВЫПУЩЕНА СЕРИЯ ПРАКТИЧЕСКИХ ПОСОБИЙ
"ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР ДЛЯ ВСЕХ"
ПОД РЕДАКЦИЕЙ Д-РА ТЕХН. НАУК ПРОФ. А.Я.САВЕЛЬЕВА.

Книга 1. Хранение и обработка информации. А.Я. Савельев, Б.А. Сазонов, С.Э. Лукьянов. 10 л.

Рассматриваются основные функции и состав оборудования персональных компьютеров, излагаются сведения об операционных системах, системах программирования и прикладных программах. Даются представления о типовом сеансе работы на персональном компьютере, практических приемах обслуживания архивов на магнитных дисках. Описываются возможности прикладного программного обеспечения.

Книга 2. Подготовка и редактирование документов. ЛЯ. Савельев, Б.А. Сазонов, С.Э. Лукьянов. 10 л.

В книге рассматриваются пакеты прикладных программ для создания текстовых и графических документов (текстовые и графические редакторы), их возможности по выбору таблиц, бланков, рисунков, схем и т.п. Дается концепция работы редакторов, получивших наибольшее распространение. Описываются команды, необходимые начинающему пользователю для компоновки и печати документов. Излагаются особенности подготовки и редактирования научных текстов.

К**нига 3. Создание и использование баз данных.** А.Я. Савельев, Б.А. Сазонов, С.Э. Лукьянов. 10 л.

В книге рассматриваются пакеты прикладных программ для создания, ведения и использования баз данных, их возможности по организации вычислений, формированию и печати выходных отчетов. Даются сведения о назначении и возможностях баз данных. Приводится сравнительный анализ наиболее распространенных СУБД, а также команды, необходимые начинающему пользователю для вывода, редактирования, выборки и просмотра данных.

Книга 4. Вычислительные и графические возможности. А.Я. Савельев, Б.А. Сазонов, С.Э. Лукьянов. 10 л.

Рассматриваются пакеты прикладных программ, обеспечивающие вычисления над данными, представленными в виде таблиц, и выдачу полученных результатов в удобном для пользователя виде. Даются сведения о назначении и возможностях электронных таблиц данных, описываются команды, необходимые начинающему пользователю для ввода, изменения данных, организации вычислений над ними. Излагаются особенности вывода результатов в табличной и графической формах.

УВАЖАЕМЫЕ ЧИТАТЕЛИ!

ПО ВОПРОСАМ ПРИОБРЕТЕНИЯ ЛИТЕРАТУРЫ ПРОСИМ ОБРАЩАТЬСЯ
В МЕСТНЫЕ ОТДЕЛЕНИЯ КНИГОТОРГА
ИЛИ КНИЖНЫЕ МАГАЗИНЫ ПО МЕСТУ ЖИТЕЛЬСТВА.

Издательство "Высшая школа"