М. И. КЛИОРИН В. Я. КАДУЛИН
В. М. СМОЛКИН

Управляющие Вычислительные Комплексы СМ-2М:

Архитектура и программное обеспечение

Под редакцией
В. В. РЕЗАНОВА

МОСКВА
ЭНЕРГОАТОМИЗДАТ
1989

ББК 32.97
К 49
УДК 681.5:581.31

Рецензент В. В. Бурляев

Клиорин М.И. и др.

К 49 Управляющие вычислительные комплексы СМ-2М: Архитектура и программное обеспечение/ М.И. Клиорин, В.Я. Кадулин, В.М. Смолкин; Под ред. В.В. Резанова. – М.: Энергоатомиздат, 1989. – 296 с.: ил.
ISBN 5-283-01483-5

Рассмотрены вопросы логической организации и программного обеспечения управляющих вычислительных комплексов на базе семейства ЭВМ М-6000, М-7000, СМ-1, СМ-2, СМ-2М. Основное внимание уделено описанию системы команд, базового языка программирования МНЕМОКОД и агрегатной системы программного обеспечения (АСПО).

Для инженерно-технических работников, программистов различной квалификации (в первую очередь начинающих).

Производственное издание

Клиорин Михаил Иосифович
Кадулин Валерий Яковлевич
Смолкин Валерий Маркович

УПРАВЛЯЮЩИЕ ВЫЧИСЛИТЕЛЬНЫЕ КОМПЛЕКСЫ СМ-2М: АРХИТЕКТУРА И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Редактор *Б. Г. Волков*, Редактор издательства *В. И. Петухова*,
Художественные редакторы *Т. А. Дворецкова*, *Т. Н. Хромова*,
Технический редактор *Е. В. Пронь*, Корректор *Л. А. Гладкова*

ИБ № 1721

Набор выполнен в издательстве. Подписано в печать с оригинала-макета 14.04.89. Т-10072. Формат 60 x 88 1/16. Бумага офсетная № 2. Печать офсетная. Усл. печ.л. 18,13. Усл.кр.-отт. 18.13. Уч.-изд.л. 20.98. Тираж 40000 экз. (1-й завод 1–20 000 экз.). Заказ 6702. Цена 1 р. 40 к.

Энергоатомиздат, 113114, Москва, М-114, Шлюзовая наб., 10.

Отпечатано в ордена Октябрьской Революции и ордена Трудового Красного Знамени МПО "Первая Образцовая типография" Союзполиграфпрома при Госкомиздате СССР. 113054, Москва, М-54, Валовая, 28.

К $\frac{2405000000\text{-}272}{051\,(01)\,\text{-}89}$ 266-88

ББК 32.97

ISBN 5-283-01483-5

ПРЕДИСЛОВИЕ ТИТУЛЬНОГО РЕДАКТОРА

Вычислительная техника все глубже и шире проникает во все сферы народного хозяйства, в деятельность человека.

Трудно представить себе в настоящее время такие отрасли, как геофизика, атомная энергетика, металлургия, химическая технология и другие, без автоматизированных систем управления сложными процессами, ядро которых составляет управляющая вычислительная техника.

В ряду вычислительных машин, успешно справляющихся с задачами управления, достойное место занимают вычислительные комплексы СМ-2М. Эти комплексы являются завершающей серией машин, ориентированных на архитектурную линию, в основу которой заложено применение интерфейса "Сопряжение 2К". В этих комплексах в полной мере нашли свое отражение концепции агрегатно-модульного подхода к построению как аппаратурных, так и программных составляющих современных вычислительных систем, которые постоянно усовершенствуются и развиваются по мере перехода к более сложным разработкам, как, например, СМ 1210, ПС 1001. Поэтому появление в литературе первых материалов с информацией об архитектуре и программном обеспечении комплексов СМ-2М должно заинтересовать большой круг читателей, которые и в дальнейшем будут ориентироваться на применение вычислительной техники, развивающей это архитектурное направление.

В предлагаемой читателю книге рассматриваются этапы программирования задач пользователя с учетом основных возможностей, обеспечиваемых архитектурными особенностями вычислительного комплекса СМ-2М. Отсутствие информации о внутрисистемной реализации тех или иных компонентов операционной среды ни в коей мере не ограничивает пользователя в принятии специфических решений при проектировании конкретных программных систем.

Книга построена как учебник и содержит достаточное число примеров и упражнений, подобранных таким образом, чтобы научить самостоятельной работе. Поэтому книга может служить хорошим введением в предмет, побуждающим к более углубленному самостоятельному изучению всего программного задела СМ-2М.

Книга рассчитана как на тех, кто уже работает с вычислительным комплексом СМ-2М, так и на тех, кто делает первые шаги в освоении новых моделей этой архитектурной линии.

В.В. Резанов

ПРЕДИСЛОВИЕ

Управляющие вычислительные комплексы СМ-2М пришли на смену первой массовой отечественной управляющей ЭВМ М-6000 и завоевали прочные позиции в таких отраслях, как атомная энергетика, геофизика, химическая технология, металлургия и др. Пользователи ЭВМ архитектурной линии М-6000/СМ-2М всегда испытывали недостаток в литературе учебно-методического характера. Восполнить пробел и призвана эта книга, написанная по материалам лекций, которые на протяжении ряда лет читались авторами в системе повышения квалификации.

Не следует думать, что материал книги способен заменить полную информацию, которая содержится в документации, сопровождающей УВК СМ-2М. Назначение книги состоит в том, чтобы помочь начинающим пользователям подготовить себя к активному восприятию этой документации, поскольку, как показывает опыт, сделать это без специальной подготовки удается далеко не всегда.

Некоторые вопросы, которые авторы считают принципиальными для освоения программного обеспечения, изложены в книге достаточно подробно [архитектура, базовый язык программирования МНЕМОКОД, структура агрегатной системы программного обеспечения (АСПО), генерация операционных систем пользователя], другие только очерчены, в них определены основные понятия и функциональные назначения соответствующих программных компонентов. Рассмотрены сравнительно новые, еще недостаточно известные разработки, такие, как Паскаль-система, система подготовки текстовой документации ОЛИМП, программные средства машинной графики и др.

Вопросы программного обеспечения многомашинных комплексов, субкомплексов и терминалов вычислительных, которые в последнее время получили значительное распространение, в книге не рассматриваются.

Первый и второй разделы книги (гл. 1–8 написаны М.И. Клиориным) посвящены логической компоновке УВК, его архитектуре и базовому языку.

Третий и четвертый разделы (гл. 9–14 написаны В.М. Смолкиным) содержат систематическое изложение структуры и методов использования АСПО, описание основных обрабатывающих программ и систем подготовки программ пользователя.

В пятом разделе (гл. 15–17 написаны В.Я. Кадулиным) рассмотрены наиболее популярные пакеты и библиотеки прикладных программ.

Авторы выражают признательность главному конструктору УВК СМ-2М, редактору книги В.В. Резанову, способствовавшему улучшению содержания книги, а также В.М. Костелянскому, А.Б. Айзенбергу, Л.В. Гомону, В.Я. Сидоренко, П.И. Санченко, В.С. Хаету, М.А. Блих и другим сотрудникам НИИ УВМ НПО "Импульс" за полезные замечания, учтенные при подготовке книги, а также В.С. Спириной и В.Г. Отвагиной, взявших на себя основной труд по оформлению рукописи.

Авторы заранее благодарны всем, кто пришлет свои замечания по книге по адресу: 113114, Москва М-114, Шлюзовая наб., 10, Энергоатомиздат.

Авторы

ВВЕДЕНИЕ

История развития управляющих вычислительных машин в нашей стране началась в первой половине шестидесятых годов. С тех пор сменилось три поколения ЭВМ, неизмеримо возросли уровень и качество их программного обеспечения (ПО).

Сегодня ПО является такой же неотъемлемой частью вычислительного комплекса (ВК), как и его аппаратные средства. Следствием этого явилось возникновение термина *внутреннее программное обеспечение.* Именно внутреннее ПО поддерживает различные режимы функционирования задач пользователя, распределяет ресурсы ВК между этими задачами, обеспечивает пользователя необходимым сервисом, средствами подготовки и отладки программ.

Пользователь современной компьютерной техники может строить свои достаточно сложные программные системы, рассматривая ЭВМ как "черный ящик", причем по мере развития и совершенствования ПО эта тенденция будет усиливаться.

В то же время для малых ЭВМ, на базе которых строятся управляющие вычислительные комплексы (УВК), характерна *открытость* внутреннего ПО, позволяющая модифицировать и расширять возможности УВК в соответствии с запросами постоянно усложняющихся задач управления.

Для того чтобы использовать указанное свойство на практике, отношение к ЭВМ как к "черному ящику" становится уже недостаточным. Возникает необходимость в понимании логической и функциональной организации ЭВМ, или, другими словами, *архитектуры* ЭВМ.

Рассматривая организацию ЭВМ на уровне архитектуры, обычно не интересуются элементной базой ЭВМ, конструктивными решениями, типами применяемых кабелей, электрическими схемами и другими атрибутами аппаратной реализации. На уровне архитектуры интерес представляют ресурсы машины, доступные пользователю: система команд, память, средства связи ЭВМ с оператором и объектом управления. Понятием архитектура ЭВМ охватываются также организация ввода-вывода, способы обращения к памяти, принципы представления информации в машине. В дополнение к этому архитектура УВК включает вопросы логической компоновки.

Развитие ЭВМ в рамках определенной архитектурной линии предполагает *преемственность* основных элементов архитектуры. Это необходимо для того, чтобы пользователи при обновлении парка ЭВМ имели

возможность использовать ранее созданный задел прикладного ПО. Принцип преемственности был соблюден при замене первой массовой отечественной управляющей ЭВМ М-6000 АСВТ-М [1] на машины третьего поколения СМ-1, СМ-2/СМ-2М СМ ЭВМ и сохранен при разработке новой ЭВМ данного семейства СМ 1210.

Вместе с тем развитие архитектуры предоставляет дополнительные возможности, повышающие эффективность использования ЭВМ. Естественно, что появление новых, более совершенных элементов архитектуры приводит к появлению новых или модификации ранее разработанных компонентов внутреннего ПО.

Основные стимулы и направления развития архитектуры и программного обеспечения семейства ЭВМ типа М-6000 были обусловлены применением их в составе УВК, где использовались многозадачные операционные системы реального времени (ДОСРВ М-6000) – наиболее характерные средства программной реализации сложных алгоритмов управления [2]. Однако со временем архитектура М-6000 и ее система программного обеспечения перестали удовлетворять потребностям практики. Повышенные требования к производительности и живучести УВК, особенно в ответственных системах управления, привели к двухпроцессорной конфигурации УВК, увеличению объема памяти, к функциональному резервированию блоков памяти и каналов связи в УВК, к введению мощных средств контроля и реконфигурации и эффективных команд дополнительного набора. Усложнение алгоритмов управления, необходимость облегчить взаимодействие разработчиков отдельных подсистем управления при создании крупных проектов вызвали необходимость поддержать многозадачные системы на уровне архитектуры ЭВМ в ее многораздельном варианте.

Такая архитектура предусматривает разбиение оперативной памяти ВК на *разделы*. При этом обеспечивается аппаратная защита памяти от несанкционированного взаимного влияния задач, находящихся в разных разделах (например, вследствие ошибок программирования). Здесь речь идет о *логических* разделах памяти, которые пользователь может рассматривать как отдельные псевдомашины, обеспечивающие выполнение программ практически независимых пользователей.

Наличие аппарата логических разделов позволяет совмещать во времени подготовку и отладку программ, проведение научно-технических расчетов и реализацию на ЭВМ различных функций управления объектами.

Именно характер задач управления и, в первую очередь, особенности самих объектов управления наложили свои требования на концепцию, структуру и функции внутреннего программного обеспечения ВК СМ-2М.

В отличие от традиционного ПО первой очереди, поставлявшегося с комплексами М-6000 АСВТ-М, все программное обеспечение комплексов СМ-2М построено по *агрегатно-модульному принципу* и получило на-

звание *агрегатной системы программного обеспечения* (АСПО) СМ ЭВМ [3].

В АСПО принцип агрегатности, который при разработке УВК на базе М-6000 касался только технических средств и позволял пользователю компоновать *агрегат*, т.е. УВК нужной конфигурации, из достаточно разнообразной номенклатуры аппаратных модулей (агрегатных средств вычислительной техники АСВТ-М), был распространен на программное обеспечение.

На чем же основывалась первая очередь программного обеспечения, чем она не удовлетворяла массового пользователя, и чем можно объяснить появление модульного подхода при проектировании второй очереди внутреннего ПО?

Ответы на эти вопросы можно получить, если напомнить, что все ПО первой очереди представляло собой совокупность некоторых автономных и функционально замкнутых программных систем, таких, как операционные системы, система подготовки программ, библиотеки стандартных подпрограмм, пакеты прикладных программ и т.п.

Характерными особенностями каждой из этих систем являлись неделимость и ненастраиваемость модулей, из которых состояли эти системы, а также замкнутость их структуры, что не позволяло изменить прежний или включить новый модуль, разработанный самим пользователем с учетом его собственных интересов, т.е. все эти системы обладали тремя основными недостатками: избыточностью, негибкостью и замкнутостью.

Для устранения этих недостатков при проектировании второй очереди ПО, ориентированного на применение агрегатно-модульного принципа, акцент был сделан на то, чтобы не поставлять пользователю функционально законченных программных систем, а предоставить ему средства и методы для *самостоятельной генерации* таких систем, добиваясь в конечном счете устранения избыточности и делая систему ПО достаточно гибкой.

Суть агрегатного подхода при проектировании ПО состоит в том, что вся система функционально разбивается на несколько подсистем, те, в свою очередь, – на более элементарные подсистемы, и так до тех пор, пока вся система не распадется на отдельные модули, разбиение которых уже не имеет смысла, так как это приведет к неоправданному усложнению процесса генерации программных систем пользователя. Совокупность этих модулей, интерфейсы между ними и программы их объединения (трансляторы, редакторы, компоновщик, макрогенератор и т.п.) в конкретные программные системы и составляют собственно программный продукт, разработанный в соответствии с агрегатным принципом. Этот продукт получил название *пакета программных модулей* (ППМ), ориентированного на работу в АСПО.

Концепция АСПО неразрывно связана не только с разбиением системы ПО на модули, но и с организацией этих модулей в виде специальных

библиотек. Так, модули АСПО собраны в две библиотеки: библиотеку макроопределений и библиотеку перемещаемых модулей.

Эти библиотеки организованы так, чтобы пользователь мог записать свои требования к создаваемой системе ПО на специальном языке – языке макрокоманд генерации системы. С использованием макрокоманд и указанных в них входных параметров на основе библиотеки макроопределений создается и настраивается по заданным параметрам *корневая часть* генерируемой системы. Корневая часть представляет собой как бы скелет будущей системы, в которой зафиксированы ссылки на те модули, которые будут выбраны из второй библиотеки – библиотеки перемещаемых модулей на завершающей стадии генерации системы.

В целом процесс генерации программной системы пользователя несложен и при наличии безошибочных копий указанных библиотек не требует подготовки системного программиста. Однако для того чтобы в полной мере оценить и использовать преимущества гибкой и адаптивной структуры АСПО, необходимо знание архитектуры УВК, базового языка программирования МНЕМОКОД. Кроме того, не обойтись без понимания *операционной среды* АСПО, умения применять разнообразные *пакеты прикладных программ,* разработанные для ЭВМ рассматриваемой архитектурной линии.

Всем этим вопросам и посвящен материал последующих разделов книги. Но прежде всего рассмотрим, как и из каких составных частей компонуются УВК СМ-2М.

РАЗДЕЛ ПЕРВЫЙ

АРХИТЕКТУРА УВК СМ-2М

Глава 1. СОСТАВ И ЛОГИЧЕСКАЯ КОМПОНОВКА УВК СМ-2М

1.1. Общая характеристика функциональной структуры УВК

По составу технических средств и условиям поставки различают *базовые* и *специфицированные* УВК СМ-2М.

Базовые вычислительные комплексы (БВК) имеют минимальный состав и используются в основном в качестве базы для компоновки специфицированных комплексов. Кроме того, БВК могут применяться в качестве ЭВМ общего назначения. В этом случае они поставляются потребителям как законченные изделия (рис. 1.1).

Специфицированные комплексы (УВКС) изготовляются по индивидуальному проекту на основе спецификации заказчика. Состав технических средств в специфицированных УВК определяют на основе анализа предполагаемого применения.

Рассмотрим некоторые особенности основных элементов УВК СМ-2М.

Оперативное запоминающее устройств (ОЗУ), или оперативная память, конструктивно выполнено в виде автономных комплектных блоков (АКБ), иначе – *кубов памяти,* емкостью 64 К байт каждый. В состав УВК СМ-2М могут входить до четырех таких АКБ.

В составе **центрального процессора (ЦПР)** имеется постоянное запоминающее устройство (ПЗУ), которое содержит стартовые программы для начальной загрузки ОЗУ, диагностические тесты, часто используемые константы, но, главное, в ПЗУ *микропрограммно* реализована вся система команд СМ-2М. Это означает, что каждая команда, которая программисту представляется элементарной, на самом деле состоит из записанной в ПЗУ последовательности микроопераций.

Наличие в составе УВК СМ-2М двух процессоров ЦПР1 и ЦПР2 повышает производительность и живучесть комплекса за счет перераспределения задач и функционального резервирования.

Блоки контроля БКНТ1 и БКНТ2 аппаратно поддерживают механизм функционального резервирования. Они выполняют функции контроля работоспособности и запуска одного процессора со стороны другого, а также обмена информацией между процессорами, минуя ОЗУ.

Согласователь ввода-вывода СВВ предназначен для подключения к УВК периферийного оборудования, т.е. различных *устройств ввода-вывода* (УВВ). Обмен информацией между парой модулей УВВ–СВВ

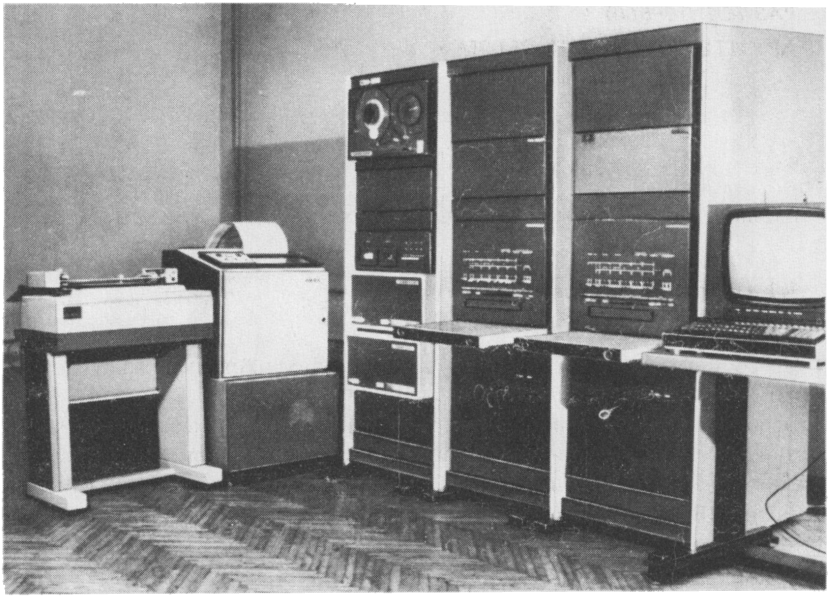


Рис. 1.1. Базовый вычислительный комплекс СМ-2М в комплектации К125-3/3

осуществляется с помощью специального интерфейсного разъема со стороны СВВ и ответной части – *интерфейсного* блока БИФ со стороны устройства (рис. 1.2).

Конструктивно СВВ выполнен в виде каркасного блока и имеет 16 мест для размещения БИФов или целиком тех УВВ, которые выполнены на одной или нескольких печатных платах. В состав УВК могут входить до трех СВВ, содержащих в общей сложности 48 интерфейсных мест и 4 места для работы с каналом прямого доступа в память (КПДП) в селекторном режиме. Если 48 интерфейсных мест оказывается недостаточно, используют *разветвители интерфейса мультиплексные* (РИМ). Разветвители используются и тогда, когда группу агрегатных модулей нужно удалить на расстояние до трех километров. Всего через цепочку СВВ–РИМ к УВК СМ-2М можно подключить до 1512 УВВ.

Канал прямого доступа в память (КПДП) предназначен для быстрого (до 1,4 млн. байт/с) обмена информацией между ОЗУ и периферийным оборудованием без участия ЦПР и представляет собой специализированное устройство для быстрой передачи информации. Каждый из двух КПДП имеет четыре подканала, и каждый подканал может обслуживать любое периферийное устройство, подключенное к СВВ. Сеанс обмена начинается по команде ЦПР и далее проходит без его участия. С точки зрения программиста КПДП представляет собой как бы один канал с восемью подканалами.

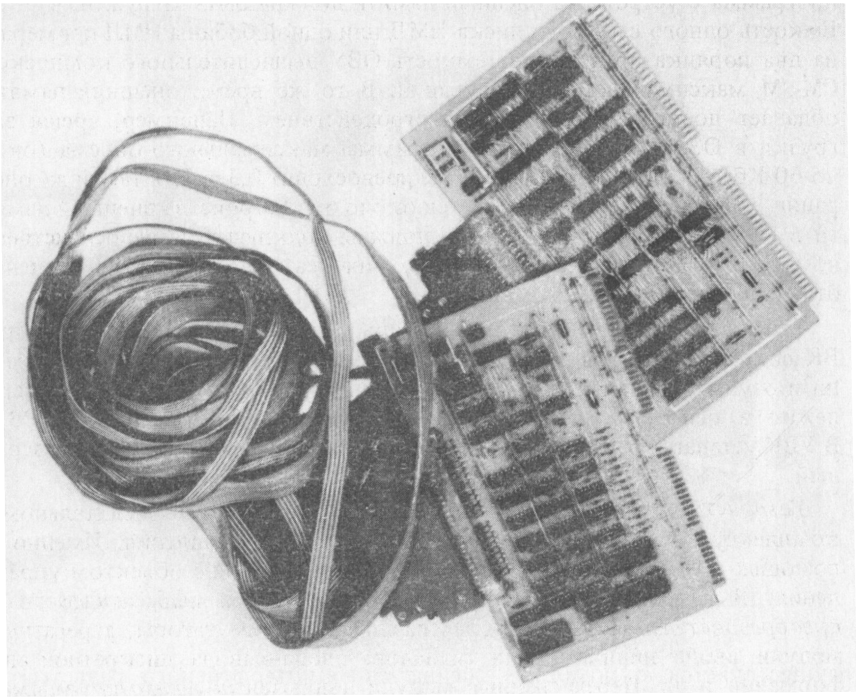


Рис. 1.2. Интерфейсный блок БИФ вместе с подводящим кабелем

Оперативные ЗУ, ЦПР, блоки контроля, СВВ и КПДП образуют *ядро* вычислительного комплекса. С другой стороны, устройства отображения, внешней памяти и связи с объектом в совокупности составляют периферийное оборудование.

К устройствам отображения относятся терминалы, в том числе экранные пульты для связи с ядром УВК, терминалы для отображения графической информации, а также печатающие устройства, позволяющие документировать при желании алфавитно-цифровую и графическую информацию. С помощью специальной аппаратуры передачи данных устройства отображения могут быть удалены от ядра комплекса на значительные расстояния: до трех километров с использованием собственных *внутрисистемных связей,* а при подключении к телеграфным и телефонным линиям – практически неограниченно.

Устройства внешней памяти представляют собой *накопители на магнитных лентах* (НМЛ) и *дисках* (НМД) и предназначены для хранения программ пользователя, системных программ, банков данных. Они являются также носителями программного обеспечения, поставляемого вместе с вычислительным комплексом. Перед выполнением любая

программа с устройства внешней памяти должна быть загружена в ОЗУ. Емкость одного сменного диска НМД или одной бобины НМЛ примерно на два порядка превосходит емкость ОЗУ вычислительного комплекса СМ-2М максимальной конфигурации. В то же время внешняя память обладает достаточно высоким быстродействием. Например, время загрузки в ОЗУ на выполнение программы максимального объема (около 60 Кбайт) с НМД ИЗОТ-1370 не превосходит 0,3 с. Для такой же операции с НМЛ ИЗОТ-5003 требуется около 6 с. Устройства внешней памяти в силу своего быстродействия должны подключаться непосредственно к СВВ (к ядру вычислительного комплекса) и не могут быть удалены от него более чем на 10 м.

Таймер (ТМР) – генератор временных отметок. Он посылает в ядро ВК сигналы через равные, программно регулируемые интервалы времени и осуществляет счет текущего времени суток. Использование таймера лежит в основе *операционных систем реального времени* (ОС РВ). В УВК устанавливаются, как правило, два таймера, один из них – резервный.

Устройства связи с объектом (УСО) позволяют вычислительному комплексу осуществлять функции контроля и управления. Именно с помощью УСО реализуется прямая и обратная связи с объектом управления. В состав УСО входят *аналого-цифровые* и *цифро-аналоговые преобразователи* (АЦП и ЦАП), различные коммутаторы, агрегатные модули ввода инициативных сигналов, ввода-вывода дискретной информации и др. Перечисленные модули являются *программируемыми*, т.е. управляемыми программой пользователя. В сочетании с высокой производительностью ЦПР и большой пропускной способностью каналов ввода-вывода это дает возможность:

программным путем задавать порядок и темп опроса датчиков, установленных на объекте;

в темпе поступления проводить как первичную (сравнение параметров с уставками, масштабирование, тарировку и т.п.), так и вторичную (усреднение, сглаживание и др.) обработку поступающей с объекта информации;

задавать порядок, темп и уровень воздействий на исполнительные механизмы системы управления.

В последнее время получили развитие *активные субкомплексы* связи с объектом (ССО), которые существенно облегчили пользователям выбор технических и программных средств для реализации различных функций управления.

Свойство активности реализуется *специальным микропроцессором*, который входит в состав субкомплекса и управляет совместной работой установленных в нем модулей. При этом пользователю в своей программе достаточно "запустить" нужную операцию в субкомплексе, после чего она проходит без участия ЦПР.

Помимо субкомплексов связи с объектом общего назначения, которые кроме программируемых модулей ввода-вывода содержат нормализаторы, усилители и другие специализированные устройства, позволяющие подключать к УВК без дополнительных согласователей практически всю номенклатуру датчиков, применяемых в АСУ ТП, разработаны терминальные субкомплексы типа "рабочее место", имеющие в своем составе средства связи с ядром ВК, функциональную клавиатуру оператора-технолога АСУ ТП, устройства печати, цветной индикации и т.п., терминальные проблемно-ориентированные субкомплексы связи с объектом для АЭС, субкомплексы внешней памяти и ряд других.

Кроме упомянутых выше устройств и субкомплексов в состав периферийного оборудования ВК могут входить также модули внутрисистемной связи и аппаратуры передачи данных.

Связь между ядром вычислительного комплекса и периферийным оборудованием имеет определенную логическую организацию и соответствующее конструктивное исполнение, т.е. определенный *интерфейс.* Вычислительные комплексы архитектурной линии М-6000/СМ-2М имеют интерфейс, называемый *сопряжением* 2К или просто 2К. Поэтому непосредственно к СВВ или РИМ комплекса СМ-2М подключаются только те устройства, которые имеют выход на этот интерфейс. Вместе с тем в номенклатуре АСВТ-М имеются согласователи интерфейса 2К практически со всеми интерфейсами, нашедшими применение в отечественной вычислительной технике.

1.2. Логическая компоновка УВК

На основе базовых вычислительных комплексов СМ-2М, подключая к ним дополнительное периферийное оборудование, можно компоновать специфицированные вычислительные комплексы.

Состав технических средств и способы их подключения к ядру УВК СМ-2М оформляются в виде *структурных электрических схем.* На рис. 1.3 представлен один из базовых комплексов К125-3/3.

На структурных электрических схемах для каждого модуля вычислительного комплекса указывается условное обозначение и шифр в соответствии с номенклатурным перечнем АСВТ-М и СМ ЭВМ. Например, программируемый таймер имеет условное обозначение ТМР и шифр А129.

Ядро БВК, представленного на рис. 1.3, имеет следующий состав:

два процессора СМ-2М А131-15;

четыре куба оперативной памяти УОП А211-20;

четыре коммутатора КМР-4 (коммутатор обеспечивает связи между модулями ядра комплекса, но конструктивно не является самостоятельным модулем, а входит в состав АКБ УОП А211-20);

три согласователя ввода-вывода СВВ А151-12.

В состав периферийного оборудования входят:

устройство внешней памяти на магнитной ленте НКМЛ СМ 5211.01;

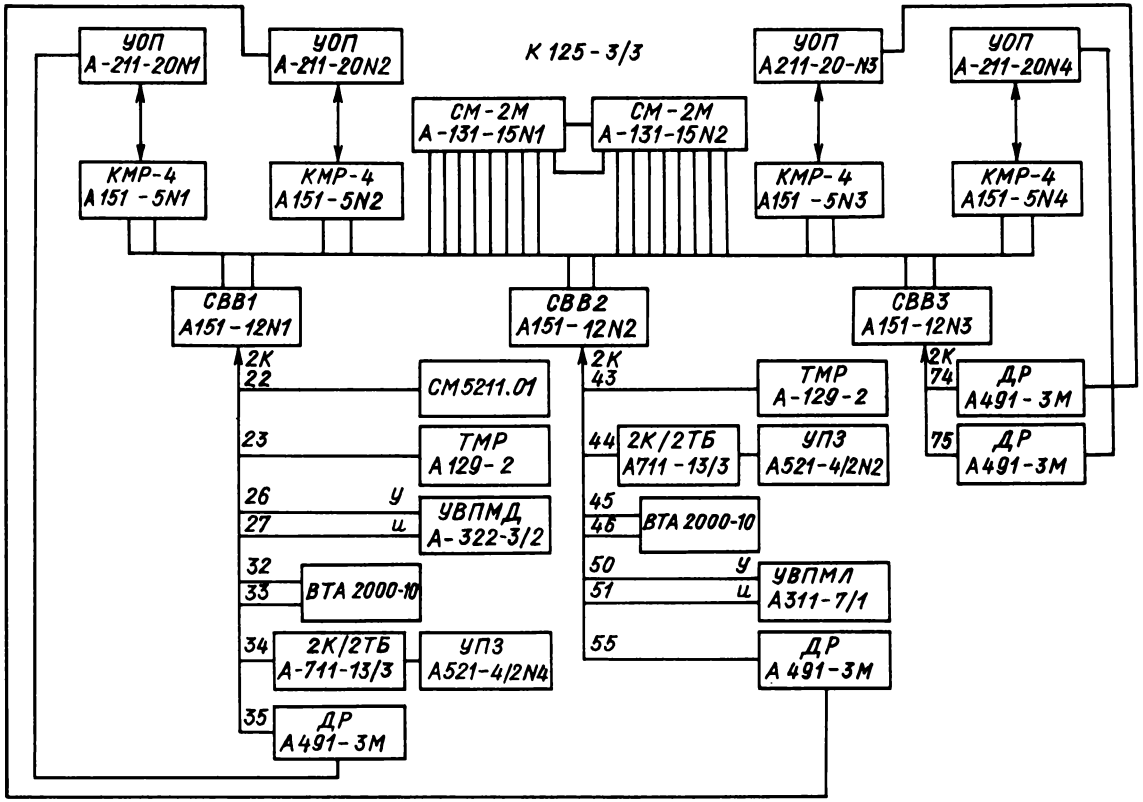


Рис. 1.3. Структурная электрическая схема БВК К125-3/3

программируемый таймер ТМР А129-2;
устройство внешней памяти на магнитных дисках УВПМД А323-3/2;
видеотерминал алфавитно-цифровой ВТА 2000-10;
согласователь интерфейсов 2К/2ТБ А711-13/3;
устройство печати знакосинтезирующее УПЗ А521-4/2;
дуплексный регистр ДР А491-3М (связь УОП-ДР используется для проверки работоспособности куба памяти специальной тестовой задачей).

На структурных электрических схемах не указываются каналы КПДП и блоки контроля БКНТ, входящие в состав процессора А131-15.

На структурных схемах УВК проставляются *коды выборки* (КВ), т.е. номера интерфейсных разъемов СВВ, к которым подключены устройства ввода-вывода. Код выборки является *адресом* устройства при обращении к нему процессора или канала прямого доступа. При этом используется восьмеричная система счисления: коды выборки с 00 по 17 адресуют элементы ядра (например, канала), коды выборки с 20 по 37 соответствуют разъемам первого СВВ, с 40 по 57 – разъемам второго СВВ и с 60 по 77 – разъемам третьего СВВ.

Различные устройства ввода-вывода могут занимать в СВВ одно, два и большее число мест. Например, УВПМД А322-3/2 подключается к комплексу двумя БИФ, для чего требуются два разъема в СВВ. Поэтому говорят, что УВПМД *занимает два выхода* на интерфейс 2К, при этом один БИФ используется для передачи управляющих сигналов (*У*), другой – информационных (*И*).

Устройства, имеющие выход на другой интерфейс, подключаются через соответствующие согласователи. На рис. 1.3 этим способом подключено устройство печати УПЗ А521-4/2.

Возможность расширения конфигурации комплекса СМ-2М определяется числом свободных мест в его СВВ. Как видно из рассмотренной структурной схемы, БВК К125-3/3 имеет 8 свободных мест в СВВ1, 9 мест в СВВ2 и 14 мест в СВВ3.

При разработке проекта УВКС пользователь должен на электрической структурной схеме отобразить его состав и указать коды выборки, к которым будут подключать дополнительное периферийное оборудование.

Занимая периферийным оборудованием свободные коды выборки, необходимо учитывать, что чем меньший КВ имеет устройство, тем более высокий приоритет получают его запросы на обслуживание модулями ядра УВК. С другой стороны, целесообразно равномерно распределять УВВ по согласователям ввода-вывода базового комплекса, чтобы не перегрузить какой-нибудь из них по суммарной токовой нагрузке.

В соответствии с согласованным проектом завод-изготовитель компонует УВКС и проверяет его работоспособность на контрольных задачах, соответствующих логической компоновке УВКС.

1.3. Компоновка одномашинных специфицированных комплексов

Основным способом расширения конфигурации базовых комплексов СМ-2М является подключение дополнительных периферийных устройств в составе субкомплексов различного назначения.

Этот вариант обладает следующими преимуществами по сравнению с непосредственным подключением агрегатных модулей через СВВ или через РИМ:

значительно уменьшается загрузка ЦПР и памяти УВК;

повышается скорость сбора и выдачи информации;

упрощается компоновка, наладка и эксплуатация комплекса.

В технически обоснованных случаях допускается подключение к комплексам СМ-2М отдельных агрегатных модулей номенклатуры АСВТ-М и СМ ЭВМ [4, 5]. Агрегатные модули могут подключаться либо непосредственно к свободным выходам на интерфейс 2К в СВВ базового комплекса, либо через мультиплексный разветвитель интерфейса РИМ А714-5.

Аналогично СВВ РИМ представляет собой каркасный блок, содержащий 16 мест для установки и подключения агрегатных модулей или согласователей интерфейса, имеющих выход на 2К.

Выпускаются две модификации РИМ. Первая модификация (РИМ-1) подключается непосредственно к ядру ВК парой модулей внутрисистемной связи МВС А723-5, соединенных между собой двухпроводной линией экранированных кабелей длиной до трех километров. При этом один из МВС устанавливается в СВВ и занимает два выхода на интерфейс 2К, а другой подключается к периферийной стороне РИМ-1 (выходов на 2К не занимает).

Вторая модификация (РИМ-2) предназначена для подключения к РИМ-1 (на расстояние не более 8 м) в целях получения дополнительных выходов на интерфейс 2К. К одному РИМ-1 можно подключить до восьми РИМ-2, каждый из которых будет содержать 16 выходов на 2К. При этом число мест в самом РИМ-1 не уменьшается. Однако не все 144 выхода на 2К, существующие в максимальной цепочке РИМ-1 – РИМ-2 № 1- . . . - РИМ-2 № 8, могут использоваться. В силу ограничений, связанных с архитектурой команд ввода-вывода, суммарное число интерфейсных разъемов, т.е. кодов выборки, используемых в цепочке РИМ-1 – РИМ-2, не должно превосходить 63. При этом на неиспользованные интерфейсные разъемы устанавливаются специальные перемычки, чтобы обеспечить последовательную нумерацию используемых.

Устройства, подключенные к ВК через РИМ, адресуются из ВК двумя кодами выборки: кодом выборки модуля МВС, которым РИМ-1 подключен к СВВ, и кодом выборки устройства в цепочке РИМ-1 – РИМ-2. Существенно, что независимо от способа подключения для программного управления устройством используется один и тот же модуль ОС (*драйвер*), который в процессе генерации операционной системы

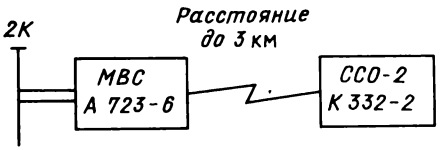


Рис. 1.4. Схема подключения ССО

настраивается на конкретный тип подключения и фактические коды выборки УВВ.

При использовании РИМ для компоновки УВКС необходимо помнить, что обмен информацией с устройствами, подключенными через РИМ, происходит по последовательному каналу – одножильному кабелю. Поэтому скорость обмена через РИМ существенно ниже, чем при непосредственном подключении устройства к СВВ. По этой причине через РИМ нельзя подключать устройства внешней памяти.

Подключение через РИМ активных субкомплексов не предусмотрено. Те из них, которые допускается удалять от ядра ВК, имеют в своем составе модули, аналогичные МВС, и подключаются, как показано на рис. 1.4. Не допускается удалять от ядра ВК более чем на 10 м субкомплексы внешней памяти.

В ситуации, когда применение субкомплексов не позволяет получить нужные параметры связи с объектом, оформляется карта заказа на ТВСО – *терминал вычислительный связи с одъектом.*

В качестве основного *вычислительного* (т.е. управляющего и обрабатывающего) звена в ТВСО используется микропроцессор модели СМ 50/60, архитектура которого совместима с мини-ЭВМ линии М-6000/СМ-2М.

Микропроцессор СМ 50/60 является свободно программируемым и доступен пользователю так же, как и ЦПР комплекса СМ-2М. По существу, вычислительный терминал представляет собой специализированную ЭВМ, способную работать как в составе УВК на нижних уровнях управляения, так и автономно.

В состав ТВСО кроме каналов связи с объектом и модулей связи с УВК могут входить устройства отображения и внешней памяти, что и позволяет использовать ТВСО как автономный ВК.

Важно отметить, что все имеющиеся в ТВСО каналы связи с объектом проходят в процессе изготовления *метрологическую аттестацию.*

1.4. Компоновка многомашинных комплексов

Два и более комплексов СМ-2М могут быть объединены в многомашинный комплекс. В связи с тем что в многомашинном комплексе возможны транзитные передачи информации (например, из ВК 1 в ВК 3 через внешнюю память ВК 2), при компоновке целесообразно закладывать минимальное число линий связи.

Для межмашинной связи могут использоваться:

пара дуплексных регистров А491-3М (допустимое расстояние такое же, как и при подключении нестандартных устройств) (см. § 1.3);

пара модулей внутрисистемной связи А723-5/1 (допустимое расстояние до 3 км, на расстояние до 1 км имеется возможность организовать быстрый режим передачи данных);

телефонные и телеграфные линии с соответствующими модемами и аппаратурой передачи данных (расстояние практически не ограничено).

Вместе с СМ-2М в многомашинный комплекс могут входить однопроцессорные комплексы той же архитектурной линии – ВК СМ 1634. Вычислительные комплексы СМ 1634 выполнены на основе микропроцессора СМ 50/60, имеющего систему команд СМ-2М (без некоторых команд дополнительного набора). Вместо интерфейса 2К комплексы СМ 1634 выходят на интерфейс ИУС, который позволяет подключать без дополнительных согласователей более совершенные модули второй очереди СМ ЭВМ.

Комплексы СМ 1634 выпускаются в различных вариантах. Например ВК СМ 1634.03 можно использовать как субкомплекс-концентратор, к которму подключают до 16 активных субкомплексов на базе микропроцессора СМ 50/60. Если учесть, что терминальные субкомплексы через РИМ не подключаются, а число мест в СВВ СМ-2М, как правило, ограничено, то становится ясна роль субкомплексов-концентраторов как своего рода разветвителей интерфейса для субкомплексов УВК СМ-2М.

В многомашинных комплексах УВВ могут подключаться к одному, двум и более ВК. Например, в конструкции РИМ-1 предусмотрена возможность подключения его к двум ВК, а терминальный субкомплекс "рабочее место оператора-технолога" РМОТ-1 может быть подключен к четырем ВК одновременно.

Поставляемые пользователям многомашинных комплексов операционные системы ОС ВС (операционная система вычислительной сети) и РОС (распределенная операционная система) обеспечивают:

доступ задач одного ВК к периферийному оборудованию других ВК (уровень доступа к периферийному оборудованию "чужого" ВК регулируется при генерации ОС);

обмен данными и событиями между задачами независимо от места их расположения (в одном ВК, в двух ВК, непосредственно связанных между собой, в двух ВК, связанных через промежуточные ВК);

централизованное управление многомашинным комплексом с одного пульта (терминала), назначенного при генерации ОС;

начальную загрузку одного ВК из другого ВК.

Важно, что на уровне пользовательских программ имеется полная программная совместимость ОС ВС и РОС с операционными системами АСПО. Это позволяет при переходе к многомашинному комплексу использовать без дополнительных доработок программный задел, накопленный в процессе эксплуатации одномашинного ВК.

В заключение отметим, что приобретение правильно скомпонованного, в полной мере соответствующего предполагаемому применению

УВКС является необходимым условием успешного решения задач автоматизации. Другое необходимое условие – соответствующая квалификация инженерных кадров. Материал гл. 2 является как бы введением в архитектуру основных элементов ядра ВК – его процессора и памяти. По-видимому, только архитектура дает отчетливое представление о логике происходящего в машине, без чего трудно рассчитывать на квалифицированную эксплуатацию УВКС.

Глава 2. АРХИТЕКТУРА ПРОЦЕССОРА И ПАМЯТИ

2.1. Представление информации в ЭВМ

Информационная модель одного куба ОЗУ УВК СМ-2М может быть представлена в виде двоичной матрицы, имеющей 32 768 строк и 16 столбцов (табл. 2.1).

Каждый элемент матрицы – двоичный разряд как элементарный носитель информации – может находиться в одном из двух состояний: 0 и 1.

Применяют следующие меры информации:

1 бит – количество информации в одном двоичном разряде.

1 байт равен 8 бит. Так как имеется 256 различных комбинаций из восьми 0 и 1, то в одном байте содержится информации ровно столько, сколько нужно, чтобы указать на одну из 256 возможностей, в частности закодировать 256 различных символов.

1 килобайт (1 Кбайт) равен 1024 байт; эта мера количества информации часто применяется для измерения объема ОЗУ. Нетрудно подсчитать, что объем одного куба ОЗУ СМ-2М равен 64 Кбайт, т.е. в нем может быть представлено более 64 тысяч различных символов – почти две газетные полосы. Максимальный объем памяти СМ-2М достигает 256 Кбайт.

1 мегабайт (1 Мбайт) равен 1024 Кбайт. Применительно к малым ЭВМ в этих единицах обычно измеряют емкость внешней памяти. Например, сменный диск НМД ИЗОТ 1370 вмещает 2,5 Мбайт информации, что в 3 раза превышает объем данной книги.

Машинное слово. Контроль по паритету. Каждая строка двоичной матрицы – информационной модели куба ОЗУ СМ-2М – вмещает 2 байта информации и представляет одну *ячейку памяти.* В отличие от машинно независимых единиц измерения информации (байт, килобайт и др.) количество информации, содержащееся в одной ячейке памяти, является машинно зависимым и называется *машинным словом.* По аналогии с килобайтами вводится машинно зависимая единица измерения информации– одно *килослово,* обозначаемая 1 К. В этих единицах объем одного куба ОЗУ СМ-2М равен 32 К, а четырех кубов в ОЗУ максимальной конфигурации 128 К.

Таблица 2.1

Адрес ячейки		Номер разряда															
Восьмеричный	Десятеричный	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	1	1	0	1	1	0	0	0	1	1	0	1	1	1
1	1	1	0	1	0	0	0	0	0	0	1	0	1	1	1	1	0
2	2	1	1	1	1	0	1	1	0	1	0	1	0	0	0	1	1
									.								
									.								
									.								
7777	4095	1	0	0	0	1	0	0	0	1	1	1	1	0	0	1	0
10000	4096	1	1	0	1	0	0	0	0	1	0	1	0	0	0	1	1
10001	4097	0	0	1	1	1	1	0	1	0	1	0	1	1	1	0	1
10002	4098	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0
10003	4099	0	0	0	1	0	1	1	1	1	1	0	1	0	1	0	1
10004	4100	0	0	1	0	1	1	0	0	1	1	1	0	1	0	0	0
77773	32763																
77774	32764	0	0	0	0	1	1	0	1	1	1	0	1	0	1	1	0
77775	32765	0	1	1	1	0	0	1	1	1	0	0	1	0	0	0	0
77776	32766	1	1	0	1	1	0	0	0	0	1	0	0	0	1	1	0
77777	32767	1	1	0	1	0	1	1	0	0	0	1	1	1	0	0	1

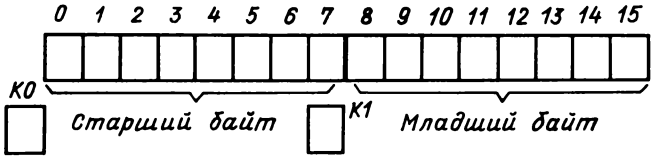


Рис. 2.1. Формат машинного слова СМ-2М:
К0, К1 – контрольные разряды

Биты в машинном слове СМ-2М нумеруются слева направо, как показано на рис. 2.1. Нулевой разряд, так же как и левый байт, считается *старшим,* пятнадцатый разряд, так же как и правый байт, – *младшим.*

В конструкции ЭВМ предусмотрен механизм контроля достоверности передачи информации. С этой целью на каждый байт информации введен *контрольный разряд.* Если число единиц в байте четно, значение контрольного разряда схемным путем устанавливается в единицу. В противном случае значение контрольного разряда устанавливается в нуль (так называемые *условия паритета*).

Передача информации ведется вместе с контрольными разрядами. Если информация в каком-то разряде машинного слова будет потеряна, сразу нарушится условие паритета, что немедленно обнаружится специальными *схемами контроля* процессора с выдачей сообщения на пульт оператора. Такой контроль называют *контролем по паритету нечетности.*

Типы данных. Всю информацию, хранящуюся в памяти машины, условно можно разделить на *команды* и *данные.* Команды в своей совокупности образуют *программу,* которая содержит следующую информацию: какие действия и в каком порядке следует выполнять ЭВМ для достижения требуемых результатов. Физически команды и данные неразличимы – и те и другие в памяти машины представляют собой последовательность нулей и единиц.

Данные, в свою очередь, делятся на числовые и логические (в том числе символьные).

Представление символов. К символам относятся буквы латинского и русского алфавитов, знаки препинания, цифры и различные специальные символы (косая черта, знак плюс и др.). Например, буква *А* русского алфавита кодируется последовательностью 1000001, а цифра *6* – последовательностью 0110110. Для кодировки числа *613* как совокупности символов потребуется уже 21 двоичный разряд. Неудобства работы с такими длинными последовательностями 0 и 1 для человека очевидны. Поэтому в инструкциях, *листингах* (распечатках) и других подобных документах применяются восьмеричные цифры, каждая из которых заменяет три двоичных разряда (*триаду*). Взаимооднозначное соответствие триад и восьмеричных цифр дано в табл. 2.2. (Более подробно см. [6].)

Восьмеричными цифрами (восьмеричным кодом) двоичные коды представляются значительно компактнее. Приведенные в табл. 2.3 основные символы СМ-2М закодированы именно таким способом.

При размещении в памяти СМ-2М каждый символ занимает один байт, в старший разряд которого записывается 0, а в семь младших – 7-разрядный код символа. Исходя из этого для размещения совокупности символов, изображающих число *613* (рис. 2.2), требуются две

Таблица 2.2

Триада	Восьмеричная цифра
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Таблица 2.3

Наименование символа	Изображение символа	Машинный восьмеричный код
Пробел		40
Восклицательный знак	!	41
Кавычки	"	42
Номер	#	43
Знак денежной единицы	¤	44
Проценты	%	45
Коммерческое ПРИ	@	46
Апостроф	'	47
Круглая скобка левая	(	50
Круглая скобка правая	)	51
Звездочка	*	52
Плюс	+	53
Запятая	,	54
Минус	–	55
Точка	.	56
Дробная черта	/	57
Цифры	0–9	60–71
Двоеточие	:	72
Точка с запятой	;	73
Меньше	<	74
Равно	=	75
Больше	>	76
Вопросительный знак	?	77
Коммерческое И (амперсенд)	&	100
Латинские прописные буквы	A–Z	101–132
Квадратная скобка левая	[	133
Обратная косая черта	\	134
Квадратная скобка правая	]	135
Стрелка вверх	↑	136
Стрелка влево	←	137
Русские прописные буквы, не совпадающие по написанию с латинскими	Ю, Б, Ц, Д, Ф, Г, И, Й, Л, П, Я, Ж, Ь, Ы, З, Ш, Э, Щ, Ч	140, 142, 143, 144, 146, 147, 151, 152, 154, 160, 161, 166, 170, 171, 162, 173, 174, 175, 176

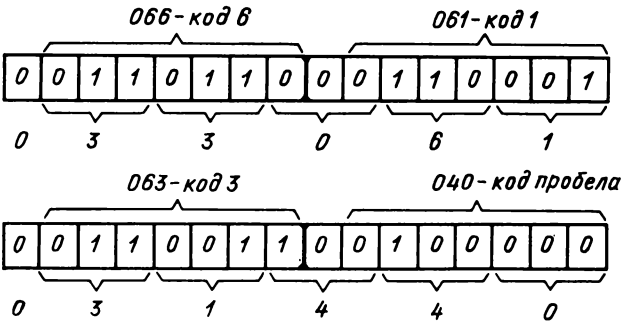


Рис. 2.2. Кодировка числа 613 как совокупности символов

ячейки памяти СМ-2М. (Заметим, что упаковка символов по два в ячейку не является самой экономной, так как 2 разряда при этом не используются. Существует код REDIX-50, описанный, например, в [7], который более сложен и представляет меньшее количество символов, но лишен этого недостатка).

Разбивая 16-разрядный код – машинное слово – на триады (справа налево, так, что старшая триада оказывается неполной), можно получить его восьмеричное представление. Например, два машинных слова на рис.2.2 представляются восьмеричными кодами 033061 и 031440. Можно сказать, что это кодировка числа *613* как совокупности символов.

Дополнительный код целых чисел. Запись числа как совокупности символов используется для его представления на устройствах отображения. Внутренний (машинный) код чисел должен быть приспособлен для выполнения арифметических операций, причем наиболее простым для аппаратурной реализации способом.

В этом смысле не может удовлетворить самое естественное, на первый взгляд, позиционное двоичное представление чисел (*прямой код*) со знаком в старшем разряде, поскольку в этом случае аппаратурная реализация сложения положительных чисел не годилась бы для отрицательных слагаемых, и наоборот.

Этого недостатка лишен *дополнительный код,* в соответствии с которым положительные числа представляются в прямом коде, а смена знака числа осуществляется *инверсией* (заменой на противоположное значение) всех разрядов в его изображении с последующим *инкрементированием* (прибавлением единицы) младшего разряда. Оказывается, что полученные таким способом отрицательные числа, как и положительные, могут суммироваться арифметическим устройством очень простой конструкции. Важно, что при этом знаковый разряд участвует в операции сложения на равных с другими разрядами числа. Проиллюстрируем

это операцией двоичного сложения чисел, записанных в ячейках 10002_8 и 10003_8 в табл.2.1:

$$\begin{array}{rl} & 1010001010000010_2 = -23934_{10} \\ + & 0001011111010101_2 = \quad 6101_{10} \\ \hline & 1011101001010111_2 = -17833_{10} \end{array}$$

Здесь знак результата получился автоматически (читатель может убедиться, что более "естественные" способы представления целых чисел таким свойством не обладают).

Свойства дополнительного кода удобно изучать, пользуясь формулой

$$N = -d_0 \cdot 2^{n-1} + d_1 \cdot 2^{n-2} + \ldots + d_{n-1} \cdot 2^0, \qquad (2.2)$$

где N – число, представленное n-разрядным дополнительным кодом; d_i – двоичные разряды этого кода.

Доказательство того, что эта формула действительно определяет значение числа в дополнительном коде, основано на том, что число N', полученное из N инверсией и инкрементированием, будет определяться соотношением

$$N' = -(1 - d_0)2^{n-1} + (1 - d_1)2^{n-2} + \ldots + (1 - d_{n-1})2^0 + 1. \qquad (2.3)$$

Нетрудно убедиться, что сумма правых частей (2.2) и (2.3) равна нулю. Отсюда следует, что N и N' равны по абсолютной величине и противоположны по знаку.

Анализируя (2.2) при n=16, устанавливаем диапазон целых чисел, представляемых одним машинным словом СМ-2М: от минус 32768 при $d_0 = 1$ и $d_1 = d_2 = \ldots = d_{n-1} = 0$ до плюс 32767 при $d_0 = 0$ и $d_1 = d_2 = \ldots = d_{n-1} = 1$.

Некоторые команды СМ-2М позволяют манипулировать целыми 32-разрядными числами (*числами двойной длины*), которые представляются двумя машинными словами. В этом случае в формуле (2.2) $n = 32$, и диапазон представления чисел – от минус 2 147 483 648 до плюс 2 147 483 647.

Представление вещественных чисел. Вещественные числа, так же как и целые двойной длины, кодируются двумя машинными словами. *Формат*, т.е. способ кодировки, определяющий назначение разрядов в двоичном представлении вещественных чисел, показан на рис.2.3. Здесь 24-разрядная мантисса и 8-разрядный двоичный порядок числа представляются в дополнительном коде.

Напомним, что значение вещественного числа

$$F = mb^p, \qquad (2.4)$$

Рис. 2.3. Формат вещественных чисел:
$d_i^{(m)}$ – разряды мантиссы; $d_i^{(p)}$ – разряды порядка

где m – мантисса числа, p – порядок; b – основание принятой системы счисления (для двоичного кода $b = 2$).

Так как абсолютная величина мантиссы m нормализованного двоичного числа должна удовлетворять неравенству

$$0{,}5 \leqslant |m| \leqslant 1, \tag{2.5}$$

то результат декодирования по (2.2) 24-разрядного кода мантиссы умножается на 2^{-23}, иначе говоря, вместо (2.2) для расчета мантиссы должно применяться соотношение

$$m = -d_0 + 2^{-1}d_1 + \ldots + 2^{-23}d_{n-1}. \tag{2.6}$$

Таким образом, для положительных чисел вида $F = 2^k$ мантисса m в условиях ограничения (2.5) может принимать единственное значение 0,5, но для чисел $F = -2^k$ в качестве отрицательной мантиссы может быть принята либо минус 1 ($d_0 = 1$ и остальные $d_i = 0$), либо минус 0,5 ($d_0 = d_1 = 1, d_i = 0$ при $i \geqslant 2$). В машине принят первый вариант, как наиболее удобный для аппаратурной реализации.

Порядок p является целым числом в диапазоне от минус 128 до плюс 127 и определяется по формуле (2.2) при $n = 8$. Особенность двоичного кода порядка состоит в том, что знаковый разряд d_0 располагается не в старшем, а в младшем разряде второго машинного слова.

Диапазон нормализованных вещественных чисел, представимых в машине,

$$2{,}9 \cdot 10^{-39} < |F| < 1{,}6 \cdot 10^{38}. \tag{2.7}$$

Его можно получить, проанализировав выражение (2.4) совместно с (2.5) при p в диапазоне от минус 128 до плюс 127.

Таким образом, не представимы (выходят за разрядную сетку) не только слишком большие числа, но и слишком маленькие, близкие к нулю. Последние заменяются *настоящим машинным нулем*, который представляется двумя машинными словами с нулями во всех разрядах.

2.2. Функции и регистры процессора

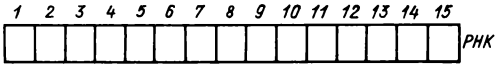
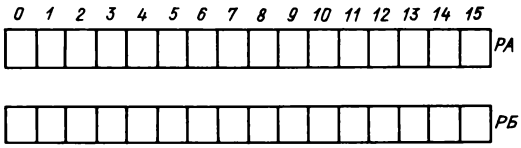
В процессе выполнения программы процессор осуществляет разнообразные операции преобразования и передачи информации. Для "рабочих записей" при проведении этих операций в конструкции процессора предусмотрены *регистры*, каждый из которых на уровне архитектуры аналогичен ячейке памяти и представляет собой совокупность двоичных разрядов.

Стремление максимально упростить процессор привело к тому, что непосредственную связь с логическими схемами процессора, в частности с *арифметическо-логическим устройством* (АЛУ), способным выполнять различные операции с двоичными кодами, в большинстве современных малых ЭВМ имеются только регистры процессора. Например, в СМ-2М результат суммирования формируется не в ячейке памяти, выделенной в программе для хранения суммы, а на специальном регистре – аккумуляторе, после чего соответствующей командой полученная сумма должна быть направлена в нужную ячейку ОЗУ.

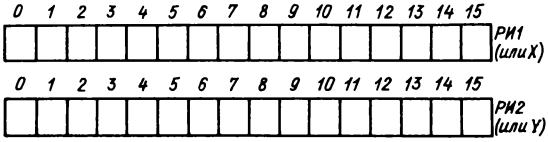
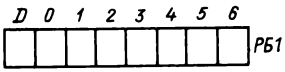
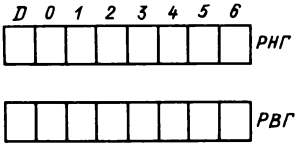
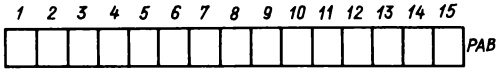
Всего в процессоре СМ-2М имеется 13 *доступных программе* регистров различной длины, содержимое которых может быть прочитано или изменено программным путем.

В табл. 2.4 представлены основные функции процессора и задействованные при их выполнении регистры.

Таблица 2.4

Функции процессора	Используемые регистры
1. Выборка очередной команды из ОЗУ для декодирования и исполнения	Регистр номера команды *РНК* 
2.1. Арифметические и логические операции 2.2. Пересылка данных по маршруту память – процессор – память 2.3. Пересылка данных по маршруту память – процессор – УВВ и обратно	Программные регистры *РА* и *РБ* (аккумуляторы) 
3. Фиксация переноса за разрядную сетку аккумулятора при арифметических и логических операциях	Одноразрядный регистр расширения *РР* 

Продолжение табл. 2.4

Функции процессора	Используемые регистры
4. Фиксация переполнения аккумулятора при арифметических операциях	Одноразрядный регистр переполнения *РП* 
5.1. Формирование относительного адреса при обработке массивов и таблиц 5.2. Косвенная адресация УВВ 5.3. Часть функции 2.1 и 2.2	Индексные регистры *РИ1* или *X* и *РИ2* или *У* 
6. Формирование абсолютного адреса в программах пользователя, размещенных в ненулевых разделах памяти	Регистр базы непривилегированного состояния *РБ1*  *D* – дополнительный разряд, обеспечивает адресацию в 17-разрядном адресном пространстве
7. Формирование абсолютного адреса в операционной системе и других программах, размещенном в нулевом разделе	Регистр базы привилегированного состояния *РБ2* 
8. Защита памяти в многораздельном режиме	Регистры верхней *РВГ* и нижней *РНГ* границ памяти 
9. Возврат в основную программу после выполнения подпрограммы или прерывания	Регистр адреса возврата *РАВ* 
10. Фиксация состояния процессора	Триггер состояния *ТС* 

Рассмотрению перечисленных в табл. 2.4 функций и посвящен весь последующий материал данного раздела книги.

Здесь же в связи с функцией, стоящей под номером 1, отметим, *что информация, хранящаяся в определенной ячейке ОЗУ, становится командой только тогда, когда номер (адрес) этой ячейки попадает на регистр номера команды (РНК)*. Адрес первой команды выполняемой программы заносится в *РНК* либо оператором с инженерной панели процессора, либо автоматически микропрограммным загрузчиком. После выполнения каждой команды, за исключением команд, осуществляющих *передачу управления,* содержимое *РНК* автоматически увеличивается на количество слов в команде. В результате на *РНК* устанавливается относительный адрес следующей команды. Такую последовательность выполнения команд называют *естественной.* После выполнения команды передачи управления на *РНК* устанавливается адрес, изменяющий естественную поледовательность – *адрес перехода.*

Если программа составлена правильно, адреса ячеек, в которых хранятся данные, никогда не попадут на *РНК.*

2.3. Организация памяти

Разделы, блоки, базовые адреса. Оперативная память УВК СМ-2М компонуется из модулей УОП А211-20 – кубов ОЗУ, каждый из которых содержит 32 К 16-разрядных ячеек. Кубы памяти нумеруются, начиная с нуля (00_2, 01_2, 10_2 и 11_2 для ОЗУ максимального объема).

Процессор обращается к ячейкам памяти ОЗУ по *адресам.* Адрес ячейки – это ее порядковый номер, причем нумерация ячеек начинается с нуля и является сквозной и непрерывной. Это означает, например, что последняя ячейка нулевого куба ОЗУ имеет адрес 77777 (здесь и далее адреса ячеек памяти задаются в восьмеричной системе счисления), а первая ячейка куба с номером 01 имеет адрес 100000. Таким образом, последняя ячейка ОЗУ максимальной емкости (т.е. последняя ячейка куба памяти с номером 11_2) имеет адрес 377777.

Для представления произвольного адреса из диапазона от 0 до 377777 требуется, как нетрудно подсчитать, 17 двоичных разрядов. В то же время в силу преемственности архитектур СМ-2М и М-6000, объем ОЗУ которой не превышал 32К, система команд СМ-2М ориентирована на работу с 15-разрядными адресами, находящимися в диапазоне от 0 до 77777. Поэтому область памяти, непосредственно доступная командам, имеет размер не более 32 К. (В машине М-6000 большего и не требовалось.)

Для расширения адресуемой памяти до 128К в конструкции процессора СМ-2М предусмотрены *базовые регистры* (функции 6 и 7 в табл. 2.4). Они предназначены для хранения базовых (начальных) 17-разрядных адресов логических разделов, на которые память СМ-2М разбивается при генерации операционной системы. (Почему при этом

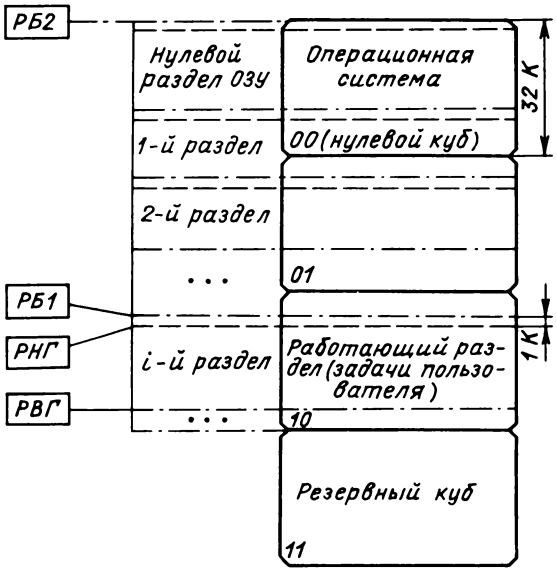


Рис. 2.4. Логическая организация памяти УВК СМ-2М:
—·— – граница раздела; – – – – граница нулевой страницы

один из базовых регистров состоит из 8 разрядов, а другой из двух, станет ясно из дальнейшего).

Кроме базового адреса каждый раздел характеризуется своим номером: 0, 1, ..., $n-1$, где n – количество разделов, созданных при генерации ОС.

Базовый адрес не является произвольным. В конструкцию машины заложено, что разделы должны состоять из целого числа *блоков* – непрерывных участков памяти по 0,5 К (512 ячеек).

Таким образом, вся память оказывается логически разбита на разделы, как показано на рис. 2.4. При этом операционная система загружается в нулевой раздел, а два первых блока каждого раздела (*нулевая страница*) используются для связи с ОС, поэтому задачам пользователя в общем случае недоступны. Отсюда следует ограничение снизу на размер одного раздела – 1,5К, хотя на практике такой раздел вряд ли может быть полезен (слишком мал!). Ограничение сверху – 32К соответствует максимальной области памяти, доступной программам одного раздела.

Относительный и абсолютный адреса. Пятнадцатиразрядный адрес, находящийся на *РНК* или полученный в результате декодирования адресной команды, называют *относительным* в отличие от 17-разрядного *абсолютного,* который является адресом ячейки при сквозной и непрерывной нумерации.

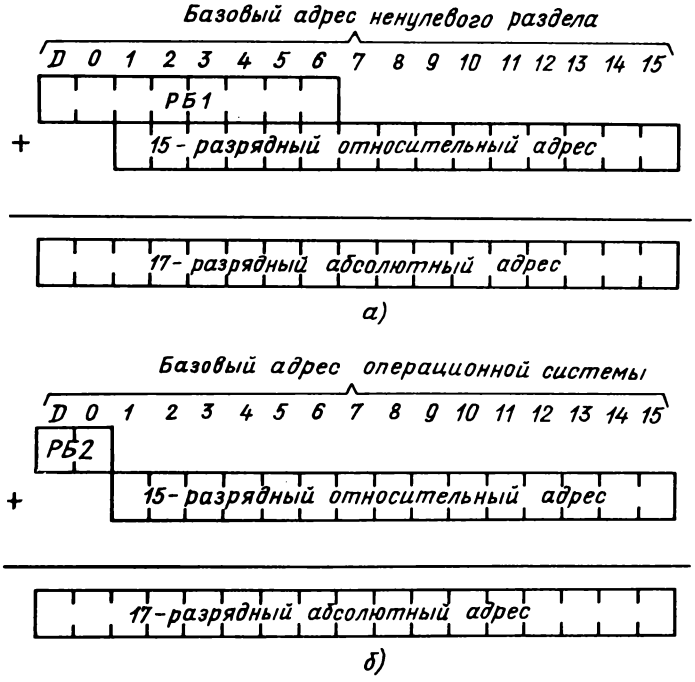


Рис. 2.5. Формирование абсолютного адреса в непривилегированном (*а*) и в привилегированном (*б*) режимах работы процессора

Поскольку логический раздел всегда состоит из целого числа блоков, базовый адрес делится на 512_{10} или 1000_8. Следовательно, для его изображения требуется 8 двоичных разрядов, так как последние 9 двоичных цифр – заведомо нули. Этот факт нашел свое отражение в архитектуре центрального процессора: регистр, предназначенный для хранения базового адреса, так называемый *регистр базы непривилегированного состояния (РБ1)*, является 8-разрядным. Схему получения абсолютного адреса по базе непривилегированного состояния можно представить, как показано на рис. 2.5, *а*.

Понятие *привилегированного* и *непривилегированного* состояний, в которых может находиться центральный процессор, имеет принципиальное значение для понимания взаимодействия программ пользователя и ОС.

Состояние процессора фиксируется на *триггере состояния (ТС)*: 0 – привилегированное, 1 – непривилегированное, и устанавливается в зависимости от того, из какого раздела команду он выполняет. Если из нулевого, то устанавливается привилегированное состояние, из любого другого – непривилегированное.

Эти два состояния *отличаются ресурсами,* доступными при выполнении команд. В привилегированном состоянии процессору доступны все ресурсы, в непривилегированном – ресурсы ограничены: не выполняются привилегированные команды, такие, как команды ввода-вывода и ряд других, а также не допускается обращение для записи информации в ячейки "чужих" разделов и ячейки своей нулевой страницы. Если из непривилегированного раздела в процессор на исполнение поступит команда, требующая привилегированных ресурсов, сразу же произойдет прерывание программы от *схем контроля* процессора с выдачей соответствующего сообщения на пульт оператора.

База привилегированного состояния. Уже отмечалось, что ОС должна загружаться в тот раздел, который при генерации ОС был объявлен нулевым. Поэтому привилегированное состояние – это состояние процессора при *выполнении программ ОС.* Говорят, что программы из нулевого раздела выполняются в *привилегированном режиме.*

Одна из особенностей архитектуры ОЗУ состоит в том, что начало нулевого раздела должно совпадать с началом одного из четырех кубов памяти, поэтому базовый адрес нулевого раздела кратен уже не 1000_8, а 100000_8. Отсюда следует, что для его представления достаточно двухразрядного регистра, каким и является *РБ2 – регистр базы привилегированного состояния.* Регистр *РБ2* (функция 7, табл. 2.4) при работе программ ОС играет ту же роль, что и *РБ1* при работе программ пользователя в непривилегированном режиме. Механизм использования *РБ2* для формирования абсолютного адреса показан на рис. 2.5, *б.*

Взаимодействие разделов. Защита памяти. Для рассмотрения этих вопросов нам придется обратиться к динамике функционирования ОС. На временной диаграмме занятости ЦПР можно выделить следующие основные моменты:

1. Работа операционной системы. Исполняются команды из нулевого раздела. Процессор находится в привилегированном состоянии.

2. Специальная программа операционной системы – диспетчер задач, сканирует таблицу задач, готовых к выполнению.

3. У самой приоритетной из готовых к выполнению задач диспетчер запрашивает, какому разделу памяти принадлежит эта задача, и затем передает ей управление: устанавливает на *РБ1* базовый адрес соответствующего раздела, на *РНК* – относительный адрес первой команды запускаемой задачи, а при переходе к задаче пользователя (в ненулевой раздел) переводит процессор в непривилегированное состояние.

4. Непосредственно перед запуском задачи, размещенной в ненулевом разделе, диспетчер задач обеспечивает защиту задач других разделов и нулевой страницы запускаемого раздела от искажений, которые могут быть следствием, например, ошибок, допущенных при программировании (функция 8, табл. 2.4). С этой целью специальными командами формирования *РНГ* и *РВГ* заносятся (см. рис. 2.4):

в регистр нижней границы – содержимое *РБ1* плюс 2, т.е. 8 старших разрядов базового адреса запускаемого раздела, увеличенного на 2000_8 (защита в том числе и нулевой страницы этого раздела);

в регистр верхней границы – содержимое *РБ1* плюс число блоков запускаемого раздела, т.е. 8 старших разрядов базового адреса, увеличенного на количество ячеек данного раздела.

Пользователь имеет возможность в своих программах обращаться к устройствам ввода-вывода, используя для этого стандартные вызовы супервизора (но не команды ввода-вывода, недоступные в непривилегированном режиме). Как только дойдет очередь до этого обращения, управление снова будет возвращено в операционную систему (непривилегированная команда – **SVC** – *вызов супервизора* – переводит процессор в привилегированное состояние и одновременно заносит в *РНК* число 6 – адрес ячейки, в которой находится команда передачи управления супервизору), процессор получит доступ к командам ввода-вывода, и запрос пользователя будет выполняться стандартными программами обслуживания УВВ. Если пользователя не устраивает эта процедура и он хочет сам программировать операции обмена с УВВ, то ему необходимо скомпоновать свои задачи вместе с операционной системой и разместить их в нулевом разделе памяти.

Однако следует иметь в виду, что работа с программным обеспечением ЭВМ на таком уровне – дело очень непростое и ответственное. Обычно ее выполняют имеющие специальную подготовку системные программисты, которых отличает детальное знакомство с операционной системой, глубокое понимание архитектуры ЭВМ и свободное владение системой команд.

2.4. Общая характеристика системы команд

По своему формату и функциональному назначению команды, реализованные в процессоре СМ-2М, разделяют на *адресные* (*основного и двойного* форматов, занимающие соответственно одно и два машинных слова), *регистровые* (в том числе команды *сдвигов* и команды группы *изменений и пропусков*), *ввода-вывода* и, наконец, команды *дополнительного набора,* которыми, в частности, процессор СМ-2М существенно отличается от своих предшественников по архитектурной линии (М-6000, СМ-1 и т.д.).

Адресные команды. Каждая команда этой группы помимо *кода операции* содержит информацию, которую логические схемы процессора преобразуют в относительный 15-разрядный адрес операнда. Из него с учетом значения базы работающего раздела формируется абсолютный адрес адресуемой ячейки памяти (рис. 2.5). Конкретное действие АЛУ над информацией, содержащейся в адресуемой ячейке, определяется кодом операции. Это может быть *пересылка* (копирование) информации из ячейки в аккумулятор и обратно, логическая или арифметическая

операция, в которой один *операнд* находится на аккумуляторе, а другой – в адресуемой ячейке. Наконец, это может быть одна из тех управляющих операций формирования *РНК* [в соответствии со значением и (или) наличием определенного признака у адресуемой информации], на основе которых реализуются логические операторы типа "**если** . . . **то** . . . **иначе**" или "**выполнять** . . . **пока** . . .".

Команды изменений и пропусков. В эту группу собраны команды изменения (установки в нуль, инверсии и инкрементирования) регистров *РА, РБ, РР* (см. табл. 2.4) и команды условного пропуска следующей ячейки, т.е. *изменения* естественного порядка выполнения программы по значению этих регистров (например, по равенству нулю, четности и т.д.). Особенностью этой группы команд является возможность кодирования в одной команде нескольких элементарных операций преобразования и анализа информации. В частности, в *одной* команде могут быть заданы операции инверсии и инкрементирования аккумулятора с последующей проверкой полученной информации на знак и на четность одновременно.

Команды сдвигов. Операнд располагается на аккумуляторе (или на двух сразу для команд сдвигов *слов двойной длины*), а в команде кодируются различные элементарные операции, в том числе и их сочетания. Однако здесь реализованы операции определенного типа – *арифметические* и *логические* сдвиги. Арифметический сдвиг двоичной информации эквивалентен умножению или делению (в зависимости от направления сдвига) на степень двойки, логические сдвиги используются, в частности, для анализа отдельных разрядов аккумулятора.

Команды ввода-вывода предназначены для пересылки информации по маршруту процессор – УВВ и обратно, а также для управления работой УВВ в составе вычислительного комплекса. Эта группа команд отражает логику интерфейса 2К и является, по существу, средством его программной поддержки.

Команды дополнительного набора включены в архитектуру СМ-2М в целях повышения ее производительности и снижения трудоемкости разработки программного обеспечения. Типичной командой дополнительного набора является команда *сканирования* массива данных, т.е. поиска в некоторой области памяти машинного слова определенного содержания.

Система команд СМ-2М обладает определенной избыточностью: в любом алгоритме, не включающем операции ввода-вывода, можно обойтись только адресными командами, да и они не все необходимы в равной мере. Однако использование других команд существенно облегчает процесс программирования и делает программы более эффективными по быстродействию и занимаемому объему. Вместе с тем логическую основу организации обработки информации на процессоре СМ-2М составляют все же адресные команды.

В заключение заметим, что, если гл. 1 и 2, как бы продолжая введение, носили ознакомительный, справочный характер, то гл. 3 и последующие имеют другую направленность. Их цель – помочь читателю научиться активно эксплуатировать программное обеспечение УВК СМ-2М, с максимальной эффективностью использовать возможности, заложенные в АСПО и пакетах прикладных программ, расширять эти возможности, разрабатывая необходимые программные модули.

Упражнение к гл. 2

При генерации операционной системы нулевой куб памяти отведен под загрузку операционной системы, третий куб – резервный. Остальная память разделена на три раздела в отношении 5 : 8 : 3.

а) Определите базовые адреса этих разделов и границы памяти, доступной командам в каждом из этих разделов.

б) Постройте нормализованный двоичный код числа минус 0,5 и определите абсолютные адреса ячеек для его занесения, если известно, что это 1100 и 1101 – ячейки нулевой страницы второго раздела.

в) Постройте код символьного представления того же числа.

Глава 3. АДРЕСНЫЕ КОМАНДЫ. ПРИЕМЫ ПРОГРАММИРОВАНИЯ

3.1. Адресация

Типы адресации. Когда говорят, что машина СМ-2М является *одноадресной,* имеют ввиду, что в каждой адресной команде кодируется один 15-разрядный относительный адрес.

На рис. 3.1 представлен формат однословной адресной команды, из которого видно, что 4 разряда занимает *код операции* (КОП) и 12 разрядов – адресная часть.

Каким же образом в 12-разрядной адресной части кодируется 15-разрядный адрес? Ясно, что здесь не обойтись без дополнительных ограничений на адресуемую область памяти либо дополнительных ячеек или регистров.

В устройстве управления процессора СМ-2М реализованы способы адресации трех типов: *прямая, косвенная* и *адресация через индексные регистры,* причем информация о том, какой конкретно способ адресации используется, содержится в самой команде, а дешифровку команды и формирование адреса осуществляют соответствующие логические схемы процессора.

Прямая адресация. Этот тип адресации связан с ограничениями на адресуемую область. Доступны:

нулевая страница (см. § 2.3) работающего раздела (прямая адресация к нулевой странице);

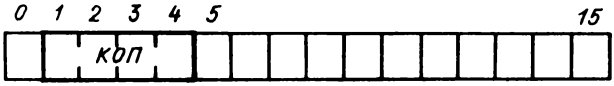


Рис. 3.1. Формат одноадресной адресной команды:
КОП – код операции, разряды 0 и 5–15 используются для кодирования адреса

текущая страница, т.е. непрерывная область памяти, размером 512 ячеек, для которой серединой является ячейка, содержащая исполняемую команду (прямая адресация к текущей странице).

Признаком прямой адресации к нулевой странице, по которому процессор распознает именно этот способ адресации, являются нули в нулевом (признак прямой адресации) и пятом (признак нулевой страницы) разрядах команды. Содержимое разрядов 6–15 воспринимается как значение адреса (без знака) в двоичной системе счисления. Например, в ячейке 77775 из табл. 2.1 представлена команда с прямой адресацией к ячейке 1620. (Следует учесть, что в непривилегированном режиме этот способ адресации может быть использован только для чтения информации.)

Признаками прямой адресации к текущей странице являются 0 в нулевом и 10_2 в пятом и шестом разрядах. Содержимое разрядов 7–15 воспринимается как положительное или отрицательное расстояние (*смещение*) местоположения операнда относительно исполняемой команды. Смещение записывается в дополнительном коде со знаком в седьмом разряде. При этом значение адреса операнда определяется путем сложения содержимого РНК (относительного адреса исполняемой команды) со смещением, значение которого определяется по формуле (2.2) при $n = 9$.

Для примера проанализируем как адресную команду содержимое ячейки 77774 в табл. 2.1. В соответствии со значениями 0, 5 и 6-го разрядов определяем, что это прямая адресация к текущей странице. Декодирование кода смещения дает минус 52. Складывая с текущим адресом, получаем 77722 – искомый относительный.

Непосредственно из (2.2) следует, что смещение лежит в пределах от минус 256 до плюс 255. Поэтому, если программа вместе с исходными данными не превышает 256 ячеек, то вся адресация в ней заведомо может быть выполнена как прямая к текущей странице. Это важно для достижения максимального быстродействия.

В адресных командах двойного формата предусмотрен еще один способ прямой адресации, при котором адрес операнда задается в 15 младших разрядах второго слова команды (дополнительная ячейка), в нулевом разряде которого в этом случае записывается 0.

Косвенная адресация. При косвенной адресации доступны все ячейки работающего раздела за счет использования дополнительных ячеек по одной или более на каждую адресную команду.

Признаками косвенной адресации являются 1 в нулевом разряде команды и 0 в пятом или шестом разрядах. Адрес дополнительной ячейки (адрес адреса) определяется как при прямой адресации. Содержимое 15 младших разрядов этой ячейки будет использовано схемами процессора в качестве адреса операнда, *если в ее нулевом разряде содержится 0.* В этом случае говорят об одноуровневой косвенной адресации.

Если же в нулевом разряде адресуемой ячейки окажется 1, то логическими схемами процессора анализ будет *продолжен.*

Рассмотрим, например, команду в ячейке с адресом 10000 в табл. 2.1. Единица в нулевом разряде и нуль в пятом определяют, что это косвенная адресация через нулевую страницу. Содержимое разрядов 6–15 указывает на 243-ю ячейку. Если в нулевом разряде этой ячейки окажется 0, то 15 младших разрядов будут использованы в качестве искомого относительного адреса операнда. Пусть не так. Предложим, что в 243-й ячейке записан код 177775, тогда следующим "кандидатом" на искомый адрес будет содержимое 15 младших разрядов ячейки 77775, и так как в табл. 2.1 в этой ячейке записан код 071620 (нуль в нулевом разряде), то адресом операнда будет 71620. Это пример двухуровневой косвенной адресации.

В адресных командах двойного формата возможна косвенная адресация с началом цепочки дополнительных ячеек во втором слове команды (признак – 1 в нулевом разряде второго слова).

При косвенной адресации анализ цепочки дополнительных ячеек ведется до тех пор, пока не будет обнаружена ячейка с 0 в нулевом разряде. Уровень косвенной адресации в приципе не ограничен, но чаще других используются одно- и двухуровневая косвенные адресации как одно из основных средств передачи адресов исходных данных при обращении к подпрограммам на языке МНЕМОКОД. Если из-за большой длины цепочки косвенной адресации команда не будет выполнена за отведенное ей время, сработают схемы контроля процессора и выполнение программы будет прервано с выдачей сообщения на пульт оператора.

Адресация через индексные регистры. Шестнадцатиразрядные индексные регистры (см. функцию 5.1 в табл. 2.4) используются тогда, когда в программе необходимо обеспечить *локальное* базирование адресов, например, при обработке таблиц или массивов. В этом случае в 15 младших разрядов индексного регистра записывают адрес первой ячейки таблицы или массива, а в самой команде указывают *смещение* относительно этого адреса.

Признаком адресации через индексный регистр являются единицы в пятом и шестом разрядах команды. При этом в нулевом разряде указывается, какой из двух равнозначных индексных регистров должен быть использован для формирования адреса операнда: 0 – *РИ1*, 1 – *РИ2.*

Существуют два способа адресации через индексные регистры: *индексация* и *автоиндексация.* Индексация применяется для обработки таблиц, автоиндексация – массивов.

Так как адрес в индексном регистре занимает 15 младших разрядов, то в его нулевой разряд вынесен признак способа: 0 в нулевом разряде используемого регистра задает режим индексации, 1 – режим автоиндексации.

При индексации 9 младших разрядов команды рассматриваются как положительное смещение операнда относительно адреса, записанного на индексном регистре. Адрес операнда определяется суммированием содержимого индексного регистра и положительного смещения, при этом содержимое индексного регистра не изменяется.

При автоиндексации смещение записывается в дополнительном коде и может быть как положительным, так и отрицательным (знак смещения в седьмом разряде команды). В отличие от индексации здесь относительный адрес операнда совпадает с содержимым 15 младших разрядов соответствующего индексного регистра, которое на заключительной фазе выполнения команды изменяется на величину смещения, настраиваясь тем самым на адрес следующего по логике обработки операнда. Применяя режим автоиндексации "в цикле", можно обрабатывать элементы массива с произвольными шагом и знаком изменения индекса.

Для примера обратимся к ячейке 10003 из табл. 2.1. Если на *РНК* попадет адрес этой ячейки, то схемами процессора ее содержимое будет воспринято как адресная команда с адресацией через индексный регистр (11_2 в пятом и шестом разрядах), причем через *РИ1* (0 – в нулевом). Какой способ адресации – индексация или автоиндексация – будет установлен и как будет восприниматься смещение – положительным или со знаком, – определяется нулевым разрядом *РИ1*. Рассмотрим два варианта.

1. На *РИ1* записан код 071250. Нуль в нулевом разряде этого кода определяет режим индексации. Значит, содержимое 9 младших разрядов ячейки 10003 (т.е. 725) будет воспринято как положительное смещение, относительный адрес операнда примет значение 71250 + 725 = 72175, значение индексного регистра не изменится.

2. На *РИ1* записан код 171250. Устанавливается режим автоиндексации. В качестве адреса операнда используется содержимое 15 младших разрядов *РИ1*. Значит 9 младших разрядов ячейки 10003 в дополнительном коде дают смещение минус 53. Индексный регистр принимает значение 171250 – 53 = 171175, так что при следующем обращении к команде в качестве операнда будет использоваться содержимое ячейки с относительным адресом 71175.

Все описанные выше способы адресации и их признаки сведены в табл. 3.1.

Остановимся на особенностях применения автоиндексации в программах операционной системы, т.е. в привилегированном режиме работы процессора.

В целях обеспечения механизма доступа программ из нулевого раздела к любой из 128 К ячеек ОЗУ в логических схемах процессора предус-

Таблица 3.1

Способ адресации	Признак способа адресации				
	Разряды команды			Нулевой разряд *РИ*	
	0	5	6	*РИ1*	*РИ2*
Прямая адресация к нулевой странице	0	0	*	*	*
Прямая адресация к текущей странице	0	1	0	*	*
Косвенная адресация через нулевую страницу	1	0	*	*	*
Косвенная адресация через текущую страницу	1	1	0	*	*
Автоиндексация с использованием *РИ1*	0	1	1	1	*
Автоиндексация с использованием *РИ2*	1	1	1	*	1
Индексация с использованием *РИ1*	0	1	1	0	*
Индексация с использованием *РИ2*	1	1	1	*	0

Примечание. * означает, что соответствующий разряд в состав признака не входит.

мотрено, что в привилегированном режиме абсолютный адрес операнда при автоиндексации определяется по *РБ1*, при других способах адресации он устанавливается по *РБ2* (см. § 2.3). Таким образом, при необходимости обратиться в программе операционной системы к ячейке ненулевого раздела вначале устанавливают соответствующее значение *РБ1*, затем на индексный регистр заносят относительный адрес нужной ячейки с 1 в нулевом разряде, определяя тем самым режим автоиндексации, и только потом применяют адресную команду с адресацией через индексный регистр. Если же автоиндексация требуется внутри нулевого раздела по ее прямому назначению, предварительно необходимо на *РБ1* установить значение базы нулевого раздела.

3.2. Однословные адресные команды

Однословные адресные команды, или адресные команды основного формата, представлены в табл. 3.2.

Хотя коды операций этих команд занимают 4 разряда, всего имеется 14, а не $16 = 2^4$, как можно было ожидать, различных КОП, так как первые три разряда КОП в адресных командах основного формата не принимают одновременно значение 0. Именно по этому признаку дешифратор команд отличает эти команды от команд других групп.

Мнемонические обозначения кодов операций происходят от соответствующих английских терминов, а также обозначений задействованных аккумуляторов (см. функцию 2.1, табл. 2.4); **ADA** от английского *add to A* – сложить с *РА*, **STB** от *store B* – занести содержимое РБ (в ячейку памяти).

Таблица 3.2

Наименование команды	Код операции (1-й–4-й разряды команды)	Мнемоника	Время выполнения, мкс
Сложение с *РА*	1000	**ADA**	2,0
Сложение с *РБ*	1001	**ADB**	2,0
Неэквивалентность	0100	**XOR**	2,0
Конъюнкция	0010	**AND**	2,0
Дизъюнкция	0110	**IOR**	2,0
Засылка на *РА*	1100	**LDA**	2,0
Засылка на *РБ*	1101	**LDB**	2,0
Безусловный переход	0101	**JMP**	1,7
Сравнение с *РА*	1010	**CPA**	2,8
Сравнение с *РБ*	1011	**CPB**	2,8
Запись содержимого *РА* в память	1110	**STA**	2,8
Запись содержимого *РБ* в память	1111	**STB**	2,8
Переход на подпрограмму	0011	**JSB**	3,8
Счет и пропуск	0111	**ISZ**	3,9

Примечание. **Время выполнения дано для прямой адресации;** при одноуровневой косвенной адресации время увеличивается на 1,3 мкс, при индексации и автоиндексации – на 1,0 мкс.

Для описания *семантики,* т.е. смысла, значения, машинных команд будем использовать следующие обозначения:

(M) – содержимое ячейки (регистра) M;

:= – операция занесения информации;

⟨=⟩ – эквивалентно.

С учетом этих обозначений семантика команды сложения записывается так:

$$\textbf{ADA}\ M \langle = \rangle\ PA := (PA) + (M) \tag{3.1}$$

что четко определяет результат суммирования и формируется на аккумуляторе *РА.* (Здесь и далее при описании семантики адресных команд через M обозначается относительный адрес адресуемой ячейки.)

Для правильного понимания дальнейшего важно четко представлять себе, что с каждой ячейкой памяти связано два объекта машинных операций: *адрес* и *содержимое* – M и (M). Операция

$$PA := (M) + 1 \tag{3.2}$$

должна быть прочитана как *заслать на регистр А* увеличенное на 1 *содержимое ячейки М,* в то время как операция

$$PA := M + 1 \tag{3.3}$$

означает засылку на РА адреса следующей за M ячейки, а операция

$$(M) := (PA) \tag{3.4}$$

имеет смысл записи содержимого *РА* в ячейку, адрес которой содержится в ячейке *М*.

Полная семантика, строго говоря, должна включать также операцию передачи управления следующей команде:

$$РНК := (РНК) + w, \tag{3.5}$$

где w – количество слов в описываемой команде (для однословных команд w = 1).

Но именно потому, что эта операция обязательна, ее можно подразумевать по умолчанию. Будем говорить об изменении *РНК* только в тех случаях, когда команда нарушает естественный порядок выполнения программы.

Описание команд, необходимое для практического программирования на МНЕМОКОДе включают формальную семантику и ряд дополнительных пояснений.

Сложение. Из табл. 3.2 видно, что команда сложения с регистром *А*, как и ряд других команд, имеет аналог для регистра *Б*. Семантику таких "парных" команд удобно описывать совместно:

$$\mathbf{ADA/B}\ M \Leftarrow РА/Б := (РА/Б) + (M). \tag{3.1a}$$

Содержимое ячейки *М* не изменяется (как и во всех других операциях, если не оговорено противное).

При выполнении команды возможен перенос из нулевого разряда *РА/Б* в *регистр расширения РР* (см. функцию 3, табл. 2.4). Если перенос в *РР* не совпадает с переносом в нулевой разряд, то *регистр переполнения РП* устанавливается в 1 (см. функцию 4, табл. 2.4). Читатель может убедиться, что перенос в *РР* не всегда означает переполнение. Это можно увидеть на примере операции (–1) + (–1). (Напомним, что машинный код минус 1 равен 177777.) Другое дело – операция 32767 + 1, при которой будет иметь место перенос в нулевой разряд без переноса в РР, т.е. будет иметь место переполнение:

$$РП := 1. \tag{3.6}$$

На основе операции сложения в СМ-2М реализованы все другие арифметические операции.

Неэквивалентность. Семантика команды:

$$\mathbf{XOR}\ M \Leftarrow РА := (РА) \oplus (M). \tag{3.7}$$

Это одна из трех *логических* операций, реализованных в системе команд СМ-2М. Логические операции являются поразрядными, т.е. определяются для пары двоичных разрядов, а осуществляются над всеми одноименными разрядами *РА* и *М* одновременно. Определение операции неэквивалентности таково: двоичные разряды d и d' в результате данной операции дадут 0, если они равны между собой, и 1 – в противном случае.

Аналогичной команды с *РБ* для данной и других логических операций нет. Это единственная причина, по которой *РА* и *РБ* нельзя считать равнозначными.

Конъюнкция – логическое поразрядное умножение:

$$\text{AND } M \langle = \rangle PA := (PA) \cap (M). \tag{3.8}$$

В соответствии с правилами булевой алгебры пара единиц в произведении дает 1, все другие комбинации двоичных разрядов дают 0.

Дизъюнкция – логическое поразрядное сложение:

$$\text{IOR } M \langle = \rangle PA := (PA) \cup (M). \tag{3.9}$$

В соответствии с правилами булевой алгебры сумма двух нулей даст 0, другие комбинации двоичных разрядов дадут 1.

Засылка на аккумулятор. Семантика команды:

$$\text{LDA/B } M \langle = \rangle PA/Б := (M). \tag{3.10}$$

Необходимо помнить, что это копирование информации, так как содержимое ячейки *М* сохраняется. Предварительной очистки аккумулятора не требуется. Естественно, что старое содержимое аккумулятора при выполнении команды стирается.

Безусловный переход

$$\text{JMP } M \langle = \rangle РНК := M. \tag{3.11}$$

По данной команде в регистр номера команды засылается не содержимое, а относительный адрес ячейки *М*. Команда применяется тогда, когда необходимо изменить естественную последовательность выполнения программы, и является одной из команд *передачи управления*.

Для команды **JMP** за ненадобностью не реализована адресация через индексные регистры. Попытка выполнить автоиндексацию или индексацию в сочетании с КОП 0101_2 (табл. 3.2) приведет к прерыванию программы от схем контроля процессора.

В привилегированном режиме команда **JMP** наряду с операцией (3.11) вызывает изменение содержимого *РБ2*, если новое значение базы привилегированного состояния было предварительно подготовлено специальной командой **LBP** (см. § 3.3) на буферном регистре этой базы:

$$РБ2 := (\text{буферный } РБ2). \tag{3.12}$$

Другими словами, команда **JMP** *М* в нулевом разделе может означать переход к новому разделу памяти с передачей ему функций раздела операционной системы и привилегированного режима работы. Таким образом, в архитектуру процессора СМ-2М заложена возможность расширения памяти для задач, требующих привилегированного режима, до двух кубов и более.

Сравнение с аккумулятором:

$$\text{CPA/B } M \langle = \rangle \text{ если } (PA/Б) \neq (M) \text{ то ПСК}. \tag{3.13}$$

Здесь используется аббревиатура ПСК - *пропуск следующей команды*. Семантика, заданная операцией (3.13), показывает, что команды **CPA** и **CPB** являются командами *условного пропуска:* если двоичные коды в аккумуляторе и в ячейке *M* не совпадают, следующая команда пропускается. Условный пропуск (чаще всего команды **JMP**) представляет собой основное средство программирования условий на машинно-ориентированных языках. Поэтому операция ПСК входит в семантику многих команд СМ-2М.

Для того чтобы не допускать ошибок при практическом применении таких команд, следует помнить, что в архитектуре СМ-2М ПСК всегда означает пропуск одной ячейки:

$$\text{ПСК} \ \langle = \rangle \ \mathit{РНК} := (\mathit{РНК}) + 2. \tag{3.14}$$

*Поэтому после команд, осуществляющих условный или безусловный ПСК, можно располагать только одно*словные *команды.*

Запись в память:

$$\textbf{STA/B} \ M \ \langle = \rangle \ M := (\mathit{РА/Б}). \tag{3.15}$$

Здесь, как и в командах **LDA** и **LDB**, происходит копирование информации, поэтому содержимое аккумулятора сохраняется, а адресуемой ячейки стирается.

Переход на подпрограмму:

$$\textbf{JSB} \ M \ \langle = \rangle \ M := (\mathit{РНК}) + 1; \tag{3.16}$$

$$\mathit{РАВ} := (\mathit{РНК}) + 1; \tag{3.17}$$

$$\mathit{РНК} := M + 1. \tag{3.18}$$

Другими словами, семантика команды **JSB** состоит в последовательном выполнении трех операций. Операции (3.16) и (3.17) означают, что по адресу *M* и в *регистр адреса возврата РАВ* (см. функцию 9, табл. 2.4) заносится текущее значение *РНК* плюс 1, т.е. адрес ячейки, следующей за командой **JSB**, а операция (3.18) [ср. с операцией (3.3)] приводит к передаче управления на ячейку *M* + 1, в которой должна быть записана первая команда подпрограммы. Содержимое ячейки *M* (рабочей ячейки подпрограммы) и регистра адреса возврата используется для *выхода* из подпрограммы, например, с помощью операции

$$\mathit{РНК} := (M), \tag{3.19}$$

а также для передачи в подпрограмму адресов фактических параметров.

Отметим, что если команда **JSB** *M* находится в *ячейке прерывания,* т.е. получает управление в результате прерывания по запросу УВВ или схем контроля процессора, то по адресу *M* и в *РАВ* записывается адрес продолжения прерванной программы, но не адрес ячейки, следующей за ячейкой прерывания, как это можно было ожидать в соответствии с (3.16) и (3.17).

Команда **JSB**, так же как и команда **JMP**, не работает с адресацией через индексные регистры и, так же как и команда **JMP**, может вызывать операцию (3.12).

Счет и пропуск:

$$\text{ISZ } M \langle = \rangle \quad M := (M) + 1; \tag{3.20}$$

$$\text{если } (M) = 0 \text{ то ПСК.} \tag{3.21}$$

Эта команда используется для организации *счетчиков*. Если первоначально в ячейке M содержится число минус n, то условие пропуска следующей команды $(M) = 0$ наступит после того, как команда **ISZ** M будет выполнена n раз.

3.3. Двухсловные адресные команды

Двухсловные адресные команды были введены в систему команд ЭВМ архитектурной линии СМ-2М в развитие группы адресных команд основного формата, причем, если на М-6000 было реализовано всего четыре такие команды (умножения и деления целых чисел, а также пересылки слов двойной длины), то в СМ-2М их насчитывается более полутора десятков.

Формат двухсловных адресных команд представлен на рис. 3.2. Младший разряд первого слова команды – *ключ Д* – определяет, каким из двух способов будет формироваться относительный адрес операнда.

При $Д = 0$ анализируется старший разряд второго слова. Если он в свою очередь окажется равен 0, то 15 младших разрядов будут использованы в качестве относительного адреса операнда (прямая адресация к произвольной ячейке памяти раздела). По-другому будет восприниматься второе слово адресной команды, если в его старшем разряде будет записана 1. В этом случае логическими схемами процессора эта ячейка будет рассматриваться как начало цепочки косвенной адресации (см. § 3.1).

При $Д = 1$ второе слово анализируется как адресная команда основного формата. Способ адресации устанавливается в соответствии с табл. 3.1, содержимое разрядов 1–4 (т.е. тех, в которых задается КОП адресной команды основного формата) игнорируется. Таким образом, обеспечивается возможность применять в двухсловных адресных командах все ранее описанные способы адресации.

Группа двухсловных адресных команд представлена в табл. 3.3.

Отметим, что в таблице представлены не все команды, имеющие формат двухсловной адресной команды. Этот же формат имеет целый ряд команд, относящихся к командам дополнительного набора. Впрочем, само деление на команды основного и дополнительного наборов достаточно условно.

Операции со словами двойной длины. Как уже упоминалось, словами двойной длины кодируются целые числа двойного формата и вещест-

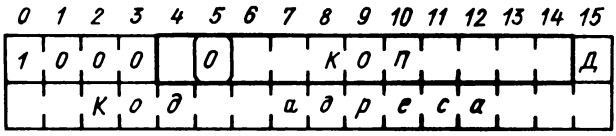


Рис. 3.2. Формат двухсловной адресной команды:
Д – ключ к коду адреса операнда

венные числа, представляемые мантиссой и порядком (*числа с плавающей запятой*). В памяти каждое такое число хранится в двух ячейках со смежными номерами *М* и *М* + 1. Поэтому, например, для хранения массива из *n* вещественных чисел требуется 2*n* ячеек памяти.

Имеются две команды для "перемещения" слов двойной длины.

Засылка двойного слова на аккумуляторы:

$$\text{DLD } M \langle = \rangle \; РА := (M); \tag{3.22}$$

$$РБ := (M + 1); \tag{3.23}$$

$$РНК := (РНК) + 2. \tag{3.24}$$

Последняя операция означает просто переход к следующей команде и в дальнейшем будет опускаться, поскольку является обязательной для всех адресных команд двойного формата.

Таблица 3.3

Наименование команды	Мнемоника	КОП (восьмеричное представление 4-го, 6–14-го разрядов	Время выполнения, мкс
Засылка двойного слова на *РБА*	DLD	1100	8,0
Запись двойного слова в память	DST	1200	7,1
Умножение целых чисел	MPY	0100	9,2
Деление целых чисел	DIV	0200	14,2
Засылка на *РИ1*	LDX	1104	6,0
Засылка на *РИ2*	LDY	1105	6,0
Запись *РИ1* в память	STX	1204	5,2
Запись *РИ2* в память	STY	1205	5,2
Запись ССП в память	STW	1206	7,6
Восстановление ССП	LDW	1106	10,6
Засылка на *РБ1*	LBN	1101	6,3
Засылка на *РБ2*	LBP	1102	6,6
Сложение вещественных чисел	FAD	1400	21,9
Вычитание вещественных чисел	FSB	1410	22,6
Умножение вещественных чисел	FMP	1420	19,8
Деление вещественных чисел	FDV	1430	35,1
Проверка и изменение (семафор)	TAS	1107	6,8

Запись двойного слова в память:

$$\textbf{DST}\quad M\ \langle=\rangle\quad M := (PA); \tag{3.25}$$

$$M+1 := (PБ). \tag{3.26}$$

Естественно, что команды **DLD** и **DST** как команды перезаписи, но не преобразования информации, являются в некотором смысле вспомогательными. В частности, они используются совместно с командами умножения и деления целых чисел.

Команда умножения MPY *M* осуществляет умножение целых чисел, одно из которых находится в ячейке *M*, а другое – в регистре *A*. Тридцатидвухразрядный результат – целое число двойной длины – размещается на двух программных регистрах *РА* и *РБ*, причем старшая часть числа – в регистре *РБ* (знак в старшем разряде этого регистра), младшая часть – в регистре *РА*. Здесь регистры *РБ* и *РА* удобно рассматривать как один 32-разрядный регистр и обозначать через *РБА*.

С учетом этого соглашения семантика команды **MPY** *M* будет иметь следующий вид:

$$\textbf{MPY}\quad M\ \langle=\rangle\quad РБА := (PA)\cdot(M). \tag{3.27}$$

Нетрудно убедиться, что при таком определении операция **MPY** не может привести к переполнению *РБА*.

Команда деления DIV *M* осуществляет операцию, обратную умножению: целое число двойной длины, записанное на *РБА*, делится на (M), частное от деления помещается на *РА*, остаток – на *РБ*:

$$\textbf{DIV}\quad M\ \langle=\rangle\quad PA := \text{целая часть}\,[(РБА):(M)_1]; \tag{3.28}$$

$$РБ := \text{остаток}\,[(РБА):(M)_1]. \tag{3.29}$$

Если частное выходит за диапазон 16-разрядных целых чисел, регистр переполнения *РП* устанавливается в 1, и результат операции не определен. В остальных случаях в *РП* записывается 0. Если делитель (M) окажется равным нулю, операция деления не выполняется, $(РБА)$ не изменяется, но в *РП* записывается 1.

Помимо описанных операций в группе регистровых команд СМ-2М реализованы различные сдвиги слов двойной длины, записанных на *РБА*. В их число входят, в частности, арифметические сдвиги влево **ASL** *n* и вправо **ASR** *n*, которые эквивалентны соответственно умножению и делению $(РБА)$ на 2^n, причем величина сдвига *n* кодируется непосредственно в формате команды. Подробнее эти операции рассмотрим вместе с соответствующими операторами МНЕМОКОДа в § 6.3.

Операции пересылок память – индексный регистр – поддержаны парными командами **LDX, LDY** (пересылки из памяти в регистры) и **STX, STY** (пересылка из регистров в память).

Засылка на индексный регистр:

$$\textbf{LDX/Y}\ M\ \langle=\rangle\ РИ1/2 := (M). \tag{3.30}$$

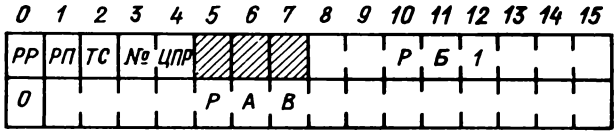


Рис. 3.3. Структура слова состояния программы. В 3-м и 4-м разрядах 1-й ячейки фиксируется номер процессора, выполняющего программу: 00 для ЦПР1 и 11 для ЦПР2; разряды с 5-го по 7-й не используются

Запись содержимого индексного регистра в память:

$$\text{STX/Y}\ \ M \Leftarrow \quad M := (РИ1/2). \tag{3.31}$$

Запись и восстановление ССП. *Слово состояния программы* (ССП), структура которого показана на рис. 3.3, физически "в сборе" начинает существовать только после того, как будет записано в две ячейки памяти командой **STW**. До этого оно существует в виде значения отдельных регистров и триггеров.

Команда записи ССП:

$$\text{STW}\ \ M \Leftarrow \quad M := \text{первое слово ССП;} \tag{3.32}$$

$$M + 1 := \text{второе слово ССП.} \tag{3.33}$$

Эта команда должна быть первой командой *реентерабельной подпрограммы* – автономного, часто используемого программного модуля, один экземпляр которого способен обслужить несколько задач, работающих одновременно. (Подробнее об организации подпрограмм см. в следующем параграфе.)

Операции, обратные операциям (3.32), (3.33), осуществляются по команде **засылка ССП:**

$$\text{LDW}\ \ M \Leftarrow \quad РР := (\text{нулевой разряд } M); \tag{3.34}$$

$$РП := (\text{первый разряд } M); \tag{3.35}$$

$$ТС := (\text{второй разряд } M); \tag{3.36}$$

$$РБ1 := (8\text{–}15\text{-й разряды } M); \tag{3.37}$$

$$РНК := (M + 1). \tag{3.38}$$

В реентерабельных подпрограммах эта команда должна быть последней. Она восстанавливает (*РР*) и (*РП*) [операции (3.34), (3.35)] и передает управление задаче, обратившейся к этой подпрограмме [операция (3.38)]. Применяя эту команду в непривилегированном режиме, нельзя допускать изменения содержимого ячейки *M*, сформированного командой **STW**, иначе последствия операций (3.36), (3.37) будут непредсказуемыми.

Напротив, в программах нулевого раздела, работающих в привилегированном режиме, в первую очередь в операционных системах, коман-

да **LDW** используется для передачи управления в любую точку произвольного раздела памяти. С этой целью в двух смежных ячейках необходимо подготовить нужную базу *РБ1* в ячейке *М* и адрес передачи управления относительно этой базы в ячейке *М* + 1. Кроме того, во втором разряде первой ячейки ССП должна быть записана 1, чтобы командой **LDW** не только сформировать нужное содержимое *РБ1* и *РНК*, но и перевести процессор в непривилегированное состояние операцией (3.36). Это единственный способ передачи управления задачам пользователя из ОС.

Формирование базовых регистров. Помимо команды **LDW** содержимое базы непривилегированного состояния может быть сформировано специальной командой **засылка на** *РБ1*:

LBN *М* ⟨=⟩ *РБ1* := (8–15-й разряды *М*). (3.39)

Команда относится к числу привилегированных и применяется в тех случаях, когда из нулевого раздела нужно без передачи управления обратиться к ячейке памяти другого раздела. Напомним (см. § 3.1), что адресные команды с признаком автоиндексации всегда, в том числе и в нулевом разделе, выполняются по *РБ1*.

Формирование новой базы привилегированного состояния осуществляется командой **засылка на** *РБ2*:

LBP *М* ⟨=⟩ буферный *РБ2* := (8, 9-й разряды *М*). (3.40)

Эта команда используется при необходимости передать функции ОС другому разделу, например резервному кубу памяти при отказе основного. В (3.40) подчеркнуто, что команда **LBP** не изменяет *РБ2* непосредственно, а только готовит новое значение базы привилегированного состояния на специальном буферном регистре. Фактический переход к новой базе осуществляется при выполнении первой после **LBP** команды передачи управления **JMP** или **JSB** [см. (3.12)].

Команды вещественной арифметики. Команды вещественной арифметики очень естественны. Они реализуют всем известные арифметические операции: сложения – команда **FAD**, вычитания – команда **FSB**, умножения – **FMP** и деления – **FDV** вещественных чисел, формат которых ранее был описан в § 2.1.

Все команды вещественной арифметики организованы одинаково. Первый операнд должен находиться на объединении аккумуляторов *РА* и *РБ*, причем на *РА* записывают старшие разряды мантиссы, а на *РБ* – младшие разряды и порядок. Второй операнд должен храниться в ячейках *М* и *М* + 1, результат формируется снова на *РА* и *БР*, так что значение первого операнда стирается.

При выполнении команд рассматриваемой группы возможно переполнение, если результат выходит за диапазон представления вещественных чисел, заданный неравенством (2.7). В этом случае регистр переполнения *РП* устанавливается в 1. При делении на нуль *РП* также уста-

навливается в 1, а на аккумуляторах записывается число, максимальное по абсолютному значению и совпадающее по знаку с делимым.

TAS-семафор. Появление ЭВМ с двухпроцессорной архитектурой породило проблему синхронизации работы процессоров при параллельном выполнении управляющих программ с общими полем памяти и (или) объектом управления.

Суть проблемы синхронизации сводится к организации обмена сообщениями о событиях *завершил опрос датчиков, провел анализ, выдал воздействие* и т.п. между двумя процессорами и к программному механизму перевода процессора в состояние ожидания того или иного события.

Все это довольно изящно реализуется с применением команды **TAS**, которую называют *проверкой и изменением* (иногда *анализом и пропуском*), но точнее, на наш взгляд, называть ее *команда-семафор.*

Формальная семантика команды такова:

TAS M ⟨⇒ **если** (0-й разряд M) = 0 **то** 0-й разряд M := 1 и ПСК (3.41)
иначе перейти к следующей команде.

Чтобы разобраться, как с помощью этой команды организовать ожидание и прием информации о событиях, возникающих в задачах на смежном процессоре, необходимо рассмотреть практические приемы программирования, т.е. *прагматику* системы команд, определяющей архитектуру ЦПР как бы в динамике.

3.4. Приемы программирования

Наряду с обозначениями, принятыми в § 3.2 для описания семантики адресных команд, в данном параграфе нам понадобятся некоторые изобразительные средства базового языка программирования МНЕМОКОД. Подробное описание этого языка будет дано во втором разделе книги. Здесь же нужно обратить внимание на следующее.

Программы на МНЕМОКОДе записываются на бланке определенного формата (рис. 3.4).

На каждой строке этого бланка размещают один оператор МНЕМОКОДа. В данном параграфе будем использовать операторы двух

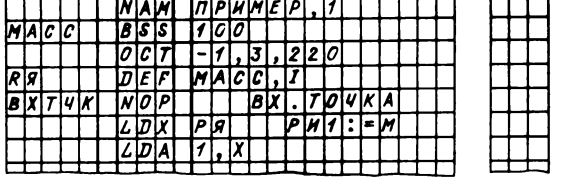


Рис. 3.4. Бланк МНЕМОКОДа с фрагментом программы

типов – *операторы машинных команд,* каждый из которых представляет собой символическую запись соответствующей машинной команды, и *операторы псевдокоманд* резервирования памяти и определения констант, которые не имеют эквивалента в виде машинных команд, а представляют собой инструкции транслятору с МНЕМОКОДа зарезервировать определенное число ячеек памяти, возможно, с записью в них тех или иных констант. На одной строке бланка может быть записано до семидесяти двух символов, которые разбиты на следующие поля: *метки, кода операции* (КОП), *операнда и комментария.*

В поле метки записываются символические адреса ячеек памяти: *М, М1, РЯ* и т.п. Максимальная длина метки – пять символов. Если в других операторах нет ссылок на соответствующую ячейку, поле метки может быть *пустым.*

В поле кода операции указывается либо мнемоническое обозначение машинной команды (для адресных команд см. табл. 3.2 и 3.3.), либо мнемоника той или иной инструкции (псевдокоманды). Естественно, что это поле должно быть всегда заполнено.

За кодом операции следует поле операнда. Оно обязательно в операторах адресных команд и псевдокоманд резервирования памяти и отсутствует за ненадобностью в операторах большинства регистровых команд.

В поле комментария располагаются поясняющие записи, которые адресованы человеку и машиной не обрабатываются.

В качестве операнда в операторах адресных команд будем использовать:

различные метки, задающие *прямую адресацию* к соответствующей ячейке. Например, **JMP** *М*, **LDA** *N1* и т.п.;

указатели адресации через *индексные регистры,* состоящие из смещения (целого числа, возможно, со знаком) и обозначения (**X** или **Y**) задействованного регистра. Например, оператор **ADA** *1*, **X** будет изображать команду сложения с адресацией через *РИ1* и смещением *1*;

метки с признаком *косвенной адресации,* в качестве которого в МНЕМОКОДе принят символ **I**. Так, оператор **STB** *М,* **I** соответствует команде записи (*РБ*) в ячейку ОЗУ, адрес которой определяется цепочкой косвенной адресации с началом в ячейке *М*;

вычисляемые адреса, или, иначе, *выражения.* Например, *М + 1* – адрес ячейки, следующей за ячейкой *М*, * – *1* – адрес ячейки, предшествующей данной (в МНЕМОКОДе символ * используется в поле операнда как указатель текущего адреса).

Из псевдокоманд на первых порах понадобятся:

псевдокоманда *резервирования* памяти с мнемоникой **BSS**. В поле операнда этой псевдокоманды указывается число резервируемых ячеек – целое десятичное или восьмеричное [восьмеричные числа отличает символ **B** (латинское), следующий за последней цифрой: *10***B**, *123***B** и т. д.]. Метка, если она есть, относится к первой резервируемой ячейке;

псевдокоманда *резервирования адресных констант* **DEF**, которая резервирует одну ячейку с записью в нее адреса, заданного в поле операнда. Например, псевдокоманда **DEF** *M* означает, что вместо нее в программе после обработки транслятором с МНЕМОКОДа появится ячейка с нулем в нулевом разряде и с относительным адресом ячейки *M* в разрядах 1–15, а псевдокоманда **DEF** *M*,**I** приведет к формированию той же адресной константы, но с единицей в нулевом разряде. Такие адресные константы используются в основном для передачи адресов фактических параметров в подпрограммы, а также для формирования содержимого индексных регистров;

псевдокоманды *резервирования восьмеричных и десятичных констант* (мнемоника **OCT** и **DEC** соответственно). Каждая такая псевдокоманда резервирует столько ячеек, сколько констант указано в поле операнда. Например, псевдокоманда **OCT** *123* резервирует одну ячейку и записывает в нее число *123***B**, а псевдокоманда **DEC** *10,20,*–1 резервирует три ячейки и записывает в первую из них 10, во вторую 20 и в третью – дополнительный код минус единицы.

Многие операции с (*РА*) и (*РБ*) удобнее всего задавать операторами *регистровых команд* (подробнее их рассмотрим в следующем разделе). При описании приемов программирования будем использовать лишь некоторые из них:

CLA – сброс (*PA*) $\Leftrightarrow$ *PA* := 0

INA – инкремент (*PA*) $\Leftrightarrow$ *PA* := (*PA*) + 1

CMA – инверсия (*PA*) $\Leftrightarrow$ *PA* := (*PA*) $\oplus$ 177777,

а также их комбинации, кодируемые в одной ячейке:

CLA,INA $\Leftrightarrow$ *PA* := 1

CMA,INA $\Leftrightarrow$ *PA* := – (*PA*)

Аналогичные команды (**CLB,INB** и т.д.) имеются для регистра *Б*. По ходу дела введем еще некоторые регистровые команды.

Излагаемые ниже приемы программирования иллюстрируют реализацию на МНЕМОКОДе *основных алгоритмических структур* [8, 9]: арифметических и логических операций, условных операторов и операторов цикла. Рассмотрены также приемы организации подпрограмм и характерных для двухпроцессорной архитектуры **TAS** семафоров.

Арифметические и логические операции. Проследим основные этапы программирования на простейшем примере операции сложения целых чисел:

$$n = n_1 + n_2 . \qquad (3.42)$$

1. Прежде всего программист выбирает символьные коды адресов ячеек памяти для хранения исходных данных и результатов. В нашем примере естественно в качестве адресов принять *N, N1* и *N2*.

Следует сразу предостеречь от отождествления адреса и содержимого: утверждение $N = n$ неверно, правильным будет равенство $(N) = n$ (ячейка с адресом N содержит значение переменной n).

В каждой операции различают *исходные данные и результат.* Не отвлекаясь на детали ввода-вывода исходных данных (подробно см. в гл. 8 и 10), предполагаем, что к моменту выполнения операции исходные данные уже находятся в нужных ячейках памяти и размещению подлежит только результат операции.

2. Следующий этап – "расщепление" программируемой операции на элементарные инструкции в соответствии с используемой системой команд. Всякая *бинарная* операция типа (3.42) (т.е. операция, которая *двум* операндам ставит в соответствие *один* результат) на одноадресной машине расщепляется на:

засылку одного из операндов на аккумулятор;

собственно операцию, где один из операндов находится на аккумуляторе, а другой – в памяти;

запись результата в память.

Таким образом, операция (3.42) реализуется тремя операторами:

```
LDA  N1 ⟨⇒   PA := (N1)                    (3.43)
ADA  N2 ⟨⇒   PA := (PA) + (N2)
STA  N  ⇔    N := (PA).
```

Решение будет полным, если "программу" (3.43) снабдить инструкциями *о распределении памяти* (в поле комментария в угловых скобках указывается, что именно записано в той или иной ячейке):

```
N1      BSS  1   ⟨n1⟩                      (3.44)
N2      BSS  1   ⟨n2⟩
N       BSS  1   ⟨n⟩.
```

Операция *вычитания* сводится к сложению:

$$n = n_2 + (-n_1). \qquad (3.45)$$

Проще всего поменять знак целого числа инверсией и инкрементированием (см. § 2.1) с применением соответствующих регистровых команд.

Поэтому программа, реализующая вычитание [операцию (3.45)], будет иметь вид

```
LDA  N1    PA :=   (N1)                    (3.46)
CMA,INA    PA :=   -(PA)
ADA  N2    PA :=   (PA) + (N2)
STA  N     N  :=   (PA).
```

Здесь распределение памяти совпадает с (3.43), а знак эквивалентности между командами и комментариями для краткости опущен.

Логические операции. С точки зрения программной реализации они выглядят так же, как и арифметические. Рассмотрим, например, операцию выделения трех младших разрядов из заданного двоичного кода *c* и занесения результата на *РБ*.

В связи с тем что логические операции выполняются только на *РА*, а результат должен быть занесен на *РБ*, необходимо выяснить, каким путем в поле операнда может быть задан *адрес регистра*. В ЭВМ архитектурной линии СМ-2М принято, что для обращения к регистрам *РА*, *РБ*, *РИ1* и *РИ2* в адресной части команды указывается *0*, *1*, *2* и *3* соответственно.

Это соглашение аппаратно реализовано в схемах процессора, поэтому первые четыре ячейки в каждом разделе памяти недоступны программам, хотя физически существуют.

Итак, программа, реализующая требуемую операцию, будет иметь вид

```
C       BSS  1   ⟨c⟩                                  (3.47)
M       OCT  7   ⟨7⟩
BGN     LDA  C   РА := (C)
        AND  M   РА := (РА) ∩ (M)
        STA  1   РБ := (РА).
```

Здесь меткой *BGN* отмечено начало *исполняемой части* программы.

Заметим, что логическое умножение (РА) на записанную по адресу *M* константу, имеющую двоичный код 00...0111, приводит к обнулению всех разрядов аккумулятора, кроме трех младших. Здесь имеем пример *разрядной маски*, применение которой представляет собой довольно распространенный прием программирования.

Программирование условий. В общем случае *условие ветвления* алгоритма может быть выражено условным оператором типа

если $\mathcal{L}$ **то** S1 **иначе** S2, (3.48)

где $\mathcal{L}$ – некоторое логическое выражение, которое может быть истинно или ложно в зависимости от значения входящих в него переменных; S1 и S2 – некоторые операции, в общем случае достаточно сложные (в частности, могут включать другие условные операторы).

Рассмотрим один частный случай оператора (3.48):

если $n_1 = n_2$ **то** S1 **иначе** S2. (3.48а)

Программно этот оператор может быть реализован следующим образом:

```
N1      BSS  1                                        (3.49)
N2      BSS  1
BGN     LDA  N1   РА := (N1)
        CPA  N2   если (РА) ≠ (N2) то ПСК
        JMP  S1   РНК := S1
        JMP  S2   РНК := S2.
```

В (3.49) метки *S1* и *S2* подразумеваются в ненаписанной части программы.

Наряду с командами сравнения **СРА** и **СРВ** для программирования логических выражений удобно использовать регистровые команды *изменений и пропусков*. Приведем наиболее употребительные из них:

SSA ⟨⇒ **если** $(PA) \geqslant 0$ **то** ПСК
SZA ⟨⇒ **если** $(PA) = 0$ **то** ПСК.

При необходимости изменить условие пропуска на противоположное к соответствующей команде добавляется операция **RSS**– так называемый *реверс условия пропуска*, например:

SSA, RSS ⟨⇒ **если** $(PA) < 0$ **то** ПСК.

В тех случаях, когда логическое выражение $\mathcal{L}$ в операторе (3.48) является *составным* и не сводится к одной команде условного пропуска, можно рекомендовать следующий прием.

1. Выражение $\mathcal{L}$ представляется в виде булевой функции простых логических выражений $\mathcal{b}_i$.

2. Для каждого логического выражения $\mathcal{b}_i$ программируется оператор

если $\mathcal{b}_i$ **то** $d_i = 1$ **иначе** $d_i = 0$. (3.50)

Если $\mathcal{b}_i$ сводится к логическому отношению $(PA) < 0$, то значение d_i из оператора (3.50) можно сформировать последовательностью команд

CLB	$PБ := 0$	(3.51)
SSA	**если** $PA \geqslant 0$ **то** ПСК	
INB	$PБ := (PБ) + 1$	
STB DI	$DI := (PБ)$.	

3. Из полученных значений d_i $(i = 1, 2 \ldots)$, используя логические операции **AND** и **IOR**, в соответствии с исходным выражением $\mathcal{L}$ формируется значение *PA*.

4. Нуль на *PA* будет означать, что составное логическое выражение ложно, поэтому необходимый условный пропуск может быть осуществлен командой **SZA**.

Для примера запрограммируем следующий условный оператор:
если $10 < n < 20$ **или** $n < 0$ **то** *n* **увеличить на** 10 (3.52)
иначе *n* **увеличить на** 20.

Представим условное выражение из (3.52) в виде в виде булевой функции:

$$10 < n\,20 \text{ или } n < 0 = (10 - n < 0) \cap (n - 20 < 0) \cup n < 0. \quad (3.53)$$

Теперь можно написать программу, реализующую оператор (3.52). Поскольку предполагается, что первые барьеры читателем уже преодолены, комментарии к этой программе будут менее подробными, однако постараемся сделать их в большей степени отражающими программи-

руемые операции: например, $PA := -n$ вместо $PA := -(PA)$. Кроме того, начинаем использовать *строки комментариев*, признаком которых является символ * в первой позиции.

```
*       Программа                                          (3.54)
*       Реализует условный оператор (3.52).
*       Разнообразие шрифтов, которыми набрана эта и
*       другие программы,служит чисто учебным целям.
*       Реально на устройствах отображения УВК СМ-2М
*       существуют только прямые прописные символы с
*       одинаковой толщиной линий.
N       BSS  1          ⟨n⟩
ДС      DEC  10         ⟨10⟩
ДВ      DEC  20         ⟨20⟩
D1      BSS  1          ⟨d1⟩
D2      BSS  1          ⟨d2⟩
D3      BSS  1          ⟨d3⟩
*       Формирование d1:
BG      LDA  N
        CMA,INA         (PA) := -n
        ADA  ДС         (PA) := -n + 10
        CLB             Здесь
        SSA             формируется d1
        INB             в соответствии
        STB  D1         с (3.51)
*       Формирование d2:
        LDA  ДВ
        CMA,INA         PA := -20
        ADA  N          PA := -20 + n
        CLB             PB := 0
        SSA             если (N) < 20 то
        INB             PB := 1
        STB  D2         D2 := (PB)
*       Формирование d3:
        LDA  N          (PA) := n
        CLB
        SSA
        INB
        STB  D3
*       Формирование PA в соответствии с (3.53):
        LDA  D1         В итоге
        AND  D2         получаем
        IOR  D3         PA := d1 ∩ d2 ∪ d3.
        SZA             Если (3.53) ложь, то
        JMP  S1         пропустить JMP S1.
        JMP  S2
*       Увеличить число n на 10:
S1      LDA  N
        ADA  ДС
        STA  N          N := n + 10
        JMP  ВЫХ
```

```
*       Увеличить число n на 20:
S2      LDA  N
        ADA  ДВ
        STA  N          N:=n + 20
*       Выход
ВЫХ     ...
```

Если читателю понятна эта программа, то он может смело двигаться вперед. Иначе рекомендуется вернуться к началу § 3.4. и проработать его еще раз. Проверьте, насколько отчетливо понятно назначение оператора **JMP** *ВЫХ* (без него после операции S1 всякий раз выполнялась бы операция S2).

Читатель мог обратить внимание на некоторую неоптимальность программы. В виде упражнения попытайтесь сделать ее несколько короче.

Программирование циклов. Для иллюстрации приемов организации циклических процессов рассмотрим простейший циклический алгоритм нахождения максимального элемента в массиве целых чисел.

Пусть массив $m = (m_1, \ldots, m_n)$ содержит не более 1000 элементов. Отметим, что максимальное число элементов должно быть оговорено заранее, так как на МНЕМОКОДе обязательно предварительное резервирование памяти инструкцией **BSS** *n*, в нашем примере **BSS** *1000*. Другими словами, здесь распределение памяти является *статическим*, как в ФОРТРАНе, а не динамическим, как, например, в АЛГОЛе.

Построим алгоритм отыскания максимального элемента в массиве m.

В системе команд СМ-2М для модификации *счетчика циклов* удобно использовать команду **ISZ**, семантика которой задана операциями (3.20) и (3.21). Поэтому начальное значение счетчика полагается равным числу повторений *со знаком минус*. Выбрав для счетчика метку *I*, а для ячейки, в которой в результате работы алгоритма должен оказаться максимальный элемент, – метку *MMAX*, запишем две операции подготовки цикла:

$$MMAX := m_1;\ I := -n. \tag{3.55}$$

Проверка условия окончания может быть выражена с помощью условного оператора типа (3.48):

$$I := (I) + 1;\ \textbf{если}\ (I) \neq 0\ \textbf{то}\ \text{перейти на}\ S \quad \textbf{иначе}\ \text{перейти на}\ ВЫХ, \tag{3.56}$$

где S – метка основной операции цикла – сравнения текущего $(MMAX)$ с очередным элементом массива m; *ВЫХ* – метка выхода после просмотра всего массива.

Основная операция (*тело цикла*) также выражается условным оператором:

$$\textbf{если}\ m_i > (MMAX)\ \textbf{то}\ MMAX := m_i. \tag{3.57}$$

Затем возвращаемся к шагу (3.56) данного алгоритма. Теперь нетрудно разобраться в следующей программе:

(3.58)

```
*       Программа
*       отыскивает максимальный элемент в массиве m
*       и запоминает его в ячейке MMAX
*       Длина массива не более 1000 элементов
*       Распределение памяти:
M       BSS   1000     ⟨m1⟩          Если n < 1000, некоторое
*                      ...           количество ячеек будет
*                      ⟨mn⟩          недоиспользовано
АДР     DEF   M,1                    Резервирование ячейки и
*                                    запись в нее адреса M с
*                                    единицей в нулевом разряде
N       BSS   1        ⟨n⟩
I       BSS   1
РЯ      BSS   1        Рабочая ячейка
MMAX    BSS   1
*       Подготовка цикла (3.55):
BGN     LDX   АДР      РИ1 := M с1 в 0-м разряде
        LDA   1,X      РА := m1 в режиме автоиндексации,
*                      т.е. РИ1 := (РИ1) + 1
        STA   MMAX     MMAX := m1
        LDA   N
        CMA,INA        РА := −n
        STA   I        I := −n
*       Проверка окончания (3.56)
П2      ISZ   I        I := (I) + 1; если (I) = 0 то
        JMP   S        пропуск команды JMP S
        JMP   ВЫХ
*       Тело цикла (3.57):
S       LDA   1,X      РА := ((РИ1)) = mi; РИ1 := (РИ1) + 1
        STA   РЯ       РЯ := mi
        CMA,INA        РА := −mi
        ADA   MMAX     РА := −mi + (MMAX)
        SSA,RSS        если (РА) < 0 то ПСК
        JMP   П2
        LDA   РЯ
        STA   MMAX     MMAX := (РА) := mi
        JMP   П2
*       Выход из цикла
ВЫХ     ...
```

В данной программе используется один из двух индексных регистров. В общем случае два индексных регистра могут обеспечить работу не более чем с двумя переменными индексами.

Поэтому интерес представляет прием программирования операций с массивами без использования индексных регистров. Этот прием основан на косвенной адресации (см. § 3.1):

```
АДР     DEF  M        (M) – адрес                                    (3.59)
РЯ      BSS  1        дополнительная (рабочая) ячейка
*       Подготовка цикла:
        LDA  АДР      РА:=M, но не (M)!
        STA  РЯ       РЯ:=M
        . . .
*       Тело цикла (косвенная адресация через ячейку РЯ):
        LDA  РЯ,I     РА:= ((РЯ)) = (M), т.е. на
*                     РА засылается содержимое
*                     ячейки, адрес которой находится
*                     в рабочей ячейке РЯ
        . . .
        ISZ  РЯ       РЯ:= (РЯ) + 1 – формирование в
                      РЯ адреса следующего элемента массива
*
```

Приведенный прием индексации был основным для М-6000, поскольку эта машина не имела индексных регистров.

Организация подпрограмм. Если условные операторы и операторы цикла являются основными структурами практического программирования, то подпрограммы отражают его модульность.

Технология программирования *сверху вниз* предполагает, что создание сложных многофункциональных программ начинается с разработки и отладки главной программы, в которой все программируемые функции увязаны по входным и выходным параметрам, а переходы к непосредственному выполнению этих функций осуществляются путем вызова соответствующих подпрограмм, причем на первых этапах эти подпрограммы вообще могут не существовать и в процессе отладки главной программы заменяться *заглушками* (см. [10, 11]).

Выбранный в главной программе способ передачи параметров в подпрограмму и из подпрограммы, а также способ перехода на выполнение подпрограммы, т.е. то, что в языке МНЕМОКОД называют *вызывающей последовательностью*, во многом определяют требования к построению подпрограммы.

По сравнению с программами, предназначенными для автономного выполнения, такими, как, например, программа (3.58), подпрограмма имеет ряд особенностей.

Во-первых, в нее в общем случае должны быть включены команды, обрабатывающие вызывающую последовательность и извлекающие из нее всю необходимую информацию о передаваемых параметрах и об адресе возврата в вызывающую программу.

Во-вторых, в ней должен быть реализован механизм возврата, а также механизм восстановления, в случае необходимости, используемых в подпрограмме регистров процессора.

В-третьих, подпрограмма должна допускать *автономную трансляцию*, т.е. в ней не должны использоваться метки, введенные в главной программе (последнее требование хотя и является самым принципиаль-

ным, но может быть несколько ослаблено специальными инструкциями МНЕМОКОДа, о которых будет идти речь в гл. 7).

Попробуем исходя из этих требований преобразовать в подпрограмму программу (3.58).

Вызывающая последовательность должна в этом случае содержать информацию об адресе *M* первого элемента массива в памяти главной программы, число элементов и адрес ячейки *MMAX*, в которой в результате работы подпрограммы должен быть размещен максимальный элемент массива *m*.

Ниже приводится один из возможных вариантов вызывающей последовательности, удовлетворяющей этим требованиям:

```
*       Вызывающая последовательность                              (3.60)
        LDA  N        PA := n – передача параметра через регистр
K       JSB  MAX      Переход на подпрограмму с входной
*                     меткой MAX
        DEF  M,I      Адреса входного M и выходного MMAX параметров
        DEF  MMAX
*       Продолжение главной программы:
ПР      ...
```

Приступая к разработке подпрограммы, необходимо учесть, что в соответствии с семантикой команды **JSB** *MAX* [операция (3.16)] в ячейку *MAX* будет записан адрес *K* + 1, т.е. адрес ячейки, в которой размещен адрес первого элемента массива. Так возникает связующее звено между списком *фактических параметров* **DEF** *M*,**I** и **DEF** *MMAX* в главной программе и ячейками памяти подпрограммы, которая, как уже говорилось, не должна зависеть от адресов фактических параметров.

С учетом указанных обстоятельств может быть написана ориентированная на вызывающую последовательность (3.60) подпрограмма:

```
*       Подпрограмма                                               (3.61)
*       отыскания максимального элемента.
*       Распределение памяти (рабочие ячейки):
АДР     BSS  1        ⟨M⟩
N       BSS  1        ⟨n⟩
I       BSS  1        ⟨i⟩
РЯ      BSS  1        Дополнительная рабочая ячейка
АДРМ    BSS  1        ⟨MMAX⟩
*       Обработка вызывающей последовательности (3.60):
MAX     NOP           Ячейка связи с главной программой.
*                     Мнемоника NOP означает то же, что
*                     и OCT 0, но традиционно используется
*                     для резервирования входных точек.
*                     По команде JSB MAX в эту ячейку
*                     заносится K+1
        STA  N        N: = (PA) = n
        LDA  MAX,I    PA := ((MAX)) = (K + 1) = M (с 1 в 0 = м
*                     разряде)
```

```
        STA   АДР      АДР:=M
        ISZ   MAX      MAX:= (MAX) + 1 = K + 2, т.е. адрес
*                      ячейки главной программы, в которой
*                      находится адрес результата MMAX
        LDA   MAX,I    PA:= ((MAX)) = (K + 2) = MMAX
        STA   АДРМ     АДРМ:= MMAX
        ISZ   MAX      MAX:= K + 3, т.е. ПР – адрес продолже-
*                      ния главной программы
        . . .
```

Далее должны быть повторены команды программы (3.58) (начиная с Подготовки цикла), за исключением двух моментов.

1. Вместо команд с фактическим параметром *MMAX* – **ADA** *MMAX* и **STA** *MMAX* – должны быть указаны команды с косвенной адресацией через ячейку *АДРМ*: **ADA** *АДРМ*, **I** и т.д. Этим достигается независимость команд подпрограммы от адресов фактических параметров.

2. Выход из подпрограммы (метка *ВЫХ*) должен включать команду **JMP** *MAX*, **I** – *команду возврата* в главную программу на метку *ПР*, используя косвенную адресацию через ячейку связи *MAX*.

Проиллюстрированный вызывающей последовательностью (3.60) и подпрограммой (3.61) способ организации подпрограмм является основным для однозначной архитектуры УВКС. Однако при переходе к многозначной архитектуре выявился существенный недостаток такой организации подпрограмм – отсутствие возможности *повторного вхождения.*

Чтобы понять, что на самом деле означает этот термин, представим себе, что двум задачам N1 и N2 потребовалось обратиться к подпрограмме (3.61), причем, когда подпрограмма работала по вызову задачи N1, в активное состояние перешла более приоритетная задача N2 и сразу же обратилась к той же подпрограмме. Нетрудно видеть, что, обслужив задачу N2, подпрограмма не сможет нормально вернуться к прерванному обслуживанию задачи N1, так как содержимое рабочих ячеек подпрограммы (*АДРМ, N* и т.д.) по состоянию их на момент прерывания будет безвозвратно потеряно при обработке запроса задачи N2.

Отсюда ясно, что подпрограмма, допускающая повторное вхождение, не должна иметь в своем составе рабочих ячеек. Такие подпрограммы называют *реентерабельными.*

Как же обойтись без рабочих ячеек? Ответ неожиданно прост: их следует зарезервировать в главной программе и адрес первой из них перед обращением к подпрограмме записать на один из индексных регистров. Тогда в подпрограмме обращение к рабочим ячейкам легко выполнить в режиме индексации или автоиндексации.

Хотя при этом возрастает объем памяти главной программы (так как каждая задача должна иметь свой экземпляр рабочих ячеек), это с лихвой окупается тем, что для любого числа одновременно работающих задач достаточно иметь по одному экземпляру используемых подпрограмм.

Обычно передача базового адреса области рабочих ячеек осуществляется через регистр *РИ2*. Для обеспечения реентерабельности подпрограммы (3.61) необходимо, в частности, в главную программу включить следующие инструкции резервирования памяти и адресной константы *АДРЯ*:

```
                                Резервирование области реентерабельности      (3.62)
P       BSS 5                   по числу рабочих ячеек подпрограммы (3.61)
*                               Резервирование двух ячеек для записи слова
ССП     BSS 2                   состояния программы
*
АДРЯ    DEF P                   ⟨P⟩
```

Напомним, что *слово состояния программы* (ССП) специальной командой **STW** записывается в двух смежных ячейках памяти.

С учетом (3.62) формирование *РИ2* может быть осуществлено командой

```
        LDY АДРЯ        РИ2:= (АДРЯ) = P,                                 (3.63)
```

за которой должна следовать вызывающая последовательность (3.60).

Важно отметить, что при обращении к реентерабельным подпрограммам не может быть использована ячейка связи с главной программой [*MAX* в подпрограмме (3.61)], так как повторное вхождение немедленно изменит ее содержимое. Поэтому для возврата в вызывающую программу здесь используют содержимое *РАВ*, значение которого формируется командой **JSB** [см. операцию (3.17)], запоминается в рабочей ячейке командой записи слова состояния программы **STW**, а из этой рабочей ячейки заносится в РНК командой **LDW** – командой восстановления ССП. Именно поэтому команда **STW** должна быть первой, а команда **LDW** – последней командой реентерабельной программы.

Для правильного понимания приведенной ниже реентерабельной подпрограммы необходимо учесть, что в области рабочих ячеек (3.62) (размещенной в главной программе) ячейка *P* будет играть роль ячейки *АДР* в подпрограмме (3.61), ячейка *P* + 1 – роль ячейки *N* и т.д.:

```
*       Реентерабельная подпрограмма                                      (3.64)
*       является переработкой нереентерабельной подпрограммы
*       (3.61) и автономной программы (3.58)
*       Обработка вызывающей последовательности:
MAX     NOP             не используется:
        STW 5,Y         запись в ячейки ССП и ССП + 1 из
*                       (3.62) слова состояния вызывающей
*                       программы, в частности, ССП + 1:= (РАВ) =
*                       = K + 1, где метка K введена в (3.60)
        STA 1,Y         P + 1:= n
        LDA 6,Y         РА:= (РЯ + 6) = (ССП + 1) = K + 1
        LDB 0,I         РБ:= ((РА)) = (K + 1) = M
        STB 0,Y         P:= M
```

```
        INA             РА := К + 2
        LDB  0, I       РБ := (К + 2) = ММАХ
        INA             РА := К + 3
        STA  6, Y       ССП + 1 := К + 3  (К + 3 = ПР – адрес продол-
*                       жения главной программы
*       Подготовка цикла:
        . . .
```

Далее, как и в подпрограмме (3.61), нужно повторить команды программы (3.58), при этом:

вместо команд с прямой адресацией к ячейкам *АДР, 1, РЯ* необходимо указать команды с адресацией через *РИ2* и смещением 0, 1, 2 и 3 соответственно: **LDX** *0*,**Y**; **LDA** *1*,**Y**; **STA** *2*,**Y** и т.д.;

каждая команда с прямой адресацией к ячейке *ММАХ*, которая в подпрограмме (3.61) должна заменяться на команду с косвенной адресацией через ячейку АДРМ, в реентерабельной подпрограмме (3.64) заменяется той же командой, но с косвенной адресацией через регистр *Б*: **STA** *1*,**I** и **ADA** *1*,**I** в котором к этому моменту должен находиться адрес ячейки *ММАХ*. В (3.64) это обеспечивается второй командой **LDB** *0*,**I** при обработке вызывающей последовательности.

Завершаться реентерабельная программа должна командой засылки ССП:

```
ВЫХ     LDW  5,Y
```

которая помимо восстановления *РР* и *РП* осуществляет возврат в вызывающую программу: *РНК* := (*ССП* + 1) = *ПР*.

Если сравнить быстродействия реентерабельной подпрограммы (3.64) и выполняющей ту же функцию нереентерабельной подпрограммы (3.61), то это сравнение будет в пользу последней, поскольку в Проверке окончания и Теле цикла подпрограммы (3.64) значительно больше команд с адресацией через индексные регистры, а, как было отмечено в § 3.2, время выполнения каждой команды возрастает при этом примерно на 1 мкс. Поэтому на практике при анализе быстропротекающих процессов может оказаться целесообразным пожертвовать свойством реентерабельности для обеспечения максимального быстродействия.

Внимательный читатель в связи с программой (3.64), наверняка, задает вопрос, каким образом при повторном вхождении в реентерабельную программу обеспечивается сохранность программных регистров *РА, РБ, РИ1* и *РИ2*?

Поскольку повторное вхождение может быть только следствием прерывания, этот вопрос сводится к более общему: каким образом сохраняются программные регистры при различного рода прерываниях (программных, по запросу УВВ и др.)?

В операционной системе СМ-2М сохранение программных регистров при многоуровневой системе прерываний обеспечивается организацией *программного стека* и некоторыми другими приемами программирования, описание которых выходит за рамки данной главы. Для нас важно

только, что это функция операционной системы, поэтому в рамках ОС АСПО можно исходить из того, что механизм запоминания и восстановления регистров *РА, РБ, РИ1* и *РИ2* при прерываниях всегда включен.

Организация TAS-семафоров. Проиллюстрируем возможности практического применения команды **TAS**, семантика которой задана операцией (3.41).

Пусть в параллельно исполняемых программах имеются следующие фрагменты:

на ЦПР1	*Ц1*	**TAS** *1000*	на ЦПР2	*Ц2*	**TAS** *1000*
		JMP *Ц1*			**JMP** *Ц2,*

причем первоначально содержимое ячейки 1000 равно нулю.

Из семантики (3.41) следует, что если первым во времени на команду **TAS** выйдет ЦПР1, то в результате ПСК он "проследует дальше", но предварительно в первый разряд ячейки 1000 запишет единицу. Поэтому, когда на команду **TAS** выйдет ЦПР2, *"семафор окажется закрытым"* (не будет выполняться ПСК) и ЦПР2 будет по очереди выполнять команды **TAS** и **JMP** до тех пор, пока ЦПР1 не откроет семафор, т.е. не зашлет в нулевой разряд тысячной ячейки нуль.

Заканчивая гл. 3, отметим, что с точки зрения основных функций управления здесь рассмотрены приемы и средства организации программ *обработки* информации, поступающей с объекта управления.

Что касается программ *сбора* и *выдачи* информации на объект, т.е. программ ввода-вывода, то здесь пользователь может применить либо стандартные *вызовы супервизора,* либо разработать собственные программы, для чего необходимо знать структуру интерфейса 2К, команды ввода-вывода и систему прерываний, т.е. все то, что составляет основу организации ввода-вывода.

Упражнения к гл. 3

1. Напишите программу, которая имитирует алгоритм построения относительного адреса при дешифровке адресной команды основного формата.

2. Объясните, почему в подпрограмме (3.64) не используется ячейка $P + 4$ области реентерабельности (3.62).

3. Напишите реентерабельную подпрограмму сложения двух векторов.

4.1. Логика интерфейса 2К

Шины интерфейса. На уровне архитектуры интерфейс 2 К может быть представлен как совокупность специальных линий – интерфейсных шин, по которым УВВ и процессор обмениваются двоичной информацией. Наличие сигнала на интерфейсной шине (в данном случае не важна его физическая природа) соответствует единице, отсутствие – нулю.

В соответствии с направлением передаваемой информации шины интерфейса делятся на две группы. Информация на шинах первой группы адресована периферийному оборудованию и всегда передается по инициативе процессора командами ввода-вывода. Источником этой информации является ядро комплекса, которое иначе называется *концентратором.* Отсюда *ШИН-К* – обозначение шин этой группы.

Информация на шины другой группы, называемой *ШИН-Т* (источником этой информации является *терминал,* под которым в данном случае подразумевается любое УВВ), адресована ядру ВК. Далее увидим, что на часть *ШИН-Т* информация выдается по инициативе УВВ (в частности, запросы на обслуживание, сообщения о неисправностях), выдача информации на другие *ШИН-Т* осуществляется по инициативе (запросу) процессора.

Разделение интерфейсных шин по признаку направления передаваемой информации дало название интерфейсу 2К – *двойной канал.*

Сигналы, адресованные терминалу. Каждый интерфейсный разъем – физический выход на интерфейс 2К – включает 27 шин группы *ШИН-К.*

Шестнадцать информационных шин ШИНО-К–ШИН15-К предназначены для выдачи 16-разрядного слова с аккумулятора процессора или из памяти на входной регистр УВВ. Интерпретация переданной информации может быть различной для различных устройств. Например, для устройств *отображения* в одной 16-разрядной посылке упаковывается два символа, подлежащих выдаче на экран или печать. Для *коммутаторов* в этой информации содержится указание, какой из датчиков, установленных на объекте, надлежит подключить к аналого-цифровому преобразователю.

Параллельно с информационными сигналами на каждое передаваемое машинное слово выдаются *контрольные разряды* по шинам *КР0-К* для младшего байта и *КР1-К* для старшего. Конструкция большинства УВВ предусматривает контроль по паритету принятой от концентратора информации и выдачу сигнала на специальную шину из группы *ШИН-Т,* если условие паритета нарушилось.

Помимо контрольных разрядов информацию на *ШИН0-К – ШИН15-К* всегда сопровождает служебный сигнал – ВЫДАНО на шине с условным обозначением *ВД-К.* Этот сигнал возбуждается командами вывода и является стробом приема этой информации на входных регистрах УВВ.

Наряду с данными различной природы процессор передает в УВВ управляющую информацию. Для этого могут использоваться как *ШИН0-К–ШИН15-К*, так и специально выделенные управляющие шины, в том числе шина сигнала ВЫПОЛНИТЬ *ВП-К*. Сигнал на этой шине возбуждается командами ввода-вывода при наличии 1 в шестом разряде команды и используется обычно для сообщений терминалу, что процессор закончил прием предыдущей порции информации. Реакция на этот сигнал специфична для каждого УВВ. Можно только сказать, что сигнал *ВП-К* – это специальный бит информации, применяемый для целей управления УВВ.

Для устройств со сложной логикой управления информационной емкости шин интерфейса 2К оказывается недостаточно. В этом случае управление УВВ реализуют *контроллеры* – входящие в состав УВВ микропроцессоры (МП), на которых та или иная функция управления запускается сигналами интерфейса, в том числе и по шине *ВП-К*, а также по другой управляющей шине *ОСТ-К*. Обычно наличие сигнала на этой шине воспринимается устройством как приказ приостановить операцию ввода-вывода.

Стандартную управляющую функцию выполняет сигнал на шине *ПР-К*, по которой процессор инициирует в устройстве операцию выдачи информации с выходного регистра на *ШИН-Т*, направленные от терминала к концентратору.

Перечисленные выше сигналы *ШИН-К* поступают одновременно на все физические выходы интерфейса 2К (соответствующие контакты интерфейсных разъемов запараллелены). Выбор конкретного устройства, а для сложных устройств – конкретной функции процессор осуществляет, возбуждая сигнал на одной из *радиальных* (т.е. незапараллеленных) шин выборки *ВБР0-К – ВБР3-К*. Чтобы понять назначение этого сигнала, вспомним, что каждое устройство в составе УВК характеризуется своим кодом выборки КВ (см. § 1.2). По существу, *код выборки для УВВ играет ту же роль, что и адрес для ячейки ОЗУ*. При выполнении команд ввода-вывода процессор дешифрует содержащийся в ней КВ и возбуждает сигнал на идущей к адресуемому устройству шине *ВБР-К*. Этот сигнал как бы устанавливает связь между процессором и устройством: основные управляющие сигналы на шинах *ВД-К, ПР-К, ВП-К, ОСТ-К* воспринимаются устройством только при наличии единицы на идущей к нему шине *ВБР-К*.

В *одном* интерфейсном разъеме имеются четыре шины сигнала выборки от *ВБР0-К* до *ВБР3-К*. Это позволяет в сложных устройствах адресовать отдельные функции, используя до четырех смежных кодов выборки. Естественно, что при этом нельзя занимать физические выходы на интерфейс 2К с задействованными кодами выборки. Например, если к интерфейсному разъему № 30 подключено сложное УВВ, использующее все четыре шины – *ВБР0-К – ВБР3-К*, то обращение к этому устройству (точнее, к его различным функциям) будет осуществляться по кодам

выборки 30–33. Поэтому физические выходы на интерфейс 2К с номерами 31–33 должны остаться свободными.

В состав интерфейса 2К входит также шина *ОСБ-К,* по которой в устройство поступает сигнал сброса – команда перевода устройства в исходное состояние. Точный смысл этого состояния специфичен для каждого устройства. Сигнал *ОСБ-К* выдается на все интерфейсные разъемы *первого уровня* (т.е. на выходы на интерфейс 2К в СВВ в отличие от выходов *второго уровня* в РИМ) при нажатии клавиши СБР СИСТ на инженерной панели процессора. Специальной командой **RIO** этот сигнал может быть индивидуализирован – направлен по конкретному коду выборки.

Сигналы, адресованные концентратору. В состав направленных от терминала к концентратору шин входят 24 *ШИН-Т.*

16 информационных шин – ШИН0-Т – ШИН15-Т – предназначены для передачи информации с выходного регистра УВВ на один из аккумуляторов процессора или непосредственно в ОЗУ (по *каналу прямого доступа в память*).

Передаваемая информация сопровождается сигналами на контрольных шинах *КР0-Т, КР1-Т,* по которым параллельно с информацией на *ШИН0-Т – ШИН15-Т* на каждый байт выдаются контрольные разряды, выработанные логическими схемами УВВ.

Если в конструкции УВВ отсутствует схема формирования контрольных разрядов, соответствующие шины *КР-Т* остаются незадействованными. Вместо этого устройство должно выставить сигнал на шину *ОК-Т,* который отменяет в концентраторе контроль по паритету принятой от УВВ информации.

Перечисленные сигналы выдаются на *ШИН-Т* только по инициативе процессора в ответ на единицу на шине *ПР-К.* По инициативе устройства выдаются сигналы на шины *ГТ-Т, ОШ-Т, КОП-Т.*

На шины *ГТ0-Т – ГТ3-Т* устройство выставляет *сигнал готовности,* если оно готово к выполнению очередной операции обмена. Реакция процессора на этот сигнал может быть различной. При определенных условиях сигнал готовности вызывает в процессоре *прерывание* последовательного выполнения команд и передачу управления ячейке, адрес которой относительно базы привилегированного состояния равен коду выборки устройства, выставившего сигнал готовности (так называемое прерывание по запросу УВВ). Четыре шины – *ГТ0-Т –ГТ3-Т* – используют сложные УВВ, которые имеют, таким образом, возможность оповещать о готовности по различным функциям. В этом случае прерывание по сигналу *ГТ0-Т* вызывает передачу управления на ячейку *КВ,* по сигналу *ГТ1-Т* – на ячейку *КВ* + 1 и т.д. Естественно, при этом на интерфейсе 2К не должны заниматься места с соответствующими номерами интерфейсных разъемов.

В отличие от немедленной реакции на сигнал *ГТ-Т* процессор реагирует на сигнал шины *ОШ-Т,* которым устройство оповещает об ошибке,

обнаруженной им в информации, принятой от концентратора или внешнего носителя (ленты, диска и т.п.), только в момент очередного обращения к этому устройству.

Аналогично обстоит дело с сигналом шины *КОП-Т* – сигналом, означающим, как правило, окончание операции ввода-вывода по инициативе терминала. Причиной может быть либо признак конца ввода на внешнем носителе (например, конец зоны на МЛ), либо возникновение ненормальной ситуации, при которой терминал не может продолжить начатую операцию (обрыв перфоленты, бумаги в печатающем устройстве и т.п.).

Сигналы *ОШ-Т, КОП-Т* вызывают *прерывание от схем контроля процессора.* Это означает передачу управления ячейке с относительным адресом 5 (сравните с прерыванием по сигналу *ГТ-Т*).

Помимо перечисленных *ШИН-К* и *ШИН-Т* в составе интерфейса 2К есть *аналоговая шина,* назначение которой – передача на вход коммутаторов и устройств типа АЦП аналоговой информации.

Сигналы интерфейса и алгоритм работы УВВ. Алгоритм работы устройства, имеющего выход на интерфейс 2К, определяет порядок следования во времени состояний и функций УВВ в соответствии с сигналами интерфейса.

На рис. 4.1 приведена упрощенная блок-схема алгоритма работы таймера ТМР А129-2. Свою работу ТМР начинает по сигналу на шине ОСБ-К в сочетании (конъюнкции) с сигналом на шине ВБР-К. Это отражает общее положение – без конъюнкции с сигналом на радиальной шине ВБР-К – шине выбора процессором именно данного устройства, УВВ управляющих сигналов интерфейса не воспринимает.

Таймер реализует две основные функции:

а) функцию "будильника", т.е. формирование сигнала готовности по истечении заданного числа временных интервалов длительностью $\Delta\tau$ (длительность интервала $\Delta\tau$ программным путем устанавливается равной либо 10 мкс, либо 100 мкс);

б) функцию "говорящих часов" – выдачу на *ШИН-Т* по запросу процессора текущего значения счетчика интервалов $\Delta\tau$.

Управляющее слово таймеру принимается с *ШИН-К* по конъюнкции сигналов *ВД-К* и *ВБР-К,* вслед за которой следует конъюнкция сигналов *ВП-К* и *ВБР-К.* Структура управляющего слова такова:

нуль в нулевом разряде воспринимается таймером как сигнал установить $\Delta\tau$ = 10 мкс, единица – как сигнал установить $\Delta\tau$ = = 100 мкс;

содержимое разрядов 1–15 воспринимается как дополнительный код со знаком в первом разряде интервала времени, по истечении которого должен быть выставлен сигнал готовности (чтобы построить код интервала времени, нужно определить, сколько раз в этом интервале укладывается выбранный шаг дискретности $\Delta\tau$, и взять полученный результат со знаком минус).

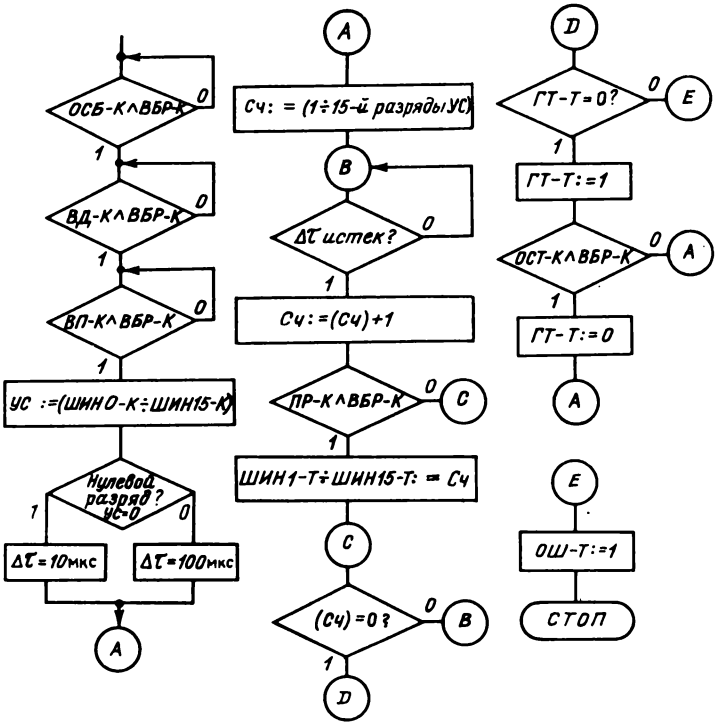


Рис. 4.1. Алгоритм работы таймера ТМР А129-2:
УС – управляющее слово таймеру, *Сч* – счетчик интервалов

После обнуления счетчика и выставления сигнала готовности таймер автоматически начинает отсчет нового интервала времени. Если к моменту его завершения ранее выставленный сигнал готовности не будет сброшен конъюнкцией сигналов *ОСТ-К* и *ВБР-К*, то таймер сформирует сигнал *ОШ-Т*, сообщив тем самым, что процессор по каким-то причинам не отреагировал на сигнал "будильника".

Функцию *"б"* таймер может осуществить в любой момент времени по запросу процессора, т.е. по конъюнкции сигналов *ПР-К* и *ВБР-К*.

Таким образом, программное управление таймером и считывание его временных отметок сводятся к формированию командами ввода-вывода определенных сигналов на шинах интерфейса 2К в тех комбинациях и в том порядке, которые соответствуют алгоритму работы таймера.

4.2. Команды ввода-вывода

Команды ввода-вывода относятся к *группе привилегированных*, т.е. их непосредственное использование допускается только в программах нулевого раздела. На рис. 4.2 представлен формат этих команд.

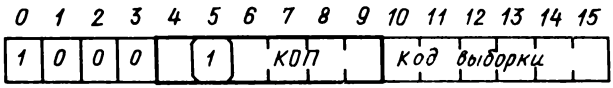


Рис. 4.2. Формат команды ввода-вывода

Из предыдущего ясно, что семантика каждой команды должна определять, на каких шинах интерфейса 2К возбуждается сигнал при ее выполнении. В этом смысле следует отметить, что все команды ввода-вывода при обращении к УВВ возбуждают сигнал на шине *ВБР-К*, соответствующей коду выборки адресуемого устройства:

$$ВБР - К := 1. \qquad (4.1)$$

Поэтому далее при описании семантики команд ввода-вывода операция (4.1) подразумевается по умолчанию.

Не все команды, которые имеют формат, показанный на рис. 4.2, возбуждают сигналы на интерфейсных шинах.

В табл. 4.1 помимо собственно команд ввода-вывода представлены команды ПСК (пропуска следующей команды) по сигналу готовности, команды включения и выключения УВВ из системы прерываний (управление маской УВВ), команда останова процессора.

Семантика команд. Ниже приводится описание семантики команд, представленных в табл. 4.1, при этом используются обозначения, принятые в § 3.2. В тех случаях, когда та или иная команда имеет смысл

Таблица 4.1

Наименование команды	Мнемоника	КОП, разряды					Время выполнения, мкс
		4	6	7	8	9	
Вывод (*РА*) на *ШИН-К*	**OTA**	0	*	1	1	0	4,3
Вывод (*РБ*) на *ШИН-К*	**OTB**	1	*	1	1	0	4,3
Ввод на *РА* с *ШИН-Т*	**LIA**	0	*	1	0	1	4,0
Ввод на *РБ* с *ШИН-Т*	**LIB**	1	*	1	0	1	4,0
Дизъюнктивный ввод на *РА*	**MIA**	0	*	1	0	0	4,0
Дизъюнктивный ввод на *РБ*	**MIB**	1	*	1	0	0	4,0
ПСК по сигналу готовности	**SFS**	0	0	0	1	1	3,5
ПСК по отсутствию сигнала готовности	**SFC**	0	0	0	1	0	3,5
Выдача сигнала *ВП-К*	**CLF**	0	1	0	0	1	3,5
Выдача сигнала *ОСТ-К*	**STF**	0	0	0	0	1	3,5
Сброс маски прерывания	**STC**	0	*	1	1	1	4,5
Установ маски прерывания	**CLC**	1	*	1	1	1	5,4
Выдача сигнала *ОСБ-К*	**RIO**	1	0	0	0	0	5,1
Останов процессора	**HLT**	0	0	0	0	0	

Примечание. * означает, что в команде шестой разряд может быть как нулем, так и единицей. В случае единицы команда помимо своей основной функции возбуждает сигнал на шине *ВП-К*.

с КВ в диапазоне от 0 до 6 (команды со *специальными кодами выборки*), это оговаривается особо.

Вывод (РА/Б) на *ШИН-К*:

$$\text{ОТА/В } КВ\ [,\text{С}]\ \langle\Rightarrow ВД\text{-}К := 1 \quad (4.2)$$
$$ШИН0\text{-}К\ \div\ ШИН15\text{-}К = (РА/Б)$$
$$[ВП\text{-}К := 1].$$

В программах на языке МНЕМОКОД команда **ОТА/В** применима как с указанием сигнала выполнить **С**, так и без него: **ОТА** *КВ*,**С** и **ОТА** *КВ*. При описании формальной семантики (4.2) *необязательные элементы взяты в квадратные скобки.* Этот прием будет часто использоваться и далее.

По команде **ОТА/В** с кодом выборки *5* младший байт (*РА/Б*) загружается в *РНГ*, при этом в *РВГ* устанавливается код 377.

С кодом выборки *6* команда **ОТА/В** формирует *РВГ*, записывая на этот регистр старший байт (*РА/Б*).

Ввод на *РА/Б* с *ШИН-Т*:

$$\text{LIA/В } КВ\ [,\text{С}]\ \langle\Rightarrow\ ПР\text{-}К := 1 \quad (4.3)$$
$$РА/Б := (ШИН0\text{-}Т\ \div\ ШИН15\text{-}Т)$$
$$[ВП\text{-}К := 1].$$

По этой команде осуществляется запись информации с выходного регистра УВВ на аккумулятор *РА/Б*. Если выходной регистр УВВ имеет меньше 16 разрядов, то информация записывается в младшие разряды *РА/Б*, а остальные обнуляются.

На *РА/Б* по команде **LIA/В** с кодом выборки *1* производится засылка содержимого *клавишного регистра* инженерной панели процессора, *4* – засылка кода выборки последнего прерывания, *5* – засылка основного *управляющего слова прерывания* УСП) – 16-разрядного слова, в котором конкретизируется причина последнего прерывания. Если в коде команды **LIA/В** *5* шестой разряд равен 1, на заключительной фазе выполнения команды все разряды УСП устанавливаются в 0. По команде **LIA/В** *6* на *РА/Б* производится засылка (*РАВ*) – содержимого регистра адреса возврата.

Дизъюнктивный ввод на *РА/Б*:

$$\text{MIA/В } КВ\ [,\text{С}]\ \langle\Rightarrow\ ПР\text{-}К := 1 \quad (4.4)$$
$$РА/Б := (РА/Б) \cup (ШИН0\text{-}Т \div ШИН15\text{-}Т)$$
$$[ВП\text{-}К := 1].$$

В отличие от команды **LIA/В**, по которой прежнее содержимое *РА/Б* безвозвратно теряется, здесь происходит поразрядная дизъюнкция (логическое сложение) (*РА/Б*) с сигналами *ШИН-Т*. Это может быть использовано для накопления на аккумуляторе информации, поступающей от УВВ во времени.

Специальные коды выборки *1*, *4*, *5* и *6* имеют тот же смысл, что и для команды **LIA/B**, но вместо операции замещения (*РА/Б*) осуществляется его поразрядная дизъюнкция.

ПСК по сигналу готовности:

SFS *КВ* ⟨⇒ **если** *ГТ-Т* = 1 **то** ПСК. (4.5)

Как следует из семантики (4.5), пара команд

К **SFS** *КВ* (4.6)
JMP *К*

переводит процессор в состояние ожидания сигнала *ГТ-Т* от адресуемого устройства.

По команде **SFS** с кодом выборки *6* осуществляется пропуск следующей команды, если команду выполняет первый из двух ЦПР.

ПСК по отсутствию сигнала *ГТ-Т*:

SFC *КВ* ⟨⇒ **если** *ГТ-Т* = 0 **то** ПСК. (4.7)

Приемом, аналогичным (4.6), можно перевести процессор в ожидание сброса сигнала готовности.

По команде **SFC** с кодом выборки *6* осуществляется пропуск следующей команды, если команду выполняет второй из двух ЦПР.

Выдача сигнала ВП-К (другое название команды – продолжение ввода-вывода):

CLF *КВ* ⟨⇒ *ВП-К* := 1. (4.8)

По этой команде в адресуемое периферийное устройство направляется сигнал ВЫПОЛНИТЬ. Поскольку этот сигнал может быть выработан и другими командами, данная команда применяется в тех случаях, когда его нужно выдать *отдельно* от приема или выдачи информации. Это дает возможность варьировать в программе время между выдачей сигналов на шины *ВД-К* или *ПР-К* и выдачей сигнала на шину *ВП-К*.

С кодом выборки *4* команда **CLF** возбуждает сигнал готовности на всех интерфейсных разъемах первого уровня (*имитатор готовности*). Это может быть использовано для тестов, а также для отладки программ ввода-вывода на УВК неполной комплектации.

Выдача сигнала *ОСТ-К* (другое название команды – останов ввода-вывода):

STF *КВ* ⟨⇒ *ОСТ-К* := 1. (4.9)

По этой команде в адресуемое периферийное устройство выдается сигнал ОСТАНОВ. Чаще всего в ответ на этот сигнал УВВ заканчивает операцию ввода-вывода и сбрасывает сигнал готовности.

Команда **STF** с кодом выборки *4* выключает имитатор готовности устройств первого уровня, с кодом выборки *6* переводит процессор в состояние *динамического останова*. В этом состоянии никаких действий

процессор не выполняет до тех пор, пока не произойдет прерывание (по запросу УВВ, схем контроля и др.) или сброс этого состояния с инженерной панели.

Команды **STC** *КВ* и **CLC** *КВ* служат соответственно для включения и выключения адресуемого устройства из системы прерываний.

Выдача сигнала ОСБ-К (или иначе – сброс ввода-вывода):

RIO *КВ* ⟨=⟩ *ОСБ-К* := 1. (4.10)

По этой команде в адресуемое устройство выдается сигнал ОБЩИЙ СБРОС. Если код выборки равен *0*, то сигнал ОБЩИЙ СБРОС выдается на все интерфейсные разъемы первого уровня.

Останов процессора HLT *КВ*. По этой команде процессор прекращает свою работу, *РНК* содержит адрес следующей команды, а на инженерной панели высвечивается код команды **HLT**: 1020*КВ*. При этом никаких сигналов на интерфейсе 2К не возбуждается, а код выборки применяют для идентификации причины останова, если останов процессора по разным причинам производится командами **HLT** с разными кодами выборки.

Управление регистром переполнения (*РП*). Для управления *РП* (см. функцию 4, табл. 2.4) используются некоторые команды ввода-вывода с кодом выборки *1*. Каждая из этих команд имеет свою собственную мнемонику.

Команда сброса регистра переполнения CLO (кодируется как команда **CLF** 1):

CLO ⟨⇒ *РП* := 0. (4.11)

Установ регистра переполнения – команда **STO** (кодируется как команда **STF** 1):

STO ⟨⇒ *РП* := 1. (4.12)

Пропуск, если *РП* взведен, – команда **SOS** (кодируется как команда **SFS** *1*):

SOS ⟨⇒ **если** (*РП*) = 1 **то** ПСК. (4.13)

Пропуск, если *РП* сброшен, – команда **SOC** (кодируется как команда **SFC** *1*):

SOC ⟨⇒ **если** (*РП*) = 0 **то** ПСК. (4.14)

Команды **CLO, STO, SOS** и **SOC** в отличие от других команд ввода-вывода являются непривилегированными.

Косвенная адресация УВВ. Адресация УВВ путем непосредственного указания в команде кода выборки устройства называется *прямой*.

Этот способ адресации является наиболее быстродействующим и в то же время связан с рядом ограничений. В частности, при использовании прямой адресации программы ввода-вывода становятся жестко

привязанными к определенным кодам выборки, возникают трудности в настройке модулей операционной системы на конкретную конфигурацию УВКС.

Кроме того, информационной емкости шести разрядов, отведенных в формате команды для КВ, не хватает, чтобы прямой адресацией обращаться к УВВ второго уровня, т.е. подключенным через РИМ.

В связи с этим в архитектуре процессора СМ-2М реализован еще один способ адресации УВВ – *косвенный.* Здесь адрес УВВ кодируется на одном из индексных регистров, а код выборки в командах ввода-вывода берется равным двум, если используется *РИ1,* и трем при использовании *РИ2.*

Для косвенной адресации в соответствующий индексный регистр должно быть предварительно загружено машинное слово следующего формата:

нулевой, второй и третий разряды – 0;

первый разряд – признак уровня: 0 – для устройств, подключенных к СВВ, 1 – для устройств, подключенных через РИМ;

разряды 4–9 – код выборки РИМа (при адресации устройств 1-го уровня в эти разряды должны быть записаны нули);

разряды 10–15 – код выборки устройства в СВВ или РИМ (в зависимости от значения первого разряда).

При косвенной адресации команды ввода-вывода исполняются несколько медленнее – время дополнительной задержки 0,8 мкс на одну команду. Другое отличие состоит в реакции процессора на сигналы *ОШ-Т* и *КОП-Т.* Косвенная адресация к устройству, выработавшему один из этих сигналов, вызывает пропуск следующей команды, а прямая адресация, как уже говорилось в § 4.1, вызывает прерывание от схем контроля процессора.

4.3. Система прерываний

Ячейка прерывания. Выше неоднократно упоминалось о прерывании как об основной реакции процессора на "внешние раздражители". Напомним, что с точки зрения пользователя суть прерывания состоит в прекращении выполнения текущей программы. При этом немедленно устанавливается привилегированное состояние процессора и управление передается *ячейке прерывания,* т.е. определенной ячейке нулевой страницы привилегированного раздела.

Адрес ячейки прерывания однозначно определяется источником прерывания. Прерывание по запросу УВВ с кодом выборки КВ приводит к передаче управления ячейке с адресом *КВ* относительно базы привилегированного состояния. (Для устройств второго уровня передача осуществляется по коду выборки РИМ.)

Заметим, что при такой организации системы прерываний для выполнения каждого конкретного прерывания требуется минимум машинных

тактов. В результате в интерфейсе 2К обеспечивается более быстрая реакция процессора на запрос УВВ по сравнения с другими интерфейсами, например *с общей шиной* [12, 13].

Прерывание от схем контроля процессора заключается в передаче управления ячейке 5. По адресу 4 передается управление при падении напряжения в сети ниже 187 В (прерывание *по нарушению питания*), по адресу 6 – при программном прерывании (см. § 2.3), которое происходит по команде *STC* с кодом выборки *6* (отсюда ясно, что мнемоника *SVC – вызов супервизора* транслируется именно в эту команду).

Очевидно, что содержимое ячейки прерывания не может быть произвольным. Смысл прерывания состоит в том, чтобы вместо прерванной программы процессор начал выполнять другую, которую называют *программой обработки прерываний.* Поэтому операционная система должна быть написана так, чтобы при ее загрузке в каждую ячейку прерываний попадала команда перехода (**JSB** или **JMP**) на соответствующую программу обработки прерывания.

Напомним, что семантика команды **JSB**, находящейся в ячейке прерывания, несколько отличается от семантики этой команды в любой другой ячейке: в адресуемую ячейку и в *PAB* записывается адрес ячейки продолжения *прерванной* программы, а не адрес ячейки, следующей за ячейкой прерывания, как можно было бы ожидать в соответствии с операцией (3.16). Следовательно, в программе обработки прерываний возврат к выполнению прерванной программы программируется точно так же, как и возврат в вызывающую программу в подпрограммах общего назначения (Подробнее об этом см. в § 3.4. Отметим, что программы обработки прерываний оформляются, как правило, в виде реентерабельных подпрограмм).

Прерывания по запросу УВВ. Необходимым условием возникновения прерывания по запросу внешнего устройства является наличие сигнала на соответствующей шине *ГТ-Т,* т.е. для прерывания необходимо, чтобы УВВ *выставило* сигнал готовности.

Однако функционирование УВКС как системы было бы довольно беспорядочным, если бы это условие являлось и достаточным.

Упорядочение прерываний обеспечивается прежде всего установлением *приоритетов.* В архитектуре СМ-2М предусмотрено, что внешнее устройство имеет тем *больший* приоритет, чем *меньше* его КВ. Это достигается блокировкой после возникновения прерывания по запросу УВВ с определенным кодом выборки всех запросов на обслуживание от этого УВВ и устройств с большими кодами выборки. Такая блокировка снимается командой **STC** *КВ.* Поэтому *драйвер* УВВ в том случае, когда он является программой обработки прерывания, в заключительной части должен содержать эту команду, чтобы разрешить прерывание от УВВ с меньшими приоритетами.

Кроме того, для управления прерываниями каждый физический выход на интерфейс 2 К снабжен специальным триггером – разрядом

маски. Командой **CLC** *КВ* этот разряд может быть установлен в 1, запретив тем самым прерывание по запросу устройства с кодом выборки КВ (выключая индивидуально данное УВВ из системы прерываний). Установ маски в 0 осуществляется командой **STC** *КВ,* которая, кроме того, осуществляет, как уже говорилось, деблокировку запросов на прерывания от устройств с большими КВ.

Наконец, еще одна возможность программного регулирования поступающего от внешних устройств потока заявок на обслуживание заключена в командах **STF** и **CLF** с кодом выборки 0. Первая из этих команд включает, а вторая выключает всю систему прерываний по запросам УВВ в целом.

В исходном состоянии УВКС *система прерываний по запросам внешних устройств включена, но все маски установлены в 1.*

Какой же из двух процессоров УВК СМ-2М прерывает свою работу для обслуживания запроса того или иного УВВ?

В *общем* случае прерывается процессор, раньше другого закончивший выполнение текущей команды. (Отметим, что прерывание не наступает непосредственно после команд ввода-вывода, переходов **JSB** и **JMP** и команды **TAS,** но может наступить в процессе выполнения некоторых команд дополнительного набора, например команды поиска заданного байта в каком-либо массиве).

Таким образом, для возникновения прерывания по запросу УВВ *необходимо и достаточно* одновременное выполнение четырех условий:

включена система прерываний по запросам УВВ (регулируется командами **STF** *0* и **CLF** *0*);

устройство не замаскировано (триггер маски управляется командами **STC** *КВ* и **CLC** *КВ*);

устройство выставило сигнал готовности (чаще всего это происходит по инициативе УВВ, в то же время для многих УВВ управление сигналом готовности может осуществляться командами **CLF** *КВ* и **STF** *КВ*);

доступный для запросов данного УВВ процессор не занят обслуживанием прерывания более высокого приоритета.

С системой прерываний по запросам УВВ непосредственно связан один из основных режимов обслуживания внешних устройств. В этом режиме – *режиме обслуживания УВВ с прерываниями* – процессор запускает на устройстве требуемую операцию (например, операцию преобразования аналог–код на АЦП) и тут же переходит к какой-либо другой готовой к выполнению задаче. Как только устройство завершит начатую операцию, оно выставит сигнал готовности. В ответ на этот сигнал процессор прервет текущую работу и займется обработкой полученной с устройства информации. Таким образом, при обслуживании медленных устройств этот режим позволяет существенно экономить процессорное время.

Другой режим обслуживания УВВ – *режим сканирования* (ожидания) *готовности* – применяется для быстродействующих устройств,

когда параллельная работа УВВ и процессора практического эффекта не дает. Программно режим сканирования готовности реализуется парой команд (4.6).

Прерывание от схем контроля. Приоритет более высокий, чем прерывание по запросу любого УВВ, имеет прерывание от схем контроля процессора.

Схемы контроля процессора установлены на входах и выходах основных агрегатных блоков ядра УВК СМ-2М (ОЗУ, ЦПР, СВВ) и контролируют ошибки паритета при чтении и записи информации, а также фиксируют другие события, препятствующие нормальному продолжению программы.

Прерывание от схем контроля аппаратно сводится, как уже говорилось, к передаче управления на пятую ячейку привилегированного раздела и, кроме того, к формированию основного и дополнительного управляющих слов прерывания (УСП и ДУСП), в которых единицами соответствующих разрядов конкретизируются причины прерывания (остальные разряды – нули):

	Разряд УСП
Обращение (прямой адресацией) к УВВ, выставившему сигнал *КОП-Т*	0
Попытка нарушения защиты памяти	1
Попытка выполнить привилегированную команду в непривилегированном режиме (нарушение режима)	2
Срабатывание внутреннего таймера (возникает, если команда не успела выполниться в установленное для нее время)	3
Ошибка по паритету чтения из ОЗУ	8
Ошибка по паритету записи в ОЗУ	9
Ошибка по паритету ввода от УВВ	10
Ошибка по паритету чтения из ПЗУ	11
Нарушение питания	12
Попытка выполнить команду с несуществующим (т.е. не обеспеченным микропрограммой) кодом операции	13
Указание на наличие информации в ДУСП	14
Обращение (прямой адресацией) к УВВ, выставившему сигнал *ОШ-Т*	15
	Разряды дополнительного УСП
Ошибка по паритету обмена с СВВ	4
Ошибка по паритету на выходе сумматора	5
Ошибка по паритету при адресации микропрограммной памяти	6
Нарушение вторичного питания в ОЗУ или СВВ	7
Ошибка по паритету обмена с РИМ	14

Управляющее слово прерывания может быть принято на *РА* или *РБ* командами **LTA, LIB, MIA** или **MIB** с кодом выборки *5*. Дополнительное УСП загружается на *РА* командой дополнительного набора **LES.**

В исходном состоянии система прерываний от схем контроля включена, но, как и любой другой вид прерываний, она может быть замаскирована (командой **CLC** *5*). Однако при разработке архитектуры СМ-2М

было признано целесообразным *исключить* из числа маскируемых ошибки по паритету чтения/записи в ОЗУ. Таким образом, независимо от состояния маски прерывания по коду выборки *5* эти ошибки всегда приводят к передаче управления на пятую ячейку привилегированного раздела. В операционных системах АСПО следствием этого является в конечном итоге передача управления специальной подсистеме анализа сбоев и реконфигурации.

4.4. Архитектура канала прямого доступа в память

Назначение и логическая организация канала. Аппаратно канал прямого доступа в память представляет собой специализированный микропроцессор ввода-вывода, предназначенный для обеспечения обмена информацией между ОЗУ и УВВ без участия центрального процессора.

Однако роль КПДП состоит не только в разгрузке центрального процессора по операциям ввода-вывода. Обладая высокой пропускной способностью, КПДП позволяет подключать к ядру ВК быстродействующие внешние устройства – в первую очередь устройства внешней памяти, которые в принципе не могут работать с имеющим ограниченную производительность *программным* каналом. Здесь под программным каналом понимается канал обмена между УВВ и ОЗУ через аккумуляторы процессора под управлением программы ввода-вывода.

Каждый из двух входящих в состав УВК СМ-2М каналов прямого доступа имеет четыре независимых подканала и поэтому способен обслуживать до четырех одновременно работающих УВВ.

Какие внешние устройства доступны каналу? Однозначно можно сказать, что это УВВ первого уровня, т.е. подключенные непосредственно к СВВ, но не к РИМ. С другой стороны, при разработке проекта УВКС заказчик с учетом особенностей УВВ и характера решаемых задач должен выбрать, в каком из двух режимов использовать каждый подканал – в *мультиплексном* или *селекторном.*

В мультиплексном режиме каждому подканалу доступно любое устройство в каждом из трех СВВ, причем связь между ними устанавливается программным путем.

В селекторном режиме устройство и подканал связаны *физически,* и связь эта устанавливается при изготовлении УВКС размещением устройства на специальном месте в СВВ. Два таких места, предназначенных для связи с первым каналом, имеются в СВВ1, для связи со вторым каналом – в СВВ2. В СВВ3 таких мест нет.

Устройства, обслуживаемые в селекторном режиме, оказываются недоступными программному каналу (в частности, потому, что специальные места в СВВ не связаны с определенными кодами выборки). Кроме того, в этом режиме канал не может обслуживать два и больше одновременно работающих УВВ. Но все эти потери окупаются скоростью

обмена: пропускная способность селекторного канала в 2 раза выше, чем мультиплексного.

Заметим, что часто фактическая скорость обмена по КПДП определяется быстродействием самого УВВ потому, что каждое очередное слово или каждый очередной байт передается только после того, как устройство сигналом *ГТ-Т* подтвердило готовность продолжить операцию обмена.

Задание операций каналу. Программно канал управляется как обычное УВВ: чтобы начать операцию обмена, процессор по определенным кодам выборки (*12* – для первого КПДП и *14* – для второго) должен выдать на *ШИН-К* пять *управляющих слов* каналу (УСК1–УСК5), при этом в первых четырех УСК даются все необходимые параметры: номер подканала, код выборки УВВ, адрес буфера, объем передаваемой информации и др., а пятое УСК непосредственно запускает операцию.

Алгоритм работы канала требует, чтобы каждое УСК выдавалось в сопровождении сигнала ВЫПОЛНИТЬ и выдача очередного УСК производилась не раньше, чем канал сигналом готовности подтвердит принятие предыдущего.

Таким образом, последовательность команд, реализующая выдачу УСК по коду выборки *12*, т.е. первому КПДП, должна иметь вид:

```
*  Выдача первого УСК                                    (4.15)
       LDA  УСК1
       OTA  12B, C
*  Ожидание готовности
       SFS  12B
       JMP  * – 1
*  Выдача второго УСК
       LDA  УСК2
       OTA  12B, C
```

и т.д.

Первое из четырех управляющих слов УСК имеет следующий формат:

в 10–15-й разряды записывается код выборки устройства;

в 6–7-й разряды записывается код операции обмена: нуль в шестом разряде задает операцию ввода, единица – операцию вывода, нуль в седьмом разряде означает, что обмен осуществляется словами, единица – байтами;

во 2–3-м разрядах кодируется номер подканала (00 – подканал 1, 01 – подканал 2 и т.д.);

в 1-м разряде нуль записывается при использовании подканала в мультиплексном режиме, единица – в селекторном;

8-й и 9-й разряды формируют с учетом специфики обслуживаемого УВВ: если УВВ требует, чтобы обмен осуществлялся в сопровождении сигнала *ВП-К*, то единица записывается в восьмой разряд, если требуется сигнал *ОСТ-К* – то в девятый;

в 5-м разряде должна быть записана единица – отличительный признак УСК1;

в нулевом разряде чаще всего записывается нуль, хотя здесь есть некоторые тонкости;

4-й разряд не используется и должен содержать нуль.

Абсолютный 17-разрядный адрес буфера (т.е. начальный адрес вводимого или выводимого массива) кодируется во втором и третьем УСК: девять старших разрядов адреса записываются в девяти младших разрядах УСК3, восемь младших разрядов – в младшем байте УСК2 (остальные разряды УСК2 и УСК3 – нули).

Длина передаваемого массива (со знаком минус в 16-разрядном дополнительном коде) задается каналу в УСК4. Эта длина измеряется в словах или байтах в зависимости от значения седьмого разряда УСК1.

Наконец, пятое УСК, непосредственно запускающее операцию обмена на выполнение, должно содержать номер подканала во втором и третьем разрядах, как и первое УСК, единицу в седьмом разряде и нули в остальных. Поэтому УСК запуска операции на первом подканале равняется 400_8, на втором подканале – 10400_8 и т.д.

Начатую операцию обмена процессор может приостановить командой **ОТА** *12***В, С** (или **ОТА** *14***В, С** для второго КПДП), если на *РА* предварительно сформировать 0 для приостанова операции на первом подканале, 10000_8 – на втором, 20000_8 – на третьем и 30000_8 – на четвертом, а продолжить ее той же командой с предварительной загрузкой на *РА* описанного выше УСК5.

Из других операций, реализованных в архитектуре КПДП, отметим возможность получения по запросу процессора в произвольный момент времени текущих значений длины и адреса последнего элемента переданного массива.

Эта и ряд других операций канала чаще всего используются в диагностических тестах, и поэтому их подробное описание выходит за рамки данной книги.

Анализ завершения работы КПДП. Канал вырабатывает сигнал готовности и формирует запрос на прерывание по коду выборки *13* (или *15* для второго КПДП), как только один из подканалов завершит заданную операцию обмена, а также в тех случаях, когда операция прекращается аварийно.

Прочитав на *РА/Б* командой **LIA/B** *13***В** *слово состояния канала* (ССК) и проанализировав его старшие четыре разряда, можно выяснить, какой из четырех подканалов закончил операцию: если единица записана в нулевом разряде ССК, то обмен закончил первый подканал, если в первом – то второй и т.д. (в ССК разряды 4–15 всегда равны нулю).

Алгоритм определения *причины окончания* работы подканала состоит в следующем:

1. Командой **ОТА** *12***В, С** с (*РА*), равным 100_8 для первого подканала, 1100_8 -- для второго, 2100_8 – для третьего и 3100_8 – для четвертого, процессор запрашивает слово состояния подканала (СП).

2. Закончив формирование СП, канал выставляет сигнал готовности по коду выборки *12.* Поэтому необходимо дождаться сигнала *ГТ-Т,* как это сделано в последовательности команд (4.15).

3. Слово состояния подканала заносится на РА командой **LIA** *12***В** (или **MIA** *12***В**).

Далее причина окончания работы подканала определяется, исходя из следующего формата СП: единица в нулевом разряде СП означает нормальное завершение операции, в первом – свидетельствует, что обслуживаемое устройство выработало сигнал *КОП-Т,* во втором – то же, но устройство выработало сигнал *ОШ-Т,* в третьем – обнаружена ошибка по паритету в принимаемой информации, в седьмом -- та же ошибка обнаружена в выдаваемой информации, в шестом – означает, что обнаружена ошибка в схемах канала, которая привела к превышению допустимого времени передачи одного слова или байта.

Упражнения к гл. 4

1. Напишите последовательность команд, реализующих алгоритм работы таймера. В частности, а) запустите таймер с интервалом $\Delta\tau$ = = 100 мкс и интервалом выставления сигнала готовности Δ*ГТ-Т,* равным 1 с; б) запросите текущее значение счетчика *Сч* интервалов $\Delta\tau$.

2. Рассмотрите возможные последствия выполнения команды **STC** *КВ,* если устройство с кодом выборки КВ выставило сигнал готовности. Как в связи с этим должна завершиться программа обработки прерываний по запросу УВВ?

РАЗДЕЛ ВТОРОЙ

БАЗОВЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ МНЕМОКОД

Глава 5. ОСНОВНЫЕ ПОНЯТИЯ

5.1. Назначение базового языка

Следуя [14], языком программирования можно назвать любую систему обозначений для описания алгоритмов и данных, хотя обычно требуется, чтобы программы, написанные на этом языке, могли выполняться на реальной машине. Чтобы обеспечить эту возможность, создаются специальные программирующие программы – *трансляторы,* которые преобразуют символическую запись на языке программирования в последовательности машинных команд.

На ЭВМ архитектурной линии СМ-2М трансляция программ с языков высокого уровня осуществляется в два этапа: сначала операторы языка транслируются в эквивалентные последовательности операторов МНЕМОКОДа, и затем транслятор с МНЕМОКОДа формирует соответствующую машинную программу.

Такая двухступенчатая организация существенно упрощает алгоритм трансляции и делает его более эффективным. Итак, одно из предназначений языка МНЕМОКОД – играть роль буферного языка при трансляции с языков высокого уровня.

Чем выше уровень языка программирования, чем дальше он отстоит от машинных команд, тем с большими потерями качества программ, изготавливаемых трансляторами, приходится мириться.

Программы, составленные опытным программистом на машинно-ориентированном языке, занимают значительно меньше памяти и выполняются, как правило, значительно быстрее, чем программы, реализующие тот же алгоритм, но "изготовленные" транслятором с языка высокого уровня (конечно это не относится к так называемым оптимизирующим трансляторам, получившим значительное распространение в последнее время). Отсюда следует вторая роль МНЕМОКОДа как языка для написания программ, к которым предъявляются повышенные требования по быстродействию и компактности.

Однако главное применение МНЕМОКОДа обусловлено, пожалуй, тем, что это единственное средство для программирования операций с непосредственным обращением к аппаратным ресурсам машины. Именно поэтому на МНЕМОКОДе и его расширениях написаны все программные модули АСПО (о расширениях МНЕМОКОДа см. гл. 8 и 13).

5.2. Элементы языка и методы его описания

Синтаксис, семантика, прагматика. Что значит владеть тем или иным языком программирования? Это значит уметь представлять средствами языка разнообразные алгоритмы, причем так, чтобы программы понимались машиной именно в том смысле, какой хотел вложить в нее автор.

Первый шаг к взаимопониманию лежит в области *синтаксиса* языка, который определяет множество всех допустимых конструкций: любая комбинация символов, не принадлежащая этому множеству, будет восприниматься транслятором как ошибочная с выдачей соответствующего сообщения на пульт оператора. Например, строка МНЕМОКОДа **ADD** *M* содержит синтаксическую ошибку, поскольку в числе операторов МНЕМОКОДа нет оператора с мнемоникой **ADD**.

Как уже говорилось, транслятор, обрабатывая каждый синтаксически правильный оператор МНЕМОКОДа, формирует либо машинную команду вместо операторов машинных команд, либо служебную информацию (области данных, константы и т.п.) вместо операторов псевдокоманд.

Соответствие между операторами языка и результатами их трансляции, а для операторов машинных команд и результатами выполнения на ЭВМ составляет *семантику* языка МНЕМОКОД. Например, семантика псевдокоманды **BSS** *10* состоит в том, что транслятор, обрабатывая эту конструкцию, резервирует 10 ячеек памяти, а семантика оператора **LDA** *1,* **X** состоит, во-первых, в формировании соответствующей адресной команды с признаком адресации через индексный регистр и единицей в качестве смещения, а во-вторых, определяется операцией (3.10) и алгоритмом формирования адреса операнда, описанным в § 3.1. Таким образом, от правильного понимания семантики языка во многом зависит, будет ли программа, "изготовленная" транслятором, вообще решать поставленную задачу.

С другой стороны, один и тот же алгоритм может быть запрограммирован далеко неоднозначно. Но одни программы при этом будут хорошими, а другие плохими. Качество программ зависит от ряда факторов, и многие из них также лежат в области семантики языка.

Что же касается трудоемкости разработки программ, то этот показатель зависит прежде всего от квалификации программиста и является достаточно субъективным потому, что одни программисты для той или иной алгоритмической структуры сразу находят наиболее подходящие средства языка, а для других это представляет существенные трудности. Рекомендации по практическому применению различных средств языка составляют его *прагматику.*

Итак, синтаксис языка определяет множество всех допустимых конструкций, семантика – что означают эти конструкции для транслятора и машины, прагматика – как и когда их применять.

Метаязык. Система обозначений, используемая для описания языка, называется *метаязыком.* Широкое распространение получил метаязык Бекуса. Именно на этом метаязыке был описан один из первых строго определенных языков программирования АЛГОЛ-60 [15].

Однако для описания машинно-ориентированных языков метаязык Бекуса неудобен в чистом виде. Используя его в качестве основы, введем некоторые дополнительные обозначения и соглашения.

Далее будем применять общую структуру синтаксических определений вида

$$\langle \text{k.l.m. } \textit{определяемый элемент} \rangle ::= \langle \textit{определение} \rangle , \qquad (5.1)$$

где ссылочный номер k.l.m состоит из номера параграфа k.l и порядкового номера m определения в данном параграфе; знак ::= читается как "есть по определению"; в угловые скобки заключаются элементы, не являющиеся *первичными* (К первичным элементам МНЕМОКОДа относятся мнемонические обозначения кодов операций и основные символы языка, перечисленные в табл. 2.3). В синтаксических определениях первичные элементы языка, так же как и постоянные (мнемоника КОП, указатели косвенной адресации, индексных регистров и др.), будут выделяться жирным шрифтом.

В правой части определений типа (5.1) будут также использоваться:

[ ] – квадратные скобки – заключенный в них элемент не является обязательным;

{ } – фигурные скобки – из заключенных в фигурные скобки обязателен один и только один элемент;

| – вертикальная черта – читается как "либо", этим знаком отделяются друг от друга синтаксически правильные варианты определяемой конструкции в правой части (5.1).

В тех случаях, когда формальное определение с использованием этих обозначений слишком громоздко, будем давать определение в словесной формулировке.

Основные элементы языка. Определим элементы языка, являющиеся в некотором смысле основными:

⟨5.2.1. *идентификатор*⟩ ::= ⟨5.2.2. *последовательность* длиной не более пяти ⟨5.2.3. *допустимых символов*⟩ и ⟨5.2.5. *цифр*⟩⟩, начинающаяся с *допустимого символа*

⟨5.2.2. *последовательность* ⟨*элементов* (любых)⟩⟩ ::= один ⟨*элемент*⟩| совокупность ⟨*элементов*⟩, выписанных друг за другом без каких-либо разделителей

⟨5.2.3. *допустимый символ*⟩ ::= ⟨5.2.4. *буква*⟩|**·|?|/|"|§|%|#|**

⟨5.2.4. *буква*⟩ ::= одна из букв латинского или русского алфавита

⟨5.2.5. *цифра*⟩ ::= ⟨5.2.6. *восьмеричная цифра*⟩|⟨5.2.7. *десятичная цифра*⟩

⟨5.2.6. *восьмеричная цифра*⟩ ::= **0|1|3|4|5|6|7**

⟨5.2.7. *десятичная цифра*⟩ ::= **0|1|2|3|4|5|6|7|8|9**

Как основной элемент языка МНЕМОКОДа *идентификатор* используется в программах для обозначения адресов ячеек памяти, например в качестве метки. Следующие примеры иллюстрируют определение ⟨5.2.1.⟩.

Правильные идентификаторы: *М* , *АЛЬФА* , *А.* , *#*

Неправильные идентификаторы: *1МК* – начинается не с ⟨5.2.3.⟩; *АЛЬФА1* – состоит более чем из пяти символов.

Наряду с идентификаторами для указания адресов используют *выражения* *М + 1*, ** – 2* и т.п.

⟨5.2.8. *выражение*⟩ ::= [{ **±** }] ⟨5.2.9. *терм*⟩ | совокупность *термов*, соединенных знаками { **±** }, причем перед первым термом может стоять знак { **±** }

⟨5.2.9. *терм*⟩ ::= ⟨5.2.1. *идентификатор*⟩ | ⟨5.2.10. *целое без знака*⟩ | ⟨ 5.2.13. *указатель текущего адреса* ⟩

⟨5.2.10. *целое без знака*⟩ ::= ⟨5.2.11. *десятичное целое без знака*⟩ | ⟨5.2.12. *восьмеричное целое без знака* ⟩

⟨5.2.11. *десятичное целое без знака*⟩ ::= ⟨5.2.2. *последовательность* ⟨ 5.2.7. *десятичных цифр* ⟩⟩

⟨5.2.12. *восьмеричное целое без знака*⟩ ::= ⟨5.2.2. *последовательность* ⟨ 5.2.6. *восьмеричных цифр* ⟩⟩ **B**

⟨5.2.13. *указатель текущего адреса*⟩ ::= ***

Цепочка определений ⟨5.2.8.⟩ –⟨5.2.13.⟩ задает все синтаксически допустимые *выражения* языка МНЕМОКОД. Видно, например, что частным случаем выражения являются идентификаторы и целые числа.

Рассмотрим семантику *выражения* языка МНЕМОКОД. Встретив *выражение* в тексте программы, транслятор вычисляет его *значение,* при этом:

значением *идентификатора* является адрес, на который указывает этот идентификатор;

значением *целого числа* является само это число;

значением *** (*указателя текущего адреса*) является адрес ячейки, в которой размещается результат трансляции оператора, содержащего этот указатель в поле операнда.

Здесь требуется уточнить, о каких адресах идет речь.

Машинная программа, получаемая в результате трансляции, может по выбору программиста иметь либо *абсолютный,* либо *перемещаемый* формат.

Программа в абсолютном формате – это программа, которая в процессе программирования *жестко* привязывается к конкретным адресам. Для правильной работы такой программы необходимо размещение ее в определенных ячейках памяти (неважно какого раздела). Программы в абсолютном формате не могут быть скомпонованы с другими программными модулями. Поэтому им недоступны, в частности, стандартные операции вызова супервизора АСПО, библиотечные подпрограммы вычисления элементарных функций, ввода-вывода и др. Прикладному

программисту СМ-2М программировать в абсолютном формате практически не приходится, поэтому больше к нему возвращаться не будем.

Основной формат программ – *перемещаемый.* Адреса ячеек программы в перемещаемом формате выдаются транслятором с точностью до начального адреса загрузки: перемещаемый адрес первой ячейки программы полагается равным нулю, второй ячейки – единице и т.д., причем настройку на реальные адреса загрузки выполняет специальная программа – *компоновщик программ.* Поэтому в дополнение к ранее введенным относительным и абсолютным адресам будем различать *перемещаемые адреса* – адреса ячеек программы в перемещаемом формате. В этом смысле идентификаторы, указывающие на ячейки программы, имеют перемещаемые значения, а указывающие на ячейки нулевой страницы, – абсолютные.

В соответствии с типами входящих в состав *выражения* термов оно может быть также либо абсолютным, либо перемещаемым. Например, разность двух идентификаторов, имеющих перемещаемые значения, будет абсолютной (по смыслу – число ячеек между соответствующими метками), а сумма такого идентификатора и целого числа – перемещаемой. Заметим, что синтаксически верное выражение может быть семантически лишено смысла. Например, нельзя разумно интерпретировать в виде адреса сумму двух перемещаемых термов.

В заключение данного параграфа определим еще два элемента МНЕМОКОДа, которые будут использоваться в дальнейшем:

⟨5.2.14. *список* ⟨*элементов* (любых)⟩⟩ ::= один ⟨*элемент*⟩ | совокупность ⟨*элементов*⟩, выписанных друг за другом через запятую (Сравните с ⟨5.2.2. *последовательностью*⟩)

⟨5.2.15. *основной символ*⟩::= произвольный символ из табл. 2.3.

Пример *списка* восьмеричных констант уже встречали в §3.4 в связи с псевдокомандой **ОСТ.**

5.3. Структура программ на МНЕМОКОДе

В общем случае программа на МНЕМОКОДе начинается с *управляющего оператора,* за которым следует оператор псевдокоманды определения *имени программы,* и включает исполняемую и неисполняемую части.

Исполняемая часть содержит в основном операторы машинных команд. Если в этой части используются операторы псевдокоманд, например, **DEF**, то необходимо следить, чтобы управление не попало на эту ячейку.

Исполняемая часть должна иметь одну или несколько *входных точек* (часто операторов **NOP**), через которые осуществляется обращение к программе со стороны других программ пользователя и модулей ОС.

Неисполняемую часть программы образуют константы и данные, а также участки памяти, зарезервированные для размещения рабочих ячеек и результатов. Здесь же перечисляются внешние метки и собственные входные точки (см. гл. 7).

Структура программы, в которой неисполняемая часть предшествует исполняемой, а последняя имеет одну входную точку, может быть представлена следующим образом:

```
ASMB,R,B,L,F                                          (5.2)
        NAM ОБЩВД,1,10,13,30,30
*       Неисполняемая часть
M       BSS 100
        . . .
*       Исполняемая часть
ВХТЧК   NOP
        . . .
        END ВХТЧК
```

Управляющий оператор. Всякая программа должна начинаться с управляющего оператора, который задает режим работы транслятора.

⟨5.3.1. *управляющий оператор*⟩ ::= **ASMB**, ⟨5.3.2. *указатель формата результирующей программы*⟩,⟨ 5.2.14, *список* ⟨5.3.3. *параметров управляющего оператора*⟩⟩

⟨5.3.2. *указатель формата результирующей программы*⟩ ::= **A**|**R**, где **A** – указатель абсолютного формата; **R** – указатель перемещаемого формата.

⟨5.3.3. *параметр управляющего оператора*⟩ ::= **B**|**L**|**T**|**N**|**Z**|**F**|**C**, где каждый параметр задает определенное указание транслятору: **B** – выдать результирующую программу на диск, точнее, на устройство с логическим номером 4 (см. гл. 10); **L** – напечатать листинг программы; **T** – напечатать таблицу идентификаторов; **N** – транслировать в результирующую программу все операторы между псевдокомандами условной трансляции **IFN** и **XIF**; **Z** – транслировать в результирующую программу все операторы между псевдокомандами **IFZ** и **XIF** (подробнее об условной трансляции см. гл. 7); **F** – указание сформировать команды вещественной арифметики (при отсутствии этого параметра операторы соответствующих команд транслируются как обращение к библиотечным подпрограммам); **C** – построить и распечатать *таблицу перекрестных ссылок* содержащую все идентификаторы программы с указанием для каждого из них номеров операторов, в которых идентификатор вводится и используется.

Например управляющий оператор в (5.2) задает трансляцию в перемещаемый формат с использованием команд вещественной арифметики, выдачей результирующей программы и печатью листинга.

Порядок следования параметров управляющего оператора несущественен, но повторения их в *списке* не допускаются.

Управляющий оператор должен начинаться с первой (крайней левой) позиции строки МНЕМОКОДа.

Оператор NAM$_1$ Этот оператор, который следует за управляющим, определяет имя программы, ее тип, приоритет и временные параметры запуска.

⟨ 5.3.4. *оператор псевдокоманды определения имени программы*⟩ ::= **NAM** ⟨5.3.5. *имя программы*⟩ [,⟨5.2.14. *список* ⟨5.3.6. *параметров оператора* **NAM**⟩⟩]

⟨5.3.5. *имя программы*⟩ ::= ⟨5.2.1. *идентификатор*⟩

Имя программы используется компоновщиком в протоколе компоновки, который выдается на пульт оператора. Не следует думать, что это имя обязательно совпадает с именем, под которым программа будет храниться на носителе. Последнее задается в команде оператора "завершить компоновку" и может быть произвольным.

⟨5.3.6. *параметр оператора* **NAM**⟩::=⟨5.3.7. *указатель типа программы*⟩|⟨5.3.9. *указатель приоритета*⟩|⟨5.3.10. *временной параметр запуска*⟩

В *списке* параметры оператора **NAM** указываются без повторения и в том порядке, в каком они перечислены в определении ⟨5.3.6.⟩.

⟨5.3.7. *указатель типа программы*⟩ ::= ⟨5.2.11. *десятичное целое без знака*⟩|⟨5.3.8. *пусто*⟩

⟨5.3.8. *пусто*⟩ ::= строка символов нулевой длины.

Указатель типа для системных программ полагается равным **0**, для главных программ (задач пользователя) – произвольному числу от **1** до **4**, для диск-резидентных программ (см. § 9.8) – **5**, для реентерабельных подпрограмм – **6** либо **7**, для реентерабельных – **16**.

Отметим, что только присваивание программе типа "главная", делает ее задачей. Число задач в многозадачном режиме определяется именно тем, сколько программ этого типа скомпоновано в один загрузочный модуль.

Программы типа **16** должны писаться в соответствии с рекомендациями § 3.4. Однако важно отметить, что для них резервирование области реентерабельности и формирование *РИ2* осуществляет компоновщик (в этом и состоит реакция компоновщика на указатель типа, равный **16**). Поэтому операторы (3.62) и (3.63) становятся ненужными.

Для обеспечения доступа реентерабельных привилегированных подпрограмм к произвольному разделу памяти, т.е. базирования по *РБ1*, компоновщик заносит на *РИ2* адрес области реентерабельности с признаком автоиндексации, поэтому реентерабельные подпрограммы пользователя следует начинать с установки в нуль знакового разряда *РИ2*.

Если указатель типа опущен, при трансляции он полагается равным нулю, т.е. программа считается системной. Указатель типа опускается вместе с предшествующей запятой, если за ним не следует никаких других параметров, иначе он принимает значение "пусто", и в списке параметров две запятые следуют друг за другом. Например, **NAM** *EXS 5.*

⟨5.3.9. *указатель приоритета*⟩ ::= ⟨5.2.11. *десятичное целое без знака*⟩|⟨5.3.8. *пусто*⟩

Чем меньшее значение имеет *указатель приоритета*, тем большим приоритетом "в борьбе за ресурсы машины" будет обладать данная задача. Если указатель приоритета опущен, он полагается транслятором равным *99*.

Указатель приоритета опускается также вместе с запятой, если он является последним в списке.

⟨5.3.10. *временной параметр запуска*⟩ ::= ⟨5.2.11. *десятичное целое без знака*⟩⟨5.3.8. *пусто*⟩

Первым из *временных параметров* в списке должен быть указан *масштаб* измерения временных интервалов: **1** (или **11**) задает масштаб 0,1 с, **2** (или **12**) – 1 с, **3** (или **13**) – 60 с и, наконец, **4** (или **14**) определяет, что время должно измеряться в часах.

Следующий временной параметр задает *интервал повторения* запуска в единицах, определенных масштабом.

Завершает список указатель, определяющий момент (*фазу*) начального запуска. Имеются два варианта задания фазы начального запуска.

Вариант первый – *задание абсолютной фазы*. Указатель фазы состоит из четырех временных параметров *ф1, ф2, ф3, ф4*, которые определяют соответственно в часах, минутах, секундах и десятых долях секунды тот момент текущего времени суток, когда должен произойти *первоначальный запуск* задачи. Признаком этого варианта является задание масштаба одной цифрой (**1, 2, 3** или **4**).

Второй вариант – задание *относительной фазы*. Указатель фазы состоит из одного временного параметра *ф1*, который в единицах, заданных масштабом, определяет интервал времени от момента ввода оператором УВК текущего времени суток до момента первоначального запуска задачи. Признаком этого варианта является задание масштаба двумя цифрами (**11, 12, 13** или **14**).

Теперь нетрудно понять, что оператор **NAM** из (5.2) определяет главную задачу пользователя (тип **1**) с десятым приоритетом, запускаемую по времени через каждые 30 мин.

Если первый временной параметр (масштаб) опущен, то должны быть опущены и остальные. В этом случае программа не запускается по времени, что характерно, например, для подпрограмм, вызываемых из других программных модулей.

Вслед за оператором **NAM** обычно следует неисполняемая часть программы, хотя некоторые программисты считают удобным располагать константы и данные в конце программы.

Исполняемая часть, как правило, начинается с помеченного оператора **NOP**, причем метка этого оператора является одной из входных точек программы. Механизм использования входных точек подпрограмм связан с семантикой команды **JSB** и рассматривался нами в § 3.4.

Оператор END. Последним оператором программы должен быть оператор **END** – оператор псевдокоманды завершения трансляции.

⟨5.3.11. *оператор псевдокоманды завершения трансляции*⟩ ::= **END** [⟨5.3.12. *метка запуска*⟩]

⟨5.3.12. *метка запуска*⟩ ::= ⟨5.2.1. *идентификатор*⟩

Метка запуска – это одна из входных точек программы. Если оператор **END** содержит метку запуска, то программа запускается на исполнение с соответствующей входной точки, иначе – со своей нулевой ячейки.

Две последующие главы посвящены синтаксису, семантике и прагматике операторов машинных команд и псевдокоманд.

Упражнения к гл. 5

1. Какой из последующих идентификаторов является синтаксически неверным? а) *???*; б) *.123*; в) *CM-2M*.

2. Определите тип и значение выражений: а) *100–100***B**; б) ***+10B**.

3. Покажите, что управляющий оператор **ASMB,R** не принадлежит множеству синтаксически верных конструкций.

Глава 6. ОПЕРАТОРЫ МАШИННЫХ КОМАНД

6.1. Операторы адресных команд основного формата

Формат операторов адресных команд в общем случае включает в себя метку, мнемоническое обозначение операции и операнд.

Естественно, что в языке МНЕМОКОД существуют средства, позволяющие задавать в поле операнда любой из описанных в третьей главе способов адресации. В дополнение к этому предусмотрена возможность *непосредственного указания данных*. Однако, как нетрудно видеть, не для всех команд это имеет смысл (сравните, например, команды **LDA** и **STA**). Последнее обстоятельство позволяет разделить адресные команды на две группы: операторы, в поле операнда которых допустимо указывать только адрес, и операторы, допускающие вместо адреса непосредственно указывать данные.

⟨6.1.1. *оператор адресной команды первой группы*⟩::=

[⟨6.1.2. *метка*⟩] { **JMP** / **JSB** / **ISZ** / **STA** / **STB** } { ⟨6.1.3. *адрес операнда*⟩ [,⟨6.1.4. *указатель косвенной адресации*⟩]. / ⟨6.1.5. *указатель адресации через индексный регистр*⟩ }

⟨6.1.2. *метка*⟩::=⟨5.2.1. *идентификатор*⟩

Как видно из определения ⟨6.1.1.⟩, оператор может быть *помеченным* либо *непомеченным*. Транслятор, встретив идентификатор в поле метки, помещает его в таблицу идентификаторов программы.

Каждый элемент таблицы помимо собственно идентификатора (совокупности символов) содержит его *значение и указатель типа*.

Значение идентификатора-метки полагается равным перемещаемому адресу ячейки, в которую записывается соответствующая машинная команда.

Метки операторов машинных команд имеют тип *перемещаемых в программе.* Кроме того, в МНЕМОКОДе существуют средства образования идентификаторов, *перемещаемых в общей области,* а также *неперемещаемых,* т.е. имеющих абсолютные значения. В программах перемещаемого формата абсолютные идентификаторы могут иметь значения только в пределах нулевой страницы, т.е. от 0 до 1777.

Для примера рассмотрим фрагмент программы:

```
ASMB,R,L                                                    (6.1)
        NAM  EXS     Обнуление счетчика ячеек
A       BSS  20B     Резервирование ячеек 0–17
BXEXS   NOP          В ячейку 20 записывается 0
M1      LDA  A
        ...
```

Здесь при трансляции оператора

M1 **LDA** *A*

будет сформирован элемент таблицы идентификаторов *M1,* имеющий перемещаемое в программе значение 21_8,

⟨6.1.3. *адрес операнда*⟩::= ⟨5.2.8. *выражение*⟩

Синтаксически в качестве адреса операнда допускается любое *выражение* МНЕМОКОДа. Однако из семантики адресных команд ясно, что оно должно быть неотрицательным и корректным по типу (см. § 5.2.). Кроме того, абсолютное выражение в поле адреса должно адресовать в пределах нулевой страницы (в программах пользователей — только для чтения информации). Записывать информацию в непривилегированном режиме можно по абсолютным адресам от 0 до 3, которые соответствуют, как уже говорилось, аккумуляторам и индексным регистрам процессора.

⟨6.1.4. *указатель косвенной адресации*⟩::= **I**

⟨6.1.5. *указатель адресации через индексный регистр*⟩::= ⟨6.1.6. *смещение*⟩,⟨6.1.7. *указатель индексного регистра*⟩

⟨6.1.6. *смещение*⟩::= ⟨5.2.8. *выражение*⟩

⟨6.1.7. *указатель индексного регистра*⟩::= **X | Y**

Встретив оператор адресной команды, транслятор анализирует способ адресации, определяет значение *адреса операнда* или *смещения,* строит соответствующую машинную команду и заносит ее в очередную ячейку программы. Таким образом, при трансляции оператора адресной команды основного формата занимается одна ячейка памяти, т.е. значение текущего адреса * увеличивается на единицу.

Из семантики ясно, что смещение должно иметь абсолютное значение, причем при автоиндексации в диапазоне $-400 \div +377_8$, а при индексации в диапазоне $0-777_8$.

Анализируя определение ⟨6.1.1⟩ операторов первой группы, внимательный читатель, наверняка, усмотрел некоторое противоречие: синтаксически верные операторы команд **JMP** и **JSB** с указателем адресации через индексные регистры семантически недопустимы, так как не реализованы в логических схемах процессора.

Переходим к рассмотрению операторов адресных команд второй группы.

⟨6.1.8. *оператор адресной команды второй группы*⟩ ::=

[⟨6.1.2. *метка*⟩] { **ADA** / **ADB** / **CPA** / **CPB** / **AND** / **XOR** / **IOR** / **LDA** / **LDB** } { ⟨6.1.3. *адрес операнда*⟩ [,⟨6.1.4. *указатель косвенной адресации*⟩] / ⟨6.1.5. *указатель адресации через индексный регистр*⟩ / ⟨6.1.9. *однословный литерал*⟩ }

В **отличие от** ⟨6.1.1.⟩ здесь добавлены *литералы,* которые используются для непосредственного указания данных.

⟨6.1.9. *однословный литерал*⟩::= ⟨6.1.10. *десятичный литерал*⟩|⟨6.1.11. *восьмеричный литерал* ⟩|⟨6.1.12. *Символьный литерал* ⟩|⟨6.1.13. *литерал типа абсолютное выражение* ⟩

⟨6.1.10. *десятичный литерал*⟩::= **=D** [{ ± }] ⟨5.2.11. *десятичное целое без знака*⟩

⟨6.1.11. *восьмеричный литерал* ⟩ ::= **=B** [{ ± }] ⟨ 5.2.2. *последовательность* ⟨5.2.6. *восьмеричных цифр* ⟩⟩

⟨6.1.12. *символьный литерал*⟩ ::= **=A**⟨5.2.15. *основной символ*⟩| **=A** ⟨основной символ⟩⟨*основной символ*⟩

⟨6.1.13. *литерал типа абсолютное выражение*⟩::= **=L**⟨5.2.8. *выражение*⟩

В определениях ⟨6.1.9.⟩ – ⟨6.1.13⟩ пары символов **=D, =B, =A** и **=L** есть не что иное, как *спецификаторы литералов.*

В адресных командах основного формата с помощью литерала можно задать операнд в виде восьмеричного или десятичного числа (например, **AND =B***7* или **ADA =D***–123,* один или два символа (**LDA =A***A* или **CPA =A***AB*) и, наконец, вычисляемую величину, заданную выражением абсолютного типа: **LDB =L***M2–M1.*

Встретив литерал, транслятор заводит в служебной области программы (т.е. не изменяя значения текущего адреса) ячейку, в которую записывает соответствующее литералу машинное слово, и затем использует адрес этой ячейки при формировании машинной команды. (Заметим, что если символьный литерал состоит из одного символа, то в младший байт машинного слова записывается код пробела). Таким образом, аппарат литералов экономит, отнюдь, не память машины, а трудоемкость программирования, избавляя от необходимости заводить соответствующие константы.

Если в программе встречаются несколько совпадающих по машинному представлению литералов, то транслятор распознает эту ситуацию и строит соответствующее машинное слово всего 1 раз.

Доступен ли адрес литерала программисту? При рассмотрении псевдокоманды **DEF** (см. гл. 7) получим положительный ответ на этот вопрос.

Наряду с литералами однословными в МНЕМОКОДе реализованы ⟨6.2.2. *двухсловные литералы*⟩, которые могут указываться в качестве операнда в тех адресных командах двойного формата, которые оперируют со словами двойной длины.

6.2. Операторы адресных команд двойного формата

Синтаксически адресные команды двойного формата строят из тех же элементов языка МНЕМОКОД, что и команды основного формата. Более того, синтаксис операторов команд **DST, STX, STY, LDW, STW** и **TAS** соответствует определению ⟨6.1.1.⟩, а команд **MPY, DIV, LBN, LBP, LDX** и **LDY** – определению ⟨6.1.8.⟩.

Новый элемент МНЕМОКОДа – ⟨6.2.2. *двухсловный литерал*⟩ – входит в адресную часть операторов команд вещественной арифметики.

⟨6.2.1. *оператор команды вещественной арифметики*⟩ ::=

$$[\langle 6.1.2.\ \textit{метка}\rangle]\ \left\{\begin{array}{l}\textbf{FAD}\\ \textbf{FSB}\\ \\ \textbf{FMP}\\ \\ \textbf{FDV}\end{array}\right\}\ \left\{\begin{array}{l}\langle 6.1.3.\ \textit{адрес операнда}\rangle\\ [,\langle 6.1.4.\ \textit{указатель косвенной адресации}\rangle]\\ \langle 6.1.5.\ \textit{указатель адресации через индексный регистр}\rangle\\ \langle 6.2.2.\ \textit{двухсловный литерал}\rangle\end{array}\right\}$$

⟨6.2.2. *двухсловный литерал*⟩::= **=F**⟨6.2.3. *дробное десятичное число*⟩

⟨6.2.3. *дробное десятичное число*⟩::= [{ ± }]⟨6.2.4. *дробное десятичное без знака*⟩

⟨6.2.4. *дробное десятичное без знака*⟩::=*n* [**E***p*] | *n*.[**E***p*] | *n.n*[**E***p*] | *n* [**E***p*]
где *n* – ⟨5.2.11. *десятичное целое без знака*⟩; *p* – ⟨6.2.5. *десятичный порядок*⟩.

⟨6.2.5. *десятичный порядок*⟩::= [{ ± }]⟨5.2.11. *десятичное целое без знака*⟩.

Таким образом, команда **FAD =F**.*5***E**–*6* означает, например, сложение (*РАБ*) с числом $0{,}5 \cdot 10^{-6}$.

Определение ⟨6.2.1⟩ задает также синтаксис команды **DLD**.

В заключение заметим, что, поскольку адресные команды двойного формата кодируются в двух ячейках, при трансляции соответствующих операторов счетчик текущего адреса меняется на два. Это необходимо, в частности, учитывать при организации условных пропусков:

Правильно	**Неправильно**
SSA	**SSA**
JMP * *+3*	**JMP** * *+2*
DLD **=F**.*5*	**DLD** **= F.***5*

6.3. Операторы регистровых команд

Регистровые команды позволяют осуществлять *сброс, инверсию* и (или) *инкрементирование* аккумуляторов, а также различные *сдвиги* содержимого аккумуляторов, в том числе вместе с *РР*, который при этом является как бы дополнительным старшим разрядом *РА* или *РБ*. Кроме того, в этой группе команд кодируются условные ПСК по знаку, равенству нулю, четности содержимого аккумуляторов и другим признакам.

Формат регистровых команд представлен на рис. 6.1. Нули в четырех старших разрядах являются признаком группы регистровых команд. Если четвертый разряд равен нулю, то преобразованию подвергается (*РА*), равен единице – операндом является (*РБ*). Пятый разряд регистровой команды – признак подгруппы: единица – для команд изменений и пропусков, нуль – для команд сдвигов слов основного формата. Разряды 6–15 – различные сочетания элементарных операций, которые выполняются за один машинный цикл. Время выполнения одной регистровой команды 1,9 мкс.

Изменения и пропуски. В одной команде может быть одновременно закодировано до *восьми* элементарных операций изменений и пропусков. Это достигается тем, что КОП в команде интерпретируется логическими схемами процессора как восемь независимых полей (рис. 6.2.).

В первом поле (разряды 6, 7) кодируются:

01 – **сброс аккумулятора** (РА или РБ в зависимости от значения четвертого разряда):

$$\text{CLA/B} \ \langle = \rangle \ РА/Б := 0, \tag{6.2}$$

10 – **инверсия аккумулятора**:

$$\text{CMA/B} \ \langle = \rangle \ РА/Б := (РА/Б) \oplus 177777, \tag{6.3}$$

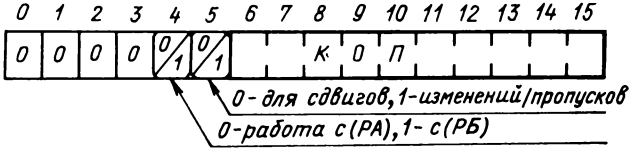


Рис. 6.1. Формат регистровой команды

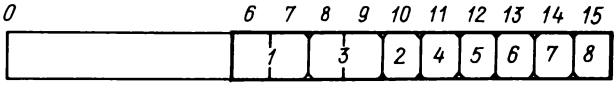


Рис. 6.2. Формат кода операции в командах изменений и пропусков. КОП разбит на поля с номерами 1–8

где операция $\oplus$ – исключающее или определена в § 3.2,
11 – **сброс и инверсия аккумулятора:**

$$\text{CCA/B} \Leftrightarrow PA/Б := 177777 = -1. \tag{6.4}$$

Единица во втором поле (разряд 10) задает **пропуск по регистру расширения** РР:

$$\text{SEZ} \Leftrightarrow \text{если } (PP) = 0 \text{ то ПСК.} \tag{6.5}$$

В третьем поле (разряды 8, 9) кодируются:
01 – **сброс регистра расширения,**

$$\text{CLE} \Leftrightarrow PP := 0; \tag{6.6}$$

10 – **инверсия регистра расширения,**

$$\text{CME} \Leftrightarrow PP := (PP) \oplus 1; \tag{6.7}$$

11 – **сброс и инверсия регистра расширения,**

$$\text{CCE} \Leftrightarrow PP := -1. \tag{6.8}$$

Единица в четвертом поле (разряд 11) задает условный **пропуск по знаку аккумулятора:**

$$\text{SSA/B} \Leftrightarrow \text{если } (PA/Б) \geqslant 0 \text{ то ПСК,} \tag{6.9}$$

а в пятом поле (разряд 12) кодируется условный **пропуск по четности,** т.е. по нулю младшего разряда $(PA/Б)$:

$$\text{SLA/B} \Leftrightarrow \text{если (младший разряд } PA/Б) = 0 \text{ то ПСК.} \tag{6.10}$$

Шестое поле (разряд 13) отведено для кодировки операции **инкрементирования аккумулятора:**

$$\text{INA/B} \Leftrightarrow PA/Б := (PA/Б) + 1. \tag{6.11}$$

При **равенстве аккумулятора нулю пропуск** будет осуществлен при наличии единицы в седьмом поле (разряд 14):

$$\text{SZA/B} \Leftrightarrow \text{если } (PA/Б) = 0 \text{ то ПСК.} \tag{6.12}$$

Наконец, если программисту удобнее использовать в операциях (6.5), (6.9), (6.10) и (6.12) *обращенное условие пропуска:* (РР) $\neq 0$, (РАБ) $<$ < 0 и т.д., то он может к команде пропуска добавить **операцию реверса RSS**, которая кодируется в восьмом поле, т.е. единицей разряда 15. Например, оператор регистровой команды

$$\text{SSA,RSS} \Leftrightarrow \text{если } (PA) < 0 \text{ то ПСК.} \tag{6.9а}$$

Нули в разрядах того или иного поля означают отсутствие соответствующей операции, а нули во всех разрядах одновременно воспринимаются как регистровая команда, не вызывающая никаких действий. В такую вырожденную команду транслируется **оператор NOP**, которым обычно резервируют ячейки для входных точек программ.

Описанный выше формат команд изменения и пропусков непосредственно определяет синтаксис соответствующих операторов МНЕМОКОДа:

⟨6.3.1. *оператор команды изменений и пропусков*⟩::=

[⟨6.1.2. *метка*⟩] { **NOP** | ⟨6.3.2. *непомеченный оператор команды изменений и пропусков с операндом в регистре А*⟩ | ⟨6.3.3. *непомеченный оператор команды изменений и пропусков с операндом в регистре Б*⟩ }

⟨6.3.2. (6.3.3) *непомеченный оператор команды изменений и пропусков с операндом в регистре А (регистре Б)*⟩ ::= ⟨5.2.14. *список* ⟨6.3.4. (6.3.5) *элементарных операций изменения/пропуска с операндом в регистре А (регистре Б)*⟩⟩ длиной не более восьми и не содержащий повторяющихся элементов

⟨6.3.4. *элементарная операция изменения/пропуска с операндом в регистре А*⟩::= { **CLA** | **CMA** | **CCA** } | **SEZ** | { **CLE** | **CME** | **CCE** } | **SSA** | **SLA** | **INA** | **SZA** | **RSS**

⟨6.3.5. *элементарная операция изменения/пропуска с операндом в регистре Б*⟩::= { **CLB** | **CMB** | **CCB** } | **SEZ** | { **CLE** | **CME** | **CCE** } | **SSB** | **SLB** | **INB** | **SZB** | **RSS**

В соответствии с форматом команд изменений и пропусков операции в *списке,* состоящем более чем из одного элемента, должны следовать в том порядке, в каком они перечислены в определениях ⟨6.3.4.⟩ и ⟨6.3.5.⟩.

Семантика сочетания различных условий пропуска состоит в ПСК по их логическому сложению (дизъюнкции) : если хотя бы одно из указанных условий выполняется, то следующая команда пропускается. В операторах с **RSS** пропуск осуществляется по дизъюнкции отрицаний этих условий. Исключение составляет сочетание в одной команде операций **SSA** и **SLA** или **SSB** и **SLB**, для которых реверс **RSS** не только отрицает условия пропуска, но и превращает дизъюнкцию в конъюнкцию. Например:

SEZ,SSA,SLA,RSS⟨⇒

если $(PP) \neq 0 \cup (PA) < 0 \cap ((PA)$ – нечетно) **то** ПСК.

Семантика сочетания элементарных операций изменения с операциями условного пропуска состоит в их последовательном выполнении в соответствии с порядком перечисления в операторе команды. Например, если $(PA) = 0$, то команда **SLA,INA** сначала настраивает *РНК* на пропуск следующей команды, а затем увеличивает (PA) на 1.

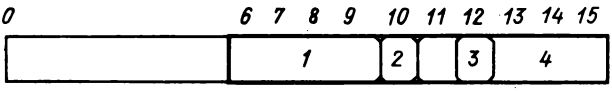


Рис. 6.3. Формат кода операции в командах сдвига слов основного формата (поле 3 справа и слева окружено разрядами поля 4)

Сдвиги слов основного формата. Если в пятом разряде регистровой команды записан нуль, то оно логическими схемами процессора анализируется как *команда сдвигов*. Это значит, что КОП в команде (разряды 6–15) интерпретируется уже не как восемь, а как четыре независимых поля (рис. 6.3) элементарных операций сдвига.

Сдвиг на один разряд представляет собой такое преобразование машинного слова, при котором каждый бит информации как бы переносится в соседний правый или левый, в зависимости от направления сдвига, разряд. Повторяя эту операцию, можно получить сдвиг на два, три и большее число разрядов.

В машинно-ориентированных языках обычно реализуют сдвиги трех типов: *арифметические, логические* и *циклические.*

Арифметические сдвиги эквивалентны делению или умножению на целую степень двойки. Естественно, что здесь знаковый разряд должен оставаться неподвижным.

В логических сдвигах знаковый разряд участвует на равных с другими разрядами машинного слова. Поэтому логические сдвиги имеют смысл только для нечисловых данных. В системе команд архитектурной линии СМ-2М для слов основного формата реализован только логический сдвиг *влево.*

Для нечисловых данных также применяют различные циклические сдвиги. Они отличаются от логических тем, что биты, "выталкиваемые" при сдвигах за разрядную сетку, не пропадают, а записываются на освободившиеся места: из младших разрядов – в старшие (при сдвигах вправо) и из старших – в младшие (при сдвигах влево).

Объектом сдвига в СМ-2М может быть либо (*РА/Б*) – содержимое аккумулятора *РА* или *РБ*, либо (*РРА/Б*) – содержимое аккумулятора, дополненное слева содержимым *РР* (так называемые сдвиги вместе с регистром расширения), либо, наконец, (*РБА*) – двойное машинное слово (*сдвиги слов двойной длины*).

Вернемся к формату КОП, представленному на рис. 6.3. Элементарные операции сдвигов, которые могут быть закодированы в первом поле этого формата, сведены в табл. 6.1.

Этот же набор элементарных операций кодируется и в четвертом поле, под которое отведены разряды 11, 13–15. Наконец, во втором и третьем полях могут задаваться уже известные операции изменения и пропуска: единица в разряде 10 задает операцию **CLE**, и единица в разряде 12 – операцию **SLA/B**.

Таблица 6.1

Наименование операции	Мнемоника	КОП* (разряды 6–9)
Арифметический сдвиг (*РА/Б*) влево на один разряд	**A/BLS**	1000
Арифметический сдвиг (*РА/Б*) вправо на один разряд	**A/BRS**	1001
Циклический сдвиг (*РА/Б*) влево на один разряд	**RA/BL**	1010
Циклический сдвиг (*РА/Б*) вправо на один разряд	**RA/BR**	1011
Логический сдвиг (*РА/Б*) влево на один разряд	**A/BLR**	1100
Циклический сдвиг (*РРА/Б*) вправо на один разряд	**ERA/B**	1101
Циклический сдвиг (*РРА/Б*) влево на один разряд	**ELA/B**	1110
Циклический сдвиг (*РА/Б*) влево на четыре разряда	**A/BLF**	1111

*КОП в пределах 0000 – 0111 эквивалентен отсутствию операции

Таким образом, общее число операций в команде сдвигов слов основного формата не может быть больше четырех, из них – собственно сдвигов – не более двух. Синтаксическое определение оператора этих команд аналогично ⟨6.3.1⟩, но с той разницей, что *список элементарных операций* общим числом не более четырех может включать одну из элементарных операций сдвига из табл. 6.1, операцию **CLE** и условный пропуск **SLA/B**, за которым снова может следовать одна из элементарных операций сдвига.

Приведем наиболее употребительные комбинации операций сдвигов:

ALF,ALF – циклический сдвиг (*РА*) на один байт, в итоге старший и младший байты (*РА*) меняются местами;

ELA,CLE,ERA – сброс (установка в нуль) знакового разряда (*РА*);

RAL,SLA – организация логической шкалы [в соответствии с (6.10) применение данной команды в цикле обеспечит ПСК на i-м шаге цикла, если значение i-го разряда первоначального содержимого *РА* равно нулю].

В заключение рассмотрим *операторы команд сдвигов слов двойной длины,* или, точнее, команды сдвигов (*РБА*).

Формат этих команд представлен на рис. 6.4. Разряды 4, 6, 9–11 отведены для кода операции и четыре разряда (12–15) – для указания числа разрядов, на которое должен быть произведен сдвиг. (Четыре нуля в этих разрядах воспринимаются как указание сдвинуть РБА на 16 разрядов.) Информационная емкость пяти разрядов КОП используется на 10% – всего реализовано шесть различных сдвигов слов двойной длины, которые представлены в табл. 6.2.

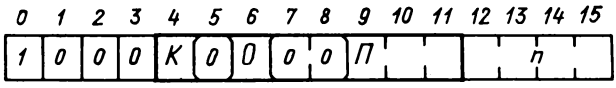


Рис. 6.4. Формат команд сдвига слов двойной длины:
n – число разрядов (величина сдвига)

Таблица 6.2

Наименование команды	Мнемоника	КОП (4, 6, 9–11 разряды команды)
Арифметический сдвиг (*РБА*) вправо на *n* разрядов	**ASR**	01001
Арифметический сдвиг (*РБА*) влево на *n* разрядов	**ASL**	00001
Логический сдвиг (*РБА*) вправо на *n* разрядов	**LSR**	01010
Логический сдвиг (*РБА*) влево на *n* разрядов	**LSL**	00010
Циклический сдвиг (*РБА*) вправо на *n* разрядов	**RRR**	01100
Циклический сдвиг (*РБА*) влево на *n* разрядов	**RRL**	00100

Синтаксис операторов рассматриваемых команд задается следующим определением:

⟨ 6.3.6. *оператор команды сдвига слова двойной длины* ⟩ ::=

[⟨ 6.1.2. *метка* ⟩] { **ASR** | **ASL** | **LSR** | **LSL** | **RRR** | **RRL** } ⟨ 6.3.7. *указатель числа разрядов* ⟩

⟨ 6.3.7. *указатель числа разрядов* ⟩ ::=⟨ 5.2.10. *целое без знака* ⟩

Указатель числа разрядов должен быть в пределах от 1 до 16, причем 16 транслируется как четыре нуля в разрядах 12–15 соответствующей команды (рис. 6.4). Используя команды сдвигов слов двойной длины, можно умножить или разделить (*РБА*) на 2^n ($n = 1, \ldots, 16$), поменять местами содержимое *РА* и *РБ* (**RRR** *16* или **RRL** *16*), организовать 32-разрядную логическую шкалу (например, командой **LSL** *1*) и т. д.

6.4. Операторы команд ввода-вывода

Большинство описанных в гл. 4 команд ввода-вывода можно разделить на три группы. Синтаксис операторов команд *первой группы* обусловливается не зафиксированным значением шестого разряда этих команд: с единицей в шестом разряде команда дополнительно к основной операции возбуждает сигнал на шине *ВП-К*. Программист регулирует значение шестого разряда, задавая или опуская в поле операнда *указатель сигнала выполнить*.

⟨ 6.4.1. *оператор команды ввода-вывода первой группы* ⟩ ::=

[⟨ 6.1.2. *метка* ⟩] { **MIA** | **MIB** | **LIA** | **LIB** | **OTA** | **OTB** | **CLC** | **STC** } ⟨ 6.4.2. *код выборки* ⟩ [, ⟨ 6.4.4. *указатель сигнала выполнить* ⟩]

Таблица 6.3

Наименование команды	Мнемоника	Команда-эквивалент
Установка *РП* в единицу	**STO**	**STF** *1*
Сброс *РП* в нуль	**CLO**	**CLF** *1*
Установить нижнюю границу : *РНГ*: = (8 мл.разряд *РА/Б*)	**LA/BI**	**OTA/B** *5*
Установить верхнюю границу *РВГ*: = (8 ст.разряд *РА/Б*)	**LA/BS**	**OTA/B** *6*
Вызов супервизора: *РНК*: = 6, *ТС*: = 0	**SVC**	**STC** *6*
Динамический останов (ожидание прерывания)	**DWT**	**STF** *6*
Пропуск по номеру процессора (первого)	**SPF**	**SFS** *6*
Пропуск по номеру процессора (второго)	**SPS**	**SFC** *6*

⟨ 6.4.2. *код выборки* ⟩ ::= ⟨ 5.2.10. *целое без знака* ⟩ | ⟨ 6.4.3. *идентификатор внешней константы* ⟩

⟨ 6.4.3. *идентификатор внешней константы* ⟩ ::= ⟨ 5.2.1. *идентификатор* ⟩

Естественно, что значение кода выборки не должно превосходить 77_8. Что касается *внешних констант,* то их значения задаются в процессе компоновки задачи, а объявление идентификаторов внешних констант осуществляется специальной псевдокомандой **ESC** (см. гл. 7).

Заметим, что внешние константы в поле операнда команд ввода-вывода позволяют писать и транслировать программы, не привязывая их к конкретным кодам выборки УВВ.

⟨ 6.4.4. *указатель сигнала выполнить* ⟩ ::= **C**

У команд *второй группы* значение шестого разряда фиксировано, поэтому указатель сигнала выполнить не допускается:

⟨ 6.4.4. *оператор команды ввода-вывода второй группы* ⟩ ::=

[⟨ 6.1.2. *метка* ⟩] { **CLF** | **STF** | **SFS** | **SFC** | **RIO** } ⟨ 6.4.2. *код выборки* ⟩

Третью группу образуют операторы команд ввода-вывода со *специальными кодами выборки,* имеющие в МНЕМОКОДе собственные мнемонические обозначения.

Эти команды сведены в табл. 6.3.

Синтаксически операторы перечисленных в табл. 6.3 команд состоят из метки (которой может не быть) и мнемонического обозначения, помещаемого в поле кода операций. Поле операнда не используется. Например, оператор команды динамического останова, помеченный меткой *M1*, имеет вид

M1 **DWT**

В заключение отметим, что операторы трех команд **SOS** (**пропуск по единице на** *РП*, эквивалентна команде **SFS** *1*), **SOC** (**пропуск по нулю на** *РП*, эквивалентна команде **SFC** *1*) и **HLT** (**останов процессора**), кодируемых в формате команд ввода-вывода, отличаются от операторов описанных выше синтаксических групп. В частности, поле операнда команд **SOS** и **SOC** может быть либо пустым, либо содержать указатель единицы шестого разряда: например, **SOS** и **SOS C**. При наличии указателя **C** выполнение этих команд начинается с операций (4.13), (4.14) и завершается операцией (4.11) – обнулением *РП*. Наконец, поле операнда команды **HLT** может быть либо пустым, либо содержать код выборки, который в сочетании с командой **HLT** никаких сигналов на шинах интерфейса не возбуждает, а играет роль указателя причин останова. (Имеется в виду, что программист для индикации на инженерной панели процессора различных причин останова будет использовать в программе команду **HLT** с различными кодами выборки.)

6.5. Операторы команд дополнительного набора

Дополнительный набор команд СМ-2М включает более 60 команд различного назначения, которые могут быть разделены на команды преобразования форматов, дополнительные системные привилегированные и непривилегированные команды, операции с битами, операции пересылок и сравнения массивов и дополнительные операции с индексными регистрами.

Далеко не все из данного множества представляет непосредственный интерес для прикладного программиста, поэтому ниже рассмотрим наиболее употребительные в программах обработки информации команды и представляющие их операторы языка МНЕМОКОД.

Преобразование форматов. Преобразование числа с плавающей запятой в целое число осуществляется командой **FIX** – однословной безадресной командой, имеющей восьмеричный код 105100. По этой команде число, представленное мантиссой и порядком в РАБ, преобразуется в 16-разрядное целое число (отрицательное – в дополнительном коде) и размещается на РА:

FIX ⟨=⟩ **если** $-2^{16} \leqslant (РАБ) \leqslant 2^{16} - 1$ **то** $РА :=$ **целая часть** $(РАБ)$
иначе ($РА :=$ **максимальное целое, совпадающее по знаку с** $(РАБ)$; $РП := 1$). (6.13)

Содержимое *РБ* по окончании этой операции непредсказуемо.

Обратная операция осуществляется командой **FLT** (восьмеричный код 105120).

FLT ⟨=⟩ $РАБ :=$ **нормализованное число, представленное мантиссой и порядком, совпадающее по величине с** (РА). (6.14)

Определим синтаксис операторов описанных выше команд:

⟨ 6.5.1. *оператор команды преобразования форматов* ⟩ ::=

[⟨ 6.1.2. *метка* ⟩] { **FIX** | **FLT** } .

В связи с тем что команды **FIX** и **FLT** безадресные, поле операнда должно быть пустым.

Команды пересылок и сканирования байтов. Если выше во всех командах адресуемой единицей информации было машинное слово, то теперь познакомимся с командами, которые могут работать с *отдельными байтами.*

Относительный *адрес байта* (по базовому адресу раздела) в архитектуре СМ-2М задается 16-разрядным машинным словом. При этом 15 старших разрядов задают относительный адрес ячейки, а младший разряд определяет, какой из двух байтов этой ячейки адресуется: единица задает младший, т. е. правый байт, нуль – старший, т. е. левый. Таким образом, адреса двух соседних байтов (неважно, старшего и младшего в одной ячейке или младшего в предыдущей и старшего в последующей) всегда отличаются на единицу.

Однословная **команда LBT** (код 105763) осуществляет засылку байта, относительный адрес которого находится на *РБ*, в младшие разряды *РА*. При этом старшие разряды *РА* устанавливаются в 0, а (*РБ*) увеличивается на 1, настраиваясь тем самым на относительный адрес соседнего байта. (Далее легко можно сообразить, как использовать эту команду, чтобы "распаковать" $2n$ байтов, занимающих n ячеек, и передать на устройство отображения по одному байту на каждую посылку.)

Обратную операцию (упаковку) можно осуществить, использовав **команду SBT** (код 105764). По этой команде содержимое восьми младших разрядов *РА* заносится по адресу, указанному на *РБ*. Смежный с адресуемым байт при этом не изменяется. (*РБ*) увеличивается на 1.

Следующая **команда SFB** (код 105767) – *сканирование байтов* – запускает на выполнение нетривиальный циклический алгоритм, записанный на ПЗУ. В соответствии с этим алгоритмом выполняется сравнение последовательности байтов (байт за байтом), начиная с байта, адрес которого указан в *РБ*, с байтом проверки – младшим байтом (*РА*) и с байтом окончания – старшим байтом (*РА*). Выполнение команды заканчивается, если очередной байт последовательности совпадает с байтом проверки или байтом окончания.

Если выполнение команды завершится по совпадению с байтом проверки, в *РБ* устанавливается адрес байта последовательности, равного байту проверки, после чего выполняется следующая за **SFB** команда.

Если первым обнаруживается совпадение с байтом окончания, следующая за **SFB** команда пропускается, а в *РБ* устанавливается адрес,

Таблица 6.4

Наименование команды	Мнемоника	Код команды (содержимое первого слова)
Пересылка массива байтов	**MBT**	105765
Пересылка массива слов	**MVW**	105777
Сравнение массива байтов	**CBT**	105766
Сравнение массива слов	**CMW**	105776

на единицу превышающий адрес того байта, который совпал с байтом окончания.

Может случиться так, что в сканируемой последовательности не встретится ни байт проверки, ни байт окончания. Тогда выполнение продолжается до обращения по относительному нулевому адресу (формируется при игнорировании переноса из старших разрядов адреса), который и есть адрес старшего байта *РА*, т. е. байта окончания.

Синтаксис операторов **LBT**, **SBT** и **SFB** по формату не отличается от заданного определением ⟨ 6.5.1.⟩ .

Команды пересылки и сравнения массивов (табл. 6.4). Команды данной группы также реализованы на ПЗУ в виде циклического процесса, но в отличие от команды **SFB** имеют трехсловный формат, причем в первом слове записывается код команды, во втором – адрес ячейки, содержащей длину массива-операнда, а третье резервируется для текущего числа обработанных элементов массива (может быть использовано, если в процессе исполнения команды возникнет прерывание), и в нее при трансляции заносится нуль.

На МНЕМОКОДе эти команды записываются двумя операторами, первый из которых транслируется в две ячейки (код команды и код адреса), а второй должен резервировать нулевую константу, например **NOP**.

Для каждой из представленных в табл. 6.4 команд операндами являются две непрерывные области памяти одинакового объема. В командах *пересылки* информация из ячеек или байтов одной области памяти переписывается в ячейки или байты другой области памяти. В командах *сравнения* происходит поэлементное сравнение массивов информации и формирование *РНК* по результатам этого сравнения.

Ясно, что для задания двух областей памяти равной длины нужно три числа: два начальных адреса и число элементов (в словах или байтах). В архитектуре СМ-2М принято, что перед исполнением команд данной группы начальные адреса первой и второй областей памяти должны быть записаны на *РА* и *РБ* соответственно, причем для команд, работающих с байтами, записывается 16-разрядный относительный адрес первого байта (организованный, как описано выше), а для команд, работающих со словами, – обычный 15-разрядный адрес первой ячейки.

Что касается числа элементов, то эта величина, как уже говорилось, должна быть записана в ячейке памяти, адрес которой кодируется во втором слове команды, причем кодируется точно так же, как и в адресных командах двойного формата (см. рис. 3.2), с $Д = 0$ для команд **CBT** и **CMW** и с $Д = 1$ для команд **MBT** и **MVW**.

Отсюда ясно, что синтаксис первого оператора команд **MBT** и **MVW** совпадает с синтаксисом адресных команд, заданным определением ⟨ 6.1.8.⟩, а для команд **CBT** и **CMW** в операторах МНЕМОКОДа нельзя использовать указатель адресации через индексный регистр.

Пример записи на МНЕМОКОДе:

```
N     BSS  1      ⟨n⟩ – число элементов                    (6.14)
      ...
      CMW  N  }
      NOP     }   трехсловная команда CMW.
```

При выполнении команд **MBT** и **MVW** нормальное (т. е. без прерываний) завершение операции сводится к пересылке всего заданного массива информации и увеличению (*РА*) и (*РБ*) на число пересланных элементов (т. е. на содержимое адресуемой ячейки).

При выполнении команд **CBT** и **CMW** происходит последовательное поэлементное сравнение массивов информации. При этом в команде **CBT** каждый байт рассматривается как положительное 8-разрядное целое число, а в команде **CMW** каждое машинное слово – как обычное 16-разрядное целое со знаком в старшем разряде.

Возможны три варианта нормального завершения операции сравнения:

все одноименные элементы первого и второго массивов равны. В этом случае выполняется следующая команда, т. е. *РНК* увеличивается на три,

i-й элемент первого массива меньше *i*-го элемента второго массива, а предыдущие $i - 1$ элемент совпадали. В этом случае пропускается одна следующая команда, т. е. *РНК* увеличивается на четыре;

i-й элемент первого массива больше *i*-го элемента второго массива. В этом случае пропускаются две следующие команды.

По окончании на *РА* записывается адрес элемента из первого массива, на котором завершилась операция сравнения, на *РБ* – адрес последнего элемента второго массива.

Команды изменения и анализа битов. Описание операторов машинных команд завершаем рассмотрением трех команд **CBS – сброс, SBS – установка** и **TBS – анализ битов.** Эти команды позволяют изменять значения и анализировать отдельные биты произвольного машинного слова. Каждая из этих команд имеет два операнда. Первый операнд – 16-разрядная *маска.* В ней единицами *отмечаются* те разряды, которые во втором операнде необходимо изменить или проанализировать.

По команде **CBS** (код 105774) отмеченные разряды сбрасываются в нуль, а по команде **SBS** (код 105773) они устанавливаются в единицу.

Анализ битов по команде **TBS** (код 105775) сводится к ПСК (увеличению *РНК* на четыре, поскольку описываемые команды трехсловные), если хотя бы один из *отмеченных* разрядов равен нулю.

Формат команд – трехсловный: в первом слове записывается код команды, во втором – адрес первого операнда и в третьем – адрес второго. На МНЕМОКОДе эти команды также записываются двумя операторами, первый из которых транслируется в две ячейки (как адресная команда двойного формата), а второй должен резервировать адресную константу, задающую адрес изменяемой (анализируемой) ячейки, например:

```
МАСКА  OCT  7                                    (6.15)
ЯЧЕЙК  BSS  1
       ...
       CBS  МАСКА
       DEF  ЯЧЕЙК
```

Синтаксис первого оператора задается определением ⟨ 6.1.8.⟩, но поскольку код команды **CBS** оканчивается на 0, т. е. ключ $Д = 0$, она не употребляется с указателем адресации через индексные регистры.

Упражнения к гл. 6

1. Определите, из какой ячейки будет занесена информация на *РА* после выполнения команд, полученных в результате трансляции операторов

```
LDX  =B200
LDA  -1,X
```

2. Программист написал последовательность операторов

```
       CCA
M1     INA
       SZA,RSS
       CLB
       JMP  M1
```

и решил, что команда **CLB** выполнится только 1 раз, а после возврата по **JMP** *M1* всякий раз будет пропускаться. Покажите ошибочность этого мнения.

3. Исходя из того, что арифметический сдвиг влево на n разрядов должен быть эквивалентен делению на 2^n как положительных, так и отрицательных чисел, сформулируйте правило заполнения освобождающихся при сдвиге старших разрядов.

4. Напишите программу определения порядкового номера слова **TASK** в списке английских слов, хранящихся в некоторой области памяти. При отсутствии слова **TASK** порядковый номер положить равным нулю.

Глава 7. ОПЕРАТОРЫ ПСЕВДОКОМАНД

7.1. Псевдокоманды введения и спецификации идентификаторов

Внешние идентификаторы. Входные точки. Как уже говорилось, для того чтобы идентификатор, который в МНЕМОКОДе всегда играет роль адреса ячейки ОЗУ, приобрел право быть использованным в программе в качестве такового, его необходимо поместить в поле метки одного из операторов программы.

Вместе с тем такой способ введения идентификаторов хотя и является основным, в ряде случаев оказывается недостаточным. В частности, он не годится для идентификаторов *входных точек* вызываемых подпрограмм – операндов команд **JSB** [см., например, вызывающую последовательность (3.60), в которой метка *MAX* является *внешней* по отношению к вызывающей программе]. Чтобы в процессе трансляции *внешние метки* не рассматривались как неопределенные, их список необходимо оговорить в каждом автономно транслируемом программном модуле. Для этой цели служит псевдокоманда **EXT.**

⟨ 7.1.1. *псевдокоманда определения внешних идентификаторов* ⟩ ::=
EXT ⟨ 5.2.14. *список* ⟨ 7.1.2. *внешних идентификаторов* ⟩⟩
⟨ 7.1.2. *внешний идентификатор* ⟩ ::= ⟨ 5.2.1. *идентификатор* ⟩ .

Идентификаторы, определенные псевдокомандой **EXT**, могут использоваться в поле операнда любых адресных команд и псевдокоманд **DEF** и **EQU** (см. ниже), но не в составе выражения.

Итак, в общем случае псевдокоманда **EXT** служит для описания идентификаторов, не принадлежащих программе, если на них приходится делать ссылки.

Встретив псевдокоманду **EXT** транслятор, не меняя значения текущего адреса (в частности, поэтому оператор **EXT** используется без метки), строит в служебной, недоступной программисту области памяти программы специальную таблицу, в которой каждому внешнему идентификатору ставится в соответствие *код-заменитель* и присваивается тип *внешний.* (В листинге результата трансляции этот тип можно распознать по символу X.) Код-заменитель внешнего идентификатора используется транслятором для формирования соответствующих адресных команд и констант на этапе автономной трансляции программ. В одной программе может быть не более 255 различных внешних иден-

тификаторов. Если список внешних идентификаторов программы не помещается на одной строчке (требует больше 72 символов), то его делят на два или большее число операторов **EXT**.

При компоновке загрузочного модуля все ячейки, содержимое которых имеет тип X (внешний), настраиваются на реальные адреса загрузки соответствующих входных точек. Соответствие между внешними идентификаторами и входными точками транслятор устанавливает по таблицам *идентификаторов входных точек* всех компонуемых в один загрузочный модуль программных компонент. Таблица входных точек программы строится транслятором в результате обработки псевдокоманд **ENT**.

⟨ 7.1.3. *псевдокоманда определения идентификаторов входных точек* ⟩ ::= **ENT** ⟨ 5.2.14. *список* ⟨ 7.1.4. *идентификаторов входных точек* ⟩⟩

⟨ 7.1.4. *идентификатор входной точки* ⟩ ::= ⟨ 5.2.1. *идентификатор* ⟩

Общее правило таково: всякий идентификатор, являющийся внешним в данной программе, должен быть указан в списке входных точек какой-нибудь другой программы, компонуемой вместе с данной.

Оператор **ENT**, так же как и **EXT**, не влияет на распределение памяти и может указываться в любом месте программы после оператора **NAM**. В случае длинного списка входных точек его можно разбить на два и большее число операторов.

Организация общей области памяти, которая может использоваться совместно несколькими программными компонентами в одном загрузочном модуле, осуществляется псевдокомандой **COM**.

⟨ 7.1.5. *псевдокоманда определения блока общей области* ⟩ ::=
[⟨ 7.1.6. имя блока ⟩] **COM** ⟨ 5.2.14. *список*
⟨ 7.1.7. *сегментов блока общей области* ⟩⟩

⟨ 7.1.6. *имя блока* ⟩ ::= ⟨ 5.2.1. *идентификатор* ⟩

⟨ 7.1.7. *сегмент блока общей области* ⟩ ::= ⟨ 7.1.8. *имя сегмента* ⟩
[⟨ (7.1.9. *указатель длины сегмента* ⟩)]

⟨ 7.1.8. *имя сегмента* ⟩ ::= ⟨ 5.2.1. *идентификатор* ⟩

⟨ 7.1.9. *указатель длины сегмента* ⟩ ::= ⟨ 5.2.10. *целое без знака* ⟩

В общей области обычно размещают наиболее важные для задачи или раздела массивы данных и константы. Данные, к которым будут обращаться все компонуемые вместе программы, размещают в *неименованном блоке*, т. е. описывают оператором **COM** с пустым полем метки. Если область данных представляет интерес только для части программ, то ее описание имеет смысл оформить в виде *именованного блока*, в котором имя блока помещается в поле метки.

Сказанное поясним на примере. Пусть задача компонуется из трех программ – одной главной с именем *GP* и двух подпрограмм *P1* и *P2*, пусть к массиву *M1*, состоящему из 100 элементов, и константе *C* предполагается обращаться во всех трех программных модулях, а к массиву *M2* длиной 50 – только в модулях *GP* и *P1*.

Описанной ситуации соответствуют три фрагмента программ с псевдокомандами **COM** следующего вида:

```
ASMB,R,B
        NAM  GP,1
        COM  M1(100), C
БЛ      COM  M2(50)
ASMB,R,B
        NAM  P2,6
        COM  M1(100),C
```

```
ASMB,R,B
        NAM  P1,6
        COM  M1(100), C
БЛ      COM  M2(50)
```

Встретив в программе оператор **COM**, транслятор заносит в таблицу идентификаторов *имена сегментов* общих областей, присваивает каждому из них тип *перемещаемый в общей области* (в листинге трансляции этот тип выделяется символом *C*) и значение, равное суммарной длине предшествующих сегментов. Поэтому при обработке операторов **COM** предыдущего примера в таблице идентификаторов программы *GP* появятся следующие элементы: *M1* с перемещаемым в общей области значением 0 (как следствие, идентификатор *M1* будет меткой нулевой ячейки общей области), идентификаторы *C* и *M2* со значениями 100 (метка сотой ячейки) и 101 (метка 101-й ячейки) соответственно.

Обращение к ячейкам общей области осуществляется в программе по именам сегментов, которые, очевидно, играют роль меток-идентификаторов ячеек общей области. Поэтому на заключительной фазе компоновки адреса, перемещаемые в общей области, настраиваются на реальные адреса загрузки.

В табл. 7.1 показано распределение меток по физическим ячейкам ОЗУ, соответствующее операторам **COM** в программе *EX* и подпрограмме *SUB*:

```
        NAM  EX,1
ИМБ     COM  C1(5),C2(3),C
```

```
        NAM  SUB,5
ИМБ     COM  Д1,Д2(4),Д3
```

В частности, видно, что физически к одной ячейке относятся метки *C2* и *Д3* (в программах *EX* и *SUB* соответственно), к этой же ячейке адресуют выражения *C–3* и *Д1+5*. Интересно, что в подпрограмме *SUB* вполне корректно будет обращение к ячейке *Д3+3*, т. е. к восьмой ячейке именованного общего блока *ИМБ*, хотя в *SUB* для общего блока заказано всего шесть ячеек памяти.

Оператор приравнивания. В МНЕМОКОДе существует средство, позволяющее вводить метки-идентификаторы и придавать им нужные программисту (неотрицательные) значения. Например, если программист часто обращается к *PA* как к ячейке, т. е. по нулевому адресу, и ему удобно вместо адреса *0* использовать идентификатор *A*, он может обеспечить себе такую возможность *псевдокомандой приравнивания*

```
A       EQU  0,
```

Таблица 7.1

Метка в *EX*	Метка в *SUB*	Адреса в общей области
C1	*Д1*	0
	Д2	1
		2
		3
		4
C2	*Д3*	5
		6
		7
C		8

помещая ее в произвольном месте программы. В сочетании с данной псевдокомандой оператор **LDA** *0*, **I**, например, становится эквивалентным оператору **LDA** *A*, **I**.

Общий синтаксис псевдокоманды **EQU** таков:

⟨ 7.1.10. *псевдокоманда приравнивания* ⟩ ::= ⟨ 7.1.11. *вводимый идентификатор* ⟩ **EQU** ⟨ 5.2.8. *выражение* ⟩

⟨ 7.1.11. *вводимый идентификатор* ⟩ ::= ⟨ 5.2.1. *идентификатор* ⟩

Вводимый идентификатор указывается в поле метки; таким образом, один оператор **EQU** определяет один идентификатор, но общее число таких операторов в программе практически неограниченно.

Встретив псевдокоманду приравнивания, транслятор вычисляет значение выражения (поскольку это делается на первом проходе двухпроходного транслятора, все термы, входящие в состав выражения, должны быть либо числами, либо определены в программе выше оператора **EQU**) и заносит в таблицу идентификаторов новый элемент, имеющий тип и значение выражения, находящегося в поле операнда оператора **EQU**. Поэтому введенный идентификатор может свободно использоваться в операторах МНЕМОКОДа, как если бы он был указан в поле метки соответствующей ячейки. Например, пара операторов

```
        LDA  РЯ
M       EQU  *
```

по результатам трансляции совершенно эквивалентна одному оператору

```
M       LDA  РЯ.
```

В обоих случаях идентификатору *M* будет присвоено одно и то же значение – значение текущего адреса.

Определение идентификаторов внешних констант. Если присвоение значений для некоторых идентификаторов программисту удобно отложить до этапа компоновки, то он должен использовать оператор **ESC**.

⟨ 7.1.12. *псевдокоманда определения идентификаторов внешних константа* ⟩ ::= **ESC** ⟨ 5.2.14. *список* ⟨ 6.4.3. *идентификаторов внешних констант* ⟩⟩

В отличие от внешних идентификаторов, фактические значения которых совпадают с адресами соответствующих входных точек, значения идентификаторов *внешних констант* задаются с пульта оператора непосредственно в процессе компоновки загрузочного модуля.

7.2. Псевдокоманды резервирования констант

Семантика различных псевдокоманд резервирования констант однотипна: встретив такую псевдокоманду, транслятор в очередной ячейке (или ячейках) памяти программы размещает константы, заданные в поле операнда псевдокоманды. Метка, если она есть, относится к первой из формируемых ячеек. Указатель текущего адреса увеличивается на число формируемых ячеек. Различают константы *числовые (восьмеричные и десятичные), символьные, адресные, абсолютные и внешние.* Естественно, что, используя в тексте программы псевдокоманды резервирования констант, программист должен следить, чтобы адреса ячеек, содержащих константы, не попадали на *РНК,* если, конечно, эти константы по замыслу программиста не должны играть роль машинных команд.

Резервирование числовых констант. Для этой цели служат псевдокоманды **ОСТ** и **DEC**:

⟨ 7.2.1. *Псевдокоманда резервирования восьмеричных констант* ⟩ ::= [⟨ 6.1.2. *метка* ⟩] **ОСТ** ⟨ 5.2.14. *список* ⟨ 7.2.2. *восьмеричных констант* ⟩⟩

⟨ 7.2.2. *восьмеричная константа* ⟩ ::= [{ ± }] ⟨ 5.2.2. *последовательность* ⟨ 5.2.6. *восьмеричных цифр* ⟩⟩

При трансляции восьмеричные константы заменяются двоичными кодами (отрицательные – дополнительным) и размещаются в последовательных ячейках памяти согласно их перечислению в *списке.*

Например, при трансляции оператора

C **ОСТ** *23, –1,1777*

в ячейку с меткой *C* будет занесено число 23_8, в две последующие ячейки – числа –1 и 1777_8 соответственно, а счетчик текущего адреса увеличится на три.

⟨ 7.2.3. *псевдокоманда резервирования десятичных констант* ⟩ ::=
[⟨ 6.1.2. *метка* ⟩] **DEC** ⟨ 5.2.14. *список*
⟨ 7.2.4. *десятичных констант* ⟩⟩

⟨ 7.2.4. *десятичная константа* ⟩ ::= [{ ± }] ⟨ 5.2.11. *десятичное целое без знака* ⟩ | [{ ± }] ⟨ 6.2.4. *десятичное дробное без знака* ⟩

Как видно из приведенного определения, в один список десятичных констант могут быть включены как целые, так и дробные десятичные числа, дробные числа при трансляции занимают по две ячейки каждое.

Резервирование символьных констант осуществляется псевдокомандой **ASC**. Необходимость в этом возникает, когда нужно запастись текстовой строкой для вывода комментариев, для задания формата ввода-вывода чисел (см. следующую главу) и в ряде других случаев.

⟨ 7.2.5. *псевдокоманда резервирования символьной константы* ⟩ ::=
[⟨ 6.1.2. *метка* ⟩] **ASC** ⟨ 7.2.6. *указатель длины константы* ⟩,
⟨ 5.2.2. *последовательность* ⟨ 5.2.15. *основных символов* ⟩⟩

⟨ 7.2.6. *указатель длины константы* ⟩ ::= ⟨ 5.2.8. *выражение* ⟩

Длина константы в псевдокоманде **ASC** задается не в символах, а в машинных словах, необходимых для размещения константы. Если число символов нечетно, к последовательности символов добавляется код пробела.

Например, если в программе предполагается использовать слово КОНСТАНТА, но в неисполняемой части программы необходимо поместить оператор

```
TX      ASC  5,КОНСТАНТА
```

в результате трансляции которого в пяти последовательных ячейках памяти, начиная с ячейки, помеченной меткой *TX*, будет записано по два символа: КО – в первой ячейке, НС – во второй и т. д., а поскольку общее число символов нечетно, в пятую ячейку транслятор запишет букву А с кодом пробела в младшем байте.

Резервирование адресных констант осуществляется хорошо знакомой (см. гл. 3) псевдокомандой **DEF**. Не будет преувеличением сказать, что это самая "популярная" псевдокоманда. Во всяком случае, она необходима всякий раз, когда в ячейке требуется получить адрес другой ячейки. Напомним, что такого рода адресные константы незаменимы при организации режимов индексации и автоиндексации, как, впрочем, и в других приемах обработки массивов и таблиц. Кроме того, в МНЕМОКОДе последовательностью псевдокоманд **DEF** при обращении к подпрограммам задаются фактические параметры, имеющие смысл адресов (начальные адреса массивов исходных данных и областей памяти для размещения результатов, адреса точек возврата, скалярных параметров и др.).

Синтаксис псевдокоманды **DEF** определяется ее прагматикой: требуемый адрес формируется либо с нулем, либо с единицей в нулевом

разряде (режим индексации и автоиндексации соответственно). Кроме того, псевдокоманда **DEF** делает доступными программе *адреса литералов:*

⟨ 7.2.7. *псевдокоманда резервирования адресной константы* ⟩ ::= [⟨ 6.1.2. *метка* ⟩] **DEF** { ⟨ 5.2.8. *выражение* ⟩ / ⟨ 7.2.9. *литерал* ⟩ } [,⟨ 7.2.8. *указатель единицы нулевого разряда* ⟩], причем *указатель* может следовать только за *выражением,* либо за *литералом типа выражение*

⟨ 7.2.8. *указатель единицы нулевого разряда* ⟩ ::= **I**

⟨ 7.2.9. *литерал* ⟩ ::= ⟨ 6.1.9. *однословный литерал* ⟩ | ⟨ 6.2.2. *двухсловный литерал* ⟩

В дополнение к известным из гл. 3 формам псевдокоманды **DEF**, таким, как **DEF** ***+3*** или **DEF** *АДР*,**I**, данное определение вводит псевдокоманду **DEF** с литералом в поле операнда: **DEF =B** *1777*, **DEF =L***177***B**,**I** и т. п.

Обратим внимание на некоторую особенность реализации псевдокоманды **DEF**: из всех литералов только литерал типа *выражение* (спецификатор **=L**) может использоваться с указателем **I**. Однако учитывая, что этим литералом может быть представлен любой другой литерал, кроме, пожалуй, литерала со спецификатором **=F** (например, литерал **=B** *23* имеет то же машинное представление, что и литерал **=L***23***B**), это ограничение не является существенным.

Другое ограничение накладывается на значение *выражения* – операнда псевдокоманды **DEF**: поскольку резервируется адресная константа, выражение должно иметь неотрицательное значение. Если, кроме того, выражение абсолютно по типу, то его значение должно адресовать ячейку в пределах нулевой страницы, т. е. быть в пределах от 0 до 1777.

Резервирование абсолютных констант. Резервирование имеющих произвольное, в том числе и отрицательное значение констант осуществляется псевдокомандой **ABS**.

⟨ 7.2.10. *псевдокоманда резервирования абсолютной константы* ⟩ ::=
::= [⟨ 6.1.2. *метка* ⟩] **ABS** ⟨ 5.2.8. *выражение* ⟩

Встретив эту псевдокоманду, транслятор определяет значение *выражения* (которое должно быть абсолютно по типу, иначе транслятор выдаст сообщение об ошибке) и записывает его в очередной ячейке программы.

Говоря о возможном применении псевдокоманды **ABS**, отметим, что в больших программах эта псевдокоманда удобна для получения величины смещения одной ячейки относительно другой.

Например, если в программе имеются метки *M1* и *M2*, то при трансляции псевдокоманды

СМЕЩ **ABS** *M2–M1*

транслятор в ячейке *СМЕЩ* запишет число ячеек программы между метками *М2* и *М1*. Причем, если метка *М2* относится к ячейке с меньшим адресом, это число будет отрицательным.

Резервирование дизъюнкции внешних и базовых констант. Все описанные ранее константы формируются в процессе трансляции программ (перемещаемые адресные константы – с точностью до начального адреса загрузки). *Внешние константы,* напротив, заносятся в память программы в процессе компоновки задач.

Другой вариант использования внешних констант связан с псевдокомандой **ECO** – псевдокомандой резервирования внешней константы (точнее, дизъюнкции внешней и *базовой* констант):

⟨ 7.2.11. *псевдокоманда резервирования внешней константы* ⟩ ::=
[⟨ 6.1.2. *метка* ⟩] **ECO** ⟨ 6.4.3. *идентификатор внешней константы* ⟩ [(⟨ 7.2.12. *указатель базовой константы* ⟩
[, ⟨ 7.2.13. *указатель сдвига внешней константы* ⟩])]

⟨ 7.2.12. *указатель базовой константы* ⟩ ::= ⟨ 5.2.8. *выражение* ⟩

⟨ 7.2.13. *указатель сдвига внешней константы* ⟩ ::= ⟨ 5.2.8. *выражение* ⟩

Встретив псевдокоманду **ECO,** транслятор резервирует ячейку памяти программы, вычисляет значение *указателя базовой константы,* которое должно быть абсолютным по типу (чаще всего в качестве базовой константы используется восьмеричное целое число без знака), и записывает это значение в зарезервированную ячейку. Кроме того, вычисляется и записывается в служебную область значение *указателя сдвига* внешней константы (число в пределах от 0 до 16). Если эти элементы псевдокоманды **ECO** опущены, они по умолчанию полагаются транслятором равными нулю.

Компоновщик формирует дизъюнкцию внешней и базовой констант следующим образом: значение идентификатора внешней константы, введенное при компоновке, сдвигается влево на число разрядов, заданное указателем сдвига, и дизъюнктивно складывается с базовой константой.

Полученный таким образом восьмеричный код может использоваться в программе так же, как и любая другая константа, но основное предназначение псевдокоманды **ECO** – резервирование *переменных* машинных команд, т. е. таких команд, окончательное формирование которых осуществляется на этапе компоновки введением значений внешних констант.

В качестве примера посмотрим, как можно с помощью псевдокоманды **ECO** обеспечить в программе чтение на регистр А содержимого ячейки нулевой страницы с восьмеричным адресом 100*n*, где *n* переменно и задается в процессе компоновки.

Выбирая из табл. 3.2 КОП засылки на *РА*, находим код 060000 – машинный эквивалент команды **LDA** *0*. Теперь его нужно использовать в качестве базовой константы, поместив вместо оператора **LDA** псевдокоманду **ECO** (*060000*B,*6*), где *6* – указатель сдвига внешней

константы. Если, кроме того, идентификатор **N** псевдокомандой **ESC** *N* специфицирован как идентификатор внешней константы, то в процессе компоновки константе *n* может быть присвоено произвольное значение (по смыслу – в пределах от 0 до 17_8). Например, если *n* при компоновке положить равным 17_8, то после сдвига на шесть разрядов окажется, что $n = 1700_8$. Поэтому на месте псевдокоманды **ECO** возникнет код 061700, т. е. машинная команда **LDA** *1700* **B**.

7.3. Псевдокоманды управления памятью

Псевдокоманды данной группы позволяют указать транслятору зарезервировать определенное число ячеек для размещения исходных данных и результатов (псевдокоманда **BSS**), разместить *сегмент* программы в нулевой странице (псевдокоманда **ORB**), произвольным образом изменить значение счетчика текущего адреса (псевдокоманда **ORG**) и, наконец, восстановить значение этого счетчика (псевдокоманда **ORR**) таким, каким оно было до псевдокоманд **ORB** и **ORG**:

⟨ 7.3.1. *псевдокоманда резервирования памяти* ⟩ ::= [⟨ 6.1.2. *метка* ⟩]
BSS ⟨ 7.3.2. *указатель числа резервируемых ячеек* ⟩

⟨ 7.3.2. *указатель числа резервируемых ячеек* ⟩ ::= ⟨ 5.2.8. *выражение* ⟩

Встретив эту псевдокоманду, транслятор определяет значение *указателя числа резервируемых ячеек*, которое должно быть положительным и абсолютным по типу целым числом, и добавляет это число к счетчику текущего адреса, резервируя тем самым область памяти требуемого объема. Кроме того, при наличии *метки* транслятор помещает новый элемент в таблицу идентификаторов программы и присваивает ему в качестве значения адрес начальной ячейки резервируемой области.

⟨ 7.3.3. *псевдокоманда размещения сегмента в нулевой странице* ⟩ ::=
ORB ⟨ 7.3.4. *указатель начального адреса сегмента* ⟩

⟨ 7.3.4. *указатель начального адреса сегмента* ⟩ ::= ⟨ 5.2.8. *выражение* ⟩

Встретив псевдокоманду **ORB**, транслятор определяет значение *указателя начального адреса сегмента*, которое в данном случае должно быть абсолютным целым числом в пределах от 0 до 1777_8, и присваивает это значение счетчику текущего адреса. Как следствие, начиная с этого момента, оттранслированные операторы программы будут размещаться в ячейках нулевой страницы.

Восстановить значение счетчика текущего адреса можно псевдокомандой **ORR**:

⟨ 7.3.5. *псевдокоманда восстановления счетчика текущего адреса* ⟩ ::=
ORR

Встретив эту псевдокоманду, транслятор присваивает счетчику текущего адреса то значение, которое он имел в момент появления первой псевдокоманды **ORB** или первой псевдокоманды **ORG** – *псевдокоманды изменения значения счетчика текущего адреса:*

⟨ 7.3.6. *псевдокоманда изменения значения счетчика текущего адреса* ⟩ ::= **ORG** ⟨ 7.3.4. *указатель начального адреса сегмента* ⟩

Как и в случае псевдокоманды **ORB**, здесь транслятор также вычисляет значение *указателя начального адреса.* Но в сочетании с мнемоникой **ORG** это значение должно быть перемещаемым в программе либо в общей области. Далее вычисленное значение присваивается счетчику текущего адреса, и дальнейшая трансляция идет в область памяти, начиная с этого адреса.

7.4. Прочие псевдокоманды

Псевдокоманды условной трансляции являются своего рода *операторными скобками,* причем роль открывающих скобок играют псевдокоманды с мнемоникой **IFN** и **IFZ**, а роль закрывающейся скобки – псевдокоманда с мнемоникой **XIF**. Поле метки и поле операнда в операторах этих псевдокоманд не используются. Операторы МНЕМОКОДа, охваченные скобками условной трансляции, транслируются только при наличии соответствующего параметра в управляющем операторе: фрагменты программы между псевдокомандами **IFN** и **XIF** будут включены транслятором в результирующую программу при наличии параметра **N**, в то время как фрагменты, открывающиеся псевдокомандой **IFZ**, будут транслироваться, если в список параметров управляющего оператора включен параметр **Z**.

Необходимо помнить, что вложение и пересечение операторных скобок условной трансляции не допускаются.

Таким образом, с использованием псевдокоманд условной трансляции на МНЕМОКОДе может быть написана *одна* программа, дающая в результате трансляции до *трех* вариантов, в зависимости от списка параметров управляющего оператора. Именно трех, а не четырех, так как если в список включить оба параметра **N** и **Z**, то трансляция будет вестись по тому из них, который в управляющем операторе указан первым.

Псевдокоманда повторения. Оператор, следующий за этой псевдокомандой, повторяется и транслирует указанное в поле операнда число раз:

⟨ 7.4.1. *псевдокоманда повторения* ⟩ ::= [⟨ 5.2.1. *идентификатор* ⟩] **REP** ⟨ 7.4.2. *указатель числа повторений* ⟩

⟨ 7.4.2. *указатель числа повторений* ⟩ ::= ⟨ 5.2.8. *выражение* ⟩

Значение *указателя числа повторений* должно быть положительным и абсолютным по типу. Если в поле метки указан *идентификатор,* то транслятор поместит его в таблицу идентификаторов и присвоит значение счетчика текущего адреса. Таким образом, идентификатор будет являться меткой первого повторения "тиражируемого" оператора.

Псевдокоманды управления печатью листинга. В заключение перечислим псевдокоманды, которые не влияют на результат трансляции и

служат лишь для придания листингу той или иной желаемой формы. В их числе:

псевдокоманда печати на каждой странице листинга *заглавия программы* (произвольной строки символов), имеющая мнемонику **HED**, за которой в поле операнда должна следовать строка символов языка МНЕМОКОД. Размещают псевдокоманду **HED** непосредственно за управляющим оператором, перед оператором **NAM**. Если псевдокоманда **HED** встретится внутри программы, листинг начнет выдаваться с новой страницы и с новым заголовком,

псевдокоманда **SKP**, появление которой в тексте программы вызывает переход к новой странице листинга (так, что предыдущая страница может оказаться неполной);

псевдокоманда **SPS**, вызывающая исключение из листинга результата трансляции *n* следующих за **SPS** операторов (число *n* задается выражением в поле операнда псевдокоманды);

псевдокоманда **UNL**, вызывающая исключение из листинга результат трансляции всех следующих за **UNL** операторов вплоть до оператора **END** или до псевдокоманды **LST**, назначение которой – отменить действие псевдокоманды **UNL**,

псевдокоманда **SUP**, подавляющая печать дополнительных строк при трансляции псевдокоманд резервирования констант в две и большее число ячеек памяти. Действие псевдокоманды **SUP** отменяется оператором псевдокоманды **UNS**.

Все команды управления печатью листинга употребляются без метки, а такие псевдокоманды, как **UNL**, **LST**, **SUP**, **UNS** и **SKP**, – и с пустым полем операнда.

Итак, исчерпан список операторов базового языка. Однако было бы ошибкой считать, что этим исчерпывается также весь арсенал его изобразительных средств. Неотъемлемой частью языка МНЕМОКОД следует считать его расширения – *основную библиотеку подпрограмм* и *макроязык*, которые рассматриваются в следующей главе.

Упражнения к гл. 7

1. Дан фрагмент программы:

```
ASMB,R,L,T
        NAM  PROG,6
M       EQU  10B
M1      OCT  -177,1000,126014,142075
M2      EQU  M+*.
```

а) Какой код транслятор разместит в ячейке с перемещаемым в программе адресом 3?

б) Какие значения получат при трансляции идентификаторы *M, M1, M2?*

2. Покажите, что в операторах команд изменения и анализа битов [см. пример (6.15)] можно непосредственно указать значение обоих операндов.

3. Оформите в пригодном для трансляции виде реентерабельную подпрограмму отыскания максимального элемента в массиве чисел. С помощью псевдокоманд условной трансляции обеспечьте, чтобы при наличии в управляющем операторе параметра **N** подпрограмма отыскивала не максимальный, а минимальный элемент массива.

Глава 8. РАСШИРЕНИЯ МНЕМОКОДа

8.1. Основная библиотека подпрограмм

Основная библиотека содержит большое число подпрограмм, осуществляющих стандартные, часто встречающиеся операции: нахождение абсолютной величины числа, вычисление квадратного корня, передачу параметров в подпрограммы, преобразование форматов и т. п.

Достаточно включить в список внешних идентификаторов (псевдокоманды **EXT**) метки входных точек нужных подпрограмм, и они станут доступными из любой перемещаемой программы пользователя, избавив его от необходимости писать довольно трудоемкие подпрограммы. К подпрограммам основной библиотеки можно обращаться как в программах на МНЕМОКОДе, так и на языках выского уровня. Кроме того, обращение к этим подпрограммам генерируется автоматически трансляторами с ФОРТРАНа и других алгоритмических языков при трансляции выражений, содержащих стандартные функции **SIN**, **ABS**, **EXP** и т. п.

В состав библиотеки входят:

подпрограммы, выполняющие математические вычисления с целыми и вещественными числами;

подпрограммы, выполняющие аналогичные вычисления с комплексными числами и вещественными числами двойной длины (каждое комп-

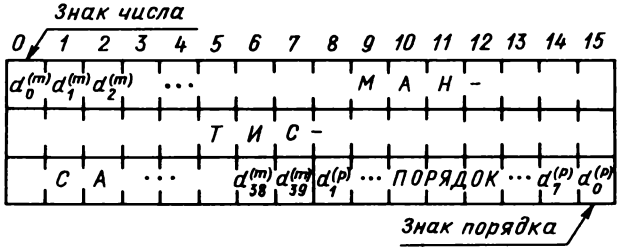


Рис. 8.1. Формат вещественного числа двойной точности

лексное число занимает четыре машинных слова, в первых двух записывается действительная часть, в последующих – мнимая; вещественное число двойной длины записывается в трех ячейках, как показано на рис. 8.1),

подпрограммы, выполняющие различные специальные функции: генерацию случайных чисел, передачу адресов параметров в подпрограммы, вычисление адреса элемента двухмерного массива и др.

Сюда входит также подпрограмма **FMTIO** (известная также под названием *форматтер*), обеспечивающая ввод-вывод числовых данных, согласно заданному формату.

8.2. Ввод-вывод числовых данных

Подпрограмма **FMTIO** может работать с различными УВВ: пультами оператора, устройствами печати, устройствами перфоввода и перфорации. При необходимости осуществить взаимодействие с данными на магнитных носителях можно воспользоваться *внутренним* преобразованием формата (одна из функций подпрограммы **FMTIO**) совместно с системой управления файлами, описанной в гл. 11.

Всего **FMTIO** имеет 17 входных точек, основные из которых представлены ниже:

.DIO. – задание параметров операции ввода-вывода алфавитно-цифровой информации
.IIO. – ввод-вывод одного целого числа
.IAY. – ввод-вывод целочисленного массива
.RIO. – ввод-вывод одного вещественного числа
.RAY. – ввод-вывод массива вещественных чисел
.XIO. – ввод-вывод одного вещественного числа двойной точности
.XAY. – ввод-вывод массива вещественных чисел двойной точности
.DTA. – завершение вывода числовых данных

В программах на МНЕМОКОДе всякая операция ввода-вывода числовой информации должна последовательно включать обращение к входной точке *.DIO.* и одно или несколько обращений собственно к операциям ввода-вывода – входным точкам *.IIO.*, *.IAY.* и т. д.

При обращении к входной точке *.DIO.* в общем случае должны передаваться четыре входных параметра: *логический номер* УВВ (о логических номерах см. § 12.6), *указатель направления операции, адрес указателя формата* и *адрес перехода по ошибке.*

Если в качестве логического номера указывается нуль, то осуществляется *внутреннее преобразование,* суть которого состоит в преобразовании числовой информации, размещенной в определенной области ОЗУ, из символьного кода в машинный или обратно без какого-либо обмена с УВВ. В этом случае требуется дополнительный, пятый параметр – *адрес буфера,* имеющий смысл адреса первого элемента символьного массива.

Обращение к входной точке *.DIO.* имеет вид

```
LDA  =L логический номер УВВ или 0                          (8.1)
LDB  =L1 – для ввода, 0 – для вывода
JSB  .DIO.
[DEF адрес буфера]
{DEF адрес указателя формата                                }
{OCT 0 – указатель свободного формата                       }
DEF  адрес перехода по ошибке
```

Указатель формата вводится в программу с помощью псевдокоманды **ASC** – резервирования символьной константы. Выбрав определенный формат ввода-вывода, программист должен записать его в виде последовательности спецификаций ввода-вывода алгоритмического языка ФОРТРАН IV и в неисполняемой части программы зарезервировать символьную константу, которая изображает эту последовательность спецификаций, взятую в круглые скобки. Адрес первой ячейки сформированной таким образом символьной константы и является адресом указателя формата. При равенстве этого параметра абсолютному нулю устанавливается режим ввода в *свободном формате,* в котором числа изображаются в любой допустимой форме (целые со знаком и вещественные в соответствии с синтаксическим определением), отделяются друг от друга произвольным числом пробелов и в конце числового материала ставится косая черта. Вывод в свободном формате не предусмотрен.

Если в спецификациях формата или в передаваемых числах будет обнаружена синтаксическая ошибка, **FMTIO** прекратит операцию ввода-вывода и передаст управление по *адресу перехода по ошибке.*

Итак, если, например, задаться целью отпечатать максимальный элемент (*MMAX*) массива по формату I6 на устройстве с логическим номером 6, то в программу необходимо включить следующие операторы:

в неисполняемой части

```
УФ      ASC 7, (6H  MMAX=, I6)                              (8.2)
```

в исполняемой части

```
        LDA  =L6   РА := логический номер УВВ                (8.3)
        CLB        РБ :=указатель направления передачи, в данном
*                  случае вывода
        JSB  .DIO.
        DEF  УФ    адрес указателя формата
        DEF  ОШ    адрес перехода по ошибке
```

В результате на печать будет выдано

MMAX=

и задача перейдет в ожидание операции вывода числовых данных по формату I6.

Обращения к операциям ввода-вывода осуществляется в форме, показанной ниже.

Ввод-вывод одного числа:

JSB { *.IIO.* – целого
.RIO. – вещественного
.XIO – двойной точности } (8.4)

DEF *адрес числа*

Ввод-вывод массива:

JSB { *.IAY.* – целого
RAY. – вещественного
.XAY. – двойной точности } (8.5)

DEF *адрес первого элемента массива*

DEC *количество элементов*

Поэтому пример, начатый операторами (8.2), (8.3), необходимо закончить так:

```
JSB  .I IO.                                    (8.6)
DEF  MMAX
JSB  .DTA.
```

Заметим в заключение, что подпрограмма **FMTIO**, являясь достаточно универсальной, неудобна тем не менее для использования в многозадачном режиме: не обладая свойством реентерабельности, она увеличивает объем каждой обращающейся к ней задачи почти на 6 К ячеек (!). В то же время в основной библиотеке имеются реентерабельные подпрограммы объемом порядка 1 К для внутреннего преобразования чисел из символьного вида в машинный код (подпрограммы *ВВЦ* и *ВВП* для целых и вещественных чисел соответственно) и для обратных преобразований (подпрограммы *П10* и *ПП32*), которые в сочетании с макрокомандой **EXIO** (см. гл. 10) могут обеспечить многие функции подпрограммы **FMTIO.**

8.3. Макрокоманды и макроопределения

На практике часто встречается ситуация, когда для реализации задуманного алгоритма программисту приходится многократно повторять однотипные последовательности операторов, отличающиеся лишь небольшими вариациями операндов и мнемонических обозначений КОП. Естественно, что при этом возникает стремление некоторым образом переобозначить всю эту последовательность в целом, предусмотрев в формате выбранного обозначения способ варьирования деталей.

Работая с МНЕМОКОДом, программист имеет возможность реализовать это стремление, воспользовавшись аппаратом макроопределений и макрокоманд.

Под *макроопределением* понимается специальным образом оформленная последовательность операторов МНЕМОКОДа с указанием варьируемых деталей – *параметров макроопределения* – и способов их конкретизации.

Что касается *макрокоманды*, то это и есть тот самый короткий заменитель последовательности операторов МНЕМОКОДа, о котором говорилось выше.

Подстановку операторов МНЕМОКОДа из макроопределений вместо макрокоманд осуществляет *макрогенератор* – одна из системных обрабатывающих программ, входящих в состав внутреннего программного обеспечения ВК архитектурной линии СМ-2М.

На вход макрогенератора подается программа, написанная на МНЕМОКОДе, расширенном различными макрокомандами, и содержащая некоторые из макроопределений. На выходе генерируется программа на базовом языке без каких-либо расширений, пригодная для обработки транслятором с МНЕМОКОДа.

Оператором окончания макрогенерации является оператор **ENDM**, поэтому в исходной программе он должен следовать непосредственно за оператором окончания трансляции **END**.

Встретив в тексте программы ту или иную макрокоманду, макрогенератор для поиска соответствующего макроопределения сначала просматривает предшествующий текст программы и лишь затем обращается к специальной библиотеке – библиотеке макроопределений, входящей в состав АСПО и содержащей макроопределения для макрокоманд генерации ОС и вызова супервизора. Заметим, что всякое новое макроопределение пользователя может быть также включено в эту библиотеку (о корректировке библиотеки макроопределений см. § 13.2). Расширяя библиотеку собственными макроопределениями, пользователь может придать базовому языку ориентацию на определенный класс задач и тем самым существенно повысить производительность своего труда и труда своих коллег.

Таким образом, макроопределение, как отмечается в [14], очень напоминает подпрограмму и отличается лишь тем, что подпрограмму транслируют отдельно и вызывают во время выполнения программы, а макроопределение подставляется вместо каждой макрокоманды во время обработки программы макрогенератором.

Вместе с тем правила написания макрокоманд являются более гибкими по сравнению с правилами написания подпрограмм. В частности, они позволяют объявить переменными один или несколько символов в идентификаторе или мнемоническом обозначении кода операции и получить, например, после обработки макрогенератором одного и того же прототипа команду **LDA** или **LDB** в зависимости от фактических параметров макрокоманды.

Структура макроопределения. В простейшем случае она такова:

заглавный оператор (8.7)
прототип-оператор
модель-оператор
. . .
модель-оператор
конечный оператор

Операторы макроязыка (к их числу относятся макрокоманды и операторы, из которых построены макроопределения) имеют формат, аналогичный формату операторов МНЕМОКОДа: поле метки, поле кода операций, поле операнда и поле комментария. Соседние поля должны отделяться друг от друга не менее чем одним пробелом.

Заглавный оператор состоит из одного служебного слова **MACRO**, которое записывается в поле КОП. Другие поля при этом остаются пустыми. С заглавного оператора должно начинаться любое макроопределение.

Прототип-оператор задает формат той макрокоманды, которая будет в программе "вызывать" соответствующее макроопределение, настроенное макрогенератором на список *фактических параметров макрокоманды.*

Настройка макрокоманды касается прежде всего *моделей-операторов,* каждый из которых представляет собой один из операторов базового языка. В нем варьируемые элементы заменены *параметрами-идентификаторами.*

Отличительным признаком параметра-идентификатора является его первый символ – **&** (амперсенд). Например, переменный код операции **LD &***Y* в модели-операторе будет превращен макрогенератором в **LDA**, в **LDB** и т.п. в зависимости от параметров макрокоманды. (Заметим, что при этом из-за ошибки в параметрах может возникнуть несуществующая в базовом языке мнемоника **LDC** или, скажем, **LDAB**, и ошибка будет обнаружена только на этапе трансляции с МНЕКОКОДа.)

Конечный оператор часто состоит из одного слова **MEND**, которое размещается в поле КОП. Этот оператор должен быть последним в каждом макроопределении.

8.4. Прототип- и модель-операторы

Прототип-оператор. В общем случае он состоит из *имени макроопределения* (обязательный элемент), *списка параметров-идентификаторов* (общим числом до двухсот) и комментария.

Имя макроопределения размещается в поле КОП. Оно может представлять собой последовательность основных символов (кроме **&**) практически любой длины, но макрогенератор воспринимает только первые пять. Встретив в программе макрокоманду, макрогенератор ищет макроопределение по имени, указанному в поле КОП макрокоманды.

Отсюда ясно, что КОП макрокоманды и имя соответствующего макроопределения должны совпадать.

Естественно, что в качестве имени макроопределения нельзя использовать мнемонику машинных команд или псевдокоманд, иначе макрогенератор не сможет отличить макрокоманду от оператора базового языка.

Список параметров-идентификаторов размещают в поле операнда прототипа-оператора. Но один из параметров-идентификаторов может быть размещен также и в поле метки. Включенная в программу макрокоманда определяет фактические значения параметров-идентификаторов. Например, в макроопределении, имеющем прототип-оператор

&NAME **EXS** *&A,&B* (8.8)

макрокоманда

M1 **EXS** *R1, R2* (8.9)

определит для параметров-идентификаторов *&NAME*, *&A* и *&B* значения *M1*, *R1* и *R2* соответственно. Эти значения при макрогенерации будут подставлены в модели-операторы на место соответствующих параметров-идентификаторов.

Операндом макрокоманды [в (8.9) операндами являются *M1*, *R1* и *R2*] может быть любая последовательность основных символов, но при необходимости включить в эту последовательность запятую ее следует заключить в кавычки, в связи с тем что запятая является разделителем в списке операндов. Поэтому макрокоманда, присваивающая в прототипе-операторе (8.8) параметру-идентификатоду &A значение *A*,**I** (*A* с признаком косвенной адресации), будет иметь вид

M1 **EXS** *A","I,R2* (8.10)

Заметим также, что в состав операнда не может входить пробел, поскольку он указывает на конец поля операнда.

Опущенные операнды. Каждому операнду (параметру-идентификатору) прототипа-оператора в общем случае соответствует операнд макрокоманды. Если один или несколько операндов макрокоманды опущены, то разделяющие их запятые не опускаются, кроме случая, когда отсутствуют один или несколько последних операндов макрокоманды. Параметру-идентификатору, соответствующему опущенному операнду макрокоманды, присваивается символьное значение *нулевой длины*.

Строки-продолжения. Некоторые операторы макроязыка, в частности макрокоманды и прототипы-операторы, список операндов которых может включать до двухсот элементов, допускается размещать на двух и большем числе строк. В этом случае каждую строку, кроме первой, называют *строкой-продолжением*. Признаками строки-продолжения являются запятая, один или несколько пробелов в конце предыдущей строки и пробел в начале строки-продолжения. Перенос части одного операнда не допускается.

Модель-оператор. Если макрокоманда и прототип-оператор могут занимать произвольное число строк, то модель-оператор должен записываться на одной строке. По существу, он представляет собой некий прообраз оператора машинной команды или псевдокоманды, в котором варьируемые символы в составе метки, КОП и операнда заменены параметрами-идентификаторами.

В макроязыке возможно объединение в созданном операторе значений нескольких параметров-идентификаторов, а также объединение значений параметров-идентификаторов с неизменяемыми символами базового языка. При всяком объединении после параметра-идентификатора может быть поставлена точка (в качестве разделителя), которая в созданный оператор не войдет. Постановка точки обязательна, если после параметра-идентификатора в объединении следует буква, цифра, открывающаяся скобка или точка.

Приведем некоторые примеры.

Для макроопределения

```
        MACRO                                          (8.11)
* Прототип-оператор
&NAME EXS1  &Y, &P, &S
* Модели-операторы:
&NAME LD&Y &NAME +6
        ST&Y * +5
        LDB   R. &P
        STB   &P. . R
        ADB   &P&S
        STA   &S
* Конечный оператор
        MEND
```

макрокоманда

```
M       EXS1  A, #, 100                                (8.12)
```

определяет следующие значения параметров-идентификаторов:

$\&NAME = M,\ \ \&Y = A,\ \ \&P = \#,\ \ \&S = 100.$

При обработке программы макрогенератором на месте макрокоманды (8.12) возникнут следующие операторы МНЕМОКОДа:

```
M       LDA   M+6                                      (8.13)
        STA   *+5
        LDB   R . #
        STB   #. R
        ADB   #100
        STA   100
```

В свою очередь, макрокоманда

```
M       EXS1  A,,100                                   (8.14)
```

с опущенным вторым операндом присваивает параметру-идентификатору *&P* символьное значение нулевой длины, и на ее месте после макрогенерации вместо последовательности операторов (8.13) будет получено

```
M    LDA  M+6                    (8.15)
     STA  *+5
     LDB  R.
     STB  .R
     ADB  100
     STA  100
```

Подсписки операндов. Одному параметру-идентификатору может соответствовать несколько операндов макрокоманды. Например, макрокоманда

```
M    EXS  (N2, N1, A3), R2       (8.16)
```

ставит в соответствие параметру-идентификатору *& A* в прототипе-операторе (8.8) сразу три операнда макрокоманды: *N2, N1* и *A3*, которые в совокупности называют *подсписком*.

Параметр *& A* в модели-операторе при обработке макрокоманды (8.16) будет заменен на последовательность символов (*N2, N1, A3*), что, по-видимому, не имеет смысла.

Основной вариант обращения к элементам подсписки состоит в *индексировании* параметров-идентификаторов, т.е. в присоединении справа от параметра-идентификатора целого числа в круглых скобках при его использовании в составе моделей-операторов. Так, в предыдущем примере индексированный параметр *& A* будет иметь следующие значения:

& A (1) = N2, & A (2) = N1, & A (3) = A3,

что позволяет вместо одного параметра-идентификатора в прототипе-операторе, иметь как бы три параметра в моделях-операторах макроопределения.

Заметим, что значение индексированных параметров-идентификаторов всегда определено независимо от того, является соответствующий параметр макрокоманды подсписком или нет. Это обеспечивается простым правилом: индексированные параметры-идентификаторы, которым "не хватило" элементов подсписка, получают символьные значения нулевой длины (считаются как бы опущенными).

Например, макрокоманда (8.9) определяет в качестве значения для *& A (1)*, так же как и для *& A*, идентификатор *R1*, а для *& A (i)* при $i > 1$ – символьные значения нулевой длины.

8.5. SET-идентификаторы и команды условной генерации

Выше было рассказано об основных изобразительных средствах макроязыка. Не ставя перед собой задачу его полного описания, огра-

ничимся в заключение краткой характеристикой средств, позволяющих управлять процессом генерации.

SET-идентификаторы. Помимо параметров-идентификаторов переменными элементами моделей-операторов могут являться SET-идентификаторы [от английского *set* – положить равным]. Внешне не отличимые от параметров-идентификаторов SET-идентификаторы могут использоваться как внутри макроопределения, так и вне его в составе операторов базового языка. SET-идентификаторы в определенном смысле аналогичны переменным языкам высокого уровня: SET-идентификаторы вводят в макроопределение или программу (т.е. объявляют) специальными операторами макроязыка, другими операторами (типа операторов присваивания) меняют их значения. Соответствующие присваивания осуществляет макрогенератор, и в процессе генерации вместо SET-идентификаторов в модели-операторы и операторы МНЕМОКОДа подставляются текущие значения SET-идентификаторов. Каждый SET-идентификатор может быть объявлен либо глобальным, т.е. одинаково понимаемым во всех макроопределениях и программах, где такое объявление имеет место, либо локальным, т.е. действующим в рамках одного макроопределения (одной программы). Аппарат глобальных SET-идентификаторов, а также возможность их включения в операторы МНЕМОКОДа вне макроопределений позволяют организовать систему макроопределений, обладающую внутренним алгоритмическим единством и связанную вместе с тем с операторами программы.

Команды условной генерации. Может ли модель-оператор порождать более одного оператора МНЕМОКОДа, и, вообще, могут ли модели-операторы обрабатываться макрооператором не в той последовательности, в которой они даны в макроопределении? Положительный ответ на этот далеко не праздный вопрос дает применение команд условной генерации.

Основной командой условной генерации является оператор условного перехода **AIF** с логическим выражением фортрановского типа в поле операнда, за которым следует *метка перехода.* В логическое выражение могут входить параметры-идентификаторы, SET-идентификаторы, целые числа и символы. Встретив оператор **AIF**, макрогенератор определяет значение логического выражения с учетом текущих значений параметров-идентификаторов и SET-идентификаторов. Если логическое выражение истинно, то макрогенератор переходит к обработке оператора, помеченного меткой перехода. В противном случае обрабатывается оператор, непосредственно следующий за **AIF**. Метки перехода в макроязыке отличаются от ⟨ 6.1.2. *меток*⟩ базового языка: начинаются символом *точка с запятой*, за которым следует *буква,* а за ней – еще не более трех *букв* или *цифр.* В соответствии со сложившейся терминологией метку перехода команд условной генерации называют *идентификатором последовательности.* Его допускается размещать в поле метки моделей-операторов, команд условной генера-

ции, заключительного оператора **MEND**, а также в поле метки макрокоманды, т.е. вне макроопределения. В этом случае, естественно, вне макроопределения должен быть размещен и оператор **AIF**.

Команды условной генерации, SET-идентификаторы, соглашение об опущенных операндах и ряд других элементов делают макроязык чрезвычайно гибким, позволяющим конструировать макроопределения для макрокоманд практически любой семантики.

Именно макроязык был и остается основой для разработки внутреннего и прикладного программного обеспечения УВК СМ-2М, рассмотрению которого посвящены последующие разделы книги.

Упражнение к гл. 8

Напишите макроопределение для макрокоманды

ВВОДМ *лн,ош,бф,n*

ввода в свободном формате целочисленного массива, где *лн* – логический номер устройства ввода; *ош* – метка перехода по ошибке; *бф* – метка буфера ввода; *n* – длина массива.

РАЗДЕЛ ТРЕТИЙ

ОПЕРАЦИОННЫЕ СИСТЕМЫ АСПО

Глава 9. УПРАВЛЕНИЕ ПРОГРАММАМИ

9.1. Основные пакеты программных модулей АСПО

Как уже отмечалось, все внутреннее программное обеспечение ВК СМ-2М построено по агрегатно-модульному принципу и получило название *агрегатной системы программного обеспечения* (АСПО). В основу концепции АСПО заложен принцип объединения программных модулей в две библиотеки: библиотеку макроопределений и библиотеку перемещаемых модулей. Эти библиотеки совместно с системными обрабатывающими программами, обеспечивающими подготовку и создание из этих библиотек требуемой программной системы, представляют собой типичный по структуре пакет программных модулей (ППМ) АСПО.

Среди всех ППМ, входящих в состав АСПО, различают следующие основные пакеты: ППМ для генерации операционных систем (ОС) и ППМ для компоновки задач пользователя, на использование которых ориентированы, в свою очередь, все другие проблемно-ориентированные пакеты АСПО (например, ППМ БАНК, ДППМСОИ и др.).

В свою очередь, ППМ для генерации ОС содержит три основных комплекта макробиблиотек и библиотек перемещаемых модулей, обеспечивающих генерацию различных по функционированию ОС, объединенных под тремя следующими названиями:

дисковые операционные системы (ДОС АСПО);

дисковые операционные системы – II (ДОС-II АСПО);

распределенные операционные системы (РОС АСПО).

Ориентация пользователя на выбор того или иного типа ОС неразрывно связана с областями применения ВК.

Дисковые операционные системы АСПО в основном предназначены для эксплуатации в системах *реального масштаба времени,* главной особенностью которых является то, что вычислительная система работает совместно с некоторым физическим процессом или объектом. При этом данные, поступающие в систему, должны быть обработаны с учетом временных ограничений, с тем чтобы результаты обработки можно было бы использовать для управления процессом [2].

Для режима реального масштаба времени в ДОС АСПО предусмотрено выполнение таких функций, как синхронизация выполнения процессов в соответствии с внешними событиями, предоставление процессоров

задачам в соответствии с их приоритетами, счет текущего времени и отсчет заданных интервалов времени, организация выполнения процессов в соответствии с определенным временным графиком и при необходимости динамическое изменение этого графика, функциональное резервирование устройств ввода-вывода, управление работой системы со стороны человека-оператора и т.п.

Кроме основного режима в ДОС АСПО также предусмотрен режим пакетной обработки, обеспечивающий использование ВК для подготовки и отладки программ и текстовых документов, построение информационно-справочных систем, решение инженерно-экономических задач и т.п.

Дисковые операционные системы – II АСПО по своим возможностям и структуре являются функциональным расширением ДОС АСПО, в которых параллельно с режимами реального времени и пакетной обработки обеспечивается функционирование *режима разделения времени.* Кроме того, в ДОС-II АСПО включены такие программные средства, которые обеспечивают построение высокоживучих ВК с развитой системой автоматического резервирования отдельных агрегатных модулей. При этом ДОС-II АСПО обеспечивает полную совместимость с ДОС АСПО на уровне всех задач пользователя.

РОС АСПО предназначены для управления *многомашинными вычислительными системами* с дистанционным использованием рассредоточенных ресурсов при обработке данных в системах, работающих в реальном масштабе времени. Основу РОС составляют программные модули, входящие в ДОС АСПО, а также дополнительные модули, позволяющие осуществлять обмен информацией между задачами и доступ к любому ресурсу многомашинного комплекса независимо от его расположения в одной машине либо в разных территориально рассредоточенных.

Из приведенного выше видно, что режим реального масштаба времени является главным режимом функционирования операционных систем ВК СМ-2М, поэтому нас в большей мере будет интересовать информация о программных средствах, используемых для генерации ДОС-II АСПО, где помимо режима реального времени будут освещены некоторые аспекты реализации пакетной обработки и режима разделения времени.

Для генерации ДОС-II АСПО из ППМ для генерации ОС используются следующие библиотеки:

МБФ – библиотека макроопределений, содержащая макроопределения всех модулей ДОС-II АСПО;

ОБОС – библиотека перемещаемых подпрограмм ДОС-II АСПО.

Вторым основным пакетом АСПО является ППМ для генерации задач пользователя.

Здесь также различают две наиболее часто используемые библиотеки:

ОБМДО – основная библиотека макроопределений;

ОБП – основная библиотека перемещаемых подпрограмм.

Какова же методика использования этих пакетов? Как научиться генерировать с их помощью необходимые программные системы, оптимально настроенные под заданные ресурсы и требования конкретного применения? Какая при этом должна быть предусмотрена последовательность действий? Как использовать все функциональные возможности библиотек? И наконец, как на практике научиться проектировать программные системы, предназначенные для работы в реальном масштабе времени?

В методическом плане следует заметить, что пользователь должен начинать свою работу с создания программных систем с задачами, используя второй пакет для компоновки задач пользователя, а к генерации необходимой ОС приступать после того, как он определился в конфигурации технических средств, выработал требования к режимам работы ОС, наметил нужный для своих задач сервис и т.п.

Для обеспечения первоначального функционирования указанных выше двух ППМ вместе с ВК СМ-2М поставляются две стартовые (базовые) операционные системы *#ОС12* и *#ОС13*, настроенные на самую минимальную конфигурацию технических средств и самый минимальный набор функций, необходимый для функционирования этих пакетов. С помощью стартовых операционных систем осуществляется разметка поверхности дисковых накопителей внешней памяти и загрузка на них поставляемого с вычислительным комплексом программного обеспечения.

Перед тем как перейти непосредственно к изложению материала об особенностях использования этих пакетов, приведем простейший, но достаточно характерный пример типового технологического процесса, который, с нашей точки зрения, наглядно иллюстрирует характер выполняемых задач технологического процесса, а следовательно, и требования, налагаемые на задачи, предназначенные для работы в реальном масштабе времени, и генерируемую для этих целей ОС.

9.2. Пример простейшего технологического процесса

Попытаемся на примере показать характерные стадии работы, которые необходимо пройти разработчику при проектировании автоматизированной системы управления технологическим процессом. Этот пример будем часто использовать в качестве иллюстраций функциональных возможностей ОС АСПО, поэтому рекомендуем обратить внимание читателя на описываемую здесь последовательность действий, которой необходимо придерживаться при проектировании программных систем, предназначенных для работы в реальном масштабе времени.

Пусть объект управления представляет собой резервуар с двумя клапанами. Через клапан *Кл1* подогреваемая однородная масса под давлением поступает в резервуар, а через клапан *Кл2* выходит на другой объект, потребляющий эту массу (рис. 9.1).

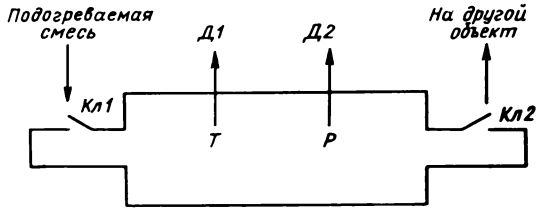


Рис. 9.1. Пример простейшего объекта управления

Кроме клапанов в резервуаре имеются два датчика *Д1* и *Д2,* предназначенные соответственно для съема значений температуры и давления внутри резервуара.

Суть технологического процесса состоит в следующем.

1. В 8 ч утра запускается вычислительный комплекс и объект управления, вводится текущее время дня.

2. В исходном состоянии оба клапана резервуара находятся в закрытом состоянии.

3. В 8 ч 05 мин должен открываться клапан *Кл1* для поступления массы в резервуар.

4. Через 10 с с датчика *Д1* начинает сниматься температура и сравниваться с первой уставкой $T1 = 280\ ^\circ C$. Съем и сравнение осуществляются периодически с интервалом $Ит1 = 20$ с.

5. Через 3 с после начала опроса температуры с датчика *Д2* с интервалом $Ир1 = 26$ с начинают сниматься давления и сравниваться с первой уставкой $P1 = 12 \cdot 10^5$ Па. Результаты опросов датчиков регистрируются в специальном файле на диске. Таким образом, эти два подпроцесса (опрос температуры и давления) должны проистекать независимо друг от друга с приведенными выше значениями технологического режима (первый технологический режим).

6. Если в некоторый момент времени температура или давление сравнились со своими первыми уставками, то должен быть изменен технологический режим, т.е. должны быть изменены периоды (*интервалы*) опроса температуры и давления (чтобы не пропустить критической ситуации), а также установлены новые значения уставок (второй технологический режим):

$Ит2 = 4$ с $\quad T2 = 440\ ^\circ C$

$Ир2 = 5$ с $\quad P2 = 16 \cdot 10^5$ Па.

7. Если хотя бы одна из контролируемых величин достигла значения своей второй уставки, то необходимо перекрыть клапан *Кл1* и через 2 с после его перекрытия открыть клапан *Кл2* для выхода массы на потребляемый ее объект.

8. При открытом клапане *Кл2* устанавливается третий режим технологического процесса:

$Ит3 = 25$ с $\quad T3 = 70\ ^\circ C$

$Ир3 = 30$ с $\quad P3 = 10 \cdot 10^5$ Па.

Таблица 9.1

Режим	Температурная уставка, °С	Временной интервал контроля температуры, с	Уставка по давлению, $\times 10^5$ Па	Временной интервал контроля давления, с
I	280	20	12	26
II	440	4	16	5
III	70	25	10	30

9. В том случае, когда измеренные значения температуры или давления снизятся до значений третьей уставки, технологический процесс повторится сначала, т.е. перекроется *Кл2*, откроется *Кл1*, и установятся значения периодов и уставок первого технологического режима работы. Значения периодов опроса и уставок каждого технологического режима приведены в сводной таблице режимов (табл. 9.1).

10. Кроме того, каждые 45 мин на печатающее устройство пульта технолога-оператора должна выдаваться оперативная информация из файла на диске, содержащая текущие значения опрашиваемых величин и время их опроса.

11. На дисплей пульта технолога-оператора в любой момент времени должно быть выведено сообщение об аварии, которая регистрируется специальным датчиком, фиксирующим неисправность в технологическом оборудовании.

12. Весь технологический процесс должен быть прекращен в 17 ч 17 мин ручными процедурами (выпущена масса, перекрыты клапаны, отключен ВК и т.п.).

Теперь, зная суть технологического процесса, попытаемся составить алгоритм управления им со стороны ВК.

Различают два подхода при решении этой задачи. Первый подход заключается в том, что весь процесс рассматривается как единый, который может быть запрограммирован одной программой пользователя. Этот подход крайне неудобен для пользователя тем, что ему приходится самому решать вопросы диспетчеризации подпроцессов, синхронизации событий, передачи информации и т.п., что всегда являлось функциями ОС. Поэтому удобен и наиболее оправдан другой подход, когда весь процесс управления функционально разбивают на подпроцессы и поручают ОС управление отдельными программами, реализующими функции этих подпроцессов. Эти программы называют *задачами пользователя*, они имеют определенный формат и являются единицами управления ОС.

Используя этот подход, в итоге получаем качественно новый результат: удается строить такие сложные программные системы, которые в виде одной программы были бы просто необозримы.

Теперь определим множество и характер задач, необходимых для реализации функций управления приведенным выше технологическим

процессом, а также выясним, какие требования надо предъявить к ОС, чтобы обеспечить функционирование и взаимодействие этих задач.

Считаем, что все управление процессом можем поручить следующим функционально независимым задачам:

задача A – управление состоянием клапана *Кл1* (открыть/закрыть);

задача B – управление состоянием клапана *Кл2*;

задача C – опрос температуры посредством датчика *Д1*;

задача D – опрос давления посредством датчика *Д2*;

задача E – выдача текущих значений опрашиваемых величин на печатающее устройство;

задача F – выдача сообщения об аварии на пульт оператора.

Далее рассмотрим несколько подробнее каждую из задач.

Задача A. Это задача в первый раз запускается по абсолютному времени дня (*абсолютная фаза*) в 8 ч 5 мин после ввода человеком-оператором текущего времени дня в систему. Получив управление в первый раз, задача блокирует свои дальнейшие запуски по времени (исключается из списка задач, запускаемых по времени), чтобы в следующий раз она могла быть запущена только из других задач. Получив управление, она выполняет следующее:

с помощью модуля устройства связи с объектом (модуль кодового управления бесконтактный – МКУБ) управляет открытием клапана *Кл1*;

включает в список задач, запускаемых по времени, задачу C, т.е. устанавливает ей следующие временные характеристики: *относительная фаза* (задержка вызова задачи относительно текущего времени дня) равна 10 с, интервал (период повторения) равен 20 с;

аналогично устанавливает временные характеристики задаче D: относительная фаза равна 13 с, интервал равен 26 с;

устанавливает признак первого режима, в котором задачи C и D осуществляют сравнение с первыми уставками опрашиваемых величин;

извещает ОС о завершении работы (в результате ОС должна перевести задачу в состояние готовности быть запущенной в очередной раз – исходное состояние).

В очередной раз задача A может быть запущена из задачи C или D для закрытия или открытия клапана *Кл1*. В этом случае при получении управления в ней должно быть реализовано разветвление по двум алгоритмам работы. С этой целью при вызове задачи в ОС должен быть предусмотрен аппарат передачи ей числовых параметров, настраивающих вызываемую задачу на выполнение той или иной ветви запрограммированного алгоритма. Если входной параметр указывает на открытие клапана *Кл1*, то задача A открывает его и полностью повторяет описанный выше алгоритм работы. Если входной параметр указывает на закрытие клапана, то, выполнив эту операцию, задача A обязана перед своим завершением запустить задачу B, указав последней на необходимость открытия клапана *Кл2*. Так как задача B выполняет те же операции откры-

тия (закрытия) клапана, то указание на выполнение операции производится с помощью аналогичного числового параметра, передаваемого ей при вызове. На рис. 9.2 приведена укрупненная блок-схема описываемого алгоритма взаимодействия задач.

Задача B. Эта задача не запускается по времени (не имеет *временных характеристик)*, а вызывается на выполнение из задач A, C и D. Указание на выполнение операции *открытие (закрытие) Кл2* передается ей значением входного числового параметра. Если указывается операция закрытия, то *Кл2* закрывается с помощью модуля МКУБ, и задача завершается.

Если входной параметр указывает на открытие клапана, то согласно описанному ранее алгоритму задача B обязана осуществить временную выдержку 2 с, а уж потом открыть клапан *Кл2*. Эту временную выдержку в дальнейшем будем называть *тайм-аутом* от задачи.

После открытия клапана устанавливаются новые временные характеристики задачам C и D, соответствующие третьему режиму выполняемого процесса. После этого задача B завершает работу.

Задача C. Эта задача запускается по времени согласно временным характеристикам, установленным в соответствии с номером технологического режима, и выполняет следующее:

с использованием подсистемы ввода аналоговой информации УСО осуществляет замер через датчик *Д1* текущего значения температуры;

запрашивает у ОС текущее время дня;

регистрирует полученные данные в специальном, едином с задачей D файле на диске;

сравнивает полученное значение температуры с текущей уставкой, соответствующей установленному режиму технологического процесса;

если сравнение не произошло, завершается работа задачи C;

при сравнении полученного значения с первой уставкой устанавливаются новые временные характеристики задаче C (т.е. самой себе) и задаче D, а также устанавливается признак второго режима для указания работы со вторыми уставками. Если осуществляется сравнение со второй уставкой, вызывается задача A с указанием закрыть клапан *Кл1*. Если же до сравнения задача работала с третьей уставкой, то согласно описанному алгоритму вызывается задача A с параметром *открыть Кл1* и задача B с параметром *закрыть Кл2*.

Задача D. Она ведет опрос давления в резервуаре и должна выполняться по аналогии с задачей C.

Задача E. Для этой задачи временные характеристики должны быть запланированы пользователем на стадии ее написания и иметь следующие значения: относительная фаза равна 45 мин, интервал равен 45 мин. Задача E выполняет чтение текущих значений опрашиваемых величин с файла на диске, а также выдачу полученных данных на печатающее устройство и завершает работу.

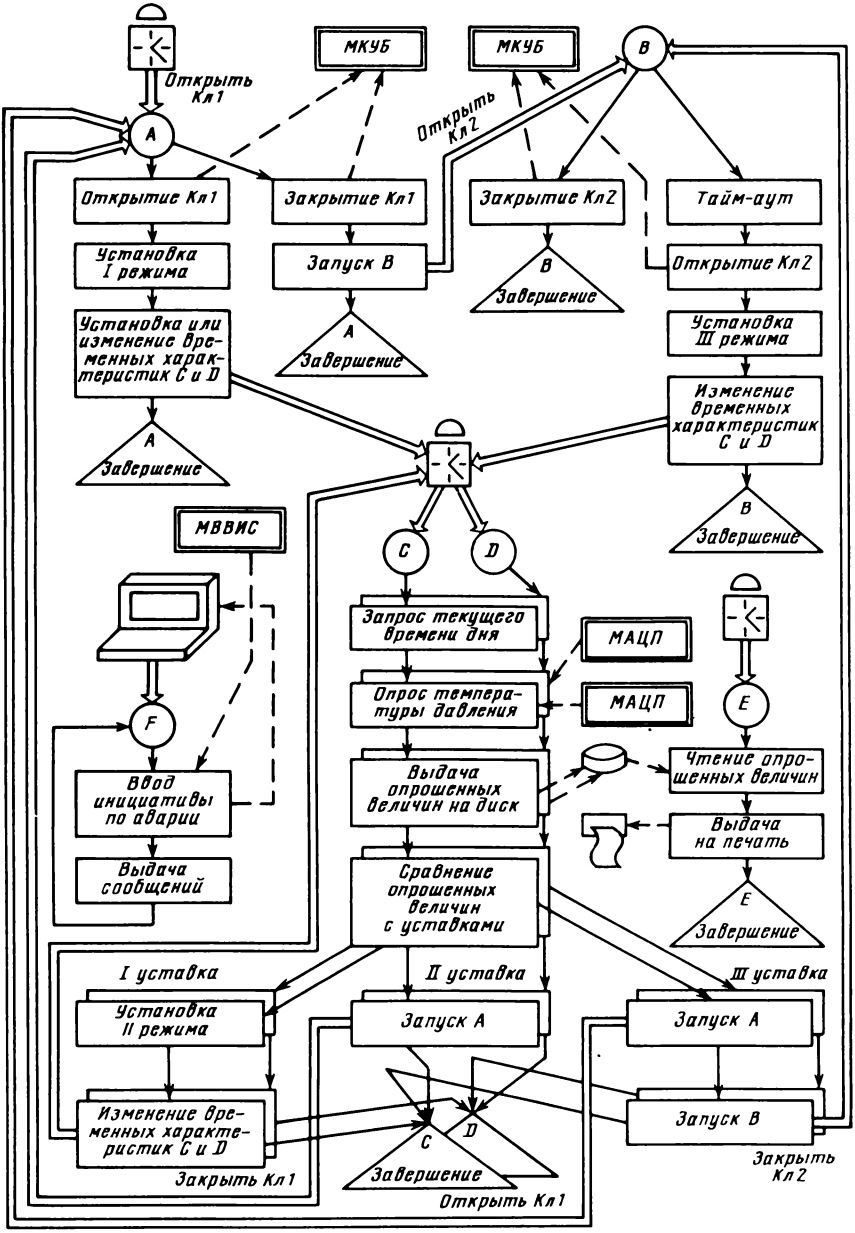


Рис. 9.2. Блок-схема алгоритма взаимодействия задач

Таблица 9.2

Используемые операции	A	B	C	D	E	F
Способы запуска задач						
Запуск с пульта оператора						+
Запуск по времени	+		+	+	+	
Запуск из другой задачи	+	+				
Операции, запрашиваемые в задачах						
Операции ввода-вывода	+	+	+	+	+	+
Запуск другой задачи с передачей ей числовых параметров	+		+	+		
Установка или изменение временных характеристик задачам	+	+	+	+		
Запрос времени дня			+	+		
Отсчет временной выдержки (тайм-аут)		+				
Операции синхронизации событий (ожидание завершения запрошенной операции, извещение о начале или конце операции)	+	+	+	+	+	+
Завершение задачи	+	+	+	+	+	

Задача F. Это аварийная задача, которая вызывается только 1 раз с пульта оператора и выполняет следующее: осуществляет запрос на ввод инициативного сигнала от модуля ввода инициативных сигналов (МВВИС), связанного с датчиком "авария", и переходит в ожидание прихода этого сигнала; если приходит сигнал от модуля МВВИС, то осуществляется выдача соответствующего сообщения об этом на пульт оператора системы; после выдачи сообщения повторяется запрос на ввод следующей возможной инициативы.

В табл. 9.2 приведены сводные данные об основных характеристиках задач.

Теперь, когда полностью определен процесс и известен весь алгоритм управления им, можно сформулировать требования к средствам управления. Под *средствами управления* будем понимать весь вычислительный комплекс в совокупности с техническими средствами и программным обеспечением. Тогда требования можно изложить в следующей последовательности.

Требования к задачам. Все перечисленные задачи могут быть написаны на любом из доступных в АСПО языках программирования. Принято использовать в этих целях МАКРОЯЗЫК, который позволяет, в частности, в сжатой и удобной форме записать обращение к ОС для выполнения какой-нибудь сервисной функции, например:

EXIO (запрос операции ввода-вывода),

STIME (временная выдержка и т.д.).

Эти обращения к ОС получили названия *макроопераций вызова супервизора.* Макроопределения этих макроопераций собраны в основной библиотеке макроопределений ОБМДО.

Из задач пользователя и библиотек, входящих в ППМ, для генерации задач пользователя, с помощью системных обрабатывающих программ компонуется программная система с задачами в виде *загрузочного модуля,* пригодного для загрузки в выделенный для этих целей раздел оперативной памяти.

Требования к техническим средствам. Для автоматизации управления данным объектом должен быть использован ВК СМ-2М со следующей минимальной конфигурацией:

ОЗУ емкостью, обеспечивающей работу не менее двух разделов (для ОС и загрузочного модуля с задачами);

накопитель внешней памяти на магнитных дисках для функционирования ОС и хранения данных задач;

модули УСО, оговоренные в алгоритмах задач;

печатающее устройство для вывода технологической информации;

пульт оператора;

таймер для обеспечения работы службы времени ОС.

Кроме перечисленного в состав ВК может быть включено дополнительное оборудование, используемое для работы системных обрабатывающих программ (магнитные ленты, печатающие устройства и т.п.).

В целях облегчения восприятия последующего материала была приведена довольно простая конфигурация технических средств, предусматривающая подключение всех УВВ, в том числе модулей УСО, непосредственно к разъемам согласователя ввода-вывода. В дальнейшем не будем интересоваться подробностями технической реализации алгоритма обмена информацией с тем или иным модулем УСО. Наше внимание должно быть уделено только интерфейсам с ОС, с помощью которых реализуются запрашиваемые от этих модулей функции.

Требования к ОС. Исходя из характера выполняемых задач и особенностей данного процесса, сформулируем требования для генерации ОС:

ОС должна быть *дисковой, многозадачной,* обеспечивающей выполнение задач в реальном масштабе времени;

в ОС должна быть предусмотрена работа с входящими в состав ВК устройствами ввода-вывода символьной и двоичной информации, модулями УСО, накопителями на магнитных дисках и т.д.;

должно быть зарезервировано два или более разделов оперативной памяти (для работы ОС и загрузочного модуля с задачами). При этом объем раздела для задач должен быть не менее объема загрузочного модуля.

Если имеются свободные разделы, то их можно использовать для других технологических процессов или для подготовки программ пользователя с помощью системных обрабатывающих программ. В общем случае может оказаться, что число задач столь велико, что они не умещаются в максимально допустимые размеры одного загрузочного модуля. В этом случае весь технологический процесс делится на два загрузочных модуля с задачами, и для управления этим процессом исполь-

зуются два раздела памяти. Тогда ОС обязана обеспечить обмен информацией между задачами, принадлежащими разным разделам;

должна быть предусмотрена работа *службы времени* для организации подсчета времени дня, поддержки временного графика выполнения задач, организации тайм-аутов от задач и т.д.;

должны быть включены все модули ОС, ответственные за выполнение макроопераций, запрашиваемых из задач;

ОС должна обеспечить связь с оператором на уровне директив, вводимых с выделенных для этих целей пультов оператора.

Помимо перечисленных требований в генерируемую ОС можно включить дополнительные средства повышения живучести, *анализа сбоев и реконфигурации.*

Определив все требования к генерируемой ОС, необходимо на МАКРОЯЗЫКе оформить *программу генерации ОС.*

Из программы генерации ОС и библиотек, входящих в ППМ, для генерации ОС, с помощью системных обрабатывающих программ компонуется программная система со сгенерированной ОС в виде загрузочного модуля, пригодного для загрузки и запуска в нулевом разделе оперативной памяти.

9.3. Основные определения, терминология

Систематизируя некоторые понятия, введенные при разборе примера, а также в целях обеспечения единой терминологии для дальнейшего, введем следующие основные понятия и определения, неразрывно связанные с архитектурными особенностями ВК СМ-2М и структурой ОС.

Программа – последовательность логически связанных между собой команд и данных, реализующих в вычислительном комплексе определенный алгоритм запрограммированного процесса. Различают несколько типов программ, которые отличаются друг от друга форматом написания и дисциплиной обслуживания (выполнения). К ним относятся *главные программы, подпрограммы и программные сегменты.*

Как уже упоминалось в гл. 5, главная программа начинается оператором **NAM** с типом **0–4**, который содержит информацию о дисциплине обслуживания программы.

При этом тип **0** используется для указания главных программ модулей ОС, тип **1–4** (все равно какой) – для указания главных программ задач пользователя. Здесь и далее мнемоника всех операторов приведена на базовом языке программирования МНЕМОКОД.

Подпрограмма – это часть главной программы, которая временно получает из нее управление и возвращает его главной программе по окончании работы (см. § 3.4).

Сегмент – это часть главной программы (по аналогии с подпрограммой), которая постоянно находится на диске и загружается в специаль-

ную область загрузки сегментов в памяти только по мере необходимости.

Программный модуль – программа на любом из языков программирования (*исходный* формат программного модуля), или на языке компоновщика программ (*объектный* формат программного модуля – результат работы транслятора), или в кодах машины (*абсолютный* формат), организованная на носителе, внешней памяти или в ОЗУ.

Все трансляторы АСПО выдают результирующую программу в едином для всех формате – формате объектного (перемещаемого) программного модуля. В этом формате программный модуль не зависит от конкретных адресов расположения его в памяти, еще не зафиксированы его связи с другими программными модулями, но в нем уже не содержится никакой информации, связанной с каким-либо конкретным языком программирования. Изготовление программы в кодах машины из таких объектных программных модулей является основной функцией специальной системной обрабатывающей программы – компоновщика программ (в дальнейшем – компоновщик), единой для всех трансляторов, входящих в состав АСПО.

Задача – одна или группа взаимосвязанных, выполняющихся последовательно программ, одна из которых обязательно является главной. Для операционной системы и, как следствие, для человека-оператора системы задача является основной единицей управления. При этом в зависимости от места расположения задач различают три вида задач: ОЗУ-резидентные, частично ОЗУ-резидентные (сегментированные) и диск-резидентные.

ОЗУ-*резидентная задача* – задача, постоянно находящаяся в памяти независимо от степени ее активности (выполняется или не выполняется). Все задачи, приведенные в предыдущем примере технологического процесса, являются ОЗУ-резидентными.

Сегментированная задача – задача, состоящая из главной программы, постоянно находящейся в оперативной памяти, и одного или несколько сегментов, хранящихся во внешней памяти (на диске) и вызываемых в оперативную память только по мере необходимости. Типичным примером сегментированной задачи является любой транслятор, осуществляющий трансляцию за несколько просмотров, где каждый просмотр оформлен в виде сегмента к главной программе.

Диск-резидентная задача – задача, постоянно хранящаяся на диске и загружаемая в память на время ее выполнения.

Задача обладает именем, роль которого играет наименование главной программы задачи.

Загрузочный модуль – программный модуль в кодах машины (в абсолютном формате), пригодный для загрузки его в память (раздел) на выполнение. Загрузочный модуль является результатом работы компоновщика, который в зависимости от типа поступаемых на компоновку программ генерирует тот или иной тип загрузочного модуля.

Фактически, загрузочный модуль есть конечный результат работы системы подготовки программ и является единственным видом программного модуля, который может быть загружен в память и выполнен под управлением операционной системы.

Раздел оперативной памяти (в дальнейшем – раздел) – участок смежных ячеек оперативной памяти, предназначенный для загрузки и выполнения загрузочных модулей (см. § 2.3).

Операционная система (управляющая программа) – программная часть вычислительного комплекса, предназначенная для управления работой этим комплексом. Все программные модули ОС компонуются в загрузочный модуль, работающий в привилегированном разделе оперативной памяти.

Обрабатывающая программа – задача пользователя или системная обрабатывающая программа, предназначенная для выполнения, как правило, в непривилегированном режиме работы процессора.

Системная обрабатывающая программа – стандартная обрабатывающая программа, поставляемая в составе программного обеспечения вычислительного комплекса: транслятор, редактор, отладчик, макрогенератор, обслуживающая программа и т.п. Как правило, каждая системная обрабатывающая программа представляет собой однозадачный загрузочный модуль (как сегментированный, так и несегментированный), выполняющийся в непривилегированном разделе оперативной памяти.

Файл – логически связанная по обработке совокупность данных, имеющая имя (дисковый файл) либо заканчивающаяся маркером конца файла (файл на последовательных устройствах – магнитная лента, миникассета и т.п.). В частности, это могут быть программа в объективном или исходном формате, загрузочный модуль или же просто промежуточные данные, хранимые только во время выполнения какой-либо программы.

Пакет программных модулей АСПО (ППМ) – совокупность, состоящая из библиотек макроопределений, библиотек перемещаемых подпрограмм и набора системных обрабатывающих программ.

9.4. Подготовка загрузочных модулей

По своему функциональному назначению, содержимому и средствам загрузки в память загрузочные модули делятся на три типа:

I – загрузочный модуль, содержащий только ОС;

II – загрузочный модуль, содержащий только обрабатывающие программы;

III – загрузочный модуль смешанного типа, содержащий модули ОС и обрабатывающие программы.

Тип загрузочного модуля определяется компоновщиком в зависимости от типа программ (в операторе **NAM**), поступаемых на компонов-

ку. С типом загрузочного модуля связано также и использование тех или иных библиотек макроопределений и библиотек перемещаемых подпрограмм.

Ниже приводятся краткая характеристика и особенности каждого типа загрузочных модулей.

Загрузочный модуль I типа. Загрузочный модуль этого типа создается с помощью ППМ для генерации ОС. Создание загрузочного модуля этого типа осуществляется в четыре этапа.

На *первом* этапе осуществляется подготовка на МАКРОЯЗЫКе программы генерации ОС, в которой пользователь определяет свои требования к генерируемой ОС, к дисциплине работы ее модулей, к конфигурации технических средств и т.п.

На *втором* этапе с помощью макрогенератора осуществляется макрогенерация программ с получением на выходе той же программы генерации, но уже в формате МНЕМОКОДа. В макрогенерации принимает участие специальная библиотека макроопределений БМФ.

На *третьем* этапе полученная программа транслируется с помощью транслятора с МНЕМОКОДа, результатом работы которого является та же программа генерации, но уже в объектном (перемещаемом) формате. Полученная таким образом программа является как бы скелетом будущей ОС, в которой зафиксированы ссылки к требуемым объектным модулям, находящимся в библиотеке перемещаемых подпрограмм ОС.

Четвертый этап является основным этапом работы, на котором осуществляется объединение корневой части ОС с теми модулями библиотеки перемещаемых подпрограмм ОС (ОБОС), к которым зафиксированы обращения в программе генерации. Объединение с библиотекой и генерация загрузочного модуля осуществляются компоновщиком программ.

Пользователь в операторе **NAM** программы генерации ОС обязан задать тип "системная программа" (тип **0**), что является указанием компоновщику скомпоновать загрузочный модуль I типа. Загрузочный модуль с ОС создается на диске со стандартным именем файла *#OCnn*, где nn – собственно номер ОС, задаваемый в программе генерации. Загрузочный модуль I типа загружается в нулевой раздел с помощью микропрограммного начального загрузчика. Перед загрузкой ОС в память номер ОС набирают на клавишном регистре инженерной панели для указания начальному загрузчику на выбранную для загрузки ОС.

Загрузочный модуль II типа. Загрузочный модуль этого типа компонуется из обрабатывающих программ и предназначен для работы в непривилегированном разделе памяти. В загрузочный модуль этого типа может входить одна или несколько задач, что определяет соответствующее название модуля: однозадачный или многозадачный. По аналогии с модулем I типа создание модуля II типа осуществляется в три, а в ряде случаев и в четыре этапа.

На *первом* этапе подготавливаются одна или несколько программ, написанных на выбранном для этих целей языке программирования

(МНЕМОКОД, МАКРОЯЗЫК, ФОРТРАН, АЛГОЛ и др.). Возможности МАКРОЯЗЫКа позволяют писать обрабатывающие программы на этом языке с использованием стандартных макрокоманд обращения (макроопераций) к ОС для выполнения той или иной сервисной функции. Очевидно, в этом случае пользователю понадобится предварительная макрообработка таких программ, которая выполняется на втором этапе.

Второй этап – обработка программ макрогенератором. Естественно, что этот этап необходим только для программ, написанных с использованием макрокоманд, в частности макрокоманд вызова супервизора.

На *третьем* этапе программы в исходном формате транслируются трансляторами с соответствующих языков программирования и получаются результирующие программы в объектном (перемещаемом) формате.

На *четвертом* этапе компоновщик объединяет транслированные программы пользователя и библиотечные подпрограммы в задачи и генерирует однозадачный или многозадачный загрузочный модуль II типа. При этом из библиотек выбираются только те подпрограммы, к которым есть внешние обращения в обрабатывающих программах. Напомним, что среди библиотечных подпрограмм, так же как и среди подпрограмм пользователя, могут быть повторно (*нереентерабельные*) и параллельно (*реентерабельные*) используемые подпрограммы.

Функцию резервирования рабочих ячеек для реентерабельных подпрограмм (областей реентерабельности) берет на себя компоновщик и для тех задач, которые применяют реентерабельные подпрограммы, дополнительно выделяет рабочие области по 45 ячеек и помещает их в конец загрузочного модуля. Адрес этой области перед вызовом задачи подготавливается в регистре *V*. Если в задаче не используется этот регистр, программирование обращений к реентерабельной подпрограмме не требует дополнительных усилий (см. § 3.4).

Естественно, что при написании пользователем задач в операторах **NAM** должен быть указан тип "Обрабатывающая программа" (тип **1–4**), что является признаком компоновки загрузочного модуля II типа.

Для каждой ОЗУ-резидентной задачи, компонуемой в этом загрузочном модуле, компоновщик формирует специальную 27-словную таблицу – *блок управления задачей* (БУЗ) и помещает БУЗ в нулевую страницу, куда также помещает ячейки связи и постоянно распределенные системные ячейки (стандартные ячейки нулевой страницы). Вся эта информация будет защищена во время работы загрузочного модуля.

В БУЗе каждой задачи записывается вся необходимая информация для будущего выполнения задачи: имя задачи, приоритет, временные характеристики, адрес запуска и др. – эта информация извлекается компоновщиком из оператора **NAM** главной программы компонуемой задачи и будет исользована диспетчером задач ОС для выбора и выполнения задач.

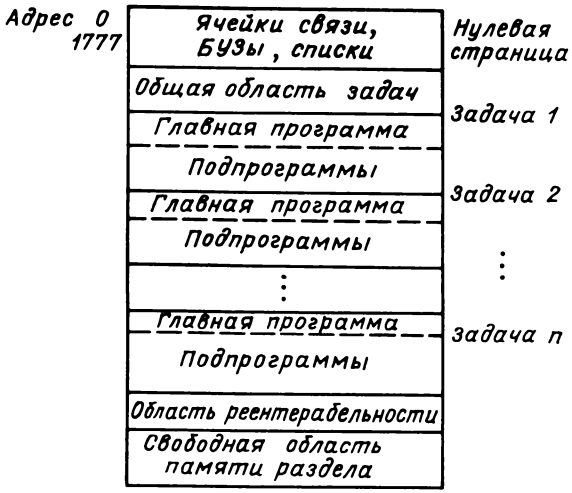


Рис. 9.3. Размещение загрузочного модуля с ОЗУ-резидентными задачами в разделе памяти

В многозадачных загрузочных модулях все БУЗы в конце компоновки связываются в *цепочку* БУЗов, упорядоченную по убыванию приоритетов задач (для организации цепочки применяются начальные ячейки каждого БУЗа). Если две задачи имеют одинаковый приоритет, то их место в цепочке определяется порядком компоновки.

После того как загрузочный модуль с ОЗУ-резидентными задачами загружен в раздел памяти, он имеет вид, показанный на рис. 9.3.

В общем случае при размещении загрузочного модуля в память в разделе остается свободная область, не занятая модулем, которая может быть использована в задачах в качестве дополнительной рабочей области. С этой целью в стандартные ячейки нулевой страницы раздела помещают адреса границ этой области.

Одна из задач (только одна), включаемая в загрузочный модуль, может быть сегментированной, т.е. состоящей из главной программы (возможно, с подпрограммами) и ее сегментов. Компоновщик включает все ОЗУ-резидентные задачи и главную программу сегментированной задачи в ОЗУ-резидентную часть загрузочного модуля, а ее сегменты – в дискрезидентную, настраивая их на загрузку в одно и то же место в памяти, называемое *областью загрузки сегментов*.

Эта область располагается за главной программой. Вслед за последним сегментом на диске формируется *таблица сегментов*, каждый шестнадцатисловный элемент которой (по аналогии с БУЗом для ОЗУ-резидентных задач) содержит информацию о выполняемом сегменте: имя сегмента, его адрес на диске и т.п.

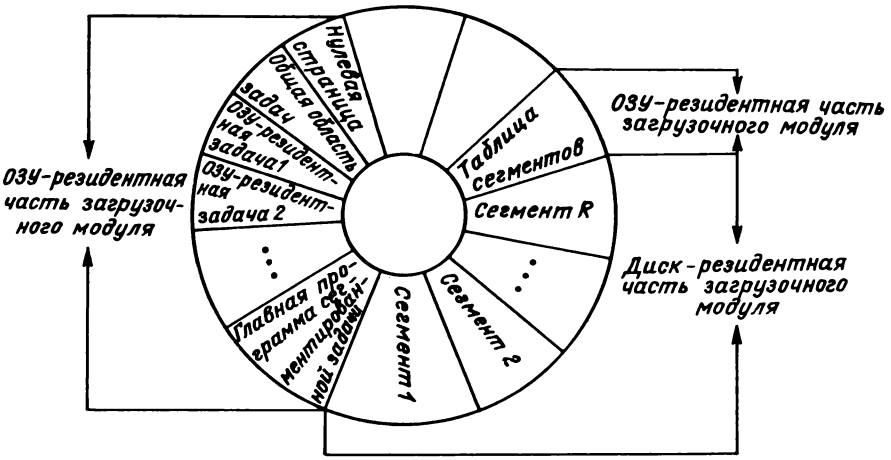


Рис. 9.4. Формат загрузочного модуля с сегментированной задачей в файле на диске

Таблица сегментов также входит в ОЗУ-резидентную часть загрузочного модуля. На рис. 9.4 приведен формат загрузочного модуля с сегментированной задачей в файле на диске.

При загрузке загрузочного модуля с сегментированной задачей с файла в раздел загружается только его ОЗУ-резидентная часть, при этом таблица сегментов помещается в конец раздела, в результате чего сокращается свободная область раздела.

Если в главной программе необходимо обратиться к сегменту, то при создании загрузочного модуля она компонуется со специальной подпрограммой загрузки сегментов (*NSEGM*), находящейся в библиотеке стандартных подпрограмм (*# ОБП*). Эта подпрограмма по имени сегмента отыскивает в таблице сегментов нужный элемент, определяет адрес расположения сегмента на диске, загружает каждый сегмент в область загрузки сегментов и передает ему управление.

Отработав, сегмент, используя стандартную подпрограмму из той же библиотеки (*RSEGM*), возвращает управление вызвавшей его главной программе. Хотя отработанный сегмент остается в памяти, при необходимости на его место будет загружен новый сегмент. Особенностью работы с сегментами является то, что каждый последующий сегмент может быть вызван на место предыдущего как из главной программы, так и из предыдущего сегмента. Такую дисциплиту работы называют *последовательной,* она обусловлена тем, что компоновщик резервирует только одну область загрузки сегментов. Это также определяет и то требование, что в загрузочном модуле может быть только одна сегментированная задача. Размещение загрузочного модуля с сегментированной задачей в разделе памяти приведено на рис. 9.5.

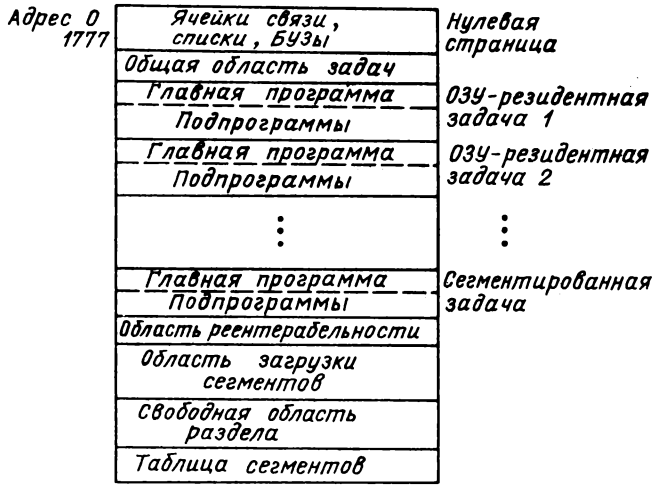


Рис. 9.5. Размещение загрузочного модуля с сегментированной задачей в разделе памяти

Загрузочный модуль II типа оформляется в виде файла на диске под именем, указанным пользователем при компоновке. Имя файла может содержать до восьми символов. Для однозадачных загрузочных модулей рекомендуется присваивать имя файлу с загрузочным модулем по имени единственной задачи. Примерами могут служить однозадачные загрузочные модули с системными обрабатывающими программами: *МНКД, АЛГОЛ, ФТН4, SLIDE, ПОФ* и др.

Загрузочный модуль II типа загружается с диска в любой непривилегированный раздел с помощью последовательности команд оператора:

:*нр*
:ВД,*имяф* [:[*имяд*] [:*пнд*]]

где *нр* – номер раздела, куда загружается модуль; *имяф* – имя файла с загрузочным модулем; *имяд* – имя диска с этим файлом; *пнд* – порядковый номер диска.

После загрузки загрузочного модуля в раздел любая задача из этого загрузочного модуля может быть вызвана на выполнение с помощью директивы:

:ВЫ[ЗОВ] ,*имяз,p1,p2, . . . ,p5,*

где *имяз* – имя задачи; *p1, p2, . . . , p5* – параметры, передаваемые задаче с пульта оператора.

Возможно совмещение загрузки файла с загрузочным модулем и вызова старшей по приоритету задачи в этом модуле:

:СТ[АРТ] ,*имяф* [:[*имяд*] [:*пнд*]]

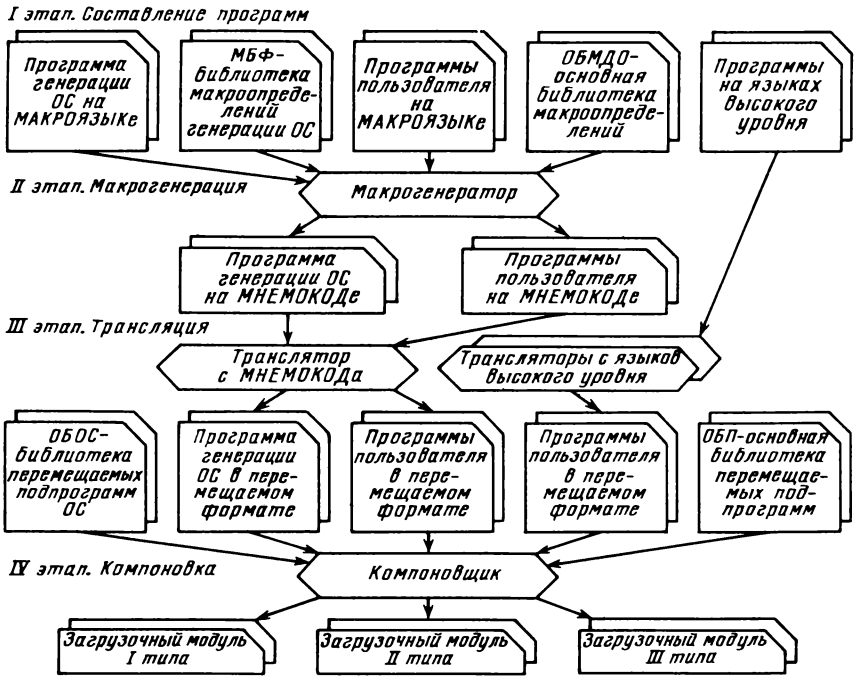


Рис. 9.6. Основные этапы создания загрузочных модулей

Этой директивой рекомендуется пользоваться при запуске однозадачных загрузочных модулей.

Загрузочный модуль III типа. Этот модуль является загрузочным модулем смешанного типа, который содержит ОС и задачи пользователя, предназначенные для выполнения в привилегированном режиме работы процессора. Вполне очевидно, что для создания модуля этого типа пользователю придется раздельно выполнить описанные ранее три этапа работ, связанных с подготовкой к компоновке модулей ОС и задач пользователя. И только на четвертом этапе осуществляется совместная компоновка модулей ОС и задач пользователя. При этом должен соблюдаться строгий порядок, который предусматривает, что первыми на компоновку должны поступать системные программы, а затем уже задачи пользователя. Естественно, что у таких загрузочных модулей нулевая страница будет одновременно содержать информацию, необходимую для работы ОС и задач пользователя.

Загрузочный модуль III типа, так же как и I типа, создается под стандартным именем *# OCnn* и загружается только в нулевой раздел с по-

мощью начального дискового загрузчика. Для вызова задач из этого раздела используются команды оператора:

:0
:**ВЫ! [ЗОВ]** ,*имяз,p1,p2, . . . ,p5*

Примером такого загрузочного модуля является стартовая операционная система *# OC13*, представляющая собой скомпонованные в единый загрузочный модуль операционную систему и системную обрабатывающую программу *РМД* – разметка диска.

Основные этапы работ по созданию загрузочных модулей различного типа представлены на рис. 9.6.

9.5. Оформление ОЗУ-резидентных задач

Освоив основные языки программирования, пользователь приступает к написанию на одном из этих языков задач, реализующих запрограммированный алгоритм управления. При этом необходимо соблюсти определенные правила оформления задач, чтобы обеспечить их функционирование под управлением операционной системы АСПО. Ниже приводятся требования, которые необходимо соблюдать пользователю при написании главной программы ОЗУ-резидентной задачи, написанной на МАКРОЯЗЫКе.

Операторы ASMB и NAM. Первым оператором программы на МАКРОЯЗЫКЕ должен быть управляющий оператор, определяющий режим работы и выход транслятора с МНЕМОКОДа. Формат управляющего оператора (см. § 5.3)

ASMB,*p1,p2, . . . ,pn*

Следующим оператором, определяющим название и основные параметры задачи, должен быть оператор NAM, имеющий следующий формат:

NAM *имяз,тип,пр,масшт,инт,ф1,ф2,ф3,ф4*

Описание значений, которые могут принимать эти параметры, было приведено в § 5.3.

Здесь же уточним некоторые ранее введенные понятия: *фаза* – это время первого запуска задачи, которое может быть абсолютным и относительным. Абсолютная фаза указывает на астрономическое время дня, в которое задача должна быть вызвана в первый раз, и оно измеряется в часах, минутах, секундах и десятках миллисекунд.

Задача А в примере технологического процесса (см. § 9.2) является ОЗУ-резидентной задачей, запускаемой по абсолютной фазе.

Относительная фаза – это первоначальная временная задержка перед первым выполнением задачи относительно текущего времени дня. На-

пример, задачи C и D из этого же примера запускаются по относительной фазе. После загрузки в память задач с запланированной относительной фазой служба времени ОС преобразует все относительные фазы в абсолютные, сложив их с текущим временем дня.

Интервал – это промежуток времени между двумя последовательными запусками задачи. Служба времени ОС из всех задач, запускаемых по времени, формирует специальный список (цепочку), используя стандартные ячейки соответствующих БУЗов.

Если параметр *масшт* в операторе **NAM** равен нулю или же опущен, то задача не будет запускаться по времени, и остальные параметры должны быть также опущены. Если *инт* равен нулю, то задача вызовется только 1 раз согласно своей фазе, а затем будет исключена из списка задач, запускаемых по времени. Иллюстрацией к этому может служить оператор задачи A из примера (см. § 9.2).

```
NAM A,1,,2,0,8,5.
```

Если временные характеристики опущены в операторе **NAM**, то предполагается, что задача будет вызываться только на однократное выполнение с пульта оператора либо из другой задачи. Однако в любой момент она может быть включена в список задач, выполняемых по времени, т.е. ей могут быть установлены временные характеристики с пульта оператора по специальной команде оператора **:ПВ** либо из другой задачи по макрооперации **TURN**. Этими же средствами задачам можно изменять прежние значения фаз и интервалов.

При компоновке все параметры оператора **NAM** помещаются компоновщиком в БУЗ компонуемой задачи.

Входные параметры. Далее идет тело программы, реализующее основной алгоритм работы задачи. Если задача запускается с пульта оператора или из другой задачи, то ей может быть передано до пяти числовых параметров, которые запоминаются в специальной области БУЗа задачи. Эти параметры всегда доступны задаче, так как адрес этой области передается в регистре *B* при входе в задачу. Например, следующая цепочка операторов МНЕМОКОДа реализует чтение первых двух параметров и запоминание их в ячейках *ПАРМ1* и *ПАРМ2:*

```
LDA  1,I
STA  ПАРМ1
INB
LDA  1,I
STA  ПАРМ2
```

Использование макрокоманд вызова супервизора. В теле задачи может быть заказано выполнение любой макрооперации вызова супервизора, с помощью которой задача запрашивает выполнение необходимой ей сервисной функции, обеспечиваемой возможностями ОС.

В программе на МНЕМОКОДе (МАКРОЯЗЫКе) вызов супервизора программируется с помощью макрокоманды, имеющей следующий общий вид:

метка имям p1,p2,. . . ,pn,

где *метка* – идентификатор данной макрокоманды в программе; *имям* – наименование макрокоманды; *p1, p2, . . . , pn* – параметры макрокоманды.

Здесь код вызова супервизора (*квс*) определяется наименованием макрокоманды и не входит в ее параметры. При обработке макрокоманды макрогенератором она преобразуется в соответствующую вызывающую последовательность на МНЕМОКОДе:

```
SVC
JMP  *+n +2
DEF  квс,I
DEF  p1,I
DEF  p2,I
...
DEF  pn,I
```

Здесь первая команда – команда вызова супервизора **SVC**, вторая – команда обхода списка параметров **JMP** * *+n +2*, где *n* – число параметров макрооперации. Далее следуют адреса параметров с единицей в нулевом разряде, позволяющей ОС выбирать их значения по базе непривилегированного состояния (с помощью автоиндексации) см. § 3.1.

Параметр *квс* указывает код вызова супервизора (код запроса) и определяет номер макрооперации, задаваемой данной вызывающей последовательностью.

Обработав программное прерывание, т.е. запустив выполнение макрооперации, ОС возвращает управление задаче, обойдя список параметров. Здесь различаются два типа макроопераций: однотактная и многотактная.

Однотактная – это такая макрооперация, которая к моменту возврата управления задаче уже полностью завершилась. Например, макрооперация **TIME**, запрашивающая текущее время дня.

Многотактная – это такая макрооперация, которая к моменту возврата управления задаче еще не завершена, а находится в запущенном состоянии, например, макрооперация **EXIO**, запрашивающая операции ввода-вывода. Получив управление после запуска макрооперации, задача может поступить двояко: либо ожидать завершения макрооперации, либо продолжать работу без ожидания завершения.

Выбор одного из этих состояний определяется требованиями алгоритма и неразрывно связан с понятием синхронизации событий, которое подробно описано в следующем параграфе. Описание всех макроопераций вызова супервизора см. в гл. 10.

Обращение к подпрограммам. В теле главной программы задачи может быть использовано обращение к собственным подпрограммам или библиотечным подпрограммам.

При обращении к реентерабельной подпрограмме в регистре *Y* автоматически подготавливается начальный адрес области реентерабельности. Поэтому при использовании этого регистра следует обеспечить его запоминание и восстановление перед обращением к реентерабельной подпрограмме.

Завершение задачи. ОЗУ-резидентная задача должна завершать свое выполнение макрооперацией EXIT, которая оповещает ОС об окончании работы задачи. По этой макрооперации задача переводится в исходное состояние – состояние готовности быть вызванной вновь.

Операторы **END и ENDM** должны быть последними операторами псевдокоманд программы (см. § 5.3 и 8.2).

Запуск ОЗУ-резидентной задачи. ОЗУ-резидентная задача может быть запущена на выполнение тремя способами: с пульта оператора, из другой задачи или по времени. Запуск с пульта оператора осуществляется командой оператора **:ВЫЗОВ** (был описан в § 9.4).

Для запуска из другой задачи используется макрооперация RUN, которая по аналогии с командой оператора **:ВЫЗОВ** запускает задачу с передачей ей до пяти числовых параметров.

По времени вызываются задачи, которые включены в список задач, выполняемых по времени, и у которых время вызова совпадает с текущим временем дня.

Остальные особенности программирования ОЗУ-резидентных задач связаны с понятием *синхронизации событий.*

9.6. Состояние задач. Синхронизация событий

Понятие синхронизации событий неразрывно связано с состоянием задач, в котором они могут пребывать, находясь внутри работающего раздела памяти.

Различают два состояния задач: активное, когда задачи готовы к выполнению, и пассивное, когда задачи не готовы к выполнению. При этом активная задача будет выполняться в данный момент, если она имеет наивысший приоритет среди задач, готовых к выполнению.

Пассивное состояние – это такое состояние, когда задачи находятся в *ожидании какого-нибудь события,* например ожидание запуска со стороны оператора, ожидание завершения операции ввода-вывода и т. п.

Задача может быть переведена в *состояние готовности* путем соответствующей пометки в БУЗе, которая делается ОС в следующих случаях:

1) введена команда оператора **:СТАРТ** или **:ВЫЗОВ**, (для задач, находящихся в исходном состоянии);

2) введена команда оператора **:ПУСК,** (для задач, переведенных в состояние приостанова по макрооперации **PAUSE** или по команде оператора **:ПАУЗА)**;

3) подсистема службы времени обнаружила, что текущее время дня совпало со временем вызова задачи;

4) выполняется макрооперация **RUN** для запуска из другой задачи;

5) если задача ожидала завершения заказанной ею макрооперации и эта макрооперация завершилась.

Задача может быть переведена в *состояние неготовности* в одном из следующих случаев:

1) введена команда оператора **:ПАУЗА** или в задаче запрашивается макрооперация **PAUSE**;

2) задача ожидает завершения заказанной ею макрооперации (события);

3) задача аварийно приостановлена по прерываниям от схем контроля процессора;

4) при выполнении в задаче макрооперации **EXIT** (т.е. при переводе задачи в исходное состояние).

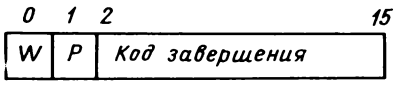
Любой перевод задачи в состояние готовности будем называть *постированием.* Соответственно любой перевод из этого состояния в состояние неготовности называется *подвешиванием.* Признак состояния задачи устанавливается в специальной ячейке **БУЗ,** которую называют *счетчиком ожидаемых событий.* Если счетчик равен нулю, то задача отпостирована (готова к выполнению); если счетчик равен минус единице, то задача подвешена.

Событием называют любое действие в системе, которое может вызвать переход задачи из одного состояния в другое: макрооперацию, запрошенную в задаче, любую команду оператора, переводящую задачу из одного состояния в другое (**:СТАРТ, :ВЫЗОВ, :ПАУЗА, :ПУСК**), момент наступления времени выполнения задачи, аварийное прекращение выполнения задачи.

Всякий переход задачи из одного состояния в другое вызывает нарушение установившегося баланса готовых и не готовых к выполнению задач, что обязывает операционную систему передавать управление своему специальному модулю—*диспетчеру задач* для временного приостанова работающей задачи и возможного запуска задачи с более высоким приоритетом, появившейся в результате нарушения этого баланса.

Диспетчер среди всех готовых к выполнению задач передает управление той, которая старше по приоритету. С этой целью он просматривает цепочку БУЗов по приоритету в поисках первой в цепочке задачи, готовой к выполнению. В общем случае это может быть та же задача, которая выполнялась до выхода на диспетчер, либо новая задача, появившаяся в результате изменения ее состояния.

Рис. 9.7. Структура БУСа



Диспетчер определяет готовую к выполнению задачу по нулевому значению счетчика ожидаемых событий и при этом не анализирует, каким из пяти способов задача была приведена в это состояние, т.е. *не существует приоритета между событиями* (существенную роль для первоочередного запуска задачи играет только ее *приоритет*).

Для идентификации состояния задачи и состояния заказанной ею макрооперации (события) в самой задаче при программировании должны резервироваться специальные ячейки, называемые *блоками управления событием* (БУС). Любой системный модуль, ответственный за выполнение макрооперации, делает отметку в БУСе о начале макрооперации (запуске события), а по окончании макрооперации – отметку о завершении (наступлении события). Тогда задача в любой момент времени может проанализировать содержимое соответствующего БУСа и узнать, в каком состоянии находится запущенная ею макрооперация, помеченная этим БУСом.

Для отметки в БУСах состояния задач по отношению к событиям все модули ОС, ответственные за выполнение событий, используют специальные *системные подпрограммы синхронизации событий POST* и *WAIT*, которые и дали название соответствующим операциям синхронизации: *постирование* и *ожидание* (*подвешивание*). В этих системных подпрограммах осуществляется перевод задачи из одного состояния в другое с формированием счетчика ожидаемых событий в соответствующих БУЗах.

Для идентификации состояния задачи по отношению к событию используются два старших разряда ячейки БУС (рис. 9.7). Эти разряды могут принимать следующие значения:

WP=00 – событие запущено, но задача не ожидает его завершения;
WP=01 – заказанное событие завершилось, но задача не ожидает его завершения;
WP=10 – задача ожидает завершения события, но оно еще не завершилось;
WP=00 – задача ожидала события, и оно завершилось.

Как видно из приведенного выше, первое и четвертое состояния задачи идентифицируются в БУСе одинаково.

Любая макрооперация, за исключением **WAITE** и **POSTE**, в БУСе, указанном в качестве параметра в этой макрооперации, при запуске устанавливает исходное значение WP = 00. После запуска макрооперации управление возвращается задаче, которая, как уже отмечалось, может поступить двояко: *ожидать завершения* макрооперации, *не ожидать завершения* макрооперации.

Если задача не ожидает завершения, то БУС остается в исходном состоянии во время работы задачи до тех пор, пока модуль ОС, ответствен-

ный за выполнение этой макрооперации, не сделает отметку о завершении макрооперации (WP = 01), обратившись к подпрограмме *POST*.

Если задача переходит в ожидание, то она использует макрооперацию ожидания завершения события **WAITE**, указав в качестве параметра адрес того БУСа, которым помечена запущенная макрооперация. В этом случае макрооперация **WAITE** вызывает системную подпрограмму *WAIT*, и если событие не завершилось, устанавливает WP = 10 и подвешивает задачу.

Конечно же, задача может не использовать макрооперацию **WAITE** в ожидании завершения события, а самостоятельно анализировать содержимое БУСа в ожидании появления WP, равного 01. В этом случае она не подвешивается, а следовательно, не дает возможности работать задачам с меньшим приоритетом.

Следует помнить, что только тогда, когда высокоприоритетные задачи не выполняются или находятся в подвешенном состоянии (применяются макрооперации **EXIT, WAITE, WAITM, PAUSE** или они приостанавливаются с пульта оператора), могут выполняться задачи с меньшим приоритетом. Это обстоятельство обязательно необходимо учитывать при проектировании задач, предназначенных для работы в реальном масштабе времени.

Для идентификации состояния задачи по отношению к событиям, связанным с ее запуском по времени, запуском или приостановом с пульта или аварийным приостановом, используются *стандартные* БУСы, резервируемые ОС непосредственно в БУЗе задачи. Здесь также любой модуль ОС, ответственный за выполнение соответствующего события, делает отметку о состоянии этого события (использует подпрограммы *POST* и *WAIT*).

9.7. Обращение к сегментам

Все правила, которые необходимо соблюдать пользователю при компоновке загрузочного модуля с сегментированной задачей, описаны в § 9.4. Ниже приводятся основные требования, которые должны выполняться пользователем при написании сегментированных задач.

1. Для вызова из главной программы или из другого сегмента с именем *имяс* используется библиотечная подпрограмма *NSEGM*, находящаяся в основной библиотеке подпрограмм (# ОБП). При этом должен соблюдаться следующий формат вызова:

```
EXT  NSEGM
...
JSB  NSEGM
DEF  *+3[+n]        n = 1,2,3,4,5.
DEF  имяс
DEF  бз
[DEF pl
```

```
...
DEF  p5|
```

Здесь *имяс* – имя сегмента; *бз* – однословный *блок завершения,* в котором при возврате в главную программу передается *код завершения; p1, . . . , p5* – числовые параметры, передаваемые вызываемому сегменту. При запуске сегмента адрес области, в которой размещаются передаваемые ему параметры, находится в регистре *Б.*

Если загрузка сегмента выполнилась с ошибками, управление возвращается главной программе в точку вслед за вызывающей последовательностью с диагностикой в блоке завершения. Кроме того, на пульт оператора последует сообщение, содержащее наименование сегмента и классификацию ошибки.

2. Для оформления сегмента оператор **NAM** записывается в следующем формате:

NAM *имяс*,**5**,*пр*

Здесь **5** – тип программы, указывающей на сегмент; *пр* – приоритет программы.

По этому оператору компоновщик формирует очередной элемент таблицы сегментов, включая его в цепочку элементов по приоритету.

3. Программы, включаемые в сегмент, могут содержать внешние обращения к главной программе задачи и к библиотечным подпрограммам. Естественно, что внешние ссылки между программами, включаемыми в различные сегменты, не допускаются.

4. В сегменте могут использоваться все макрооперации вызова супервизора, обеспечивающие те же функции, что и для ОЗУ-резидентных задач. При этом накладывается ограничение, суть которого состоит в том, что все события, запрошенные в макрооперациях, к моменту окончания работы сегмента обязаны быть завершенными. Это легко осуществить с помощью макрооперации **WAITE.**

5. Возврат в главную программу из сегмента осуществляется с помощью библиотечной подпрограммы *RSEGM.* Для этого достаточно включить в программу следующие операторы:

```
EXT  RSEGM
...
JSB  RSEGM
```

Если при выполнении цепочки сегментов (когда главная программа вызывает один сегмент, этот сегмент вызывает на свое место другой и т.д.) будет вызван сегмент, который обратится к подпрограмме *RSEGM,* управление в этом случае возвратится в главную программу задачи, а не в предыдущий сегмент. Это же происходит и в случае ошибок при загрузке какого-либо из сегментов цепочки, а также при неправильном указании имени следующего сегмента.

6. Сегмент может завершить выполнение всей сегментированной задачи, выполнив макрооперацию **EXIT.**

9.8. Диск-резидентные задачи

Диск-резидентные задачи оформляются как сегменты, главной программой для которых является системная обрабатывающая программа – *диспетчер диск-резидентных программ реального времени* (*ДДПРВ*). Диспетчер ДПРВ компонуется вместе с диск-резидентными программами в одну сегментированную программу, которая запускается по времени с интервалом 0,1 или 1 с (определяется пользователем в команде оператора:ПВ) (см. § 12.6). Такой интервал выбран потому, что при запуске какой-либо диск-резидентной программы осуществляется ее предварительная загрузка с диска. Время реакции таких программ значительно больше, чем ОЗУ-резидентных задач, и находится в диапазоне 0,1–1 с. В качестве диск-резидентных программ рекомендуется выбирать такие задачи, время выполнения которых значительно меньше их интервалов повторения. Наиболее благоприятным интервалом для работы *ДДПРВ* с диск-резидентными задачами является секунда. Среди диск-резидентных задач, выполняемых под управлением ДДПРВ, могут быть задачи, запускаемые по времени, с пульта оператора или из других диск-резидентных задач.

Диспетчер ДПРВ организует график выполнения диск-резидентных задач по времени и следит за соблюдением этого графика. Однако дисциплина последовательной загрузки сегментов и значительное время их реакции на внешние события могут вызвать нежелательные задержки при выполнении задач. В этом случае *ДДПРВ* следит за очередью выполнения диск-резидентных задач, вызывая их непрерывно до тех пор, пока не ликвидируются все временные задержки.

Так как диск-резидентные задачи оформляются в виде сегментов к *ДДПРВ*, то оператор **NAM** у них имеет следующий вид:

NAM *имяс,5,пр,масшт,инт,ф1,ф2,ф3,ф4*

Здесь *имяс* – имя диск-резидентной задачи, а все остальные параметры имеют то же назначение, что и для ОЗУ-резидентных задач.

Для диск-резидентных задач, так же как и для обычных сегментов, разрешено использование всех макроопераций вызова супервизора, за исключением макрооперации **EXIT**. Диск-резидентная задача обязана завершить свою работу либо вызовом другой диск-резидентной задачи с помощью библиотечной подпрограммы *NSEGM*, либо передачей управлений *ДДПРВ* с помощью вызова подпрограммы *RSEGM* (см. § 9.7).

Для установки новых временных характеристик диск-резидентной задаче используется еще одна специальная подпрограмма *TURNS*, функции которой аналогичны функциям макрооперации **TURN** ДЛЯ ОЗУ-резидентных задач (см. § 10.6).

При генерации операционной системы, учитывающей, что в одном из разделов оперативной памяти будет работать диспетчер диск-резидентных программ реального времени, необходимо выделить в этом разделе

пульт оператора для связи ДДПРВ с оператором системы. Этот пульт используется диспетчером для ввода специальных указаний оператора – команд оператора диспетчера. С помощью этих команд оператор может запускать диск-резидентные задачи, изменять им приоритеты и устанавливать новые параметры времени.

Эти команды имеют формат, аналогичный формату общего набора системных команд оператора (см. § 12.5) без указания первого символа : в команде оператора. Например, **ВЫ** вместо **:ВЫ** и т.д.

9.9. Организация псевдопараллельного выполнения диск-резидентных задач

Наряду с последовательной дисциплиной выполнения диск-резидентных программ под управлением *ДДПРВ* существует вторая дисциплина – псевдопараллельное выполнение диск-резидентных задач под управлением специальной обрабатывающей программы–диспетчера диск-резидентных задач (*ДДРЗ*). Эти задачи оформляются как обычные ОЗУ-резидентные задачи с типом в операторе **NAM**, равным **15**. Для этих задач в процессе компоновки создаются БУЗы, аналогичные БУЗам для ОЗУ-резидентных задач, поэтому дисциплина их запуска полностью идентична дисциплине запуска ОЗУ-резидентных задач : либо с пульта оператора (команды оператора **:СТАРТ**, **:ВЫЗОВ**, **:ПУСК**), либо по времени (макрооперация **TURN**, команда оператора **:ПВ**), либо из другой задачи (макрооперация **RUN**).

На время выполнения этих задач *ДДРЗ* загружает их в память раздела по адресам, закрепленным в процессе компоновки. Для реализации псевдопараллельной дисциплины работы в моменты прерываний осуществляется запоминание текущего состояния каждой прерванной диск-резидентной задачи в специальных файлах на дисках (*файлах свопинга*). Указателем диска для свопинга является параметр, который передается диспетчеру *ДДРЗ* при его вызове. Свопинг осуществляется всякий раз, когда появляется необходимость выполнения более высокой по приоритету диск-резидентной задачи. Исключением является ситуация, когда в диск-резидентных задачах запрашиваются макрооперации ввода-вывода, на время которых свопинг автоматически запрещается. Для остальных макроопераций вызова супервизора накладывается дополнительное ограничение, выраженное в том, что все БУСы этих макроопераций должны резервироваться в общей области загрузочного модуля.

При необходимости свопинг может быть запрещен непосредственно из задачи с помощью следующей вызывающей последовательности:

```
EXT  ZSWP
JSB  ZSWP
DEF  *+2
DEF  кз
```

Здесь *кз* – код завершения заказанной операции.

Разрешение свопинга осуществляется в такой последовательности:

```
EXT  RSWP
JSB  RSWP
DEF  * +2
DEF  кз
```

В одном загрузочном модуле наряду с диск-резидентными задачами может быть выполнено несколько несегментированных ОЗУ-резидентных задач.

Кроме того, любая из диск-резидентных задач, в свою очередь, может иметь свои собственные сегменты, откуда и появился термин "сегментированная диск-резидентная задача".

Порядок компоновки этого загрузочного модуля таков, что сначала на компоновку поступает ДДРЗ с соответствующими библиотеками, затем поступают ОЗУ-резидентные задачи, а последними – диск-резидентные задачи с необходимыми для их работы подпрограммами. При наличии сегментированных диск-резидентных задач для каждой из них сегменты поступают на компоновку сразу же после соответствующей главной части задачи.

Сегменты оформляются с типом в операторе **NAM**, равным пяти. Таким образом, для каждой сегментированной диск-резидентной задачи в диск-резидентной части загрузочного модуля в файле на диске формируется собственная область загрузки сегментов и собственная таблица сегментов.

Если диск-резидентная задача требует в своей работе свободную область памяти раздела, то это должно указываться специальным параметром при завершении компоновки. Тогда текущее состояние свободной области будет также запоминаться в файле свопинга.

Псевдопараллельная дисциплина выполнения диск-резидентных задач задается специальными макрокомандами в программе генерации ОС. Кроме того, при генерации ОС в этом случае требуется предварительная корректировка соответствующих библиотек системных модулей как для ДОС АСПО, так и для ДОС-II АСПО.

Упражнения к гл. 9.

1. Учитывая алгоритм выполнения задачи Е (см. § 9.2), запишите оператор **NAM** для этой задачи.

2. Перечислите основные этапы создания загрузочного модуля III типа.

3. Опишите последовательность команд оператора, необходимых для загрузки диспетчера диск-резидентных программ реального времени с диск-резидентными задачами во второй раздел оперативной памяти.

4. В какое состояние перейдут два старших разряда ячейки БУС после выполнения подпрограммы **WAIT**, если в исходном состоянии они имели значения WP = 01?

10.1. Таблица макроопераций. Общие сведения

Упорядоченные по коду вызова супервизора макрооперации, выполнение которых может обеспечить ОС АСПО, приведены в табл. 10.1.

По своему функциональному назначению макрооперации могут быть разделены на пять основных групп: управления вводом-выводом, синхронизации событий, управления выполнением задач, службы времени и реализации аппарата контрольных точек.

Таблица 10.1

Код вызова супервизора	Наименование макрооперации	Смысл макрооперации
0	**WAITE**	Ожидание завершения события
1	**EXIO**	Запустить операцию ввода-вывода
2	**EXIT**	Завершить выполнение задачи
3	**PAUSE**	Приостановить выполнение задачи (перевести в паузу)
4	**STAT**	Получить состояние устройства ввода-вывода
5	**TIME**	Получить текущее время суток
6	**STIME**	Установить отсчет заданного промежутка времени (генерировать временную метку)
7	**POSTE**	Отметить завершение события (отпостировать)
8	**RUN**	Запустить задачу на выполнение
9	**TURN**	Включить задачу на выполнение с заданными параметрами времени (установить временные характеристики)
10	**SCHPT**	Установить контрольную точку
11	**DCHPT**	Отменить контрольную точку
12	**WAITM**	Ждать завершения хотя бы одного события из группы событий
13	**RESET**	Сброс операций ввода-вывода и интервалов времени
14	**CHAP**	Изменить приоритет задачи

10.2. Управление вводом-выводом

В задачах пользователя все допустимые операции ввода-вывода заказываются с помощью макроопераций **EXIO, WAITE, STAT** и **RESET.**

Мкрооперация **EXIO** используется для запуска какой-либо из допустимых операций ввода-вывода на указанном устройстве ввода-вывода или файле.

После запуска операции на выполнение супервизор ввода-вывода возвращает управление задаче, затребовавшей операцию, которая, в свою очередь, может запросить другую операцию на том же или другом

устройстве ввода-вывода или выполнять вычисления, не связанные с результатами запущенной операции.

Если задача не может продолжать работу до завершения заказанной операции ввода-вывода, то она должна использовать макрооперацию **WAITE**. Операционная система по этой макрооперации переводит задачу в состояние ожидания, и задача продолжит работу только после завершения ожидаемой операции. Если к моменту выполнения макрооперации **WAITE** запущенная операция ввода-вывода уже завершилась, то операционная система возвращает управление задаче без перевода ее в состояние ожидания.

В процессе работы вычислительного комплекса устройства ввода-вывода могут выключаться из системы для проведения профилактических работ, замены носителей и т.п. Задача может определить состояние какого-либо из устройств с помощью макрооперации **STAT**. По этой макрооперации операционная система выдает состояние устройства, т.е. доступно оно для выполнения операций ввода-вывода или нет, а также код типа устройства, определяющий конкретные характеристики используемого устройства. В зависимости от полученных данных в задаче могут быть предусмотрены изменение хода вычислений, печать сообщения оператору и др.

Если в ходе выполнения задача обнаруживает, что продолжение заказанной ранее операции ввода-вывода нецелесообразно, то она может воспользоваться макрооперацией **RESET** для срочного прекращения выполнения операции (или же отмены заказа, если операция еще не была запущена на выполнение).

Макрооперация WAITE. Формат макрооперации:

WAITE *бус*

Здесь *бус* – метка блока управления событием (БУС), завершение которого ожидает задача (см. § 9.6).

Факт приостанова задачи отмечается в БУСе единицей в старшем разряде (W = 1). После завершения ожидаемого события задача будет отпостирована, т.е. ей будет передано управление в точку возврата (после макрооперации **WAITE**) для продолжения работы. При этом в БУС будет записан код завершения, и задача может проанализировать его в любой момент времени.

Макрооперация EXIO. Формат макрооперации:

EXIO *лн,ус,бз*[,*p1*[,...[,*pN*]]]

Здесь *лн* – логический номер УВВ; *ус* – управляющее слово, определяющее тип операции ввода-вывода и ее модификацию; *бз* – блок завершения; *p1*,. . ., *PN* – дополнительные параметры, определяющие, в частности, адрес буфера и длину передаваемой информации.

Логические и порядковые номера. Для обращения к УВВ в макрооперациях **EXIO** используется логический уровень, определяемый логичес-

кими номерами (ЛН). Логический номер – это произвольное десятичное число ($\leqslant 255$), которое указывает не на физические характеристики конкретного УВВ, а на определенную функцию ввода-вывода, ожидаемую в общем случае от любого УВВ, способного выполнить эту функцию. Это позволяет программировать задачи, не привязываясь к конкретным УВВ, а ориентируясь только на функции, осуществляемые этими устройствами. Физические же характеристики УВВ определяются их порядковыми номерами (ПН) – положительными десятичными числами, указывающими на порядок появления макрокоманд, описывающих конкретные УВВ, в программе генерации ОС.

Например, если в программе генерации ОС в следующем порядке перечислены макрокоманды:

```
#МВПК
#УБП
#УПП
#МДК
#ГИУ,
```

то порядковые номера будут соответствовать:

1 – МВПК (модуль внешней памяти кассетный);
2 – УБП (устройство быстрой печати);
3 – УПП (устройство последовательной печати);
4 – МДК (накопитель на магнитных дисках комбинированный);
5 – ГИУ (псевдоустройство группы инициативных устройств).

При работе с операционными системами АСПО пользователю доступны оба способа обращения к устройству: как по логическому номеру, так и по порядковому. Однако в обоих случаях должно быть установлено соответствие между порядковым и логическим номерами.

Это соответствие устанавливается для идентификации по порядковому номеру в специальной *таблице устройств* (ТУ), а для идентификации по логическому номеру – *в таблице логических номеров* раздела (ТЛН).

Таблица устройств формируется и заполняется во время генерации ОС и отражает взаимооднозначное соответствие между порядковыми номерами УВВ и адресами соответствующих *блоков управления устройствами* (БУУ).

Блок УУ формируется в процессе генерации ОС и является как бы паспортом УВВ, в котором хранится информация о типе УВВ, адресе и способе подключения, адресах соответствующих программ обработки (секции *драйвера*) и т.п.

Таблица логических номеров формируется во время генерации ОС, а заполняется либо с помощью директивы оператора **:ЛН** перед запуском задачи, в которой используется макрооперация **EXIO** с логическим уровнем доступа (например, **:ЛН,5,***1*), либо статически с помощью макрокоманды **#ЛН** непосредственно в самой программе генерации ОС (на-

пример, #**ЛН**,**5**,*1*). Здесь **5** – логический номер; *1* – порядковый номер УВВ.

При генерации может быть сформировано столько таблиц логических номеров, сколько разделов в системе, так как каждый раздел может иметь свою индивидуальную систему назначений. Во время генерации ОС в любом разделе может быть зарезервировано произвольное число логических номеров. Резервирование производится по максимальному ЛН в целях выделения памяти для ТЛН. В процессе работы в ТЛН могут скопиться занятые ЛН, которым присвоены ПН, и свободные. Учитывая это и пользуясь специальным форматом **EXIO**, можно произвести назначение непосредственно в теле программы. Если неизвестны логические номера, то назначение производится любому совободному ЛН. В этом случае ОС определит свободный ЛН, произведет назначение и возвратит этот номер задаче пользователя. После завершения работы с назначенным устройством в задаче необходимо освободить как устройство, так и логический номер для применения в других программах. Особенно часто это используется при работе с файлами на диске (см. гл. 12).

Для работы системных обрабатывающих программ в ОС АСПО приняты следующие *стандартные логические номера,* назначение которым производится статически во время генерации ОС: **1** – ввод и вывод символьной информации с пульта оператора системы, определенного для данного раздела; **2** – системный (используется для загрузки сегментов программ); **3** – системный (используется для работы с библиотеками); **4** – вывод результирующей информации; **5** – ввод исходной информации; **6** – вывод на печать; **7** – ввод аналоговой информации от УСО; **8** – ввод дискретной информации от УСО; **9** – вывод аналоговой и дискретной информации.

Если какие-то из перечисленных функций не требуются в загрузочном модуле, их номера, так же как и номера больше девяти, могут использоваться по усмотрению программиста.

Управляющее слово. Управляющее слово, указанное во втором параметре макрооперации **EXIO**, имеет структуру, приведенную на рис. 10.1.

Код операции КОП (разряды 14–15) определяет основную операцию, которую необходимо выполнить на устройстве, и принимает следующие значения:

01 – операция ввода информации от УВВ в ОЗУ;

10 – операция вывода информации из ОЗУ на УВВ;

11 – операция управления (установка на начало, пропуск, перемотка и т.п.).

Модификация основной операции (разряды 0–13) более детально указывает на выполняемую операцию ввода-вывода, например, для устройств ввода-вывода символьной и двоичной информации (устройства печати, МВПК, дисплеи, диски и т.п.). Структура поля модификации приведена на рис. 10.2.

Рис. 10.1. Структура управляющего слова EXIO

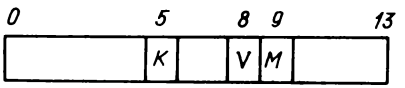


Рис. 10.2. Структура поля модификации

Разряд *9* (*M*) поля модификации при операциях ввода-вывода указывает на тип информации: $M = 0$ – символьная; $M = 1$ – двоичная.

Разряд *8* (*V*) при вводе двоичной информации задает тип записи: $V = 0$ – длина вводимой информации, определяется длиной буфера в ОЗУ для ввода; $V = 1$ – длина информации, содержится в первом байте вводимой информации.

Разряд *5* (*K*) определяет способ завершения операции на устройстве: $K = 0$ – анализируется (при вводе) и оформляется при выводе специальный признак конца информации (записи); $K = 1$ – анализ и выдача конца записи не производится. Остальные разряды должны быть равны нулю.

Единицей обмена информацией за одну макрооперацию **EXIO** для устройств ввода-вывода символьной и двоичной информации является *запись*. Например, для печатающих устройств записью является одна распечатываемая строка. Для распечатки нескольких строк потребуется многократное использование макрооперации **EXIO**. Для двоичной информации записью является последовательность данных в двоичном формате, обрамленная специальными признаками начала и конца записи.

Если задана операция ввода или вывода, параметры *p1* и *p2* в формате макрооперации **EXIO** должны соответственно определять буфер для данных и его длину. При этом если длина буфера указывается в словах, ее значение должно быть положительным числом, если в байтах – отрицательным.

Блок завершения. Блок завершения (БЗ) представляет собой две или три смежные ячейки памяти. Первая ячейка резервируется в задаче в качестве БУСа, предназначенного для идентификации состояния макрооперации **EXIO**. Во вторую ячейку БЗ записывается длина переданной информации для завершения заданной операции ввода-вывода. Третья ячейка используется только для операций, выполняемых с *групповыми* УВВ (см. § 12.2).

Макрооперация **EXIO** обрабатывается ОС следующим образом. Сначала определяется возможность запуска требуемой операции на заданном УВВ. Если операция невозможна из-за ошибки в формате макрооперации (не найдено заданное устройство, недопустимая операция и т.п.), система завершает операцию с *кодом завершения*, содержащим диагностику об ошибке. Этот код завершения помещается в БУС запрошенной макрооперации **EXIO** и может быть проанализирован в задаче. Если операция задана правильно, она запускается на выполнение (при условии незанятости устройства) или *ставится в очередь* к устройству.

В любом случае управление возвращается задаче. С момента запуска операции ввода-вывода и до ее завершения не должны изменяться ни вызывающая последовательность, ни все ее параметры, в том числе и содержимое буфера данных.

При программировании операции **EXIO** на языках высокого уровня подпрограммы-согласователи всегда обеспечивают ее выполнение с макрооперацией **WAITE**.

Формат макрооперации **EXIO** и смысл ее параметров определяются особенностями работы с конкретными УВВ и файлами, с которыми осуществляется обмен.

Примеры использования **EXIO**. Рассмотрим использование макрооперации **EXIO** для вывода на пульт оператора сообщения об аварии в задаче F (см. § 9.2):

```
EXIO   =L1,=L2,БЗ,БУФ,=L16
WAITE  БЗ
```

Здесь **=L***1* – первый параметр, указывает на логический номер пульта оператора; **=L***2* – управляющее слово, задающее операцию вывода символьной информации; *БЗ* – метка буфера из двух ячеек для блока завершения, где первая ячейка является БУСом этой макрооперации; *БУФ* – метка буфера, содержащая символьную информацию АВАРИЯ! ТРЕБУЕТСЯ ВМЕШАТЕЛЬСТВО; **=L***16* – длина выводимой информации.

Следующий пример иллюстрирует использование макрооперации **EXIO** для закрытия или открытия клапана *Кл1* в задаче А. Напомним, что для управления работой клапана используется модуль кодового управления бесконтактный (МКУБ) А641-9. Этот модуль в программе генерации ОС описывается специальной макрокомандой # **ВВДИ** (см. § 12.4). Перед запуском задачи порядковый номер этой макрокоманды должен быть назначен стандартному логическому номеру **9**. Макрокоманда **EXIO** в этом случае имеет вид

```
EXIO =L9,=L102B,БЗ,БУФ ,=L1,=L0
```

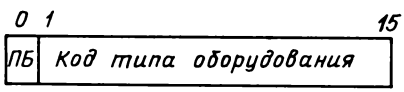
Здесь работой клапана управляет содержимое единственной ячейки буфера БУФ:

100000 – открыть;
000000 – закрыть.

Последний параметр **=L***0* указывает, что в группе модулей УСО, собранных в данной подсистеме, существует только один МКУБ, к которому и осуществляется обращение.

Интересно, что данная макрооперация **EXIO** в отличие от предыдущей является однотактной, так как к моменту возврата управления задаче она полностью завершает свое действие.

Рис. 10.3. Структура слова состояния УВВ



Еще один пример демонстрирует использование **EXIO** для опроса температуры (или давления) из задач C и D в более сложной подсистеме устройств связи с объектом.

Для опроса этих значений используются два коммутатора бесконтактных (КБ) А612-11, связанных по аналоговой шине сопряжения 2К с двумя модулями аналого-цифрового преобразования (МАЦП) А611-19. Адреса подключения этих модулей, типы коммутации, а также диапазон преобразования вводимой аналоговой информации определяются в макрокоманде **#МАЦП** в программе генерации ОС.

Формат **EXIO** в этом случае следующий:

EXIO =L*7*,=L*101*B,*БЗ*,*БУФ*,=L*1*,=L*0*

По этой макрооперации в ячейку БУФ записывается двоичный код с опрошенного датчика в диапазоне от –8192 до +8192, что соответствует напряжению на выходе МАЦП в диапазоне от 0 до +10 В при условии, что такой диапазон задан при генерации ОС. Уставки для температуры и давления, с которыми будут сравниваться вводимые величины, должны задаваться в задачах C и D в тех же двоичных кодах, что и опрашиваемые величины.

Более детальная информация об операциях ввода-вывода на отдельных УВВ и других операциях, выполняемых с помощью макроопераций **EXIO**, приведена в документации сопровождения ПО ВК СМ-2М и носит характер инструкций и руководств по пользованию.

Макрооперация STAT. Формат макрооперации:

STAT *лн,су*

Здесь *лн* – логический номер УВВ; *су* – ячейка памяти, в которую записывается информация о состоянии устройства.

Слово состояния УВВ всегда хранится в БУУ и имеет структуру, представленную на рис. 10.3.

Нулевой разряд определяет доступность устройства для выполнения операций ввода-вывода (признак *блокировки*): если ПБ = 0, устройство доступно; при ПБ = 1 устройство заблокировано. Устройство может быть заблокировано ОС при обнаружении его неработоспособности либо по указанию оператора с помощью команды оператора :**АВТОНОМ**. В любом случае оно становится доступным после выполнения команды оператора :**ЦЕНТР** (см. § 12.6).

Код типа оборудования указывает на тип и номер модели УВВ и содержит цифровую часть его шифра Annn-мм. Например, если устройство печати параллельное и имеет шифр А522-1, то его тип (разряды 1–9) равен 522, а номер модели (разряды 10–15) равен единице.

Заметим, что в макрооперации **STAT** отсутствует параметр *бус*. Это обусловлено тем, что данная макрооперация является однотактной, а для кода завершения (если неправильно задан логический номер) используется ячейка *су*.

Макрооперация RESET. По этой многотактной макрооперации осуществляется аварийное прекращение операций ввода-вывода или же отсчета временных интервалов. Формат макрооперации:

RESET *бус* [,*лн*]

Здесь *бус* – блок управления событием, отражающий результат выполнения макрооперации; *лн* – логический номер УВВ, на котором необходимо прекратить заданные с помощью **EXIO** операции ввода-вывода. Если этот параметр опущен, прекращается отсчет временных интервалов, заданных макрооперацией **STIME**.

По макрокоманде **RESET**, если опущен параметр *лн*, система уничтожает все запросы на отсчет интервалов времени, установленных данной задачей. Если указан логический номер УВВ и он отличен от нуля, все запросы на операции, находящиеся в очереди к указанному устройству и принадлежащие задаче, выполнившей макрооперацию **RESET**, уничтожаются. При задании нулевого логического номера выполняется уничтожение операций, запущенных задачей, на всех устройствах ввода-вывода.

Так как **RESET** – многотактная макрооперация, то рекомендуется использование после нее макрооперации **WAITE**.

10.3. Работа с инициативными устройствами

Операционная система обеспечивает выполнение операций ввода-вывода двух типов: операции с немедленным запуском и операции, запускаемые на выполнение при наличии сигнала готовности (*инициативы*) от устройства.

Операции с немедленным запуском используются для работы с такими устройствами, как печатающие устройства, устройства ввода-вывода на магнитную ленту, диски и т.п. Эти операции запускаются на выполнение сразу же, как только устройство ввода-вывода становится доступным для запуска операции.

Операции ввода, выполняемые по инициативе устройств, позволяют задачам оперативно реагировать на изменения внешней среды и используются для работы с инициативными устройствами связи с объектом, ввода оперативной информации по инициативе человека с пульта оператора, абонентских пультов и т.п.

Безусловно, задача, запросившая инициативу, должна быть ОЗУ-резидентной. С одним и тем же физическим УВВ может вестись работа как по инициативе задачи (операции с немедленным запуском), так и по инициативе устройства. В этом случае устройство рассматривается как два логических устройства, из которых одно является инициативным.

Способ запуска операций на каком-либо устройстве ввода-вывода (немедленный запуск или по инициативе устройства) определяется во время генерации ОС и не требует перепрограммирования задач, так как обращение к инициативным и неинициативным устройствам производится с помощью одинаковых макроопераций **EXIO**.

При небольшом числе инициативных устройств в системе обслуживание каждого из них может быть организовано следующим образом.

В исходном состоянии задача должна находиться в ожидании завершения ввода инициативы, т.е. должны быть выполнены макрооперации **EXIO** и **WAITE**. После появления инициативы задача будет отпостирована и сможет проанализировать введенную информацию. Задача выдает сообщение на пульт оператора, производит необходимую обработку, выдает корректирующие воздействия на объект и т.п. После этого задача снова должна запросить ввод инициативы и уйти в ожидание ее появления.

Каждое инициативное устройство определяется в программе генерации ОС макрокомандой **# ГИУ**.

При небольшом числе относительно медленных инициативных устройств в системе (многопультовые системы, системы коммутации сообщений) появляется необходимость обслуживать группу устройств в одной задаче. С этой целью в ОС включена возможность выполнения операций ввода с инициативных устройств по *групповому логическому номеру*. При генерации ОС определяется группа инициативных устройств и эту группу назначают одному логическому номеру. Такое групповое устройство называют *псевдоустройством* группы.

Если в макрооперации **EXIO** задача указывает этот логический номер, запрос ставится в очередь к псевдоустройству группы и операция ввода запускается на первом проявившем инициативу устройстве, входящем в группу. Если при завершении групповой операции ввода задаче требуется логический номер того устройства, который действительно выполнил ввод, она должна в макрооперации **EXIO** указать логический номер как отрицательное число и предоставить трехсловный блок завершения, в третье слово которого ОС поместит логический номер устройства, выполнившего операцию. Полученный логический номер может использоваться задачей при анализе введенной информации, выводе ответного сообщения и т.п.

Группа инициативных устройств, так же как и отдельные устройства, задаются в программе генерации ОС макрокомандой

 # ГИУ.

Примером использования инициативного устройства является задача F из § 9.2, которая запрашивает ввод инициативы в случае аварии от модуля ввода инициативных сигналов (МВВИС) А622-8. Этот модуль описывается в программе генерации ОС макрокомандами

```
M       #ВВДИ кв
        #ГИУ  M
```

В макрокоманде **#ВВДИ** указывается код выборки подключения модуля МВВИС, а в макрокоманде **#ГИУ** – метка *M* соответствующей макрокоманды **#ВВДИ**, описывающей единственное инициативное устройство в группе.

Перед запуском задачи логическому номеру *11*, используемому в макрооперации **EXIO** для ввода инициативы от МВВИС, должен быть назначен порядковый номер, соответствующий макрокоманде **#ГИУ**.

Все пояснения к примеру содержатся в комментариях к операторам задачи **F** :

```
                                ЗАДАЧА F
ASMB,R,B,L,T
      NAM F,3
* ЗАДАЧА F ЗАПУСКАЕТСЯ С ПУЛЬТА ОПЕРАТОРА И ПО ИНИЦИАТИВЕ ОТ МВВИС
* ВЫДАЕТ СООБЩЕНИЕ ОБ АВАРИИ НА ПУЛЬТ ОПЕРАТОРА
ВВОД  EXIO =L11,=L1010B,Б3,БУФ1,=L1,=L0
      WAITE Б3  ОЖИДАНИЕ ВВОДА ИНИЦИАТИВЫ
* ЕСЛИ ПРИШЕЛ СИГНАЛ, ТО ВЫВОД СООБЩЕНИЯ
      EXIO =L1,=L2,Б3,БУФ,=L16
      WAITE Б3
* ПОСЛЕ ВЫВОДА СООБЩЕНИЯ ПЕРЕХОД НА ЗАПРОС НОВОЙ ИНИЦИАТИВЫ
      JMP ВВОД
* РАБОЧИЕ БУФЕРЫ
Б3    BSS 2  - БЛОК ЗАВЕРШЕНИЯ
БУФ1  BSS 1    ОДНОСЛОВНЫЙ БУФЕР ДЛЯ ВВОДА ИНИЦИАТИВЫ
БУФ   ASS 16,АВАРИЯ! ТРЕБУЕТСЯ ВМЕШАТЕЛЬСТВО
      END
      ENDM
```

Здесь и далее примеры приведены в рабочем формате, пригодном для ввода в ЭВМ.

В приведенном примере нет макрооперации **EXIT**, из чего следует, что задача никогда не завершается, а постоянно настроена на ввод инициативы.

10.4. Обмен информацией между задачами

Обмен информацией между задачами можно осуществлять несколькими способами, один из которых заключается в использовании внешней памяти. Действительно, достаточно одной задаче организовать файл на диске, в который она будет помещать информацию, предназначенную для другой задачи, и другая задача может обращаться к этому файлу, последовательно читая переданную ей информацию.

Однако на осуществление такого способа обмена информацией затрачивается большое количество процессорного времени, связанного с постоянным обращением к дискам. Кроме того, здесь придется вводить определенные средства синхронизации для осуществления обмена информацией в обе стороны.

В операционных системах АСПО предусмотрены специальные средства для осуществления обмена информацией между задачами без использования внешней памяти. Обмен осуществляется на уровне обычных

операций ввода-вывода, выполняемых специальным *псевдоустройством межзадачного обмена информацией* (МЗОИ). Здесь под псевдоустройством понимается некоторая специальная программа ОС (*псевдодрайвер* МЗОИ), выполняющая роль накопителя (по аналогии с накопителем внешней памяти).

Это псевдоустройство включается в программу генерации ОС как обычное УВВ (по макрокоманде **#МЗОИ**), и ему присваивается порядковый номер. Назначив этот порядковый номер выбранному логическому номеру, задача обращается к псевдоустройству МЗОИ с помощью обычной макрооперации **EXIO**. Следовательно, для обмена информацией одна задача с помощью **EXIO** выполняет вывод информации из своего буфера на псевдоустройство МЗОИ, а другая задача с помощью другого формата **EXIO** осуществляет ввод с этого псевдоустройства.

Такой алгоритм обмена позволяет осуществить обмен информацией даже между задачами, расположенными в разных разделах памяти.

При получении обоих запросов на обмен информацией (запроса на передачу и запроса на прием) псевдодрайвер межзадачного обмена просто пересылает информацию из буфера одной задачи в буфер другой задачи.

Сгенерировав в программе генерации ОС несколько псевдоустройств МЗОИ, пользователь может организовать обмен информацией в обе стороны не только для двух, но и для нескольких задач.

10.5. Макрооперации синхронизации событий

К макрооперациям этой группы относятся **WAITE, POSTE** и **WAITM.** Макрооперация **WAITE** была описана в § 10.2.

Макрооперация POSTE. Эта макрооперация используется для извещения о том, что событие, идентифицируемое блоком управления событием, произошло. Для синхронизации работы двух связанных между собой задач, как правило, макрооперация **POSTE** используется совместно с **WAITE.** Формат макрооперации:

POSTE *бус*[,*кз*]

Здесь *бус* – блок управления событием, отметку о завершении которого выполняет задача; *кз* – код завершения, определяющий, как завершилось событие.

Макрооперации **POSTE** и **WAITE** могут использоваться, например, для управления доступом к последовательно применяемому ресурсу (общее для двух задач устройство ввода-вывода, общий участок оперативной памяти, с которыми задачи могут работать только поочередно). Для этого в задачах определяется общая ячейка памяти (см. оператор **COM** в § 7.1) – блок управления событием.

В исходном состоянии, когда ресурсом не пользуется ни одна задача, разряды WP установлены в БУСе в 01. Алгоритм использования макро-

операций следующий: когда любая задача занимает ресурс, она применяет **WAITE** по этому БУСу; освобождение ресурса должно сопровождаться макрооперацией **POSTE** по этому же БУСу.

Допустим, что первая задача захватила ресурс и установила макрооперацию **WAITE**. Тогда согласно алгоритму работы подпрограммы **WAIT** (см. § 9.6) задача не подвешивается, а разряды WP устанавливаются в 00. Теперь, если второй задаче потребуется ресурс, то она также использует **WAITE**, но с переводом себя в состояние ожидания и установкой WP, равным 10. Вывести вторую задачу из состояния ожидания, т.е. разрешить ей применять общий ресурс, сможет только макрооперация **POSTE**, выполненная первой задачей после освобождения ресурса. После того как вторая задача использует **POSTE**, блок управления событием возвратится в исходное состояние (WP = 01).

Следовательно, в каждой задаче участки, работающие с общим ресурсом, должны быть обрамлены макрооперациями **WAITE** и **POSTE**:

```
...
WAITE бус
* использование общего ресурса
...
POSTE бус,кз
...
```

Код завершения может быть проанализирован после выполнения **WAITE** и, следовательно, может понадобиться для передачи между задачами дополнительной информации (в частности, о состоянии ресурса).

Макрооперация WAITM. По этой макрооперации задача переходит в состояние ожидания хотя бы одного события из указанной в макрооперации группы событий. Формат макрооперации:

WAITM *рбуф*, *бус1*, . . . ,*бусN*

Здесь *рбуф* – двухсловный рабочий буфер, используемый ОС для выполнения макрооперации. После возобновления работы задачи первое слово этого буфера содержит порядковый номер того события, по завершении которого задача была выведена из состояния ожидания; *бус1*, . . . , *бусN* – блоки управления событиями, завершения которых (хотя бы одного) ждет задача. Порядковый номер события определяется порядком указания соответствующего БУСа в этом списке.

Операционная система выполняет отработку этой макрооперации следующим образом. Во время вызова происходит просмотр БУС в указанном в вызывающей последовательности порядке. Если в каком-либо из них обнаруживается признак завершения события (разряды WP = = 01), то в первую ячейку буфера *рбуф* заносится его номер, WP устанавливаются в 00 и управление возвращается задаче. Если же к моменту вызова ни одно событие еще не завершилось, задача переводится в

состояние ожидания и в каждом БУСе устанавливается признак ожидания (WP = 10). По первому же завершившемуся событию задача будет отпостирована (выведена из состояния ожидания). При этом в блоке управления завершившегося события будет содержаться код завершения с установленными WP = 00, а номер события будет занесен в первое слово рабочего буфера.

Интересно, что после завершения одного события макрооперация **WAITM** может быть использована повторно в цикле для ожидания поочередного завершения оставшихся незавершенных событий. При этом все ранее завершенные события не будут участвовать в постировании задачи.

10.6. Управление выполнением задач

К макрооперациям этой группы относятся макрооперации **RUN, TURN, CHAP, EXIT** и **PAUSE.**

Макрооперация RUN. Эту многотактную макрооперацию применяют для запуска ОЗУ-резидентной задачи (*подчиненной*) из любой другой задачи (*главной*). Обе задачи должны принадлежать одному и тому же загрузочному модулю. Формат макрооперации:

RUN *имяз*,*бус*[,p1[,. . .[,p5]]]

Здесь *имяз* – имя запускаемой задачи (до пяти символов); *бус* – блок управления событием, по которому запускающая задача может ждать завершения запущенной задачи; *p1, . . . ,p5* – параметры, передаваемые запускаемой задаче при вызове.

Запуск задачи заключается в следующем: по имени, указанному в макрооперации, отыскивается БУЗ запускаемой задачи, и если задача находится в исходном состоянии (ожидания запуска с начальной точки), то она постируется по стандартному БУСу запуска.

В результате выполнения макрооперации **RUN** обе задачи (и главная, и подчиненная) будут находиться в состоянии, готовом к выполнению, но выполняться будет только та, которая старше по приоритету. Если главной задаче необходимо дождаться завершения запущенной задачи, то она выполняет макрооперацию **WAITE** с указанием БУСа, которым помечена макрооперация **RUN**. При этом главная задача подвешивается и будет отпостирована по окончании работы запущенной задачи с получением в БУСе кода ее завершения.

При запуске подчиненной задачи адрес области, в которой размещаются переданные ей параметры, находится в регистре *B*.

В задачах C и D из примера § 9.2 используются две макрооперации **RUN** для запуска задач A и B с передачей им параметра, указывающего на открытие или закрытие клапанов:

```
RUN   A,БУС,=L0
WAITE БУС
RUN   B,БУС,=L1
WAITE БУС
```

Здесь во второй макрооперации **RUN** повторно использована ячейка *БУС,* полностью отработанного в первой макрооперации.

Макрооперация TURN. По этой макрооперации устанавливаются новые значения фазы и интервала ОЗУ-резидентной задаче. Эта макрооперация – однотактная, т.е. к моменту возврата управления задаче она полностью выполнена. Однако в ней используется БУС для того, чтобы в случае неудачного выполнения в него был занесен код завершения. Формат макрооперации:

TURN *имяз,бус,масшт,инт,ф1*[*,ф2,ф3,ф4*]

Здесь *имяз* – имя задачи, которой изменяются или устанавливаются новые временные характеристики. Легко заметить, что, указав собственное имя, задача может установить временные характеристики самой себе; *масшт* – масштаб, указывающий на единицы, в которых измеряется интервал и относительная фаза (см. параметр *мш* в макрокоманде **STIME**, описанной в § 10.7); *инт* – интервал не более 24 ч, задаваемый в единицах масштаба; *ф1–ф4* – параметры, определяющие фазу. Если фаза относительная, то она задается отрицательным значением *ф1* в единицах масштаба, а все остальные параметры *ф2–ф4* должны быть опущены. Абсолютная фаза задается положительными значениями *ф1–ф4,* указывающими на часы, минуты, секунды и десятки миллисекунд, любое из которых может быть опущено.

Здесь, так же как и в операторе **NAM**, исключение задачи из списка по времени задается нулевым значением масштаба, а однократное выполнение задачи – нулевым значением интервала.

Примером использования макрооперации **TURN** является установка начальных временных характеристик задачам C и D из задачи A (см. § 9.2):

```
TURN  C,БУС1,=L2,=L20,=L-10
TURN  D,БУС1,=L2,=L26,=L-13
```

Как видно из примера, для однотактных макроопераций с правильно заданными параметрами нет смысла в использовании макрооперации **WAITE.**

Макрооперация CHAP. По этой макрооперации задаче, имя которой указано в запросе, устанавливается новый приоритет. Как задача, которая выполняет макрооперацию, так и задача, у которой изменяется приоритет, должны принадлежать одному и тому же загрузочному модулю. В частности, задача может изменить себе приоритет, указав свое имя. Формат макрооперации:

CHAP *имяз,бус,пр*

Здесь *имяз* – имя задачи, которой присваивается новый приоритет; *бус* – блок управления событием, в котором отражается результат вы-

полнения макрооперации; *пр* – число (от 0 до 255), определяющее приоритет.

Так как **CHAP** – однотактная макрооперация, то возможно ее применение без макрооперации **WAITE**.

Макрооперация EXIT. Эта макрооперация используется для уведомления ОС о завершении выполнения задачи. По этой макрооперации задача переводится в исходное состояние, т.е. в состояние ожидания запуска ее с начального адреса с пульта оператора, по времени или из другой задачи. Формат макрооперации:

EXIT [*кз*]

Здесь *кз* – код завершения задачи. Этот код завершения применяют, если данная задача была запущена из другой задачи по макрооперации **RUN**. Этим кодом завершения задача постирует ожидавшую ее завершения задачу (из которой она была запущена), указывая ей причину своего завершения.

Приведенный выше формат **EXIT** называют *основным форматом.* Помимо этого формата существует также его расширенный вариант.

Расширенный формат макрооперации **EXIT** обеспечивает последовательную загрузку и выполнение без вмешательства оператора нескольких однозадачных загрузочных модулей, хранящихся на диске. В этом случае загрузочный модуль, завершающий работу, использует расширенный вариант **EXIT**, в котором указывается файл со следующим загрузочным модулем, которому должно быть передано управление. Следующий загрузочный модуль загружается в тот же раздел на место вызвавшего его загрузочного модуля и т.д. Если какой-либо из очередных модулей выполнит макрооперацию **EXIT** основного формата, то на его место повторно будет вызван первый из всей цепочки загрузочный модуль и ему будет передан код завершения закончившего выполнение модуля. Исходя из этого, первый загрузочный модуль, применивший расширенный формат **EXIT**, обязан располагаться на одном из системных дисков. В отличие от других загрузочных модулей в цепочке на него не налагается требование однозадачности, однако задачи, из которых он компонуется, не должны выполняться по времени. Расширенный формат **EXIT** используется в системной обрабатывающей программе, называемой *диспетчером пакетной обработки* (ДПОМ). Этот диспетчер, применяя описанный выше принцип, вызывает по очереди загрузочные модули с системными обрабатывающими программами, осуществляя тем самым подготовку программ пользователя в пакетном режиме. Расширенный формат макрооперации **EXIT**:

EXIT *бус,буф,дл,р1, . . . , р5*

Здесь *бус* – блок управления событием, в который при повторной загрузке модуля, выполнившего макрооперацию расширенного формата, заносится код завершения работавшего вместо него загрузочного моду-

ля (если он выполнил **EXIT** основного формата); *буф* – четырех-, семи- или восьмисловный буфер (в зависимости от параметра *дл*), в котором должна содержаться информация о дисковом файле со следующим загружаемым загрузочным модулем; *дл* – длина буфера, которая определяет, какие атрибуты включены в буфер: имя файла, имя диска и порядковый номер дисковода; *p1, . . . , p5* – параметры, передаваемые задаче в вызываемом загрузочном модуле.

Работа с расширенным форматом **EXIT** обеспечивается включением в программу генерации ОС макрокоманды **#ДПО.**

Макрооперация PAUSE. По этой макрооперации осуществляется приостанов выполнения задачи до возобновления ее выполнения командой оператора **:ПУСК.** Использование этой макрооперации предполагает, что дальнейшее функционирование задачи невозможно без вмешательства человека-оператора. Причина приостанова может быть предварительно выведена из задачи на пульт оператора с помощью макрооперации **EXIO.** Оператор продолжает выполнение приостановленной задачи, передавая ей в команде оператора **:ПУСК** до пяти числовых параметров. Задача запускается на выполнение с точки приостанова с регистром *Б*, содержащим адрес массива передаваемых параметров. Эти входные параметры настраивают задачу на необходимую в дальнейшем дисциплину работы. Формат макрооперации:

```
PAUSE
```

Для этой макрооперации в задаче нет необходимости резервировать ячейку БУС, так как для синхронизации применяется стандартная ячейка, хранящаяся в БУЗе. Резервирование ячейки БУС непосредственно в БУЗе вызвано тем, что этим же БУСом кроме макрооперации **PAUSE** для аналогичного подвешивания задачи используется также команда оператора **:ПАУЗА.** Очевидно, что команда оператора **:ПУСК** использует этот БУС для постирования задачи.

10.7. Макрооперации службы времени

Макрооперация TIME. По этой макрооперации задача получает значения текущего времени суток и дату. Формат макрооперации:

```
TIME время[,год]
```

Здесь *время* – метка буфера из пяти слов, в который заносится значение времени суток и день в году в следующей последовательности: первое слово – миллисекунды (*0–999*), второе слово – секунды (*0–59*), третье слово – минуты (*0–59*), четвертое слово – часы (*0–23*), пятое слово – день года (*1–366*); *год* – необязательный параметр, определяющий ячейку, в которую заносится текущий год.

Макрооперация STIME. По этой многотактной макрооперации устанавливается отсчет заданного времени, по истечении которого задача

должна быть отпостирована. Эту временную выдержку называют тайм-аутом от задачи. Запросив тайм-аут, задача может перейти в ожидание его завершения, используя макрооперацию **WAITE**. Формат макрооперации:

STIME *бус,мш,мн*

Здесь *бус* – блок управления событием, связанным с истечением промежутка времени; *мш* – масштаб, в котором измеряется время; **1** – десятки миллисекунд; **2** – секунды; **3** – минуты; **4** – часы; *мн* – промежуток времени (множитель) в единицах, определенных значением масштаба. Множитель может быть не более 24 ч.

Для задачи В из примера в § 9.2 макрооперация **STIME** осуществляющая выдержку, равную 2 с, имеет следующий вид:

STIME *БУС*,=L2,=L2
WAITE *БУС*

10.8. Аппарат контрольных точек

Макрооперация SCHPT и DCHPT. Эти макрооперации являются составной частью подсистемы анализа сбоев и реконфигурации ОС АСПО и предназначены для реализации простейшего варианта аппарата контрольных точек, суть которого состоит в следующем.

В ходе выполнения задачи могут возникать различного рода ошибочные ситуации, которые могут быть вызваны как сбоями в работе аппаратных средств, так и ошибками в программах. Эти ситуации обнаруживаются схемами контроля процессора и вызывают соответствующее прерывание (см. § 4.3).

Любая задача может установить контролируемый линейный участок, который начинается с макрооперации **SCHPT** и заканчивается **DCHPT**. Этот участок называют *линейным*, так как на нем, кроме указанных макроопераций, не должны быть использованы другие макрооперации.

В случае возникновения ошибочной ситуации на контролируемом участке задаче предоставляется возможность повторить выполнение этого участка программы. Если ошибка возникла в результате случайного сбоя и не повлекла за собой нарушений программы, повторение участка программы может восстановить нормальное выполнение задачи.

При установке контрольной точки с помощью макрооперации **SCHPT** запоминается текущее состояние задачи в буфере, резервируемом в задаче. Информация, запомненная в этом буфере, используется операционной системой при перезапуске задачи в случае сбоя.

Существует следующая стратегия обработки прерываний от схем контроля процессора.

1. По любому прерыванию от схем контроля процессора управление передается специальной подпрограмме ОС обработки этих прерываний, и если прерывание возникло на неконтролируемом участке задачи, то задача аварийно приостанавливается, а на пульт оператора выдается стандартное сообщение, фиксирующее состояние задачи в момент прерывания.

2. Если прерывание возникло на контролируемом участке, а в запросе **SCHPT** был указан режим работы с *автоперезапуском,* то осуществляется немедленный автоперезапуск с контрольной точки. Если автоперезапуск не был указан в макрооперации **SCHPT**, то задача аварийно приостанавливается с выдачей стандартного сообщения о состоянии задачи в момент прерывания (точке приостанова). Повторить выполнение программы в этом случае с контрольной точки можно с помощью команды оператора

:ПУСК

При любом безуспешном перезапуске задача также аварийно подвешивается с выдачей стандартного сообщения оператору.

Формат макрооперации **SCHPT**:

SCHPT *ус,буф*[,*твх*]

Здесь *ус* – управляющее слово, определяющее режим работы с автоперезапуском (*ус* = 1) или без него (*ус* = 0); *буф* – рабочий буфер из 14 слов, необходимый для функционирования аппарата контрольных точек; *твх* – точка входа (метка) в подпрограмму пользователя. Если этот параметр указан, то при возникновении прерывания управление передается не на повторение контролируемого участка (в случае автоперезапуска немедленно, иначе – после команды оператора **:ПУСК**), а этой подпрограмме.

Формат макрооперации **DCHPT**:

DCHPT

По этой макрооперации отменяется установленная ранее контрольная точка. Установка новой контрольной точки отменяет предыдущую контрольную точку без макрооперации **DCHPT**.

10.9. Примеры использования макроопераций вызова супервизора

В этом параграфе приведены два примера использовния макроопераций вызова супервизора в задачах А и В описанного в § 9.2 примера технологического процесса.

Все пояснения к примерам содержатся в комментариях к операторам задач:

ЗАДАЧА А

```
ASMB,R,B,L,T
      NAM A,3,,2,0,8,5
* ЗАДАЧА А ЗАПУСКАЕТСЯ В 8 ЧАС 5 МИНУТ И ИСКЛЮЧАЕТСЯ ИЗ СПИСКА ПО ВРЕМЕНИ
ОБЛ1  COM РЕЖИМ
* РЕЗЕРВИРУЕТСЯ ОДНА ЯЧЕЙКА В ОБЩЕЙ ОБЛАСТИ, В КОТОРУЮ БУДЕТ
* ПОМЕЩАТЬСЯ НОМЕР ТЕХНОЛОГИЧЕСКОГО РЕЖИМА
      LDA 1,I  АНАЛИЗ 1-ГО ВХОДНОГО ПАРАМЕТРА,
      SZA
      JMP A1   ЕСЛИ НЕ НУЛЬ, НА ЗАКРЫТИЕ КЛ1.
      EXIO =L9,=L1028,B3,=L10000008,=L1,=L0  ОТКРЫТИЕ КЛАПАНА КЛ1
      WAITE B3
* ПЕРЕД ЗАПУСКОМ ЗАДАЧИ А ЛОГИЧЕСКОМУ НОМЕРУ 9 ДОЛЖЕН БЫТЬ НАЗНАЧЕН
* ПОРЯДКОВЫЙ НОМЕР МАКРОКОМАНДЫ #ВВДИ, ОПИСЫВАЮЩЕЙ В ПРОГРАММЕ
* ГЕНЕРАЦИИ ОС МКУБ (А641-9), ПРЕДНАЗНАЧЕННЫЙ ДЛЯ УПРАВЛЕНИЯ РАБОТОЙ
* КЛАПАНА КЛ1.
      LDB =B1    УСТАНОВКА ПЕРВОГО ТЕХНОЛОГИЧЕСКОГО
      STB РЕЖИМ  РЕЖИМА
* УСТАНОВКА ВРЕМЕННЫХ ХАРАКТЕРИСТИК ЗАДАЧАМ С И D,
* СООТВЕТСТВУЮЩИХ ПЕРВОМУ ТЕХНОЛОГИЧЕСКОМУ РЕЖИМУ
      TURN C,БУС1,=L2,=L20,=L-10
      TURN D,БУС1,=L2,=L26,=L-13
A2    EXIT       ЗАВЕРШЕНИЕ ЗАДАЧИ
A1    EXIO =L9,=L1028,B3,=L0,=L1,=L0    ЗАКРЫТИЕ КЛАПАНА КЛ1
      WAITE B3
* ЗАПУСК ЗАДАЧИ В С ПАРАМЕТРОМ, УКАЗЫВАЮЩИМ НА ОТКРЫТИЕ КЛАПАНА КЛ2
      RUN B,БУС,=L0
      WAITE БУС
      JMP A2     -НА ЗАВЕРШЕНИЕ ЗАДАЧИ
B3    BSS 2       ДЛЯ EXIO РАБОЧИЕ ЯЧЕЙКИ
БУС   BSS 1       ДЛЯ RUN
БУС1  BSS 1       ДЛЯ TURN
      END
      ENDM
```

ЗАДАЧА В

```
ASMB,R,B,L,T
      NAM B,3
* ЗАДАЧА В ЗАПУСКАЕТСЯ С ПУЛЬТА ОПЕРАТОРА ИЛИ ИЗ ДРУГОЙ ЗАДАЧИ
ОБЛ1  COM РЕЖИМ
* РЕЗЕРВИРУЕТСЯ ЯЧЕЙКА В ОБЩЕЙ ОБЛАСТИ ДЛЯ ЗАНЕСЕНИЯ НОМЕРА
* ТЕХНОЛОГИЧЕСКОГО РЕЖИМА
      LDA 1,I    АНАЛИЗ 1-ГО ВХОДНОГО ПАРАМЕТРА,
      SZA
      JMP B1     ЕСЛИ НЕ НУЛЬ, НА ЗАКРЫТИЕ КЛ2
      STIME БУС,=L2,=L2    ВЫПОЛНЕНИЕ ТАЙМ-АУТА, РАВНОГО 2 СЕК
      WAITE БУС
      EXIO =L10,=L1028,B3,=L10000008,=L1,=L0  ОТКРЫТИЕ КЛАПАНА КЛ2
      WAITE B3
* ПЕРЕД ЗАПУСКОМ ЗАДАЧИ В ЛОГИЧЕСКОМУ НОМЕРУ 10 ДОЛЖЕН БЫТЬ НАЗНАЧЕН
* ПОРЯДКОВЫЙ НОМЕР СООТВЕТСТВУЮЩЕЙ МАКРОКОМАНДЫ #ВВДИ, ОПИСЫВАЮЩЕЙ
* В ПРОГРАММЕ ГЕНЕРАЦИИ ОС МКУБ (А641-9), ПРЕДНАЗНАЧЕННЫЙ ДЛЯ
* УПРАВЛЕНИЯ РАБОТОЙ КЛАПАНА КЛ2.
      LDA =B3    УСТАНОВКА ТРЕТЬЕГО
      STA РЕЖИМ  ТЕХНОЛОГИЧЕСКОГО РЕЖИМА
* УСТАНОВКА ВРЕМЕННЫХ ХАРАКТЕРИСТИК ЗАДАЧАМ С И D,
* СООТВЕТСТВУЮЩИХ ТРЕТЬЕМУ ТЕХНОЛОГИЧЕСКОМУ РЕЖИМУ
      TURN C,БУС1,=L2,=L25,=L-25
      TURN D,БУС1,=L2,=L30,=L-30
B2    EXIT       ЗАВЕРШЕНИЕ ЗАДАЧИ
* ЕСЛИ ВХОДНОЙ ПАРАМЕТР НЕ РАВЕН НУЛЮ, ТО ЗАКРЫТИЕ КЛАПАНА КЛ2
B1    EXIO =L10,=L1028,B3,=L0,=L1,=L0
      WAITE B3
      JMP B2     -НА ЗАВЕРШЕНИЕ ЗАДАЧИ
B3    BSS 2      ДЛЯ EXIO  РАБОЧИЕ ЯЧЕЙКИ
БУС   BSS 1      ДЛЯ STIME
БУС1  BSS 1      ДЛЯ TURN
      END
      ENDM
```

Упражнения к гл. 10

1. Попробуйте сократить область рабочих ячеек в задачах A и B, применив в каждой из них один БУС для идентификации событий во всех выполняемых макрооперациях. Обоснуйте правомерность этих действий в приведенных примерах.

2. Используя макрооперацию **WAITM**, запрограммируйте ОЗУ-резидентную задачу M, которая выполняет следующее:

а) запускает ОЗУ-резидентную задачу N с передачей ей двух числовых параметров: *–16,234*;

б) устанавливает временную выдержку 820 мс;

в) если задача N не уложилась в отведенную ей временную выдержку, то на устройство печати выдается сообщение :ВРЕМЯ ВЫПОЛНЕНИЯ ИСТЕКЛО и завершается работа.

3. Опишите алгоритм макроопераций **WAITE** и **POSTE** при работе с общим ресурсом.

4. Запрограммируйте ОЗУ-резидентную задачу L, которая выполняет следующее:

а) запускает по времени ОЗУ-резидентную задачу N с относительной фазой, равной 22 мин, 14 с и интервалом – 6 мин;

б) переходит в паузу, ожидая запуска с пульта оператора;

в) если входной параметр после запуска равен нулю, то завершается работа;

г) если входной параметр после запуска отличен от нуля, то на пульт оператора выдается сообщение КОНЕЦ РАБОТЫ и завершается выполнение задачи.

Глава 11. СИСТЕМА УПРАВЛЕНИЯ ФАЙЛАМИ

11.1. Общие сведения

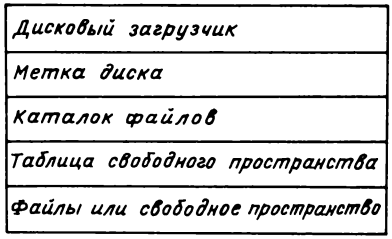
В операционных системах АСПО внешняя память (дисковая память с произвольным доступом) находится в ведении *системы управления файлами* (СУФ), обеспечивающей автоматическое распределение дисковой памяти, запоминание, хранение, выборку и модификацию поименованных наборов данных – файлов.

В целях защиты файлов одного пользователя от несанкционированного доступа со стороны программ другого пользователя в систему управления файлами введен специальный механизм *кодов защиты*. Защищенный файл становится доступным для обработки только при указании правильного кода защиты. Это делает возможным хранение и обработку файлов различных пользователей на одном дисковом носителе.

Для хранения файлов пользователя, а также системных файлов в операционных системах одновременно могут применяться несколько устройств внешней памяти с произвольным доступом. Это могут быть устройства со съемными пакетами дисков, с фиксированными дисками или гибкие диски различной емкости.

Структура дискового носителя. Перед использованием какого-либо дискового носителя информации он должен быть подготовлен специаль-

Рис. 11.1. Структура дискового носителя



ной системной обрабатывающей подпрограммой – *программной разметки.* Эта программа разбивает поверхность дискового пространства на секторы по 128 слов каждый. Сектор является минимальным адресуемым блоком дискового носителя. Секторы нумеруются в пределах всего носителя, и нумерация начинается с нуля. Программа разметки производит тестирование поверхности носителя и на основании этого формирует таблицу дефектных участков данного носителя, т.е. участков, где нарушено магнитное покрытие и операции записи и чтения информации выполняются неверно. Программа разметки формирует метку диска, область каталога файлов и таблицу свободного пространства данного носителя, а также записывает дисковый загрузчик.

Метка диска содержит имя диска (1–6 символов), код защиты (одно слово), адреса расположения на диске, длину в секторах каталога файлов и таблицы свободного пространства, а также таблицу дефектных участков носителя. Метка диска занимает 128 слов и размещается, как правило, во втором секторе. Если сектор оказался дефектным, то метка диска располагается в другом, недефектном секторе.

Каталог файлов располагается за меткой диска. Каждый элемент каталога – шестнадцатисловная метка файла. Метка файла содержит имя файла, его код защиты, номер последнего расширения файла (экстента), адрес и размер дискового пространства, выделенного файлу при его создании, дату создания файла, номер и адрес последней записи файла.

Кроме меток файлов в каталоге располагаются шестнадцатисловные метки экстентов. Экстент – несмежный участок диска, по размеру равный основному файлу, который создается в дополнение к основному, когда в нем не хватает места для данных.

Метка расширения (экстента) файла появляется в каталоге при выделении дополнительного пространства этому файлу и помимо имени файла содержит порядковый номер экстента, адрес и длину выделенного ему пространства. При уничтожении основного файла элементы каталога, относящиеся к нему, используются повторно для описания вновь создаваемых файлов, автоматически освобождаются экстенты файлов, а освобождаемое пространство возвращается таблице свободных участков.

Таблица свободных участков располагается вслед за каталогом файлов и служит как справочник при распределении дискового пространства. Она корректируется каждый раз при создании нового или уничтожении существующего файла на дисковом носителе. Структура дискового носителя приведена на рис. 11.1.

Состав СУФ. Система управления файлами включает в себя три различные по функциональному назначению подсистемы:

подсистема управления внешней памятью;

подсистема управления данными;

обслуживающие программы.

Первая подсистема состоит из программ управления внешней памятью, которые являются ядром всей СУФ и собраны в виде модулей ОС в ППМ для генерации ОС. Они включаются в программу генерации ОС макрокомандой **#СУФ**. Эти модули ОС обеспечивают распределение пространства, создание, поиск, уничтожение файлов на дисках по абсолютному или относительному номеру сектора. К ним возможен доступ из задач с помощью специальных форматов **EXIO** для чтения и записи информации по относительному номеру сектора в файле, т.е. этот уровень СУФ предполагает хорошую осведомленность пользователя в секторной структуре обрабатываемого файла.

Вторая подсистема представляет собой совокупность программ управления данными, которые обеспечивают логические методы организации и доступа к данным в файлах или других устройствах ввода-вывода и дают возможность пользователю писать программы, зависящие только от методов обработки данных, а не от физических характеристик устройств, где эти данные содержатся. Программы этой подсистемы собраны в ППМ для генерации задач пользователя в библиотеке подпрограмм #ОБП.

Эти программы дают возможность доступа к файлам на уровне записей, т.е. позволяют обрабатывать файлы, содержащие записи постоянной и переменной длины, осуществляют последовательный и прямой методы доступа к ним, не привязываясь к номерам секторов, в которых располагаются эти записи.

Последовательный метод доступа обеспечивается программами *OPEN, GET, PUT, FCONT, STORE, RESTR, AREST, EOV, CLOSE*, которые осуществляют прием (*GET*) или выдачу (*PUT*) очередной записи файла, пропуск (*FCONT*) заданного количества записей, автоматическое выделение файлу дополнительного дискового пространства и т.д. Этот метод используется в основном при работе с УВВ последовательного доступа (магнитная лента, мини-кассета и т.п.) и с дисковыми файлами постоянной и переменной длины.

Подпрограммы *прямого метода доступа OPEN, READ, WRITE, CLOSE* в отличие от подпрограмм последовательного метода доступа обеспечивают чтение (*READ*) или запись (*WRITE*) данных по произвольному номеру записи относительно начала файла и применяются только для дисковых файлов, состоящих из записей постоянной длины.

Подпрограммы *третьей подсистемы* собраны в специальной системной обрабатывающей программе под названием *программы обслуживания файлов* (*ПОФ*). Доступ к ним возможен только с помощью ди-

ректив, вводимых с пульта оператора, которые расшифровываются программой ПОФ с передачей управления требуемой подпрограмме этой подсистемы. С помощью директив ПОФ возможно осуществить распечатку файлов, копирование файлов, распечатку каталога файлов и др.

Заметим, что при использовании любой из этих подсистем пользователю приходится оперировать с такой единицей управления, как файл, который характеризуется следующими тремя атрибутами: *именем файла* (до восьми символов), *именем диска* (до шести символов), на котором расположен этот файл, и *порядковым номером дискового накопителя.*

Для повышения удобств работы очень часто такие атрибуты, как имя диска и порядковый номер, могут опускаться. В этом случае подразумевается, что файл располагается на одном из системных дисков, который должен быть определен таковым в макрокоманде #**СУФ** при генерации ОС.

Доступ к файлам. Для обеспечения доступа к дисковым файлам в задачах пользователя с помощью подпрограмм первой и второй подсистем СУФ необходимо выполнить следующее:

осуществить назначение логическому номеру имени файла на диске;

открыть дисковый файл, т.е. обратиться к ОС с требованием на его поиск, если файл создан ранее, или же на его создание, если файл не был создан;

после открытия файла выполнить операцию чтения и (или) записи очередной порции данных;

закрыть файл и, если не требуется повторное его открытие, отменить проведенное ранее назначение логическому номеру файла.

Файл данных может занимать один или несколько несмежных участков на диске. В первом случае вся работа с файлом ограничивается приведенным выше набором операций. Во втором случае, когда файл располагается в нескольких несмежных участках (экстентах), пользователю необходимо выполнять дополнительные операции открытия экстентов, если он работает с программами первой подсистемы СУФ. При работе с программами второй подсистемы СУФ пользователь оперирует не с относительными номерами секторов в файле, а с записями, что подразумевает автоматическое открытие экстентов подпрограммами СУФ при первой же необходимости.

11.2. Подготовка файлов к работе

Закрепление УВВ и дисковых файлов за логическими номерами, используемыми в задачах, как уже отмечалось ранее, обычно осуществляется командой оператора :**ЛН** перед запуском задачи на выполнение или макрокомандой #**ЛН** в программе генерации ОС (например, :**ЛН**,5,*МНКД*).

Однако в ситуациях, когда самой задаче известны порядковые номера устройств ввода-вывода и (или) имена файлов, требуемых для работы, допускается назначение таких устройств или файлов с помощью макрооперации **EXIO** следующего специального формата.

Приведем пример назначения свободному логическому номеру имени файла *ТЕМПДАВЛ*, который должен располагаться на диске с порядковым номером 14 и использоваться в задачах C и D для записи текущих значений температуры и давления (см. § 9.2):

```
EXIO  =L0,=L20002B,БЗ,БУФ1,=L9
WAITE БЗ
```

Здесь

```
БУФ1   OCT  0                Логический номер
       ASC  4,ТЕМПДАВЛ       Имя файла
       OCT  0,0,0            Имя диска (системного)
       DEC  14               Порядковый номер диска
БЗ     BSS  2
```

Параметры макрооперации **EXIO** имеют следующий смысл: **=L***20002***B** – управляющее слово, определяющее операцию назначения, **=L***0* – указание нулевого логического номера, интерпретируемое ОС как требование найти и использовать свободный (незадействованный) логический номер (записывается системой во вторую ячейку *БЗ*), **=L***9* – длина требуемого в данном случае *БУФ1*.

В общем случае длина буфера может равняться: 1 – для операции освобождения ранее назначенного логического номера; 2 – для операции назначения логического номера, записанного в первом слове буфера, устройству, порядковый номер которого записан во втором слове; 5 – для операции назначения именованному файлу на системном диске (имя файла должно располагаться в двух–пяти словах буфера); 8 – то же для именованного диска, имя которого должно быть записано в шести–восьми словах; 9 – когда задаче необходимо указать, кроме того, порядковый номер диска, который и записывается в девятом слове буфера.

11.3. Подсистема управления данными

Работая с этой подсистемой СУФ, пользователь должен рассматривать файл как последовательность записей, содержащих данные, с которыми пользователь оперирует, не привязываясь к номерам секторов, отводимых под эти записи. Записи в дисковом файле имеют тот же смысл, что и для недисковых УВВ, и могут быть фиксированной (постоянной) либо переменной длины.

Запись постоянной длины не имеет специальных признаков начала и конца записи и определяется только своей постоянной длиной, которая

задается 1 раз при создании файла и хранится в дальнейшем в его метке.

Запись переменной длины в файле на диске обрамлена специальными признаками начала и конца записи, представляющими собой отрицательную длину записи (в байтах), записанную в первое и последнее слова каждой записи.

Эти признаки формируются СУФ только при выводе записи на диск: В буфере же пользователя запись переменной длины содержится без них.

Тип записей, из которых составляется файл, выбирается пользователем в зависимости от требований задачи, и этот же тип определяет выбор последовательного или прямого доступа к записям. Запись является единицей обмена информации с файлом, которую задача может прочитать либо записать в файл за одно обращение к подпрограммам управления данными.

Так как записи располагаются в файле последовательно, то указателем записи, с которой работает задача, для СУФ является ее порядковый номер внутри файла, всегда начинающийся с единицы. При этом, если используются программы последовательного доступа СУФ, номер записи автоматически наращивается ими на единицу при переходе к следующей записи. При обращении к программам прямого доступа этот номер всякий раз должен быть указан явно в качестве параметра подпрограммы СУФ.

Отсюда и различие в дисциплине работы: последовательный доступ обеспечивает последовательную работу с записями постоянной и переменной длины и, если понадобится прочитать или изменить не следующую запись, а какую-нибудь другую, выполняет две операции – установку на нужный номер записи и чтение (или изменение) этой записи.

Для программ прямого метода доступа, обеспечивающих работу только с записями постоянной длины, возможно выполнение операции чтения, записи или изменения любой записи за одно обращение к СУФ.

В целом, при применении любого из этих методов можно выполнить четыре основные операции с записями файла:

чтение (ввод) записи из ранее созданного файла;

запись (вывод) новой записи в создаваемый файл;

изменение старой записи на новую такой же длины в ранее созданном файле;

расширение ранее созданного файла новыми записями, записываемыми в конец файла.

Все подпрограммы СУФ требуют, чтобы в задачах обращения к ним резервировалась дополнительная рабочая область, называемая *блоком управления данными* (БУД). Эта область состоит из 144 ячеек и используется подпрограммами СУФ как промежуточный буфер для временного хранения данных, читаемых либо записываемых на диск за одно физическое обращение к нему.

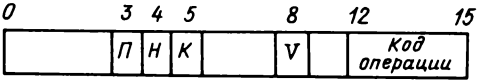


Рис. 11.2. Структура слова режима открытия файла

Число БУД, резервируемых в задаче, определяется количеством одновременно обрабатываемых файлов в задаче. В некоторых случаях при необходимости увеличения скорости обмена информацией с диском это можно достичь за счет увеличения числа ячеек, резервируемых под БУД. Число ячеек задается по формуле 16 + 128*n*, где *n* может быть равно 1 и более.

Открытие файла. Для открытия файла независимо от применяемого метода доступа используется подпрограмма *OPEN*. Здесь и далее вызовы подпрограмм из библиотеки СУФ даны в формате макрокоманд, получивших название *макроопераций вызова СУФ*. Макроопределения этих макрокоманд собраны в той же основной библиотеке макроопределений, что и макроопределения всех макроопераций вызова супервизора.

Однако в отличие от последних макрооперации вызова СУФ генерируют макрорасширения в виде вызывающей последовательности к соответствующей подпрограмме СУФ.

Формат макрооперации **OPEN** следующий:

OPEN *буд*,*лн*,*бз*,*реж*,=L*144* [,*кзащ*,] [,*длф*] [,*длз*] [,*дпм*]

Здесь *буд* – блок управления данными; *лн* – логический номер, используемый для идентификации файла; *бз* – двухсловный блок завершения; *реж* – режим открытия файла; =L*144* – длина рабочей области – БУД; *кзащ* – код защиты файла; *длф* – длина дискового файла, задаваемая только при создании файла. Если этот параметр опущен, то файл создается размером в 32 сектора. Если этот размер недостаточен для создания новых записей или расширения существующего файла, то автоматически образуются экстенты такого же размера; *длз* – длина записи, которая задается только при создании файла, состоящего из записей постоянной длины. Если параметр опущен, его значение приравнивается 128 словам; *дпм* – единственный параметр, который может быть записан задачей в метку файла при его создании.

Режим открытия файла задается в слове, имеющем структуру, изображенную на рис. 11.2.

Здесь код операции (разряды 12–15) указывает на режим открытия:

0001 – поиск существующего файла для чтения или изменения в нем ранее созданных записей;

0010 – создание нового файла. Этот режим используется для вывода новых записей с начала файла;

0100 – поиск существующего файла для расширения его. Этот режим применяется для расширения существующего файла новыми записями, заносимыми в конец файла.

Разряд *V* указывает на тип записей, из которых составляется файл:

V = 0 – запись постоянной длины;

V = 1 – запись переменной длины.

Разряд *K* используется только для записи переменной длины и указывает на допустимость замены старых записей внутри файла:

K = 0 – замена разрешена, тогда длина заданного буфера при выводе записи должна совпадать с длиной заменяемой записи в файле на диске;

K = 1 – замена запрещена. Любая попытка заменить запись при этом запрете приведет к тому, что выведенная запись станет последней записью в файле.

Для записей постоянной длины замена всегда разрешена, так как заменяющая и заменяемая записи равны по длине.

Разряд *Н*, равный единице, задает монопольное открытие файла.

Разряд *П*, равный единице, используется только для прямого метода доступа. Он является указателем того, что новые записи могут выводиться во вновь создаваемый файл не последовательно, а произвольно, т. е. разрешается создание новых записей в любом "пустом" месте файла.

Закрытие файла. Закрытие файла осуществляется макрооперацией **CLOSE**, имеющей следующий формат:

CLOSE *бул,бз* [*,режим*] [*,дпм*]

Здесь *буд* – блок управления данными, зарезервированный при открытии файла. Так как логический номер закрываемого файла не указывается в этой макрооперации, то параметр *буд* является единственным указателем закрываемого файла; *бз* – двухсловный блок завершения, имеющий прежний смысл; *режим* – этот параметр имеет тот же смысл, что и в макрооперации **EXIO**, используемой для закрытия файла; *дпм* – дополнительный параметр, который может быть записан задачей в метку файла при его закрытии.

11.4. Последовательный метод доступа

Рассмотрим макрооперации, используемые при последовательном методе доступа.

Макрооперация GET. Эта макрооперация применяется для чтения очередной записи постоянной или переменной длины из файла на диске в буфер пользователя. Предварительно файл должен быть открыт в режиме поиска. Если эта макрооперация повторяется многократно для последовательного чтения записей, то номер очередной записи формируется автоматически при переходе от одной записи к следующей. При выборочном чтении записи потребуется предварительная установка на номер выбранной записи с помощью макрооперации **FCONT**. Если у файла имеются экстенты, то они будут открываться автоматически по мере необходимости. Признаком конца файла является специальный код за-

вершения, помещаемый в первое слово *бз* при очередном выполнении макрооперации **GET**.

Формат макрооперации **GET**:

GET *буд,бз,буф,дл*

Здесь *буд* – блок управления данными, указанный при открытии в соответствующей макрооперации **OPEN**; *бз* – двухсловный блок завершения, имеющий прежний смысл; *буф* – буфер, в который принимается запись; *дл* – длина буфера.

Макрооперация PUT. Эта макрооперация осуществляет одну из следующих четырех операций:

вывод очередной записи постоянной или переменной длины из буфера пользователя во вновь создаваемый файл на диске. Предварительно этот файл должен быть открыт в режиме создания. Здесь разрешен только последовательный вывод записей, осуществляемый с автоматическим наращиванием номера записи;

замену любой старой записи переменной длины на новую такой же длины. Предварительно этот файл должен быть открыт в режиме поиска с установкой признака разрешения замены. Для выбора записи, подлежащей замене, потребуется предварительная установка на номер выбранной записи с помощью макрооперации **FCONT**. Если длина заменяющей записи не совпадает с длиной заменяемой записи, то выведенная запись становится последней в файле;

расширение ранее созданного файла новыми записями. Предварительно этот файл должен быть открыт в режиме расширения. Каждая новая выводимая запись с помощью **PUT** будет выводиться в конец файла. Предварительного использования макрооперации **FCONT** для установки номера записи, на единицу большего номера последней записи в файле, не потребуется, так как эта установка производится автоматически при открытии файла в данном режиме;

расширение ранее созданного файла с возможностью замены старых записей. Предварительно этот файл должен быть открыт в режиме расширения с установкой признака разрешения замены. Здесь разрешено совмещение второй и третьей операций, перечисленных выше.

Формат макрооперации **PUT**:

PUT *буд,бз,буф,дл*

Здесь все параметры имеют тот же смысл, что и в макрооперации **GET**. Следует обратить внимание только на то, что *буф* содержит выводимую информацию.

Макрооперация FCONT. Эта макрооперация пропускает записи переменной или постоянной длины в целях установки на необходимый номер записи. Предварительно файл должен быть открыт в требуемом режиме.

Пропуск осуществляется относительно текущего номера записи.

Формат макрооперации:

FCONT *буд,бз,коп,чз*

Здесь параметры *буд* и *бз* имеют прежний смысл; *коп* – код выполняемой операции: *коп* = 2 – пропуск вперед; *коп* = 3 – пропуск назад; *чз* – число пропускаемых записей.

Эта макрооперация может осуществляться и в других целях при работе с недисковыми УВВ, для которых параметр *коп* будет принимать другие значения.

Остальные макрооперации последовательного доступа (**STORE, RESTR, AREST** и **EOV**) применяют довольно редко и предназначены для запоминания и восстановления текущего состояния в буфере задачи, аварийного восстановления конца дискового файла и для работы с магнитной лентой.

11.5. Прямой метод доступа

Рассмотрим макрооперации, используемые при прямом методе доступа.

Макрооперация READ. Эта макрооперация используется для последовательного или выборочного чтения записи постоянной длины из файла на диске в буфер пользователя. Чтение записи осуществляется по ее номеру, указанному в запросе задачи.

Файл должен быть предварительно открыт в режиме поиска. Если осуществляется последовательное чтение записей, то задача обязана самостоятельно следить за правильностью указания очередного номера записи.

Формат макрооперации **READ**:

READ *буд,бз,нз,буф,дл*

Здесь *буд* – блок управления данными, указанный при открытии в соответствующей макрооперации **OPEN**; *бз* – двухсловный блок завершения, имеющий прежний смысл; *нз* – номер считываемой записи относительно начала файла; *буф* – буфер, резервируемый для приема записи. Длина этого буфера должна рассчитываться так, чтобы быть большей либо равной постоянной длине записи, записанной в метке файла; *дл* – длина буфера.

Макрооперация WRITE. Эта макрооперация используется для осуществления одной из следующих четырех операций с обязательным указанием номера записи в запросе:

вывод очередной записи постоянной длины из буфера задачи во вновь создаваемый файл на диске. Предварительно этот файл должен быть

открыт в режиме создания. Здесь разрешен только последовательный вывод записей с обязательным указанием очередного номера выводимой записи в запросе;

вывод записи постоянной длины из буфера задачи в любое место вновь создаваемого файла на диске. Предварительно этот файл должен быть открыт в режиме создания с установкой разряда П = 1 (см. макрооперацию **OPEN**). Это позволяет заполнять файл новыми записями не последовательно, а произвольно, в любом порядке;

расширение ранее созданного файла новыми записями. Предварительно этот файл должен быть открыт в режиме расширения. Каждая новая выводимая запись с помощью **WRITE** должна записываться в конец файла. При этом задаче приходится самой следить, чтобы номер очередной записи в запросе был на единицу больше номера предыдущей выведенной записи;

вывод записи из буфера задачи в произвольное место вновь создаваемого файла с возможностью его дальнейшего расширения. Предварительно этот файл должен быть открыт в режиме расширения с установкой разряда П = 1. Фактически здесь разрешено совмещение второй и третьей операций. Замена старых записей на новые здесь не выделена в отдельную операцию, так как она всегда разрешена при любом режиме открытия файла. Таким образом, операция замены может быть совмещена с любой из перечисленных ранее операций этого метода доступа.

Формат макрооперации **WRITE**:

WRITE *буд,бз,нз,буф,дл*

Здесь все параметры имеют тот же смысл, что и в макрооперации **READ**. Следует только обратить внимание на то, что *буф* должен содержать выводимую информацию постоянной длины, которая должна совпадать с длиной существующих записей в файле.

Примеры использования некоторых макроопераций СУФ последовательного доступа приведены в § 11.7.

11.6. Директивы программы обслуживания файлов

Приведем методику использования директив оператора программы обслуживания файлов (*ПОФ*), в которой собраны подпрограммы третьей подсистемы СУФ. Программа ПОФ представляет собой системную обрабатывающую программу, которая загружается в раздел и вызывается на выполнение системной командой оператора

:СТ [АРТ] , *ПОФ*

После запуска *ПОФ* и выдачи ею на пульт оператора разрешающего символа / пользователь может вводить с пульта двухсимвольные дирек-

тивы с параметрами. Ниже приводятся форматы некоторых часто используемых директив и их назначение:

ПС, *пнд* – распечатка справочника каталога файлов диска.

Здесь и далее *пнд* – порядковый номер диска; *имяд* – имя диска; *имяф* – имя файла; *кзащ* – код защиты файла.

ПМ,*имяф* [:[*имяд*] :*пнд*]] [:*кзащ*] – распечатка метки файла;

ПФ,*имяф* [:[*имяд*] [:*пнд*]] [:*кваш*] – распечатка символьного файла;

СФ,*имяф* [:[*имяд*] [:*пнд*]] [:*кваш*] [,*длф*] [,*типз*] [,*длз*] – создание файла, т.е. резервирование места на диске под будущий файл.

Здесь *длф* – длина файла в секторах; *типз* – тип записи (переменной или постоянной длины); *длз* – длина записей постоянной длины.

УФ,*имяф* [:[*имяд*] [:*пнд*]] [:*кваш*] – уничтожение файла;

НФ,*имяф1* [:[*имяд*] [:*пнд*]] [:*кзащ*],*имяф2* – переименование файла;

КФ,*имяф1* [:[*имяд1*] [:*пнд1*]] [:*кзащ1*], *имяф2* [[:*имяд2*] [:*пнд2*]] – копирование файла.

11.7. Примеры использования СУФ

В этом параграфе приведены примеры использования макроопераций вызова СУФ последовательного доступа в задачах С и Е, описанных в § 9.2.

Все пояснения к примерам содержатся в комментариях к операторам задач:

```
                        ЗАДАЧА С
ASMB R,B,L,T
      NAM C,3
OБЛ1  COM РЕЖИМ
OБЛ2  COM БУСО
      EXT П1Ф
* ЭТА ЗАДАЧА ЗАПУСКАЕТСЯ ПО ВРЕМЕНИ ИЗ ЗАДАЧИ А, РЕГИСТР Y НЕ ИСПОЛЬ-
* ЗУЕТСЯ, ТАК КАК В НЕМ ЗАПИСАН АДРЕС ОБЛАСТИ РЕЕНТЕРАБЕЛЬНОСТИ ДЛЯ П1Ф.
      TIME ВРЕМЯ    ЗАПРОС ТЕКУЩЕГО ВРЕМЕНИ С ПОМОЩЬЮ МАКРООПЕРАЦИИ TIME
* ПЕРЕВОД ПОЛУЧЕННОГО ВРЕМЕНИ В СИМВОЛЬНЫЙ ФОРМАТ
* И ЗАНЕСЕНИЕ ЕГО В БУФЕР ДЛЯ ВЫВОДА
      LDA ВРЕМЯ+3  - ПРЕВОДИМАЯ ВЕЛИЧИНА
      LDX АДБУФ    - РАБОЧИЙ БУФЕР ИЗ ТРЕХ ЯЧЕЕК
      JSB П1Ф      - ПОДПРОГРАММА ПЕРЕВОДА
      LDB АДБУФ+3
      STB ЧЧ       - ЗАНЕСЕНИЕ ЧАСОВ
      LDA ВРЕМЯ+2
      LDX АДБУФ
      JSB П1Ф
      LDB АДБУФ+3
      STB ММ       - ЗАНЕСЕНИЕ МИНУТ
      LDA ВРЕМЯ+1
      LDX АДБУФ
      JSB П1Ф
      LDB АДБУФ+3
      STB СС       - ЗАНЕСЕНИЕ СЕКУНД
      EXIO =L7,=L1Ф1В,Б3,БУФ,=L1,=L0   ОПРОС ТЕМПЕРАТУРЫ
      WAITE Б3
* ПЕРЕД ЗАПУСКОМ ЗАДАЧИ ЛОГИЧЕСКОМУ НОМЕРУ 7 ДОЛЖЕН БЫТЬ НАЗНАЧЕН
* ПОРЯДКОВЫЙ НОМЕР МАКРОКОМАНДЫ #МАЦП, ОПИСЫВАЮЩЕЙ В ПРОГРАММЕ
* ГЕНЕРАЦИИ ОС ПОДСИСТЕМУ УСО: КБ(А612-11)-АЦП(А611-19),
* ПРЕДНАЗНАЧЕННУЮ ДЛЯ ОПРОСА ТЕМПЕРАТУРЫ.
```

```
      LDA БУФ
      LDX ATT
      JSB П1Ф    - ЗАНЕСЕНИЕ ТЕМПЕРАТУРЫ
* ИСПОЛЬЗОВАНИЕ АППАРАТА WAITE-POSTE С ОБЩИМ БУС ДЛЯ РАБОТЫ С ОБЩИМ
* РЕСУРСОМ - ДИСКОВЫМ ФАЙЛОМ ТЕМПДАВЛ. В ЭТОТ ФАЙЛ ВЫВОДЯТСЯ ЗАПИСИ
* СО ЗНАЧЕНИЯМИ ТЕМПЕРАТУРЫ И ДАВЛЕНИЯ ИЗ ЗАДАЧ С И D.
      WAITE БУСФ  - ЗАНЯТИЕ РЕСУРСА
* НАЗНАЧЕНИЕ СВОБОДНОМУ ЛОГИЧЕСКОМУ НОМЕРУ ИМЕНИ ФАЙЛА ТЕМПДАВЛ, РАС-
* ПОЛОЖЕННОМУ НА ДИСКЕ С ПОРЯДКОВЫМ НОМЕРОМ, РАВНЫМ 14
      EXIO =L0,=L20002B,Б3,БУФ1,=L9
      WAITE Б3
      OPEN БУД,БУФ1,Б3.=L204B,=L144  ОТКРЫТИЕ ФАЙЛА В РЕЖИМЕ РАСШИРЕНИЯ
      LDA Б3     АНАЛИЗ КОДА ЗАВЕРШЕНИЯ НА НАЛИЧИЕ ФАЙЛА
      CPA =B5Ф
      JMP *+2
      JMP ВЫВОД
* ЕСЛИ ФАЙЛА НЕТ (УНИЧТОЖЕН В ЗАДАЧЕ E), ТО ОТКРЫТИЕ В РЕЖИМЕ СОЗДАНИЯ
      OPEN БУД,БУФ1,Б3.=L202B,=L144
ВЫВОД PUT  БУД,Б3,БУФВ.=L21  ВЫВОД ОЧЕРЕДНОЙ ЗАПИСИ ИЗ БУФЕРА БУФВ
      CLOSE БУД,Б3   ЗАКРЫТИЕ ФАЙЛА
      EXIO  =L0,=L20002B,Б3,БУФ1,=L1  ОСВОБОЖДЕНИЕ ЛОГИЧЕСКОГО НОМЕРА
      WAITE Б3
      POSTE БУСФ    - ОСВОБОЖДЕНИЕ РЕСУРСА
      LDB РЕЖИМ     - АНАЛИЗ НОМЕРА ТЕХНОЛОГИЧЕСКОГО РЕЖИМА
      CPB =B1
      JMP C1
      CPB =B2
      JMP C2
* ТРЕТИЙ ТЕХНОЛОГИЧЕСКИЙ РЕЖИМ. СРАВНЕНИЕ С 3-Й УСТАВКОЙ
      LDA БУФ
      ADA УСТ3
      SSA,RSS   ЕСЛИ ТЕМПЕРАТУРА БОЛЬШЕ  ЛИБО РАВНА ТРЕТЬЕЙ
КОНЕЦ EXIT      УСТАВКЕ, ТО ЗАВЕРШИТЬ ЗАДАЧУ
* ЕСЛИ ТЕМПЕРАТУРА МЕНЬШЕ 3-Й УСТАВКИ, ТО ЗАПУСКАЕТСЯ ЗАДАЧА В
* ДЛЯ ЗАКРЫТИЯ КЛАПАНА КЛ2 И ЗАДАЧА A ДЛЯ ОТКРЫТИЯ КЛАПАНА КЛ1
      RUN B,Б3,=L1
      WAITE Б3
      RUN A,Б3,=L0
      WAITE Б3
      JMP КОНЕЦ
* ПЕРВЫЙ ТЕХНОЛОГИЧЕСКИЙ РЕЖИМ. СРАВНЕНИЕ С 1-Й УСТАВКОЙ
C1    LDA БУФ
      ADA УСТ1
      SSA       ТЕМПЕРАТУРА МЕНЬШЕ 1-Й УСТАВКИ,
      JMP КОНЕЦ       ЗАВЕРШИТЬ ЗАДАЧУ
* ТЕМПЕРАТУРА БОЛЬШЕ ЛИБО РАВНА 1-Й УСТАВКЕ. УСТАНОВКА 2-ГО ТЕХНОЛОГИ-
* ЧЕСКОГО РЕЖИМА И НОВЫХ ВРЕМЕННЫХ ХАРАКТЕРИСТИК ЗАДАЧАМ C И D
      LDA =B2
      STA РЕЖИМ
      TURN C,Б3,=L2,=L4,=L-4
      TURN D,Б3,=L2,=L5,=L-5
      JMP КОНЕЦ    - НА ЗАВЕРШЕНИЕ
* ВТОРОЙ ТЕХНОЛОГИЧЕСКИЙ РЕЖИМ. СРАВНЕНИЕ СО 2-Й УСТАВКОЙ
C2    LDA БУФ
      ADA УСТ2
      SSA             - ТЕМПЕРАТУРА МЕНЬШЕ 2-Й УСТАВКИ,
      JMP КОНЕЦ          ЗАВЕРШЕНИЕ
* ТЕМПЕРАТУРА БОЛЬШЕ ЛИБО РАВНА 2-Й УСТАВКЕ. ЗАПУСК ЗАДАЧИ A ДЛЯ
* ЗАКРЫТИЯ КЛАПАНА
      RUN A,Б3,=L1
      WAITE Б3
      JMP КОНЕЦ
* РАБОЧИЕ ЯЧЕЙКИ, БУФЕРЫ, Б3, БУД
ВРЕМЯ BSS 5     БУФЕР ДЛЯ TIME
АДБУФ DEF *+1  АДРЕС БУФЕРА ДЛЯ П1Ф
      BSS 3
* БУФЕР ВЫВОДА СИМВОЛЬНОЙ ЗАПИСИ В ФАЙЛ ТЕМПДАВЛ
БУФВ  ASC 3,ВРЕМЯ
ЧЧ    NOP
```

```
        ASC 3,'ЧАС'.
ЧМ      NOP
        ASC 3,МИН.
СС      NOP
        ASC 6,СЕК.ТЕМП=
ТТ      BSS 3
ATT     DEF TT
БЗ      BSS 2          - БЛОК ЗАВЕРШЕНИЯ
БУФ     BSS 1          - БУФЕР ДЛЯ ВВОДА ТЕМПЕРАТУРЫ
БУФ1    OCT 0          - БУФЕР, СОДЕРЖАЩИЙ ИМЯ ФАЙЛА И ПНД=14
        ASC 4,ТЕМПДАВЛ
        OCT 0,0,0
        DEC 14
БУД     BSS 144        - БУД ДЛЯ ФАЙЛА
УСТ1    DEC            - ПЕРВАЯ УСТАВКА
УСТ2    DEC            - ВТОРАЯ УСТАВКА
УСТ3    DEC            - ТРЕТЬЯ УСТАВКА
        ORG БУСО  РЕЗЕРВИРОВАНИЕ В БУСО
        OCT 40000 ИСХОДНОГО СОСТОЯНИЯ МР=01
        OKR
        END
        ENDM


                          ЗАДАЧА Б

ASMB,R,B,L,T
        ЧАМ Б,3,,13,45,45
* ЭТА ЗАДАЧА ЗАПУСКАЕТСЯ ПО ВРЕМЕНИ, ВВОДИТ ЗАПИСИ С ФАЙЛА ТЕМПДАВЛ И
* ВЫВОДИТ ИХ НА ПЕЧАТЬ.
        EXIU =L0,=L20000B,БЗ,БУФ1,=L9  НАЗНАЧЕНИЕ СВОБ.ЛОГИЧЕСКОМУ НОМЕРУ
        WAITE БЗ                  ИМЕНИ ФАЙЛА ТЕМПДАВЛ::14
        OPEN БУД,БУФ1,БЗ,=L2018,=L144   ОТКРЫТИЕ ФАЙЛА В РЕЖИМЕ ПОИСКА
ВВОД    GET БУД,БЗ,БУФ,=L25   ВВОД ОЧЕРЕДНОЙ ЗАПИСИ В БУФЕР ЗАДАЧИ
        LDA БЗ      АНАЛИЗ ПРИЗНАКА КОНЦА ФАЙЛА.
        CPA =B40    ЕСЛИ КОНЕЦ ФАЙЛА, ТО
        JMP ЗАКР    УНИЧТОЖИТЬ ФАЙЛ
        LDA БЗ+1      - ПОЛУЧЕНИЕ ДЛИНЫ ВВЕДЕННОЙ
        STA ДЛ                ЗАПИСИ
        EXIO =L0,=L2,БЗ,БУФ,ДЛ  ВЫВОД ВВЕДЕННОЙ ЗАПИСИ НА ПЕЧАТЬ
        WAITE БЗ
        JMP ВВОД       - НА ВВОД СЛЕДУЮЩЕЙ ЗАПИСИ
ЗАКР    CLOSE БУД,БЗ,=1=1  УНИЧТОЖЕНИЕ ФАЙЛА, КОГДА ВСЕ ЗАПИСИ РАСПЕЧАТАНЫ
        EXIO =L0,=L20002B,БЗ,БУФ1,=L1  ОСВОБОЖДЕНИЕ ЛОГ. НОМЕРА
        WAITE БЗ
        CLA                  ПОДГОТОВКА ДН=0
        STA БУФ1
        EXIT                 ЗАВЕРШЕНИЕ ЗАДАЧИ
БУФ1    OCT 0  БУФЕР ДЛЯ НАЗНАЧЕНИЯ - ЛОГ НОМЕР
        ASC 4,ТЕМПДАВЛ           - ИМЯ ФАЙЛА
        OCT 0,0,0                - ИМЯ ДИСКА
        DEC 14                   - ПОРЯД. НОМЕР ДИСКА
БУД     BSS 144                  - БУД ДЛЯ ФАЙЛА
БЗ      BSS 2                    - БЛОК ЗАВЕРШЕНИЯ
БУФ     BSS 25                   - БУФЕР ДЛЯ ВВОДА ЗАПИСИ С ФАЙЛА
ДЛ      NOP                      - РАБ. ЯЧЕЙКА ДЛЯ ДЛИНЫ
        END
        ENDM
```

11.8. Пример компоновки загрузочного модуля с задачами

Подведя итог всему изложенному в предыдущих главах, рассмотрим характерный порядок работы оператора ЭВМ при создании загрузочного модуля с задачами, описанными в § 10.3, 10.9 и 11.7.

Для подготовки загрузочных модулей может быть использован любой выделенный для этих целей раздел пользователя во вновь сгенерированной ОС или в стартовой операционной системе. Поэтому предполагается, что выбранная для работы ОС с помощью начального загрузчика уже размещена в нулевом разделе, запущена и находится в состоянии ожидания ввода команд оператора. Оператор вводит нужный ему номер раздела и выполняет последовательность операций, требуемых для формирования загрузочного модуля.

Как уже отмечалось в § 9.4, для создания загрузочного модуля II типа необходимо выполнить четыре этапа работ. Будем считать, что первый этап нами успешно завершен и запрограммированные на МАКРОЯЗЫКе задачи в исходном состоянии сформированы на одном из системных дисков под соответствующими именами файлов: *АИ, ВИ, СИ, DИ, ЕИ, FИ.* Здесь символ *И* указывает на исходный формат. Для подготовки и загрузки на диск этих программ могут быть использованы любые стандартные средства, обеспечивающие работу с символьными данными (например, системная обрабатывающая программа ПОФ, см. § 11.6).

На втором этапе каждый из этих файлов должен быть подвергнут макрогенерации с помощью макрогенератора МГД, входом для которого являются задача на МАКРОЯЗЫКе и основная библиотека макроопределений, организованная на системном диске под названием ОБМД. Выходом макрогенератора является файл со сгенерированной задачей на МНЕМОКОДе.

Здесь и далее будем применять системные диски для удобства записи команд оператора, хотя в общем случае файлы с программами могут располагаться на любом дисковом накопителе пользователя.

Для осуществления макрогенерации задачи А последовательность команд операторов следующая:

:ЛН,5,*АИ*
:ЛН,3,*ОБМД*
:ЛН,4,*АС*
:ЛН,6,*пнуп*
:СТ [АРТ],*МГД*

Здесь *пнуп* – порядковый номер устройства печати листинга. Результирующий файл создается на системном диске под именем *АС.* Очевидно, что для макрогенерации остальных задач должны использоваться аналогичные последовательности команд оператора, но с указанием соответствующих результирующих файлов: *ВС, СС, DC, ЕС* и *FC.* Здесь символ *С* условно указывает на символьный формат.

На третьем этапе каждая из полученных в результате макрогенерации программ должна быть транслирована транслятором с МНЕМОКОДа (*МНКД*) с получением тех же программ, но в объектном формате.

Последовательность команд оператора в этом случае следующая:

:ЛН,5,*АС*
:ЛН,4,*АД*
:ЛН,6,*лкуп*
:СТ [АРТ] ,*МНКД*

Результирующий файл трансляции создается под именем *АД*. Очевидно, что для трансляции остальных задач используются аналогичные последовательности команд, но с указанием соответствующих результирующих файлов: *ВД, СД, ДД, ЕД* и *FД*. Здесь символ *Д* условно указывает на двоичный формат.

На четвертом этапе с помощью компоновщика программ (*КОМПД*) осуществляется компоновка транслированных программ с необходимыми библиотечными подпрограммами в единый многозадачный загрузочный модуль. Компоновщик запускается на выполнение командой

:СТ [АРТ] , *КОМПД*

и выдает на пульт оператора следующую информацию:

КОМПОНОВЩИК ПРОГРАММ
РЕЖИМ?

В ответ на каждый запрос о режиме пользователь вводит необходимую директиву для компоновщика с указанием имени файла, содержащего компонуемую программу или библиотеку. По мере поступления программ на компоновку компоновщик распечатывает имена тех программ, которые вошли в загрузочный модуль, и их граничные адреса внутри модуля.

По завершении компоновки на пульт оператора выводится информация о распределении адресов между программами внутри загрузочного модуля, о наличии свободной памяти в нулевой странице и размере всего модуля.

Ниже приводятся форматы директив компоновщику, вводимых оператором в ответ на запросы о режимах. Следует обратить внимание на то, что задачи C, D и E компонуются с подпрограммами из основной библиотеки #ОБП. Так как используемые здесь подпрограммы являются реентерабельными и требуют наличия только одной их копии в загрузочном модуле, то соответствующие библиотеки поступают на компоновку только 1 раз с задачей C:

РЕЖИМ?ПП,*АД*
РЕЖИМ?ПП,*ВД*
РЕЖИМ?ПП,*СД*
РЕЖИМ?БП,# *ОБП*
РЕЖИМ?ПП,*ДД*
РЕЖИМ?ПП,*ЕД*
РЕЖИМ?ПП,*FД*
РЕЖИМ?ПК, *ABCDEF*

Последняя директива указывает компоновщику на то, что должен быть сгенерирован загрузочный модуль на одном из системных дисков под именем *ABCDEF.*

После того как загрузочный модуль с задачами скомпонован на диске под именем *ABCDEF,* он может быть загружен в раздел, предназначенный для работы этих задач с помощью команды оператора:

:**ВД**,*ABCDEF*

Так как все задачи, кроме F, связаны с выполнением по времени, то в систему должно быть введено время дня. С этого момента процесс запустится и будет выполняться согласно установленному в задачах временному расписанию.

Задача F запускается с пульта оператора с помощью команды оператора :**ВЫЗОВ**,*F*

Упражнения к гл. 11

1. Перечислите основные этапы, обеспечивающие доступ к дисковым файлам.

2. Запишите формат макрооперации **EXIO**, используемый для назначения свободному логическому номеру файла *ОЛИМП*::21.

3. Запрограммируйте задачу D, учитывая следующие особенности:

буфер вывода символьной записи в файл ТЕМПДАВЛ должен содержать следующую информацию длиной в 23 слова:

ВРЕМЯ чч ЧАС. мм МИН. сс СЕК. Давление ****** ;

логический номер, используемый для ввода значения давления, должен быть равен восьми.

Глава 12. ГЕНЕРАЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ

12.1. Типы ОС

С помощью ППМ для генерации ОС можно создавать ОС, предназначенные для эксплуатации в системах реального времени, многопультовых системах, системах пакетной обработки, а также в комбинированных системах многофункционального назначения. При этом ОС делится на три основных типа: однозадачная, многозадачная, двухпроцессорная.

Однозадачная ОС. В однозадачной ОС одна задача компонуется с модулями ОС в загрузочный модуль III типа, при этом задача может быть ОЗУ-резидентной и сегментированной.

Модули однозадачной ОС обеспечивают обработку запросов на операции ввода-вывода (**EXIO**), адресуемых как непосредственно к устройствам ввода-вывода, так и к дисковым файлам, приостанов выполнения задач до завершения операций ввода-вывода (**WAITE**) и завершение задачи (**EXIT**). Дополнительно в однозадачную ОС могут быть включены модули, обеспечивающие по запросам от обрабатывающих программ выдачу текущего времени суток (**TIME**), отсчет заданного в программе

интервала времени (**STIME**), приостанов выполнения программ до продолжения ее с пульта оператора системы (**PAUSE**), и макрооперация **STAT**. Для оперативного вмешательства в работу вычислительного комплекса без нарушения его нормального функционирования в однозадачную систему могут включаться специальные средства – программы связи с оператором системы, обеспечивающие ввод и обработку директив оператора с какого-либо из устройств клавишного ввода символьной информации. Основными директивами оператора для однозадачной системы являются команды назначения устройств или файлов, вызова задачи на выполнение, приостанова ее выполнения, а также перезапуска задачи после приостанова. По желанию пользователя в систему могут быть добавлены другие директивы оператора.

При необходимости однозадачные ОС можно приспособить и для работы в многозадачном режиме. В этом случае задачи оформляются в виде сегментов, для которых в качестве главной программы необходимо использовать диспетчер диск-резидентных программ реального времени (см. § 9.8).

Однозадачная ОС применяется, как правило, в одноразделных вычислительных комплексах и задается в программе генерации ОС с помощью макрокоманды **#ОС О.**

Многозадачная ОС. Многозадачная ОС обеспечивает выполнение задач в мультипрограммном режиме на однопроцессорных конфигурациях вычислительного комплекса. Задачи компонуются в многозадачные загрузочные модули II типа с учетом выполнения их в многораздельных вычислительных комплексах. Количество разделов, предназначенных для работы этих модулей, резервируется при генерации ОС с помощью макрокоманд **#РОП.** Для всех разделов устанавливается естественный приоритет, который убывает по мере возрастания номера раздела.

В многозадачной ОС алгоритм работы диспетчера задач таков, что среди всех готовых к выполнению задач всегда будет запускаться или выполняться самая старшая по приоритету задача. Приоритет задачи в многораздельном режиме будет определяться не только ее местом в цепочке БУЗов по приоритету внутри загрузочного модуля, но и местом, куда загрузочный модуль будет помещен, т.е. номером раздела. Для поиска старшей по приоритету задачи, готовой к выполнению, ОС сначала просматривает цепочку БУЗов в нулевом разделе и, если в нем нет готовых к выполнению задач, просматривает цепочку БУЗов из следующего раздела и т.д.

Таким образом, низкоприоритетные задачи, находящиеся в состоянии готовности, получают управление только тогда, когда задачи с более высокими приоритетами окажутся в состоянии ожидания, т.е. низкоприоритетные задачи получают управление в периоды ожиданий задач с более высокими приоритетами.

Многозадачная ОС обеспечивает возможность переключения процессора с одной задачи на другую при возникновении приоритетного запроса

на обработку. Последнее означает, что программный комплекс, созданный на основе многозадачной операционной системы, оперативно реагирует и приспосабливается к изменениям внешней среды, что является характерным для систем, работающих в реальном масштабе времени. Учитывая это обстоятельство, пользователь должен умело распределять готовые загрузочные модули с задачами в нужные разделы памяти. Очевидно, что загрузочный модуль с задачами, выполняемыми в реальном масштабе времени, всегда должен помещаться в высокоприоритетный раздел. Многозадачная ОС обеспечивает выполнение всех макроопераций вызова супервизора. Для управления работой задач из разных разделов и системой в целом при генерации многозадачной ОС может быть предусмотрено несколько пультов оператора, для каждого из которых может генерироваться свой набор директив оператора.

Как правило, многозадачная ОС используется для работы в многораздельных вычислительных комплексах с общим объемом оперативной памяти 32–128 К слов. В программе генерации многозадачная ОС задается с помощью макрокоманды **# ОС М.**

Двухпроцессорная ОС. Дальнейшим развитием режима мультипрограммной обработки является режим двухпроцессорной ОС. Двухпроцессорная ОС АСПО обеспечивает режим мультипроцессорной обработки в конфигурациях вычислительных комплексов с двумя процессорами, работающими на общее поле оперативной памяти и общую часть устройств ввода-вывода. Это означает, что оба процессора обсолютно идентичны по отношению к ячейкам оперативной памяти и устройствам ввода-вывода.

Если при обычном мультипрограммировании время только одного процессора разделяется между задачами, квазиодновременно работающими на вычислительном комплексе, то в случае двухпроцессорной обработки для выполнения задач используется время уже двух процессоров. Это позволяет достичь значительного повышения производительности по отношению к однопроцессорному вычислительному комплексу и, что особенно важно для систем реального времени, достичь значительно меньшего времени реакции системы на внешние события [16].

Последнее достигается за счет того, что любой из процессоров может оперативно отреагировать на прерывание, если другой в это время работает с каким-нибудь модулем ОС при выключенной системе прерываний. Кроме того, система становится менее уязвимой по отношению к сбоям и отказам процессора, поскольку в системе предусмотрено такое "горячее" резервирование процессоров, что в ряде случаев выход из строя одного процессора может не отразиться на работоспособности системы в целом.

В этой системе, так же как и в многозадачной ОС, задачи компонуются в многозадачные загрузочные модули II типа с учетом выполнения их в многораздельных вычислительных комплексах. Так как в двухпроцессорной системе обеспечивается параллельность работы двух задач, то

алгоритм выбора задач таков, что среди всех готовых задач оба процессора в каждый момент времени выполняют две самые старшие по приоритету.

Если задача, выполняющаяся одним из процессоров, переходит в состояние ожидания какого-либо внешнего по отношению к ней события (например, завершения операции ввода–вывода, истечения интервала времени), то процессор, закончивший эту задачу, переключается на менее важную, которая не выполняется другим процессором. Если для процессора не находится задачи, готовой к работе, процессор переводится в состояние динамического останова. При обратном изменении состояния какой-либо из задач (переход из состояния ожидания в состояние готовности) на ее реализацию переходит процессор, выполнявший задачу с меньшим приоритетом по сравнению с другим процессором.

Существуют две дисциплины работы диспетчера задач ОС. *Одна дисциплина* предусматривает, что всегда будут выполняться две старшие по приоритету задачи независимо от их принадлежности к одному и тому же или разным загрузочным модулям. Иначе все задачи рассматриваются состоящими как бы в едином списке, упорядоченном по приоритету с учетом приоритета задач внутри загрузочного модуля и приоритетом раздела, в котором работает загрузочный модуль.

Однако имеется возможность задать при генерации ОС *другую дисциплину* распределения процессоров между задачами, при котором кроме естественного приоритета задач учитывается их размещение в модулях оперативной памяти. В этом режиме в каждый момент времени выполняются две старшие по приоритету задачи, но обязательно расположенные в разных разделах памяти, т.е. не могут одновременно протекать две задачи из одного и того же загрузочного модуля. Теперь, если разделы закрепить за разными физическими модулями памяти, то можно достигнуть более высокой производительности системы в целом, так как ликвидируются задержки, связанные с одновременной работой двух процессоров с одним и тем же кубом памяти.

Дисциплина работы диспетчера задач в двухпроцессорной ОС задается в программе генерации макрокомандой

```
# ОС тип
```

Здесь *тип* = **З** указывает на первую дисциплину работы диспетчера (по задачам); *тип* = **Р** указывает на вторую дисциплину работы диспетчера (по разделам).

12.2. Состав ОС

Применяя принцип агрегатности, в частности разбиение всей системы на отдельные модули, ОС можно представить как совокупность следующих основных подсистем: ядро ОС (ядро супервизора), супервизор задач, подсистема ввода-вывода (супервизор ввода-вывода), подсистема

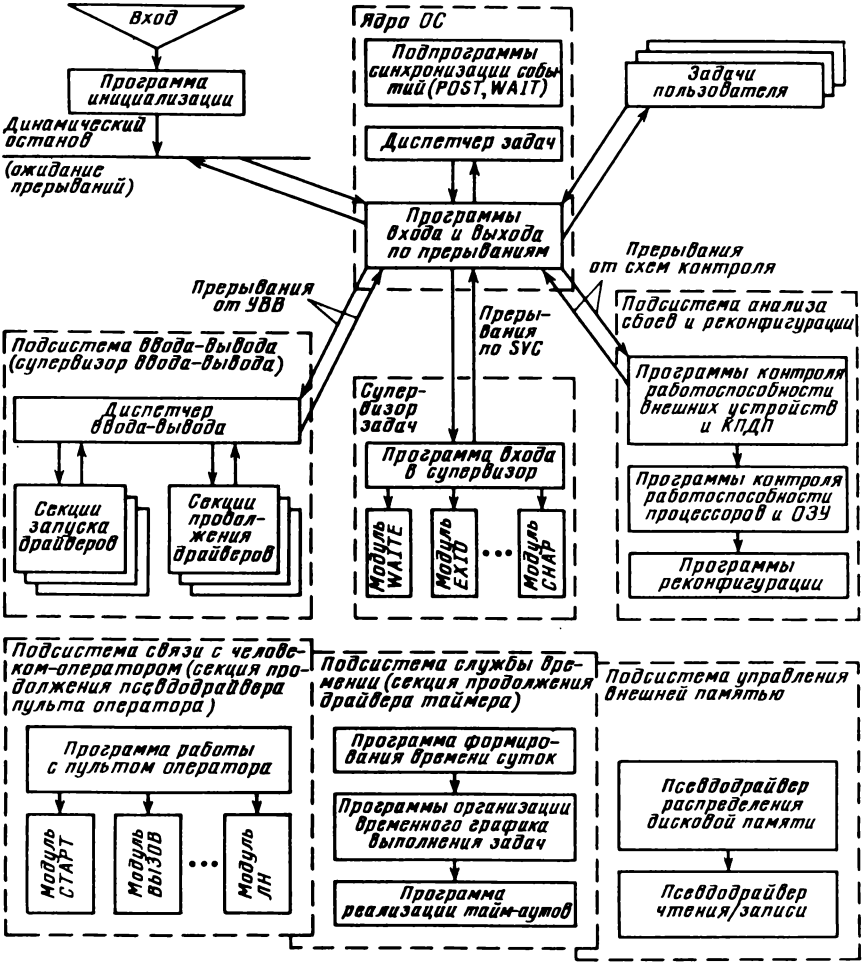


Рис. 12.1. Общая структура ОС

связи с человеком-оператором, подсистема службы времени, подсистема анализа сбоев и реконфигурации, подсистема управления внешней памятью, программа инициализации работы.

Общая структура ОС приведена на рис. 12.1.

Ядро ОС. Ядро ОС является связующим звеном между всеми остальными подсистемами ОС, которое обеспечивает распознавание прерываний и раздачу управления соответствующим подсистемам ОС. Кроме того, именно в ядре обеспечивается распределение процессорного времени и синхронизация событий.

Таким образом, ядро ОС представляет собой совокупность трех более мелких подсистем: программы входа и выхода по прерываниям; диспетчера задач, который распределяет процессорное время между задачами пользователя; подсистемы синхронизации событий, которая осуществляет синхронизацию задач и модулей ОС в связи с возникновением определенных ситуаций (событий) в системе (подпрограммы POST, WAIT).

Работа программ входа и выхода по прерыванию неразрывно связана с организацией стека прерываний. Так как основное время процессоров постоянно используется на выполнение одной или двух задач, то приостанов задач и передача управления любой из перечисленных подсистем ОС осуществляются при возникновении прерывания. Это может быть прерывание, вызванное самой задачей по макрооперации, либо прерывание от УВВ (от пульта оператора, таймера, инициативного УВВ, диска и т.п.), либо прерывание от схем контроля процессора.

Прерванное состояние задачи запоминается программой входа по прерыванию в *стеке прерываний.* По коду выборки прерывания анализируется вид прерывания и управление передается соответствующей подсистеме ОС:

прерывание от схем контроля процессора (код выборки равен *4* либо *5*) – подсистеме анализа сбоев и реконфигурации;

программное прерывание (код выборки равен *6*) – супервизору задач;

прерывания от УВВ (код выборки от *10* до *77*В) – подсистеме ввода-вывода.

Во время обработки очередного прерывания работа модуля ОС, обрабатывающего это прерывание, может быть прервана другим прерыванием, у которого более высокий приоритет. В этом случае появляется следующий *элемент стека.* Очевидно, что число элементов стека будет определяться глубиной (*уровнями*) прерываний. Это число указывается пользователем в программе генерации ОС в макрокоманде #**СВВ** и зависит от того, какое число одновременно работающих УВВ планируется пользователем в реально работающей системе. Здесь уместно напомнить, что в двухпроцессорной системе у каждого процессора имеется свой собственный стек прерываний.

Супервизор задач. Супервизор задач является основной подсистемой операционной системы, представляющий собой совокупность системных модулей, реализующих выполнение запрашиваемых в задачах макроопераций вызова супервизора. Супервизор задач состоит из программы входа в супервизор, осуществляющей распознавание макрооперации, и набора самих модулей супервизора, которые выполняют предписанные макрооперацией функции.

Для установления соответствия между кодом вызова супервизора и модулем обработки этого вызова в ОС формируется *таблица модулей супервизора* в процессе генерации ОС по макрокомандам #**ОС** и #**СЗД**. Для таблицы в памяти резервируется место, устанавливается ее длина,

а сами элементы заполняются адресами соответствующих модулей по макрокомандам # **ПОВС** и # **МОДУЛЬ**.

Подсистема ввода-вывода. Подсистема ввода-вывода (или супервизор ввода-вывода) представляет собой совокупность системных модулей, в функции которых входят прием, выполнение и обработка завершений операций ввода-вывода. Структурно супервизор состоит из *диспетчера ввода-вывода* и набора *драйверов*—подпрограмм, непосредственно реализующих алгоритмы обмена информацией с устройствами ввода-вывода.

Супервизор задач, получив запрос на выполнение макрооперации **EXIO**, передает управление подсистеме ввода-вывода, которая включает в работу диспетчер ввода-вывода. Тот, в свою очередь, формирует *элемент,* содержащий информацию о запрошенной операции, и ставит этот элемент в очередь к устройству, возвращая управление задаче, заказавшей эту макрооперацию. Как только сформированный элемент ввода-вывода окажется первым в очереди к указанному устройству, выполнится операция ввода-вывода.

По завершении операции задача извещается об этом (постируется), и запускается следующая операция из очереди к этому устройству. Число элементов очереди резервируется пользователем в программе генерации ОС и определяется количеством запросов, одновременно направляемых к одному и тому же УВВ.

Выполнением операций на устройствах ввода-вывода, т.е. реализацией алгоритма непосредственного обмена с УВВ, управляют специальные подпрограммы – *драйверы* устройств ввода-вывода. Получив управление от диспетчера ввода-вывода, драйвер УВВ (*секция запуска*) инициирует требуемую операцию и возвращает управление диспетчеру. После этого драйвер (*секция продолжения*) получает управление от программы обработки прерываний ядра операционной системы по запросам от устройства до тех пор, пока не будет полностью выполнена запущенная операция.

Здесь уместно ввести понятие *псевдодрайвера* как системной программы, выполняющей те же функции и имеющей те же интерфейсы, что и драйвер физического УВВ, но работающей не с физическим УВВ, а с некоторым *псевдоустройством,* с которым пользователь контактирует в своей задаче с помощью все той же макрооперации **EXIO**.

При работе с ОС АСПО пользователю довольно часто приходится иметь дело с такими псевдоустройствами. К ним относятся всевозможные групповые устройства (группы инициативных устройств, группы резервируемых устройств и т.п.), псевдоустройства межзадачного обмена, пульты операторов, дисковые файлы и т.д. Отличительной особенностью псевдодрайверов этих псевдоустройств является то, что сами они не выполняют физических операций ввода-вывода, а ведут только некоторую необходимую предварительную обработку и уже потом через драйверы физических УВВ выходят на конкретные физические устройства.

С точки зрения ОС и задач пользователя псевдоустройства полностью идентичны обычным УВВ, для них существуют одинаковые средства генерации, создаются идентичные таблицы и блоки управления.

Все качественные и количественные характеристики подсистемы ввода-вывода (уровень контроля, количество УВВ, количество элементов очереди, стеков и т.п.) определяются макрокомандой **#СВВ** в программе генерации ОС.

Подсистема связи с человеком-оператором. Подсистема связи с человеком-оператором предназначена для оперативного вмешательства человека в работу операционной системы и задач пользователя посредством ввода *команд* (директив) *оператора* со специально предназначенных для этих целей *пультов операторов* (ПО).

Подсистема связи с человеком-оператором состоит из программы работы с пультом оператора (программы ПРП) и набора модулей обработки команд оператора. Программа ПРП осуществляет ввод и первоначальный контекстный анализ команды оператора. В соответствии с результатом анализа управление передается модулю обработки распознанной команды оператора, который и осуществляет действия, предписанные данной командой.

Операционная система начинает свою работу с передачи управления специальной программе инициализации, которая сбрасывает *маски прерываний* всем устройствам, выбранным в качестве пультов операторов. Теперь вся работа ОС будет управляться командами оператора, вводимыми с этих пультов.

Число пультов оператора в системе может быть произвольным и определяется на стадии генерации ОС. Здесь же для каждого пульта генерируется *таблица команд оператора* (ТКО), которая, как правило, может быть общей для всех пультов, чтобы обеспечить их одинаковые возможности.

Таблица команд оператора устанавливает соответствие между идентификатором команды оператора (первыми двумя символами команды) и модулем обработки этой команды оператора.

Формируются таблицы команд оператора в процессе генерации: по макрокомандам **#ТКО** для них отводится место в оперативной памяти, а по макрокомандам **#КО** производится заполнение элементов таблицы соответствующей информацией.

Каждый пульт в программе генерации ОС описывается как псевдоустройство с помощью макрокоманды **#ПО**. Между псевдоустройствами ПО и устройствами, выбранными в качестве пультов оператора, устанавливается взаимно однозначное соответствие. В параметрах макрокоманды **#ПО** указывается идентификатор, которым помечена макрокоманда устройства пульта оператора. Независимо от числа пультов, сгенерированных в системе, хотя бы один из них, называемый главным, кроме выполнения взаимосвязи с человеком-оператором, должен использоваться также для вывода системных сообщений, в

частности сообщений о неработоспособности устройств, процессоров, ОЗУ и т.п.

И наконец, любой пульт оператора помимо ввода команд оператора и возможной выдачи на него системных сообщений может также применяться как обычное устройство для ввода и вывода информации по запросам из задач. Информация, вводимая с пульта оператора для таких задач, не должна включать в себя в качестве первого символа двоеточие (:), так как именно двоеточие является признаком команды оператора.

После отработки очередной команды на соответствующий пульт оператора выдается разрешающий символ * (звездочка), означающий, что система готова к приему следующей команды. Ввод новой команды до выдачи разрешающего символа может привести к непредсказуемым ситуациям.

Набор команд оператора для каждой конкретной системы определяется пользователем в зависимости от типа и назначения генерируемой системы.

Подсистема службы времени. Подсистема службы времени обеспечивает в системе:

формирование и коррекцию текущего времени суток и текущей даты;

отсчет заданных промежутков времени;

дисциплину периодического выполнения задач с заданной фазой (временем первоначального запуска) и заданным интервалом (временным периодом между двумя последовательными запусками задач на выполнение).

Подсистема службы времени включает в себя драйвер таймера — программу, которой передается управление по прерыванию от таймера и которая формирует текущее время суток и дату, и программу обработки текущего времени суток.

Кроме того, по выполняемым функциям к данной подсистеме следует отнести подпрограммы работы с тайм-аутом, формально относящиеся к вспомогательным подпрограммам подсистемы ввода-вывода, и модули обработки команд оператора **:ВРЕМЯ** и **:ДАТА**.

Другие функции подсистемы службы времени являются сервисом, предоставляемым задачам для организации выдержек времени по макрооперациям **STIME**, установки новых или корректировки прежних фаз и интервалов задачам по макрооперации **TURN** или драйверам УВВ для реализации тайм-аутов.

Подсистема управления внешней памятью. Подсистема управления внешней памятью обеспечивает выполнение операций нижнего уровня системы управления файлами — операций назначения логических номеров файлу или устройству, открытия и закрытия файла, чтения данных из файла и записи информации в файл.

Управление дисковой памятью осуществляется с помощью подпрограмм-псевдодрайверов и драйверов дисковых механизмов.

Подсистема анализа сбоев и реконфигурации. Подсистема анализа сбоев и реконфигурации призвана обеспечивать оперативную реакцию со стороны операционной системы на сбой и отказ в работе того или иного аппаратного модуля (процессора, модуля оперативной памяти, коммутатора связи, согласователя ввода-вывода и т.п.) и перестройку работы системы, если такая возможна, в целях продолжения работы при возникновении отказа.

Эта подсистема состоит из двух функциональных подсистем, первая из которых относится к контролю работы внешних устройств, каналов и т.п., а вторая – к работе процессоров и ОЗУ.

Программные средства, составляющие первую подсистему, обеспечивают резервирование УВВ, если они сгенерированы пользователем в группы резервируемых устройств, автоматическое резервирование подканалов при отказе одного из них, а также функционирование аппарата тайм-аута для тех устройств, которым он был запланирован при составлении программы генерации ОС.

Для резервирования функций ввода-вывода необходимо объединять в группы несколько устройств, одинаковых по выполняемым функциям: ГИУ – группа инициативных устройств; ГРУ – группа резервируемых устройств; ГРДУ – группа динамически распределяемых устройств.

Включение этих групповых устройств в план генерации ОС осуществляется соответствующими макрокомандами: **# ГИУ, # ГРУ, # ГРДУ.**

В этом случае групповые устройства рассматриваются как псевдоустройства, которым присваиваются порядковые номера и для которых формируются соответствующие БУУ. Теперь задача может обратиться к групповому устройству по логическому номеру, которому предварительно должен быть назначен порядковый номер псевдоустройства группы.

Для перечисления устройств, составляющих группу, метки макрокоманд, описывающих эти устройства в программе генерации ОС, указываются в соответствующих макрокомандах групповых УВВ. При этом не допускается одновременное использование одних и тех же устройств в разных группах.

Если при завершении групповой операции ввода-вывода задаче требуется логический номер того устройства в группе, которое действительно выполнило операцию, то она должна в макрооперации **EXIO** указать логический номер группы как отрицательное число и предоставить трехсловный блок завершения, в третье слово которого ОС занесет логический номер устройства, которое выполнило операцию.

Группа инициативных устройств объединяет в себе инициативные УВВ, которые, по определению, начинают операцию ввода не сразу, по запросу из задачи, а только по инициативе самого УВВ.

Для таких устройств маска прерывания сбрасывается не тогда, когда приходит запрос из задачи, а в начале работы ОС в программе инициализации. По любой инициативе от любого устройства из этой группы будет отпостирована задача, ожидающая эту инициативу. Таким образом, даже при отказе какого-нибудь УВВ из этой группы другие продолжают работу с задачей, не задерживая, в целом, работу всей системы.

При обращении к *группе резервируемых устройств* операция ввода-вывода всегда запускается на первом УВВ в группе, все остальные находятся в "хододном" резерве, если они автономно не используются в других задачах. Если первое УВВ неработоспособно или занято, то автоматический переход на другое УВВ не осуществляется, а управление возвращается к задаче с диагностикой (кодом завершения) об ошибке. Задача повторяет запрос к группе, который только теперь вызовет автоматическую передачу управления второму УВВ в группе, и т.д.

Группа динамически распределяемых устройств предусматривает такую дисциплину работы, при которой операция ввода-вывода всегда будет автоматически запускаться на любом первом же свободном и работоспособном устройстве в группе.

Примером использования группового псевдоустройства может служить следующая задача ЗГМК, используемая для заготовки символьной информации с пульта оператора на устройстве внешней памяти с кассетной магнитной лентой СМ5211.

Пусть оба накопителя этого устройства объединены в группу резервируемых устройств, порядковый номер которой присваивается логическому номеру *12.* Соответственно порядковые номера левого и правого накопителей присвоены логическим номерам *13* и *14.*

В программе генерации ОС последовательность макрокоманд, описывающих эту группу, имеет следующий вид:

```
M1      #КНМЛ 22          22 – код выборки СМ5211
M2      #КНМЛ 22,1
        #ГРУ  (M1, M2)
```

Суть выполняемой задачи заключается в том, что информация, построчно вводимая с клавиатуры пульта оператора, выводится на устройство с кассетной магнитной лентой по групповому логическому номеру *12.* В случае отказа одного из устройств в группе запрос на вывод повторяется и информация выводится на другой накопитель. По окончании вывода на пульт оператора поступает сообщение о логическом номере того накопителя в группе, который действительно выполнил операцию.

Ниже приводится текст этой программы на МАКРОЯЗЫКе с необходимыми комментариями:

```
ASMB,R,B,L
      NAM ЗГМК
НАЧ.  ExIO =L1,=L2,БЗ,БУФ1,=L9
      WAITE БЗ
```

```
        EXIO =L1,=L1,Б3,БУФ2,=L40  ВЫВОД СООБЩЕНИЯ О ВВОДЕ ИНФОРМАЦИИ
        WAITE Б3
        LDA Б3      ВВОД ИНФОРМАЦИИ С ПУЛЬТА ОПЕРАТОРА
        CPA =B40    ЕСЛИ ПРИШЕЛ ПРИЗНАК "КОНЕЦ ТЕКСТА", ТО
        JMP КОНЕЦ   НА ЗАВЕРШЕНИЕ РАБОТЫ
        LDA Б3+1
        STA ДЛИНА
ВЫВОД EXIO =L-12,=L2,Б3,БУФ2,ДЛИНА  ВЫВОД ПО ГРУППОВОМУ ЛОГИЧЕСКОМУ НОМЕРУ
        WAITE Б3
        LDA Б3
        SZA         ЕСЛИ ПРИШЕЛ ОТКАЗ, НА ПОВТОРЕНИЕ ВЫВОДА
        JMP ВЫВОД
        LDA Б3+2
        LDB =A14
        SLA   ПОДГОТОВКА ЛОГИЧЕСКОГО НОМЕРА В БУФЕРЕ ДЛЯ ПЕЧАТИ
        LDB =A13
        STB НОМЕР
        EXIO =L1,=L2,Б3,БУФ3,=L14
        WAITE Б3    ВЫВОД СООБЩЕНИЯ О ЛОГИЧЕСКОМ НОМЕРЕ
        JMP НАЧ.    ПЕРЕХОД НА ВВОД СЛЕДУЮЩЕЙ СТРОКИ
КОНЕЦ EXIO =L1,=L2,БУ3.БУФ4,=L6
        WAITE Б3    ВЫВОД СООБЩЕНИЯ О КОНЦЕ РАБОТЫ
        EXIT        ЗАВЕРШЕНИЕ ЗАДАЧИ
Б3      BSS 3
БУФ1    ASC 9,ВВЕДИТЕ ИНФОРМАЦИЮ
БУФ2    BSS 40  БУФЕР ДЛЯ ВВОДА И ВЫВОДА ИНФОРМАЦИИ
ДЛИНА BSS 1
БУФ3    ASC 13,ИНФ-ЦИЯ ВЫДАНА УВВ С ЛН=
НОМЕР NOP
БУФ4    ASC 6,КОНЕЦ РАБОТЫ
        END
        ENDM
```

Используя эту задачу, можно самостоятельно проверить реакцию системы на отказ одного из накопителей при объединении их в группу динамически распределяемых устройств (ГРДУ). Отказ одного из накопителей можно имитировать, открыв карман соответствующего накопителя с мини-кассетой.

В обычных условиях эту задачу применяют для заготовки информации на мини-кассету, предварительно установив ее на нужный файл с помощью команды опертора **:УСТАНОВ.**

Резервирование подканалов осуществляется в специальной подпрограмме работы с КПДП, которая предусматривает автоматическую передачу управления первому свободному и работоспособному подканалу в случае отказа одного из них. Работа с КПДП включаются в программу генерации по макрокоманде **# КПДП.**

Аппарат тайм-аута используется в драйверах ввода-вывода для контроля времени возникновения прерывания от данного устройства. Одновременно с запуском очередной операции ввода-вывода драйвер устанавливает временную выдержку (тайм-аут), которая заведомо больше максимального времени ожидания сигнала готовности от УВВ. Если тайм-аут исчерпался раньше, чем пришел сигнал готовности, то это указывает на неработоспособность данного УВВ, и драйвер аварийно завершает всю операцию ввода-вывода, извещая об этом задачу и человека-оператора. Включение аппарата тайм-аута определяется при написании программы генерации ОС в макрокоманде **# СВВ.**

К программам, составляющим вторую подсистему, относятся программа обработки прерываний от схем контроля, программа обработки прерываний по нарушению питания, драйвер блока контроля (БКНТ); модули макроопераций *Установить контрольную точку* и *Отменить контрольную точку.*

Программа обработки прерываний от схем контроля процессора получает управление по прерыванию от этих схем, которые контролируют работоспособность ОЗУ, самого процессора, ПЗУ, согласователя ввода-вывода, следят за правильностью использования ресурсов задачами и т.п. Любое отклонение от нормального функционирования этих аппаратных модулей фиксируется в *управляющем слое прерывания* (УСП). Читая это УСП, программа обработки прерываний анализирует, произошел сбой (или отказ) при работе ОС или при работе задач пользователя.

Если сбой или отказ зафиксирован при работе ОС, то в программе обработки прерываний с помощью микропрограмм может быть предусмотрен автоматический переход на *резервную* ОС, которая предварительно была загружена в резервный куб памяти. В этом случае выдается сообщение оператору и вся работа ОС начинается сначала (экономится время на перезапуск системы). Во всех остальных случаях, во-первых, инициализируется печать сообщения об ошибке на главном пульте оператора, во-вторых, при использовании аппарата контрольных точек осуществляется попытка перезапустить выполнение задачи с установленной контрольной точки.

Программа обработки прерываний по нарушению питания получает управление при кратковременном понижении напряжения в первичной сети, и если в течение установленного промежутка времени напряжение восстанавливается, то эта программа обеспечивает выполнение работы системы с прерванного состояния.

Драйвер блока контроля представляет собой набор подпрограмм, обеспечивающих автоматический переход системы из двухпроцессорного режима работы в однопроцессорный и обратно в случае отказа одного из процессоров. Эти подпрограммы обеспечивают контроль за работоспособностью одного из процессоров со стороны другого, обмен оперативной информацией между процессорами, минуя память, в целях синхронизации, запуск остановленного процессора со стороны работающего при восстановлении двухпроцессорной системы.

Модули супервизора *Установить контрольную точку* и *Отменить контрольную точку* выполняют обработку запросов макроопераций **SCHPT** и **DCHPT**, описанных в § 10.8. Уровень схемного контроля задается в макрокоманде **#СЗД** при генерации ОС.

Программа инициализации работы ОС. Программа инициализации начинает работу ОС, выполняется однократно самой первой из всех программ ОС и при дальнейшей работе системы может затираться.

Основными ее функциями является вывод сообщения о начале работы на главный пульт оператора и сброс масок прерывания инициативным устройствам.

Программа инициализации завершает свою работу выдачей сообщения на главный пульт оператора ***МНОГОЗАДАЧНАЯ СИСТЕМА*** и передачей управления на динамический останов системы. Начиная с этого момента, вся дальнейшая работа системы зависит от человека-оператора, который может вывести систему из динамического останова, введя необходимую ему команду оператора.

12.3. Режимы функционирования ОС АСПО

Понятие режима функционирования операционных систем АСПО и правильная ориентация пользователей на выбор того или иного режима генерируемых ОС неразрывно связаны с характером выполняемых задач. Генерируемые в рамках АСПО многозадачные и двухпроцессорные операционные системы позволяют организовать три режима работы: реального времени, пакетной обработки, разделения времени.

Режим реального времени. Этот режим является основным режимом функционирования ОС и предназначен для выполнения задач, связанных с реальным объектом управления. Для этих задач в полном объеме используются все ресурсы ВК и возможности, предоставляемые ОС. Задачи реального времени, как правило, оформляются в многозадачные загрузочные модули, которые могут содержать ОЗУ-резидентные, сегментированные и диск-резидентные задачи. Для этих задач характерно широкое использование макроопераций вызова супервизора, использование достаточно большого набора команд оператора и других возможностей, которые описаны в предыдущих главах книги. Так как задачи реального масштаба времени требуют высокой реактивности на изменение внешних событий, то загрузочные модули с этими задачами загружаются и выполняются в самых высокоприоритетных разделах (функционирующих в режиме реального времени).

Менее приоритетные разделы, не задействованные для режима реального времени, могут быть использованы для организации двух режимов: пакетной обработки и разделения времени.

Режим пакетной обработки. Для этого режима характерно использование выделенных низкоприоритетных разделов памяти, куда с помощью команд оператора с закрепленных для этой цели пультов вводятся загрузочные модули с системными обрабатывающими программами или программами пользователя, не предназначенными для работы в реальном масштабе времени. Все эти программы оформляются в однозадачные загрузочные модули, запускаются на однократное выполнение и применяют ограниченный набор макроопераций вызова супервизора (**WAITE, EXIO , EXIT, PAUSE** и **STAT**) и команд оператора.

Так как все они работают в низкоприоритетных разделах, то можем говорить о том, что они будут выполняться на фоне задач реального времени.

Дисциплина запуска задач в пакетном режиме довольно простая: а) с помощью команд оператора :**ЛН** производится назначение стандартным логическим номерам требуемых устройств либо файлов; б) по команде оператора :**СТАРТ** вводится загрузочный модуль и запускается задача с передачей ей до пяти числовых параметров (см. § 9.5).

Однако термин "пакетный" режим выбран здесь, скорее, как традиционный, чем смысловой, так как этот режим функционирования обеспечивает и диалоговую (описанную выше), и собственно пакетную обработку программ.

При *диалоговой* работе производительность вычислительного комплекса в большей мере определяется скоростью работы человека за пультом. Короткие промежутки выполнения программы пользователя и системных обрабатывающих программ чередуются интервалами, во время которых система находится в состоянии ожидания команд оператора.

В *пакетной* обработке периоды ожидания новых инструкций для работы системы сводятся к минимуму. Здесь указания пользователя по работе системы вместе с данными и программами, если это необходимо, подготавливаются заранее на носителях или в файлах на диске и обрабатываются системной обрабатывающей программой – *диспетчером пакетной обработки* (ДПОМ).

Этот диспетчер использует расширенный формат макрооперации **EXIT**, описанный в § 10.6, и требует включения в программу генерации ОС макрокоманды **# ДПО**.

Входной информацией для ДПОМ является задание, представляющее собой последовательность управляющих операторов и данных какого-либо пользователя, которая обрамляется операторами начала и конца задания. Для сохранения единого языка общения для диалоговой и пакетной обработки все управляющие операторы в задании на пакетную обработку имеют формат обычных команд оператора, используемых в диалоге общения оператора с системой. Несколько таких заданий, подлежащих последовательному выполнению без вмешательства оператора, определяют входной поток заданий, обрабатываемый ДПОМ за один сеанс работы. Каждое задание может содержать требование на трансляцию, компоновку, отладку и выполнение одной или нескольких задач пользователя. Помимо этого, в задании обычно содержатся управляющие операторы, обеспечивающие назначение устройств или же файлов логическому уровню, применяемому в задачах, вызываемых на выполнение. Вслед за требованием на запуск какой-либо задачи могут располагаться данные, обрабатываемые этой задачей.

При необходимости в системе можно зарезервировать несколько разделов памяти для организации пакетного режима функционирования. Закрепив за этими разделами отдельные пульты оператора, фактически обеспечиваем многопультовую работу, где число пультов зависит от количества разделов памяти.

Однако такая организация многопультовой работы не может в полной мере обеспечить работу пользователей, так как ограничена количеством разделов памяти, которого в общем случае недостаточно для организации такой работы. Поэтому используется третий режим функционирования ОС, предназначенный для выполнения тех же задач, что и в пакетном режиме, и называемый режимом разделения времени.

Режим разделения времени. Этот режим характерен тем, что обеспечивает работу большого числа пультов оператора при ограниченном количестве разделов оперативной памяти. Разделы оперативной памяти, специально выделенные для функционирования режима разделения времени, указываются в макрокоманде **#СРВ** в программе генерации ОС. Пульты оператора, нужные для организации многопультовой работы в режиме разделения времени, получили названия *абонентов* этого режима и включаются в программу генерации ОС макрокомандами **#ПО**.

Смысл режима разделения времени заключается в том, что все ресурсы вычислительного комплекса (время процессоров, оперативная память) распределяются ОС между *работающими загрузочными модулями* II типа равномерно. Работающим загрузочным модулем называется такой модуль, в котором есть задача, готовая к выполнению. Разделение времени достигается тем, что каждому такому модулю для пребывания в разделе памяти и выполнения отводится одинаковая порция времени (*квант*), по истечении которого работа текущего модуля приостанавливается (записывается на диск) и будет вновь продолжена после того, когда все остальные загрузочные модули побывают в памяти и отработают свои кванты. Естественно, что при таком алгоритме работы выделенные для этих целей разделы оперативной памяти предоставляются работающим загрузочным модулям согласно установленному временному графику.

Основным понятием системы с разделением времени является понятие *виртуального раздела.* Виртуальный раздел представляет собой абстрактную область памяти, которой в любой момент времени может быть сопоставлен один раздел оперативной памяти, либо файл на диске, в который было записано содержимое раздела памяти по истечении кванта времени.

С точки зрения человека-оператора виртуальный раздел практически ничем не отличается от раздела памяти, используемого в пакетном режиме: имеется возможность загружать загрузочные модули, запускать в них задачи, назначать логические номера и т.п. Виртуальные разделы идентифицируются так же, как и разделы памяти, хотя в отличие от систем реального времени номера виртуальных разделов не определяют их приоритет при выделении им основных вычислительных ресурсов.

В процессе функционирования системы разделения времени ведется *динамическое перераспределение* виртуальных разделов между абонентами. По специальной команде оператора **НАЧАЛО СЕАНСА**, по

которой терминал объявляется абонентом разделения времени, абоненту выделяется "свободный" (не используемый в это время другими абонентами) виртуальный раздел и выводится на терминал его номер, идентифицирующий этот виртуальный раздел. Все дальнейшие команды оператора, управляющие работой задач, будут относиться к выделенному виртуальному разделу. С этого момента команда оператора **УСТАНОВИТЬ НОМЕР РАЗДЕЛА** становится недопустимой, но запоминается номер раздела памяти, за которым был закреплен данный пульт до начала сеанса. После освобождения виртуального раздела этот пульт восстановит свое закрепление за разделом памяти, которое имело место до сеанса. Виртуальный раздел освобождается по команде оператора **КОНЕЦ СЕАНСА**, завершающей использование терминала в качестве режима разделения времени.

В процессе функционирования системы разделения времени осуществляется *учет расходования* основных ресурсов вычислительного комплекса (для каждого абонента отдельно). Подсчитывается общее время сеанса и время работы какого-либо раздела памяти.

Если в систему при генерации включена соответствующая возможность, подсчитывается также время, израсходованное процессорами на выполнение задач, происходящим по командам абонента. При этом учитывается только время работы процессоров в задаче и не учитывается время супервизорных функций, запрошенных задачами в процессе выполнения. Накопленные данные сообщаются абоненту в конце сеанса.

Такой режим применения вычислительного комплекса, с одной стороны, удобен для пользователя, который работает в непосредственной связи с ЭВМ в диалоге с ней, с другой стороны, относительно большое число терминалов позволяет в достаточной степени загрузить вычислительный комплекс, существенно повысив эффективность его работы.

12.4. Макрокоманды генерации ОС

Программа генерации ОС по форме представляет собой обычную программу на МАКРОЯЗЫКе и поэтому должна оформляться так, как было показано в гл. 8. В эту программу входят макрокоманды генерации ОС, макроопределения которых собраны в библиотеке МБФ (см. § 9.1). Ниже приводятся описания макрокоманд, обязательных для программ генерации ОС, а также описания наиболее часто используемых макрокоманд и их форматов.

Макрокоманды генерации ОС описаны преимущественно в том порядке, в каком рекомендуется включать каждую из них в программу генерации. Заметим, что в составе АСПО есть средства, помогающие пользователю в необходимых случаях расширять библиотеку стандартных макроопределений генерации ОС.

Рамки данной книги не позволяют дать полную информацию обо всех параметрах, задаваемых в макрокомандах генерации ОС. Поэтому

для простых макрокоманд описания этих параметров будут опущены. Более подробную информацию по всем нюансам использования макрокоманд генерации ОС, как уже отмечалось, можно получить из документации сопровождения программного обеспечения ВК СМ-2М.

#ОС – определить тип генерируемой ОС. Эта макрокоманда обязана быть первой в программе генерации ОС, определяет тип создаваемой ОС и дисциплину работы диспетчера задач (см. § 12.2). Формат макрокоманды

#ОС *тип*

Здесь *тип* = **О** – однозадачная ОС; = **М** – многозадачная ОС; = **З** – двухпроцессорная с первой дисциплиной работы диспетчера задач (по задачам); = **Р** – двухпроцессорная со второй дисциплиной работы диспетчера задач (по разделам).

#РОП – определить раздел оперативной памяти. Это обязательная макрокоманда, предназначенная для разбиения оперативной памяти ВК на разделы. В программе генерации могут быть одна или несколько макрокоманд **#РОП**, которые определяют соответственно одно- или многораздельную работу. Первая макрокоманда **#РОП** резервирует раздел для работы ОС и должна задаваться без параметров. Формат команды

#РОП *кстр* [,*надр*]

Здесь *кстр* – число, определяющее объем раздела в блоках (от 3 до 64); *надр* – число, определяющее номер первого блока раздела (от 0 до 255). Если параметр опущен, пространство под текущий раздел выделяется вслед за предыдущим разделом.

#СРВ – включить в ОС возможности работы в режиме разделения времени. По этой макрокоманде в систему включаются программные средства управления задачами в режиме разделения времени. Формат макрокоманды

#СРВ *снр*, [*кпо*], [*пндс*], [*млн*] [,*стат*]

Здесь *снр* – подсписок из двух элементов, указывающих номера первого и последнего разделов памяти, которые будут использоваться при работе в режиме разделения времени; *кпо* – максимальное число пультов-абонентов, работающих в режиме разделения времени; *пндс* – порядковый номер дискового устройства, на котором будут создаваться файлы для записи состояния разделов памяти в процессе замены задач в режиме разделения времени. Если параметр опущен, файлы создаются на системных дисках; *млн* – максимально возможный логический номер в виртуальных разделах; *стат* – если опущен, то ОС подсчитывает только общее время сеанса и время использования памяти. Если в качестве параметра указана буква **С**, подсчитывается и время, израсходованное процессорами на решение задач.

В качестве разделов памяти в режиме разделения времени будут использоваться разделы из диапазона, указанного в подсписке *снр.* Для эффективного выполнения режима разделения времени необходимо указать не менее трех разделов, которые должны иметь одинаковый размер и не должны применяться для работы в режиме реального времени. Размеры этих разделов не следует выбирать с большим запасом, так как увеличение размера разделов памяти увеличивает размер области диска, которая отводится под организацию файлов для копирования разделов, и увеличивается время смены задач, что существенно скажется на эффективности использования режима разделения времени. Кроме того, для обеспечения той же эффективности в качестве диска для создания файлов виртуальных разделов (*диска свопинга*) целесообразно указывать дисковое устройство, к которому достаточно редко обращаются из задач. В качестве устройства диска свопинга целесообразно применять накопители с минимальным временем позиционирования, например диски с фиксированными головками.

Макрокоманда **#СРВ** должна задаваться в программе генерации 1 раз и располагаться после всех макрокоманд **#РОП** и перед макрокомандой **#СЗД.**

#СЗД – определить параметры супервизора задач. Это обязательная макрокоманда, которая формирует таблицу макроопераций вызова супервизора и определяет уровни схемного и программного контроля в генерируемой ОС. Формат команды

#СЗД [*мквс*],[*ск*] [,*пк*]

Здесь *мквс* – число, определяющее максимальный код вызова супервизора (если параметр опущен, его значение приравнивается к 11); *ск* – уровень схемного контроля, который задается символами **Б** или **БА**; *пк* – задание контроля входных параметров.

Если уровень схемного контроля определен символом **Б**, то в ОС не включается подсистема анализа сбоев и реконфигурации. При задании символов **БА** в ОС включается подсистема анализа сбоев и реконфигурации, выполняющая функции резервирования памяти и процессоров, а также реализующая аппарат контрольных точек (макрооперации **SCHPT** и **DCHPT**).

Если задан супервизор с контролем входных параметров (указан параметр *пк* = **К**), при каждом вызове супервизора осуществляется контроль вызывающей последовательности на корректность.

#ПОВС – включить в ОС модули обработки макроопераций вызова супервизора. По этой макрокоманде в систему включается группа модулей, ответственных за выполнение макроопераций.

Группа определяется начальным (*нквс*) и конечным (*кквс*) кодами вызова супервизора. Формат макрокоманды

#ПОВС*нквс* [,*кквс*]

Так как в каждую ОС обязательно включаются макрооперации **WAITE** и **EXIO**, поэтому они не должны указываться в макрокомандах **#ПОВС**. Если в ОС предусматривается работа с системными обрабатывающими программами, то обязательно включение макроопераций с кодами вызова от 2 до 4.

В программе генерации должно быть столько макрокоманд **#ПОВС**, сколько групп или же отдельных модулей обработки макроопераций включается в систему.

Включение в ОС нестандартных макроопераций, разработанных самим пользователем, осуществляется макрокомандой **#МОДУЛЬ**, имеющей следующий формат:

#МОДУЛЬ *квс,твх*

Здесь *квс* – код вызова супервизора новой макрооперации; *твх* – метка входной точки разработанного модуля ОС. Этот модуль должен быть включен в состав библиотеки перемещаемых модулей ОС.

#ДПО – включить в систему возможности работы с расширенным вариантом макрооперации **EXIT** (см. § 10.6). Формат макрокоманды

#ДПО

Эта макрокоманда используется без параметров и может быть только одна в программе генерации многозадачной или двухпроцессорной ОС с несколькими разделами памяти. По этой макрокоманде применяют диспетчер пакетной обработки (ДПОМ) для осуществления подготовки программ в пакетном режиме.

#СВВ – определить параметры супервизора ввода-вывода. Формат команды

#СВВ *кувв*, [*мкв*], [*кэсп*], [*кэо*] [,**Т**]

Здесь *кувв* – число УВВ, включаемых в генерируемую ОС (это число определяет размер таблицы порядковых номеров устройств и должно быть не менее общего числа макрокоманд, описывающих конкретные физические УВВ и псевдоустройства); *мкв* – восьмеричное число (не превышающее 77), указывающее максимальный из кодов выборки устройств, включаемых в систему; *кэсп* – число элементов стека прерываний для одного процессора (при опущенном параметре в системе отводится по два элемента на каждый процессор); *кэо* – число элементов, резервируемых для формирования очередей к устройствам ввода-вывода; **Т** – если этот символ указан в качестве последнего параметра, в систему включаются средства контроля УВВ по времени (таймаут УВВ).

Указанная макрокоманда должна быть одна в ОС. Вслед за ней располагают макрокоманды, описывающие конкретные УВВ. В первую очередь должны быть описаны системные УВВ: таймер (**#ТАЙМЕР**) и КПДП (**#КПДП**). После них располагаются макрокоманды включения

в систему всех остальных УВВ. В соответствии с порядком включения им присваиваются порядковые номера: УВВ, описанному первым после КПДП, присваивается номер 1, вторым – 2 и т.д. Если УВВ подключается через *разветвитель интерфейса мультиплексный* (РИМ), то в макрокомандах генерации ОС необходимо указывать код выборки (КВ) устройства в виде

XXYY

где XX – меньший код выборки РИМ на СВВ; YY – адрес устройства на РИМ (0–76).

Например, макрокоманда включения в систему УБП, подключенного через РИМ, может выглядеть следующим образом: **#УБП** *5201.*

#ТАЙМЕР –включить в систему программы службы времени. Это обязательная макрокоманда, которая включается после макрокоманды **#СВВ.** Формат макрокоманды

#ТАЙМЕР *кв,дт,*[**,И**]

Здесь *кв* – код выборки таймера; *дт* – дискретность таймера, требуемая для работы ОС и задач пользователя: *дт* = **1** – 1 мс; **2** – 10 мс; **3** – 100 мс; **4** – 1 с; **И** – если этот символ указан в качестве третьего параметра, то в ОС включается вся подсистема службы времени. Если параметр опущен, то включаются только программы учета текущего времени дня.

#КПДП – включить в ОС программы работы с КПДП. Это обязательная макрокоманда, которая указывается после макрокоманды **#ТАЙМЕР.** Формат макрокоманды

#КПДП*12* [**,***чпк*]

Здесь *12* – код выборки КПДП; *чпк* – число подканалов в системе (0–7). Если параметр опущен, подразумевается, что в системе только четыре подканала (один канал).

При наличии в системе хотя бы одного подканала драйверы всех УВВ, при работе с которыми целесообразно использовать КПДП, автоматически настраиваются на работу с ним.

#ТРУ, #КОММ – макрокоманды, изменяющие тип распределяемости УВВ. Распределение УВВ как ресурсов системы производится по следующей стратегии: каждому устройству присваивается один из трех типов распределяемости – общедоступный, распределяемый и нераспределяемый.

Общедоступный тип означает, что данное УВВ может быть назначено любому логическому номеру в любом разделе.

Распределяемый тип указывает, что УВВ в любой момент времени может быть назначено логическим номерам только в одном (любом) разделе; если устройство назначено хотя бы одному логическому номеру в каком-либо разделе, то оно может быть назначено любым другим

логическим номерам в этом же разделе и не может быть назначено логическому номеру ни в одном другом разделе. Если же необходимо назначить УВВ логическому номеру в другом разделе, предварительно должны быть отменены все назначения этого устройства другим логическим номерам.

Нераспределяемый тип означает, что произведенное в программе генерации ОС назначение не может быть динамически изменено в процессе работы ОС.

С помощью макрокоманды **#ТРУ** можно задать тип распределяемости устройству, макрокоманда описания которого стоит в программе генерации ОС сразу же после макрокоманды **#ТРУ**. Формат макрокоманды

#ТРУ *тр*

Здесь *тр* – тип распределяемости; *тр* = **О** – общедоступный; **Р** – распределяемый; = **Н** – нераспределяемый.

По макрокоманде **#КОММ** все последующие УВВ объявляются общедоступными, за исключением тех, тип распределяемости которых указан макрокомандой **#ТРУ**. Формат макрокоманды

#КОММ

Следует учесть, что макрокоманды **#ТРУ** и **#КОММ**, описываемые среди макрокоманд включения УВВ, не влияют на присвоение порядковых номеров этим устройствам.

Далее приведем наименования всех макрокоманд, описывающих УВВ, и для части макрокоманд укажем их форматы.

#КНМЛ – включить в ОС накопитель на кассетной магнитной ленте. Эта макрокоманда обеспечивает включение драйвера одного накопителя устройства внешней памяти на кассетной магнитной ленте СМ 5211.

В системе может быть несколько СМ 5211, в каждой из которых два накопителя. Макрокоманды включения обоих накопителей одной СМ 5211 должны располагаться подряд в программе генерации, и им будут присвоены два последовательных порядковых номера.

#УБП – включить в ОС устройство быстрой печати. Эта макрокоманда обеспечивает включение драйвера устройства быстрой печати (УБП).

#УПП – включить в ОС устройство последовательной печати. Эта макрокоманда обеспечивает включение драйвера устройства последовательной печати (УПП).

#УПЗ – включить в ОС устройство печати знакосинтезирующее. По этой макрокоманде включается устройство печати знакосинтезирующее (УПЗ).

#МДК – включить в ОС один накопитель дискового механизма комбинированного типа. Эта макрокоманда обеспечивает включение в ОС драйвера накопителя устройства внешней памяти на магнитных дисках (ИЗОТ-1370). Фиксированный и съемный диски одного механизма

ИЗОТ-1370 рассматриваются как отдельные дисковые накопители. Формат макрокоманды

[*метка*] #**МДК** *кв*,[*нн*],[*нсмд*],[*ксд*] [,*вз*]

Здесь *метка* – метка макрокоманды, которая указывается тогда, когда данный накопитель будет использоваться в качестве системного в макрокоманде СУФ; *нн* – номер накопителя, который может принимать значения от 0 до 7; *нсмд* – номер сектора, содержащего метку диска; *ксд* – число, определяющее, сколько 128-словных секторов находится на одной физической дорожке; *вз* – любое число, задающее включение программной защиты диска. Если параметр опущен, защита не включается.

Все накопители одного устройства управления должны включаться в систему подряд расположенными макрокомандами #**МДК**, и им будут присвоены соответствующие порядковые номера.

#**ГИУ**, #**ГРУ**, #**ГРДУ** – включить в ОС групповые псевдоустройства. Эти макрокоманды используются соответственно для включения в систему группы инициативных устройств, группы резервируемых УВВ и группы динамически распределяемых устройств.

Формат макрокоманд

[*метка*] #**ГИУ** (*имя1*, . . . , *имяN*)
[*метка*] #**ГРУ** (*имя1*, . . . , *имяN*)
[*метка*] #**ГРДУ** (*имя1*, . . . ,*имяN*)

Здесь *имя1*, . . . ,имя*N* – одна или несколько меток макрокоманд описания устройств, включаемых в соответствующие группы. При этом не допускается использование одних и тех же устройств в разных группах.

#**МАЦП** – включить в ОС подсистемы ввода аналоговой информации на базе модуля аналого-цифрового преобразователя А611-19. По этой макрокоманде в систему включаются программы для работы с подсистемой ввода аналоговой информации на базе одного или нескольких коммутаторов бесконтактных (КБ) А612-11, связанных с модулем аналого-цифрового преобразования (МАЦП) А611-19 по аналоговой шине сопряжения 2 К.

#**ВВДИ** – включить в ОС подсистемы ввода-вывода дискретной информации. Эта макрокоманда обеспечивает работу программ с группой подключенных на соседние коды выборки сопряжения 2 К модулей ввода инициативных сигналов (МВВИС) А622-8, ввода импульсных сигналов (МВВИМС) А622-3, ввода число-импульсных сигналов (МВВЧИС) А623-2 и А623-3, модулей кодового управления бесконтактных (МКУБ) А641-9, преобразователей код–ток А631-6.

#**НМЛ** – включить в ОС накопитель на магнитной ленте. Эта макрокоманда обеспечивает включение драйвера одного накопителя устройства внешней памяти на магнитной ленте (УВПМЛ) А311-7 (ИЗОТ-5003), А311-12 (ИЗОТ-5006) либо А311-3 (ЕС-5012).

#ДММ включить в ОС дисплейный модуль **VIDEOTON**-340. Эта макрокоманда включает в ОС драйвер дисплейного модуля **VIDEOTON-340** (ДМ-340).

#ДМС включить в ОС дисплейный модуль ДМ-2000. Эта макрокоманда может быть использована для включения в систему драйвера дисплейного модуля ДМ-2000. Несмотря на то что этот дисплей не входит в номенклатуру периферийного оборудования ВК СМ-2М, его довольно часто используют при работе с различными редакторами текста, входящими в состав программного обеспечения ВК СМ-2М. Суть использования дисплея как средства редактирования заключается в следующем. Экран дисплея делится программно (а для ВТА 2000-31 и других это заложено в аппаратуре) на две части: область редактирования и служебную область (*роллинг*). В этом случае дисплей рассматривается как совмещенное устройство для прямого доступа в любой участок области редактирования экрана и как пульт оператора, использующего область роллинга. *Область редактирования* занимает верхние строки экрана и предназначена для просмотров кадров с текстами и редактирования их с помощью курсора, управляемого с клавиатуры дисплея в автономном режиме. *Область роллинга* (не менее трех строк) располагается под областью редактирования и используется для ввода директив оператора и вывода системных сообщений.

Формат макрокоманды, включающей в систему дисплейный модуль ДМ-2000,

[*метка*] **#ДМС** *тп*,*кв*,[*ж*] [,*гр*]

Здесь *тп* – метка макрокоманды #ТПТК, формирующей таблицу кодировок функциональных (технологических) клавиш ДМ-2000; *ж* – параметр, который, если принимает любое ненулевое значение, указывает на работу с жетоном (здесь необходимо знакомство с документацией сопровождения ВК СМ-2М); *гр* – число, указывающее на номер строки на экране, после которой располагается область роллинга. Если этот параметр опущен, то предполагается, что он равен нулю. Последнее означает, что весь экран отдается под пульт оператора.

#ВТА1 – включить в ОС ВТА2000(500)-11, ВТА2000(500)-15. Эта макрокоманда используется для включения в систему драйверов видеотерминалов алфавитно-цифровых ВТА2000(500)-11 или ВТА2000(500)-15. Формат макрокоманды

[*метка*] **#ВТА1***тпфк*,*кв*,[*нпр*],[*тип*],[*кт*],[*гр*],[*реж*] [,*ж*]

Здесь *тпфк* – метка макрокоманды **#ТПФК**, по которой формируется таблица кодирования функциональных (технологических) клавиш; *нпр* – номер программы обработки: *нпр*-**1** для ВТА2000(500)-11, *нпр*-**3** для ВТА2000(500)-15; *кт* – режим использования ВТА. Если этот параметр опущен, ВТА используется в консольном варианте (тумблер "консоль/терминал" в положении "консоль"). Если этот параметр ра-

вен коду выборки ВТА, то устанавливается терминальный вариант (тумблер в положении "терминал"); *гр* – номер строки на экране от 0 до 23 для ВТА2000 (или от 0 до 15 для ВТА500), после которой устанавливается область роллинга (см. предыдущую макрокоманду); *реж* – режим кодирования управляющих символов в буфере пользователя; *реж-д* – кодирование соответствует кодированию управляющих символов дисплейного модуля ДМ-2000, если параметр опущен, то кодирование управляющих символов соответствует кодированию управляющих символов ВТА; *ж* – если параметр опущен, предполагается отсутствие жетона.

Если в макрокоманде # **ДМС** или # **ВТА1** применяется идентификатор *метка*, то предполагается, что любой из этих дисплеев может использоваться в качестве пульта оператора. Тогда эта метка нужна в макрокоманде # **ПО**.

ПО – определить пульт оператора. Эта макрокоманда включает в систему псевдоустройство – пульт оператора. Формат макрокоманды

ПО *имяув*,*табл*,[*пнуз*] [,*нр*]

Здесь *имяув* – метка макрокоманды, описывающей устройство, выбранное в качестве пульта оператора; *табл* – метка макрокоманды # ТКО, определяющей команды, которые могут быть введены и обработаны с данного пульта; *пнуз* – порядковый номер устройства ввода загрузочного модуля по команде оператора :**ВВОД** для данного пульта; *нр* – число, определяющее номер раздела, за которым закрепляется данный пульт. Если номер раздела определен как положительное число или нуль, для пульта включается команда оператора *:нр*, позволяющая изменять сферу действия пульта во время работы системы. Если же номер раздела указан со знаком минус, пульт закрепляется за разделом и оператору не дается возможности изменить его назначение (команда *:нр* не включается). Если этот параметр задан отрицательным числом, по абсолютной величине большим максимального номера раздела, то пульт используется только в качестве абонента системы разделения времени.

В системе может быть один или несколько пультов оператора. Каждый пульт должен описываться в программе генерации своей макрокомандой # **ПО**. Общесистемные сообщения всегда выдаются на системный пульт, описанный первым.

Таблица команд оператора может быть одна и та же для нескольких пультов или же своя для каждого пульта.

Псевдоустройству # **ПО** присваивается свой порядковый номер, и оно может, так же как и устройство, описанное макрокомандой # **ГИУ**, выполнять инициативные операции, т.е. помимо ввода команд оператора осуществлять ввод символьных записей по запросам от задач. В последнем случае информация, вводимая с пульта, не должна содержать в первой позиции символа двоеточие (:).

 # МЗОИ – включить в ОС псевдоустройство межзадачного обмена. Эта макрокоманда включает в систему программы, обеспечивающие обмен информацией между задачами, работающими в одном или различных разделах оперативной памяти (см. § 10.4).

 # СУФ – включить систему управления файлами. Это обязательная макрокоманда, которая включает в генерируемую ОС программы подсистемы управления внешней памятью, обеспечивающие физический доступ к дисковым файлам и являющиеся основой всей СУФ. Формат макрокоманды

#СУФ (*метка1, . . . ,меткаN*)

Здесь *метка1, . . . ,меткаN* – одна или несколько меток макрокоманд, описывающих дисковые накопители, предназначенные для использования в качестве системных дисков.

 # ТПТК, # ТПФК – макрокоманды задания таблиц кодировок технологических клавиш. По этим макрокомандам формируются таблицы кодировок технологических клавиш соответственно для ДМ-2000 и ВТА2000(500)-11 или ВТА2000(500)-15. Смысл кодировок этих клавиш заключается в том, что любой технологической клавише выбранного дисплея можно на стадии генерации ОС присвоить любое символьное выражение, но одинаковой длины для всех клавиш. Тогда нажатие какой-нибудь технологической клавиши вызовет ввод с дисплея соответствующего символьного выражения (текста). При этом ввод может осуществляться в любую область экрана с места расположения курсора. Этим свойством можно воспользоваться для присвоения технологическим клавишам наиболее часто используемых директив оператора при работе с некоторыми системными обрабатывающими программами, основанными на применении диалога с человеком-оператором. К таким программам относятся различные дисплейные текстовые редакторы (IET, SLIDE, ОЛИМП), составляющие основу средств редактирования и документирования (см. гл. 14). Формат макрокоманд

метка **# ТПТК** *длина*
метка **# ТПФК** *длина*

Здесь *метка* – обязательная метка, на которую ссылаются соответствующие макрокоманды **# ДМС** и **# ВТА1**; *длина* – постоянная длина текста, присваиваемого технологическим клавишам.

 # ПТК, # ПФК – макрокоманды назначения технологических клавиш. Эти макрокоманды применяют для текстовых выражений номерам технологических клавиш соответстуующих дисплеев ДМ-2000 и ВТА2000(500)-11 или ВТА2000(500)-15. Формат макрокоманд

#ПТК *нк,текст*
#ПФК *нк,текст*

 # ТКО – определить таблицу команд оператора. Эта макрокоманда резервирует пространство для таблицы команд оператора, обрабатывае-

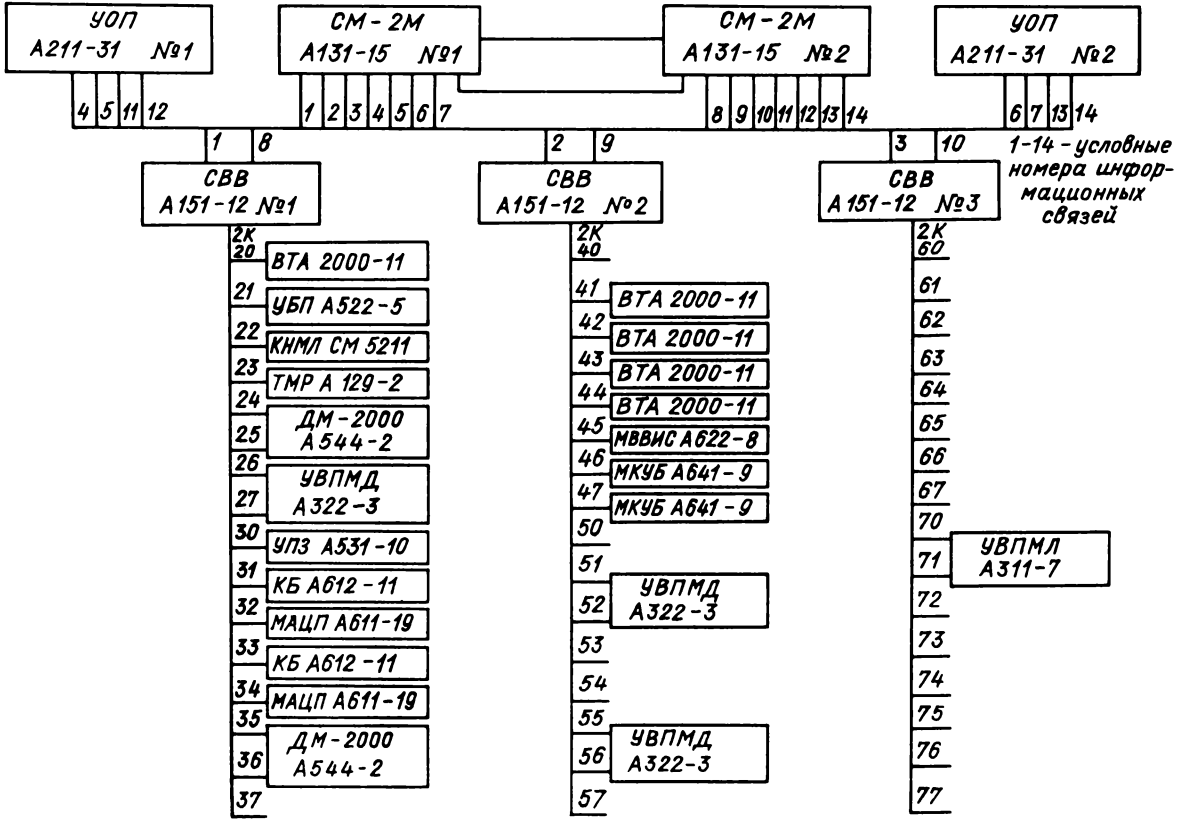


Рис. 12.2. Структурная схема УВКС для технологического процесса в примере § 9.2

мых одним или несколькими разными пультами. За этой макрокомандой должны следовать макрокоманды **#КО**, определяющие конкретные команды оператора, включаемые для соответствующего пульта оператора.

#КО – включить команду оператора. Эта макрокоманда включает в систему программу обработки соответствующей команды оператора (см. § 12.6).

#ТЛН – определить таблицу логических номеров УВВ. Эта макрокоманда резервирует память под таблицу логических номеров для одного или нескольких разделов оперативной памяти. Число команд **#ТЛН** определяется числом разделов, для которых формируются индивидуальные ТЛН. Каждая макрокоманда **#ТЛН** может сопровождаться группой макрокоманд **#ЛН**, осуществляющих статическое назначение логическим номерам физических УВВ.

#ЛН – назначить логическому номеру УВВ. Эта макрокоманда обеспечивает назначение логическим номерам устройств ввода-вывода на этапе генерации ОС. Безусловно, это назначение может быть динамически изменено в процессе работы командами оператора **:ЛН** или непосредственно в задачах пользователя. Как правило, эти макрокоманды применяются для статического назначения УВВ стандартным логическим номерам, используемым в системных обрабатывающих программах.

#КОС – конец программы генерации ОС. Эта макрокоманда завершает программу генерации ОС. После этой макрокоманды должен следовать оператор **ENDM**, указывающий на завершение макрообработки.

12.5. Пример программы генерации ОС

Ниже приводится пример программы генерации ОС, предназначенной для выполнения задач в режиме реального времени, а также в режимах пакетной обработки и разделения времени. При этом в программу генерации ОС включены все средства, необходимые для функционирования задач простейшего технологического процесса из примера, описанного в § 9.2. Структурная схема соответствующего УВКС приведена на рис. 12.2. Необходимые пояснения к программе генерации ОС даны в виде операторов-комментариев внутри самой программы, написанной на МАКРОЯЗЫКе.

```
ASM8,K,B,L,T
   #ОС 3    ДВУХПРОЦЕССОРНАЯ ОС С РАСПРЕДЕЛЕНИЕМ ПРОЦЕССОРОВ ПО ЗАДАЧАМ
   #РОП 64  РЕЗЕРВИРОВАНИЕ НУЛЕВОГО РАЗДЕЛА ДЛЯ ОС
   #РОП 64  РЕЗЕРВИРОВАНИЕ ПЕРВОГО РАЗДЕЛА ДЛЯ ЗАДАЧ РЕАЛЬНОГО ВРЕМЕНИ
   #РОП 42  РЕЗЕРВИРОВАНИЕ ВТОРОГО,
   #РОП 42  ТРЕТЬЕГО
   #РОП 42  И ЧЕТВЕРТОГО РАЗДЕЛОВ ДЛЯ РЕЖИМА РАЗДЕЛЕНИЯ ВРЕМЕНИ
   #СРВ (2,4),,29,30  ТАК КАК ВТОРОЙ ПАРАМЕТР В МАКРОКОМАНДЕ #СРВ
* ОПУЩЕН, ТО МАКСИМАЛЬНОЕ КОЛИЧЕСТВО ПУЛЬТОВ ДЛЯ РЕЖИМА РАЗДЕЛЕНИЯ
* ВРЕМЕНИ РАВНО КОЛИЧЕСТВУ ДЕЙСТВИТЕЛЬНО СУЩЕСТВУЮЩИХ ПУЛЬТОВ,
* СОЗДАВАЕМЫХ МАКРОКОМАНДАМИ #ПО
   #СЗД 14,6А,К  В ГЕНЕРИРУЕМУЮ ОС ВКЛЮЧАЕТСЯ ВСЯ ПОДСИСТЕМА
```

```
* АНАЛИЗА СБОЕВ И РЕКОНФИГУРАЦИИ
   #ПОВС 2,9  В ГЕНЕРИРУЕМУЮ ОС ВКЛЮЧАЮТСЯ ВСЕ МАКРООПЕРАЦИИ ВЫЗОВА СУПЕР-
   #ПОВС 13,14 ВИЗОРА, ЗА ИСКЛЮЧЕНИЕМ МАКРООПЕРАЦИЙ SCHPT, DCHPT, WAITM
   #ДПО    ВКЛЮЧЕНИЕ РАСШИРЕННОГО ВАРИАНТА МАКРООПЕРАЦИИ EXIT ДЛЯ
* ОБЕСПЕЧЕНИЯ РАБОТЫ ДИСПЕТЧЕРА ПАКЕТНОЙ ОБРАБОТКИ
   #СВВ 90,,7,12,Т  ЗДЕСЬ МАКСИМАЛЬНОЕ КОЛИЧЕСТВО УВВ РАВНО 90,
* МАКСИМАЛЬНЫЙ КОД ВЫБОРКИ РАВЕН 77, КОЛИЧЕСТВО ЭЛЕМЕНТОВ СТЕКОВ ДЛЯ
* КАЖДОГО ПРОЦЕССОРА РАВНО 7, КОЛИЧЕСТВО ЭЛЕМЕНТОВ ОЧЕРЕДИ РАВНО 12,
* ВКЛЮЧЕНЫ СРЕДСТВА КОНТРОЛЯ УВВ ПО ТАЙМЕРУ
   #ТАЙМЕР 23,3,И  ЗДЕСЬ СИМВОЛ "И" УКАЗЫВАЕТ НА ВКЛЮЧЕНИЕ ПОЛНОЙ
* ПОДСИСТЕМЫ СЛУЖБЫ ВРЕМЕНИ
   #КПДП 12,6   ВКЛЮЧЕНИЕ ШЕСТИ ПОДКАНАЛОВ КПДП
   #КОММ        ВСЕ ПОСЛЕДУЮЩИЕ УВВ ОБЪЯВЛЯЮТСЯ ОБЩЕДОСТУПНЫМИ И
* ПЕРЕНАЗНАЧАЕМЫМИ
М1   #КНМЛ 22          ВКЛЮЧЕНИЕ В ГРУППУ ДИНАМИЧЕСКИ РАСПРЕДЕЛЯЕМЫХ
М2   #КНМЛ 22,1        УСТРОЙСТВ ДВУХ НАКОПИТЕЛЕЙ УСТРОЙСТВА ВНЕШНЕЙ
     #ГРДУ (М1,М2)     ПАМЯТИ НА КАССЕТНОЙ МАГНИТНОЙ ЛЕНТЕ СМ5211
П1   #ВТА1 ,20,1,,,,Д ВКЛЮЧЕНИЕ В ОС В КАЧЕСТВЕ ГЛАВНОГО ПУЛЬТА ОПЕРА-
     #ПО П1,Т1,1       ТОРА ВИДЕОТЕРМИНАЛА ВТА2000-11 ДЛЯ РАБОТЫ В
* НУЛЕВОМ И ПЕРВОМ РАЗДЕЛАХ
     #УЬП 21           ВКЛЮЧИТЬ УСТРОЙСТВА ПЕЧАТИ
     #УПЗ Т,30
Д1   #МДК 26,1         ВКЛЮЧЕНИЕ ЧЕТЫРЕХ НАКОПИТЕЛЕЙ НА МАГНИТНЫХ ДИСКАХ
Д2   #МДК 26,,,,А      КОМБИНИРОВАННОГО ТИПА А322-3 С ОДНИМ УСТРОЙСТВОМ
     #МДК 26,3         УПРАВЛЕНИЯ. ЗДЕСЬ ДЛЯ ВТОРОГО НАКОПИТЕЛЯ УСТАНАВ-
     #МДК 26,2         ЛИВАЕТСЯ ПРОГРАММНАЯ ЗАЩИТА
С1   #МДК 51,1         ВКЛЮЧЕНИЕ ЧЕТЫРЕХ НАКОПИТЕЛЕЙ С ОДНИМ УСТРОЙСТВОМ
С2   #МДК 51,,,,И      УПРАВЛЕНИЯ, ПОДКЛЮЧЕННЫМ К КОДАМ ВЫБОРКИ ВТОРОГО СВВ
     #МДК 51,3
     #МДК 51,2
П2    #ДМС ТП,24,,18    ВКЛЮЧЕНИЕ ДВУХ ПУЛЬТОВ ОПЕРАТОРА НА БАЗЕ УСТРОЙСТВ
     #ПО П2,Т2,1,-8     ДМ-2000, ПРЕДНАЗНАЧЕННЫХ ДЛЯ РАБОТЫ В РЕЖИМЕ
П3    #ДМС ТП,35,,18    РАЗДЕЛЕНИЯ ВРЕМЕНИ
     #ПО П3,Т2,1,-8
П4    #ВТА1 Т0,41,1,,,,Д  ВКЛЮЧЕНИЕ ЧЕТЫРЕХ ПУЛЬТОВ ОПЕРАТОРА НА БАЗЕ
     #ПО П4,Т2,1,-8       ВТА2000-11, ПРЕДНАЗНАЧЕННЫХ ДЛЯ РАБОТЫ В
П5    #ВТА1 Т0,42,1,,,,Д  РЕЖИМЕ РАЗДЕЛЕНИЯ ВРЕМЕНИ
     #ПО П5,Т2,1,-8
П6    #ВТА1 Т0,43,1,,,,Д
     #ПО П6,Т2,1,-8
П7    #ВТА1 Т0,44,1,,,,Д
     #ПО П7,Т2,1,-8
     #МДК 45,1     ВКЛЮЧЕНИЕ ОТДЕЛЬНОГО НАКОПИТЕЛЯ А322-3 В КАЧЕСТВЕ
     #МДК 45       ДИСКА СВОПИНГА В РЕЖИМЕ РАЗДЕЛЕНИЯ ВРЕМЕНИ ДЛЯ
* МАКРОКОМАНДЫ #СВВ
    #НМЛ 70      ВКЛЮЧЕНИЕ ДВУХ НАКОПИТЕЛЕЙ НА МАГНИТНОЙ ЛЕНТЕ
    #НМЛ 70,1
    #МАЦП 32,31,1,1,1  ВКЛЮЧЕНИЕ МОДУЛЯ АНАЛОГО-ЦИФРОВОГО ПРЕОБРАЗО-
* ВАНИЯ А611-19 С ОДНИМ КОММУТАТОРОМ БЕСКОНТАКТНЫМ А612-11
* (КОД ВЫБОРКИ РАВЕН 31) ДЛЯ СЪЕМА ТЕМПЕРАТУРЫ, ТИП КОММУТАЦИИ
* ОДНОПОЛЮСНЫЙ, ДИАПАЗОН 0-10Б
    #МАЦП 34,33,1,1,1    ВКЛЮЧЕНИЕ МАЦП С ОДНИМ КБ (КОД ВЫБОРКИ
* РАВЕН 33) ДЛЯ СЪЕМА ДАВЛЕНИЯ В ЗАДАЧЕ D
ВВ   #ВВДИ 45   ВКЛЮЧЕНИЕ МВВИС А622-8 ДЛЯ ЗАДАЧИ Е
    #ГИУ ВВ
    #ВВДИ 46   ВКЛЮЧЕНИЕ МОДУЛЕЙ КОДОВОГО УПРАВЛЕНИЯ БЕСКОНТАКТНЫХ
    #ВВДИ 47   А641-9 ДЛЯ УПРАВЛЕНИЯ КЛАПАНАМИ В ЗАДАЧАХ А, В.
* ПРИ ЭТОМ ПЕРВАЯ МАКРОКОМАНДА #ВВДИ ОПИСЫВАЕТ МКУБ, ИСПОЛЬЗУЕМЫЙ
* ДЛЯ УПРАВЛЕНИЯ КЛ1 В ЗАДАЧЕ А, ВТОРАЯ - ДЛЯ УПРАВЛЕНИЯ КЛ2 В ЗАДАЧЕ В
    #МЗОИ   ВКЛЮЧЕНИЕ ДВУХ ПСЕВДОУСТРОЙСТВ МЕЖЗАДАЧНОГО ОБМЕНА
    #МЗОИ
    #СУФ (Д1,Д2)  ВКЛЮЧЕНИЕ ПОДСИСТЕМЫ УПРАВЛЕНИЯ ВНЕШНЕЙ ПАМЯТЬЮ
* И ОПРЕДЕЛЕНИЕ СИСТЕМНЫХ ДИСКОВ С УКАЗАННЫМИ МЕТКАМИ Д1 И Д2
ТП   #ТПТК 2        ОПИСАНИЕ ТАБЛИЦЫ КОДИРОВОК ДЛЯ ТЕХНОЛОГИЧЕСКИХ
    #ПТК 1,#О      КЛАВИШ ДМ-2000, В КОТОРУЮ ДОЛЖНЫ БЫТЬ ВКЛЮЧЕНЫ
*     ...          НАИБОЛЕЕ ЧАСТО ИСПОЛЬЗУЕМЫЕ ДИРЕКТИВЫ СИСТЕМЫ
    #ПТК 15,#М     РЕДАКТИРОВАНИЯ ОЛИМП
Т0   #ТПФК 2,200
    #ПФК 1,#О   ОПИСАНИЕ ТАБЛИЦЫ КОДИРОВОК ДЛЯ ФУНКЦИОНАЛЬНЫХ КЛАВИШ
```

```
*    ...          ВТА2000-11 (ЗДЕСЬ ВМЕСТО ... МОГУТ БЫТЬ УКАЗАНЫ ДРУГИЕ
     #ПФК 11,UU",* ДИРЕКТИВЫ, ВЫБРАННЫЕ ПОЛЬЗОВАТЕЛЕМ)
T1   #ТКО 21      ТАБЛИЦА КОМАНД ОПЕРАТОРА ДЛЯ НУЛЕВОГО И ПЕРВОГО РАЗДЕЛОВ,
      #КО ВР      ОБЕСПЕЧИВАЮЩИХ ВЫПОЛНЕНИЕ ЗАДАЧ В РЕЖИМЕ РЕАЛЬНОГО
*    ...          ВРЕМЕНИ. ЗДЕСЬ ВМЕСТО ... ДОЛЖНЫ БЫТЬ УКАЗАНЫ КОМАНДЫ
      #КО УС      ОС,ПП,ЦЕ,АВ,АК,ЦК,БД,СБ,ВД,СТ,ВЫ,ПУ,ПА,ЛН,ПВ,ТЛ,СО,ОР,ДН
T2   #ТКО 23      ТАБЛИЦА КОМАНД ОПЕРАТОРА (С КОМАНДАМИ ОТЛАДКИ)
      #КО НС      ДЛЯ ПУЛЬТОВ ОПЕРАТОРА, ПРЕДНАЗНАЧЕННЫХ ДЛЯ
*    ...          РАБОТЫ В РЕЖИМЕ РАЗДЕЛЕНИЯ ВРЕМЕНИ. ЗДЕСЬ ВМЕСТО ...
      #КО UU      ДОЛЖНЫ БЫТЬ УКАЗАНЫ КОМАНДЫ КС,ОР,СО,УС,ЧС,ЧВ,ЧД,ЗС,ЗВ,ЗД
      #ТЛН(0,1),25  ФОРМИРОВАНИЕ ТАБЛИЦЫ СТАНДАРТНЫХ ЛОГИЧЕСКИХ
      #ЛН 1,5        НОМЕРОВ ДЛЯ НУЛЕВОГО И ПЕРВОГО РАЗДЕЛОВ.
      #ЛН 5,3        СТАТИЧЕСКОЕ НАЗНАЧЕНИЕ ДРУГИХ ЛОГИЧЕСКИХ
      #ЛН 6,6        НОМЕРОВ ПРОИЗВЕДЕНО СОГЛАСНО ТРЕБОВАНИЯМ
      #ЛН 7,32       ЗАДАЧ ПРОСТЕЙШЕГО ТЕХНОЛОГИЧЕСКОГО ПРОЦЕССА
      #ЛН 8,33
      #ЛН 9,36
      #ЛН 10,37
      #ЛН 11,35
      #ТЛН 2,6       РЕЗЕРВИРОВАНИЕ МЕСТА ДЛЯ ТЛН РАЗДЕЛАМ,
      #ТЛН 3,6       ПРЕДНАЗНАЧЕННЫМ ДЛЯ РАБОТЫ В РЕЖИМЕ
      #ТЛН 4,30      РАЗДЕЛЕНИЯ ВРЕМЕНИ
      #КОС          КОНЕЦ ПРОГРАММЫ ГЕНЕРАЦИИ ОС
      ENDM
```

12.6. Основные команды оператора

Общий формат команды оператора. С момента запуска ОС и выдачи ею на главный пульт оператора сообщения ***МНОГОЗАДАЧНАЯ СИСТЕМА*** вся дальнейшая работа ОС управляется со стороны человека-оператора с помощью команд, вводимых с одного или нескольких пультов.

С помощью команд оператор управляет распределением ресурсов системы, выполнением задач в любом из выбранных режимов, следит за работоспособностью оборудования, ведет отладку программ и т.п.

Команда оператора представляет собой строку алфавитно-цифровых символов, первым символом которой является двоеточие (:). Строка должна содержать не более 72 символов и заканчиваться признаком конца строки. Формат команды оператора следующий:

:ико,p1,p2, . . . ,pN

Здесь *ико* – идентификатор, определяющий название команды оператора; *p1, p2, . . . ,pN* – параметры команды оператора, уточняющие ее действия.

Пробелы между : (двоеточием) и идентификатором команды оператора не допускаются.

После обработки очередной команды оператора ОС выдает на пульт оператора, с которого была введена команда, разрешающий символ * (звездочка), означающий, что ОС готова принимать следующую команду.

Все команды оператора можно распределить по следующим основным группам: команды, управляющие основными ресурсами ОС; команды, управляющие выполнением задач; команды, управляющие работой

периферийного оборудования; команды документирования работы с пультами; команды реконфигурации системы по устранению последствий сбоев и отказов в работе оборудования.

Ниже приведены наименования наиболее часто используемых команд оператора из каждой группы. По мере необходимости для некоторых команд указаны также и их форматы.

Команды, управляющие основными ресурсами ОС. Опишем эти команды.

ВРЕМЯ – ввод времени дня или индикация текущего времени. Эта команда применяется для ввода времени суток при запуске системы, а также для выдачи на пульт оператора текущего времени при работе системы. Часы, минуты, секунды задаются в общепринятых диапазонах. Если какой-то параметр опущен, его значение приравнивается нулю.

ДН – ввод даты или ее индикация. Эта команда используется для ввода в систему календарной даты или вывода ее текущего значения.

ПП – пуск второго процессора. Эта команда выполняется только в двухпроцессорной ОС и служит для включения в работу второго процессора при начальном запуске системы либо включения в работу процессора, остановленного ранее командой оператора **:ОСТАНОВ.**

ОСТАНОВ – останов процессора. Эта команда может применяться только во время работы двухпроцессорной ОС и предназначена для останова одного из процессоров.

нр – назначить раздел оперативной памяти. Эта команда оператора назначает пульту оператора номер раздела оперативной памяти, в котором могут работать команды, управляющие выполнением задач. Формат команды

:нр

Здесь *нр* – десятичное число, определяющее номер раздела, назначенного пульту, с которого введена команда. Фактически эта команда осуществляет переназначение пульту раздела памяти, так как первоначальное закрепление пульта за разделом осуществляется в программе генерации ОС макрокомандой **# ПО.** Кроме того, переназначение раздела может быть произведено только тем пультам, которым в программе генерации ОС было разрешено изменять сферу действий. В общем же случае команда *:нр* разрешена для разделов, используемых в режимах реального времени и пакетной обработки, и запрещена для тех разделов, которые выбраны для выполнения задач в режиме разделения времени.

НС – начало сеанса. По этой команде пульт оператора объявляется абонентом режима разделения времени, т.е. ему выделяется свободный виртуальный раздел.

Следует обратить внимание на то, что в выделенном по команде **:НС** виртуальном разделе не назначено ни одно устройство (или файл) ни одному логическому номеру. Поэтому после команды оператора

:НС необходимо выполнить назначение логическим номерам устройств и файлов с помощью команд оператора **:ЛН**.

КС – конец сеанса. По этой команде оператора заканчивается сеанс работы пульта в режиме разделения времени.

После выполнения этой команды восстанавливается номер раздела памяти, за которым, возможно, был закреплен пульт оператора до начала сеанса работы в режиме разделения времени.

Команды, управляющие выполнением задач. Описанные здесь команды оператора **:ВЫЗОВ**, **:СТАРТ**, **:ПАУЗА**, **:ПУСК**, **:ПВ**, **:ИП**, **:СО** используются для управления ОЗУ-резидентными и сегментированными задачами. Команды оператора, необходимые для управления выполнением диск-резидентных задач, были описаны в § 9.8.

ЛН – назначение устройства либо дискового файла. По этой команде устанавливается соответствие между логическими номерами, применяемыми в задачах для выполнения операций ввода-вывода в данном разделе, и порядковыми номерами УВВ либо файлами, с которыми выполняются эти операции. Формат команды

:ЛН*,лн* [*,пн*]

или

:ЛН*,лн,имяф* [: [*имяд*] [*:пнд*]]

Здесь *лн* – логический номер устройства в задаче, которому назначается порядковый номер (*пн*) или же дисковый файл (*имяф*) с атрибутами имени диска *имяд* и порядковым номером диска *пнд*.

Если в команде указан только один параметр *лн*, производится освобождение УВВ либо дискового файла, ранее назначенного этому логическому номеру, и это устройство либо файл становится доступным для использования в задачах других разделов.

ТЛ – распечатать ТЛН. По этой команде на пульт оператора выводится содержимое таблицы логических номеров раздела либо отдельного ее элемента.

ВВ – ввод загрузочного модуля с носителя. По этой команде с миникассеты либо магнитной ленты вводится загрузочный модуль в раздел оперативной памяти, определенный командой оператора *:нр*.

Размещение загрузочного модуля может производиться только в свободный раздел, т.е. в раздел, в котором нет ни одной работающей задачи. После ввода загрузочного модуля для задач, выполняемых по времени, корректируется время их первоначального запуска (относительная фаза складывается со временем окончания загрузки). Перед загрузкой файл с загрузочным модулем должен быть подведен под считывающие головки с помощью команды оператора **:УСТАНОВ**.

ВД – ввод загрузочного модуля с дискового файла. По этой команде в раздел памяти, если он свободен, вводится загрузочный модуль с файла на диске.

Выполнение команды :**ВД** аналогично выполнению команды :**ВВОД**.

ВЫЗОВ – вызов задачи на выполнение. По этой команде осуществляется запуск с пульта оператора ОЗУ-резидентной или сегментированной задачи на однократное выполнение. При вызове ей может быть передано до пяти числовых параметров (см. § 9.4).

СТАРТ – ввод загрузочного модуля с запуском задачи. По этой команде в раздел памяти, если он свободен, вводится загрузочный модуль с файла на диске и запускается старшая по приоритету (или единственная) в нем задача с передачей ей до пяти параметров.

ПАУЗА – приостанов выполнения задачи. Эта команда предназначена для приостанова задачи со стороны оператора. После приостанова задача может быть продолжена по команде оператора :**ПУСК**.

ПУСК – продолжение выполнения приостановленной задачи. Эта команда предназначена для возобновления задачи, приостановленной по команде оператора :**ПАУЗА** (или по собственной инициативе, когда в задаче использовалась макрооперация **PAUSE**). В последнем случае при возобновлении выполнения задачи ей может быть передано до пяти числовых параметров.

Эту же команду можно применять для возобновления задачи после аварийного приостанова (см. макрооперацию **SCHPT**, § 10.8).

ПВ – установка временных характеристик задаче. Эта команда предназначена для первоначальной установки или изменения предыдущих значений фазы и интервала задачам, которые должны запускаться по времени. Формат команды

:**ПВ**,*имяз*,*масшт*,*инт*,*ф1*,*ф2*,*ф3*,*ф4*

Здесь параметры и значения, которые они могут принимать полностью, аналогичны тем же параметрам, которые используются в макрооперации **TURN** (см. § 10.6).

ИП – изменение приоритета задачи. Эта команда устанавливает новый приоритет задачи, согласно которому в данном разделе переформировывается цепочка БУЗа по приоритету. Действия этой команды аналогичны действиям, выполняемым по макрооперации **CHAP** (см. § 10.6).

СО – распечатка состояния задачи. По этой команде на пульт оператора выдается текущее состояние и содержимое регистров задачи.

Состояние идентифицируется специальными символами, указывающими на одно из следующих возможных состояний: выполнение, готовность к работе, ожидание события, завершение, приостанов и т. д.

ОР – освобождение раздела оперативной памяти. Эту команду применяют для аварийного завершения всех задач, работающих в одном разделе памяти.

По этой команде прекращается выполнение задач, операций ввода-вывода и отсчетов временных промежутков, заказанных в задачах макрооперациями **STIME**. Дисковые файлы, назначенные всем логическим

номерам раздела, закрываются. Раздел становится свободным и может использоваться для ввода в него нового загрузочного модуля.

Команды, управляющие работой периферийного оборудования. Опишем эти команды.

АВТОНОМ – перевод УВВ в автономный режим работы. Эту команду применяют для того, чтобы объявить УВВ не готовым к работе с системой.

Любая попытка обратиться в задачах к УВВ, переведенному в автономный режим, вызовет завершение макроопераций **EXIO** с ненулевыми кодами завершения и выдачу сообщения на пульт оператора о требовании вмешательства. Команду :**АВТОНОМ** целесообразно использовать в тех случаях, когда требуется вмешательство оператора в работу устройства. Например, если оператор обнаруживает конец бумаги на печатающем устройстве или же конец ленты в накопителе, он может объявить устройство не готовым к работе, вставить бумагу или ленту и затем перевести его в режим работы с ОС с помощью команды оператора :**ЦЕНТР**, описание которой приведено ниже. Перевод устройства в автономный режим осуществляется также автоматически системой, если при выполнении операции на УВВ возникает ситуация, требующая вмешательства оператора в работу устройства (обрыв носителя, отключение питания и т.п.).

ЦЕНТР – перевод УВВ в состояние готовности работы с системой. По этой команде УВВ переводится в состояние готовности к работе, и на нем начинается выполнение операций ввода-вывода по запросам от задач.

УСТАНОВ – установка мини-кассеты или магнитной ленты к началу файла. На мини-кассете и магнитной ленте файлы с данными либо программами отделяются друг от друга маркерами конца файла. Нумерация файлов начинается с единицы, т.е. первый файл от начала ленты имеет номер 1, второй – 2 и т.д. Для ввода в память загрузочного модуля с какого-либо файла либо для работы программ, в которых автоматический поиск файлов не предусмотрен, требуемый файл необходимо предварительно подвести под считывающий механизм. Для этого и служит команда оператора :**УСТАНОВ**, которая осуществляет перемотку ленты, а затем пропуск соответствующего числа маркеров. Формат команды

:**УС [ТАНОВ]** ,*пн* ,*нф*

Здесь *пн* – порядковый номер накопителя, на котором производится установка; *нф* – номер файла, к началу которого необходимо подвести мини-кассету либо магнитную ленту.

БД – блокировка диска. Эта команда предназначена для блокировки работы программ управления внешней памятью с указанным диском. Обязательно выполнение этой команды перед заменой одного дискового пакета данного накопителя на другой либо перед включением программ упаковки дискового пространства данного пакета. Блокировка диска может осуществляться только тогда, когда все файлы на нем находят-

ся в закрытом состоянии. Для этого можно применять команду оператора **:ОР.**

СБД – сброс блокировки диска. Эта команда снимает блокировку диска и разрешает доступ к нему программ управления внешней памятью.

Команды документирования работы с пультами. В данную группу входят команды оператора, предназначенные для документирования работы пультов оператора. Под документированием подразумевается установка для пульта такого режима, когда все введенные с него и выполненные без ошибок команды оператора записываются в специальный файл на диске. К этим командам относятся:

РД –установить режим документирования;

КД – завершить документирование;

СД – определить состояние документирования;

ОД – отменить документирование.

Команды для реконфигурации системы по устранению последствий сбоев и отказов в работе оборудования. В эту группу входят команды объявления работоспособным или неработоспособным раздел оперативной памяти и команды, управляющие процессом перезапуска задач с установленных контрольных точек. К этим командам относятся:

АР – заблокировать раздел оперативной памяти;

ЦР – отменить блокировку раздела оперативной памяти;

КТ – перезапуск с контрольной точки.

Все команды оператора, описанные в данной главе, включаются в программу генерации ОС с помощью макрокоманд **#КО,** приведенных в § 12.4.

12.7. Пример компоновки загрузочного модуля с ОС

Рассмотрим последовательность действий, выполняемых оператором при создании загрузочного модуля с ОС на примере ОС, программа генерации которой приведена в § 12.5.

Для подготовки загрузочного модуля ОС, так же как и для загрузочного модуля II типа (см. § 11.8), может быть применен любой раздел пользователя в ранее сгенерированной или стартовой операционной системе. Поэтому предполагается, что выбранная для работы ОС с помощью начального загрузчика уже загружена в нулевой раздел, запущена и находится в состоянии ожидания ввода команд оператора. Оператор вводит нужный ему номер раздела и выполняет последовательность работ, необходимую для формирования загрузочного модуля I типа.

Напомним, что для создания загрузочного модуля I типа обязательны все четыре этапа работ, описанных в § 9.4. Будем считать, что первый этап нами успешно завершен и запрограммированная на МАКРО-ЯЗЫКе программа генерации ОС сформирована на одном из системных дисков под именем *ПЛАНОС.* При этом для подготовки и загруз-

ки ее на диск могут быть использованы любые стандартные средства, обеспечивающие работу с символьными данными (например, системная обрабатывающая программа ПОФ, см. § 11.6).

На втором этапе программа генерации ОС подвергается макрогенерации с применением библиотеки макроопределений *МБФ*. Выходом макрогенератора *МГД* является та же корневая часть программы генерации ОС, но в формате операторов МНЕМОКОДа.

Здесь и далее будем использовать системные диски для удобства записи команд, хотя в общем случае файл с программой генерации, так же как и библиотеки, может располагаться на любом дисковом накопителе пользователя.

Для осуществления макрогенерации должна использоваться следующая последовательность команд операторов:

:ЛН,5,*ПЛАНОС*
:ЛН,3,*МБФ*
:ЛН,6,*пнуп*
:ЛН,4,*ПЛАНОСС*
:СТ [АРТ],*МГД*

Здесь *пнуп* – порядковый номер устройства печати листинга. Результирующий файл создается на системном диске под именем *ПЛАНОСС*, где последний символ *С* условно указывает на символьный формат.

На третьем этапе полученная в результате макрогенерации программа должна быть транслирована транслятором с МНЕМОКОДа (*МНКД*) с получением той же программы генерации ОС, но в объектном формате.

Последовательность команд оператора в этом случае следующая:

:ЛН,5,*ПЛАНОСС*
:ЛН,4,*ПЛАНОСД*
:ЛН,6,*пнуп*
:СТ [АРТ],*МНКД*

Результирующий файл трансляции создается под именем *ПЛАНОСД*. Здесь символ *Д* условно обозначает двоичный формат.

На четвертом этапе с помощью компоновщика программ (*КОМПД*) осуществляется компоновка транслированной программы с библиотекой подпрограмм для генерации ОС (ОБОС) в единый загрузочный модуль. Компоновщик вызывается на выполнение командой оператора

:СТ [АРТ],*КОМПД*

и выдает на пульт оператора следующую информацию:

КОМПОНОВЩИК ПРОГРАММ

РЕЖИМ?

В ответ на запрос о режиме пользователь вводит необходимую директиву для компоновщика с указанием имени файла, содержащего компо-

нуемую программу генерации или библиотеку. По мере поступления программ на компоновку компоновщик распечатывает имена тех программ, которые вошли в загрузочный модуль, и их граничные адреса внутри модуля.

По завершении компоновки на пульт оператора выводится информация о распределении адресов между программами внутри загрузочного модуля, о наличии свободной памяти в нулевой странице и размере всего модуля.

Ниже приводятся форматы директив компоновщику, вводимых оператором в ответ на запросы о режимах:

РЕЖИМ?ПП,*ПЛАНОСД*
РЕЖИМ?БП,*ОБОС*
РЕЖИМ?КП, *# ОС15*

Последняя директива указывает компоновщику на то, что должен быть сгенерирован загрузочный модуль с ОС на одном из системных дисков под именем *# ОС15*. Здесь число *15* указывает на номер сгенерированной ОС (см. § 9.4).

Теперь загрузочный модуль со сгенерированной ОС может быть загружен в память с помощью начального загрузчика. Для этого после включения ВК выполняется последовательность действий на инженерной панели:

а) установить диск, содержащий файл с загружаемой ОС, в выбранный для этой цели накопитель;

б) установить на клавишном регистре инженерной панели следующие данные:

Назначение	Разряды
Признак автозапуска	0
Номер загружаемой ОС	1–6
Тип диска (100-фиксированный диск, 101-съемный диск А322-3)	7–9
Код выборки диска (управляющей карты)	10–15

в) нажать клавишу **ЗАГР** на инженерной панели.

Далее пользователь получает доступ ко всем ресурсам ВК, предусмотренным в программе генерации ОС. При этом подразумевается, что обращение к этим ресурсам осуществляется либо в программах пользователя, либо в процессе их подготовки с пульта оператора. В следующем разделе рассматриваются программные средства, составившие систему подготовки программ.

РАЗДЕЛ ЧЕТВЕРТЫЙ

СИСТЕМА ПОДГОТОВКИ ПРОГРАММ

Глава 13. ТРАНСЛИРУЮЩИЕ СИСТЕМЫ, КОМПИЛЯТОРЫ И МАКРОСРЕДСТВА

13.1. Общая характеристика системных обрабатывающих программ

Системные обрабатывающие программы позволяют проводить программирование и отладку больших программных комплексов по частям с применением различных языков программирования. При этом для каждой части может использоваться наиболее подходящий язык. С этой целью пользователю поставляется ряд трансляторов и компиляторов с алгоритмических и процедурных языков программирования, а также средства отладки на их основе.

Все компиляторы и трансляторы с языков программирования выполняют обработку исходных текстов программ, вырабатывая результирующие программные модули в едином для всех языков объектном (перемещаемом) формате. Объединение произведенных трасляторами частей в единую выполняемую программу – загрузочный модуль – производится компоновщиком программ, который помимо вновь протранслированных программных модулей включает в нее требуемые библиотечные подпрограммы.

Все системые обрабатывающие программы работают под управлением операционных систем АСПО и, как правило, поставляются пользователю в виде однозадачных загрузочных модулей, предназначенных для работы в непривилегированных разделах памяти.

Для функционирования этих программ используется минимальная конфигурация технических средств, включая пульт оператора, устройство печати и один или несколько накопителей внешней памяти.

Для осуществления операций ввода-вывода в генерируемую ОС должна быть включена возможность выполнения макроопераций **WAITE, EXIO, STAT, PAUSE** и **TIME.**

Ввод исходных текстов для системных обрабатывающих программ допускается с любых устройств ввода-вывода и из файлов на дисках; результирующие программы помещаются на дисках и могут выводиться на различные носители типа магнитной ленты, мини-кассеты и т.п. Для указания этих устройств используются стандартные логические номера.

Ниже даны краткие характеристики системных обрабатывающих программ и их основных функциональных возможностей.

13.2. Система программирования на базе языка МНЕМОКОД

Базовый язык программирования МНЕМОКОД является машинно-ориентированным языком программирования, подробное описание которого приведено во втором разделе данной книги. Там же приведено описание МАКРОЯЗЫКа, являющегося проблемно-ориентированным расширением языка МНЕМОКОД, в котором используются глобальные и локальные индексированные переменные, условная компиляция и другие "традиционные" атрибуты макропроцессоров [17].

Основу системы программирования на базе языка МНЕМОКОД составляют транслятор с МНЕМОКОДа (*МНКД*) и компоновщик программ (*КОМПД*). Кроме того, к этой же системе относят макрогенератор (*МГД*) и ряд сервисных программ: различные редакторы, специально ориентированные на ассемблероподобные языки, ретранслятор объектных модулей, программы корректировки библиотек и др.

Транслятор с МНЕМОКОДа. Входным языком транслятора является ассемблероподобный язык МНЕМОКОД, ориентированный на архитектуру СМ-2М.

Для запуска транслятора на выполнение, так же как и для запуска всех обрабатывающих программ, должно быть произведено первоначальное назначение стандартным логическим номерам исходного и результирующего файлов, а также устройства печати.

Запуск транслятора осуществляется командой оператора :**СТАРТ** с передачей ему необходимых числовых параметров.

Так как описание процедуры запуска однозадачных загрузочных модулей носит вполне стандартный характер и было приведено ранее, то при дальнейшем изложении материала описание этой процедуры будем опускать.

Макрогенератор. Входной язык макрогенератора (*МГД*) – МАКРОЯЗЫК – является расширением МНЕМОКОДа, позволяющим пользователю развить этот машинно-ориентированный язык и сделать его удобным для конкретных применений. В МАКРОЯЗЫКе имеется возможность определять часто встречающиеся последовательности операторов МНЕМОКОДа как макроопределения, доступные в дальнейшем для использования в программах. При этом программист пишет только один оператор-макрокоманду. Из одного и того же макроопределения при написании макрокоманд с различными параметрами будут генерироваться различные последовательности операторов МНЕМОКОДа.

При генерации программы на МАКРОЯЗЫКе с помощью макрогенератора МГД макрокоманды заменяются последовательностью операторов из макроопределений, т.е. *макрорасширениями.*

Компоновщик программы. Компоновщик программ *КОМПД* предназначен для генерации загрузочных модулей различного типа, входом для которого являются программные модули в объектном формате,

вырабатываемые трансляторами с различных языков программирования: выходом является программа в абсолютном формате (загрузочный модуль), пригодная для ввода ее в какой-либо раздел оперативной памяти для выполнения. Компоновщик обеспечивает обработку как отдельных программных модулей, так и модулей, содержащихся в различных библиотеках. В состав загрузочного модуля из библиотек выбираются только те модули, к которым есть внешние обращения в программных модулях, обработанных до библиотеки.

Управление работой компоновщика осуществляется командами оператора, которые вводятся после выдачи компоновщиком сообщения РЕЖИМ?

Приведем форматы наиболее часто используемых команд оператора, с помощью которых осуществляется основная функция компоновщика – создание из нескольких модулей в объектном формате файла с загрузочным модулем

ПП,*имяф* [: [*имяд*] [:*пнд*]]

По этой команде программа, расположенная в указанном файле, обрабатывается согласно ее типу, заданному при написании в операторе **NAM**. В процессе загрузки на печатающее устройство или на пульт оператора выдаются ее имя и границы размещения в загрузочном модуле:

БП,*имяф* [: [*имяд*] [:*пнд*]

По этой команде все вводимые программы и подпрограммы обрабатываются как библиотечные подпрограммы:

КП,*имяф* [: [*имяд*] [:*пнд*]]

По этой команде завершается работа компоновщика, создается под указанным именем файл с загрузочным модулем и производится печать протокола компоновки. В протокол входит печать наименований программ с границами загрузки, таблицы идентификаторов и информации о границах расположения загрузочного модуля.

Кроме перечисленных функций компоновщик обеспечивает выполнение дополнительных операций таких, как корректировка адреса загрузки и параметров программ, распечатка протокола компоновки и др.

Редактор символьной информации. Для внесения изменений в исходные программы, содержащие символьные записи на МНЕМОКОДе или других языках программирования, служит редактор символьной информации *РСИД*. Этот редактор позволяет в пакетном режиме выполнять построчное редактирование текстовой информации (вставка, вычеркивание и замена целых строк или отдельных символов в строках), а также копирование, объединение или распечатку символьных файлов.

Так как указателями объектов редактирования являются их координаты (номера записей или номера символов в записях файла), то наибо-

лее целесообразным считается применение этого редактора для редактирования позиционных языков программирования типа МНЕМОКОД.

Редактор комментариев. Этот редактор используется для внесения изменений в поле комментария программ, написанных на МНЕМОКОДе. Программа позволяет вставлять, заменять, удалять комментарии, форматировать комментарий по левой и правой границам. При этом задания номера позиции от пользователя не требуется: поиск свободной позиции либо текста, указанного в управляющей команде, осуществляется автоматически.

Редактор объектных модулей (РОМ). Редактор предназначен для внесения изменений в двоичные перемещаемые программы, написанные на МНЕМОКОДе. Это позволяет миновать этап повторной трансляции всей программы, так как при работе редактора транслируются лишь изменения.

Все изменения (имеются ввиду модификации ячеек памяти и команд) оформляются в виде программы на МНЕМОКОДе – редакторского модуля, которая после трансляции является входной информацией для редактора наряду с редактируемой программой и директивами редактора. На выходе редактора получается скорректированный модуль в объектном (перемещаемом) формате.

Ретранслятор перемещаемых и абсолютных программ. При разработке программного обеспечения часто возникает проблема получения листинга и модуля в исходном формате из объектного модуля. Данную проблему до некоторой степени решает ретранслятор перемещаемых и абсолютных программ РПАП. В процессе отладки с помощью РОМ пользователь может получить объектный модуль, для которого еще нет листинга и символьного носителя (файла). Ретранслятор РПАП позволяет из исправленного модуля получить новый листинг и символьный файл, обеспечивает восстановление на уровне МНЕМОКОДа содержимого двоичных файлов как перемещаемого, так и абсолютного форматов.

Программа корректировки библиотеки (КБД). Она предназначена для корректировки существующих и компоновки новых библиотек перемещаемых подпрограмм и позволяет вставлять новые подпрограммы в библиотеку, удалять и заменять старые подпрограммы библиотеки, объединять несколько библиотек в одну, получать копию библиотеки, каталог наименований всех библиотечных подпрограмм, а также идентификаторов входных точек, внешних идентификаторов и внешних констант.

Программа корректировки макробиблиотеки (КМБ). Программа предназначена для корректировки существующих и компоновки новых библиотек макроопределений, используемых макрогенератором *МГД* при обработке программ, написанных на МАКРОЯЗЫКе. Эта программа позволяет вставлять макроопределения в библиотеку макроопределений, расположенную в файле на диске, заменять макроопределения в библиотеке (без изменения имени), удалять макроопределения из биб-

лиотеки; заменять имя макроопределения, выводить макроопределения на устройство печати, получать каталог библиотеки, выводить макроопределения на внешний носитель.

13.3. Компилирующие системы с языков высокого уровня

Компилирующая система с языка ФОРТРАН-IV. Алгоритмический язык ФОРТРАН-IV относится к числу наиболее распространенных, что объясняется его простотой по сравнению с другими языками (например, с АЛГОЛом) и довольно высокой эффективностью получаемых после трансляции программ. В то же время ФОРТРАН подходит для программирования подавляющего большинства вычислительных алгоритмов и алгоритмов управления [18].

В компилирующую систему с расширенного стандартного языка ФОРТРАН вошли компилятор с этого языка и отладчик на уровне входного языка.

Полученные в результате компиляции перемещаемые программы на стадии компоновки могут объединяться в единое целое с программами, написанными на других языках, и библиотеками общего назначения, входящими в состав ПО СМ-2М.

Описание работы отладчика приведено в гл. 14.

Транслятор с языка АЛГОЛ. Алгоритмический язык АЛГОЛ, используемый в рамках АСПО, является диалектом языка АЛГОЛ-60 [19]. Программа на языке АЛГОЛ может быть разбита на несколько самостоятельных блоков, каждая из которых допускает автономную трансляцию. Это осуществляется с помощью внешней процедуры – кода. Эти же средства языка АЛГОЛ могут использоваться для объединения программ, написанных на языке АЛГОЛ, с программами, написанными на других языках.

Система поддержки выполнения транслированных программ представляет собой набор подпрограмм, помещенных в основную библиотеку подпрограмм # ОБП.

Компилирующая система с языка КОБОЛ. Универсальный проблемно-ориентированный язык КОБОЛ предназначен для программирования задач учета, планирования и статистики, основное содержание которых составляют обработка файлов, формирование и печать разнообразных отчетов.

В качестве подмножества языка КОБОЛ для АСПО в компилирующей системе выбрано минимальное подмножество (согласно [20]), включающее первые уровни ядра, модуля обработки таблиц и модуля поледовательного ввода-вывода.

Программа в объектном формате, созданная компилятором с КОБОЛ, перед выполнением должна быть объединена (при помощи компоновщика) с разработанной для этой цели библиотекой # CBL.

Компилирующая система с языка ПАСКАЛЬ. Язык программирования ПАСКАЛЬ получил широкое распространение как практический язык для написания эффективных и надежных программ в области прикладного и системного программирования [21].

Сфера применения языка ПАСКАЛЬ интенсивно расширяется, охватывая самые разнообразные области. Многие из них соответствуют основной ориентации в использовании вычислительных комплексов СМ-2М и старшей модели СМ 1210. Язык ПАСКАЛЬ-АСПО является расширенной версией языка ПАСКАЛЬ [22]. Эти расширения позволяют более полно использовать возможности ОС и архитектуры УВК.

От стандартной версии ПАСКАЛЬ-АСПО отличают средства раздельной компиляции, реализованные в рамках ОС АСПО, универсальный тип данных, позволяющий непосредственно адресовать машинную память и осуществить необходимую обработку информации, возможность определять константы любого типа, в том числе массивы и записи, расширенный набор управляющих операторов, средства работы с файлами прямого доступа, аппарат обработки исключительных ситуаций на уровне входного языка, расширенный набор предопределенных функций и процедур.

Программы, написанные на стандартном языке ПАСКАЛЬ, могут обрабатываться компилятором расширенной версии языка без всяких изменений.

Система поддержки выполнения ПАСКАЛЬ-программ (RUN-TIME) включает в себя поддержку ввода-вывода, динамического распределения памяти, рекурсии, операций со множествами, обработки ошибок с соответствующими сообщениями и др.

Компиляторы с языков системного программирования. Языки высокого уровня для системного программирования ориентированы в основном на автоматизацию процесса проектирования внутрисистемного программного обеспечения.

Из языков этого класса в программное обеспечение УВК СМ-2М входят: машинно-зависимый язык системного программирования ЯСП, машинно-независимый язык системного программирования МАС.

Входным языком компилятора с языка ЯСП является алголоподобный машинно-ориентированный язык ЯСП, обладающий свойствами, присущими языкам высокого уровня: свободный формат программы, блочная структура программы, описание памяти и констант, общие блоки памяти, описание входных точек и внешних ссылок, арифметические и условные выражения, операторы цикла, выбирающие выражения, составные выражения и структурные переходы, подпрограммы и сопрограммы.

Машинную ориентацию языку придает возможность описания памяти и доступ к ней по именам и с помощью адресных констант, двоичное представление данных, использование косвенной и абсолютной адресации и др.

Система поддержки выполнения результирующей программы представлена набором подпрограмм, объединенных в отдельную библиотеку # ЯСПБИБ, которая обеспечивает выполнение операций с байтами, перевода чисел из одного формата в другой, распечатки данных в различных форматах и т.д.

Входным языком компилятора с языком МАС является машинно-независимый язык высокого уровня МАС, который используется для системного и прикладного программирования. В состав операторов языка входят оператор присваивания, оператор перехода, оператор вызова подпрограмм, условный оператор, оператор цикла, оператор переопределения структуры данных, операторы ввода-вывода.

Обширный набор операций над *строковыми* данными (катенация, дублирование, выделение подстроки, вычисление длины строки, поиск текста, замена текста, подсчет вхождений и т.п.) обеспечивает применение языка для задач обработки текстов, программирования компиляторов, ассемблеров, редакторов и т.д.

В качестве системы поддержки МАС-программ используется специальная библиотека #МАС, обеспечивающая выполнение таких функций, как ввод-вывод, работа с файлами, преобразование типов данных, текстовая обработка, вычисление адреса элемента однородного и неоднородного массивов и т.д.

Система программирования на основе языка управления заданиями. Система программирования на основе языка управления заданиями лежит в основе пакетной обработки, обеспечиваемой операционными системами АСПО. Система реализована в виде диспетчера пакетной обработки (ДПОМ), предназначенного для выполнения вычислительных и управляющих операций в соответствии с указаниями в пакете заданий. Пакет заданий представляет собой последовательность операторов, написанных с использованием *языка управления заданиями* (ЯУЗ) и содержащих описание *действий-директив,* выполняющихся в процессе обработки пакета ДПОМ.

Для упрощения составления пакета ряд часто встречающихся фрагментов пакета можно выделить в отдельные "подпакеты" – *каталогизированные процедуры,* снабдить их наименованиями и поместить в библиотеку каталогизированных процедур.

Средства отладки реализованы в виде команд оператора, вводимых с пульта оператора в процессе работы диспетчера в отладочном режиме. Как правило, отладка ведется в *пошаговом* режиме функционирования, суть которого заключается в том, что перед выполнением каждого оператора работа ДПОМ приостанавливается, а на пульт оператора выводится сообщение, содержащее текст оператора.

После приостанова программист, отлаживающий пакет или каталогизированную процедуру, имеет возможность запросить значение любой переменной, выполнить произвольный оператор, текст которого вводится с пульта, установить или отменить контрольный установ по номеру

оператора в пакете или в библиотеке каталогизированных процедур и т.п.

Макропроцессор МП25. Он предназначен для генерации произвольных текстов. В отличие от макрогенератора МГД, с которым у макропроцессора МП25 возможно пересечение областей применения, у последнего нет никаких ограничений на характер исходного и выходного текстов. Это может быть, например, программа для ЭВМ на каком-нибудь языке программирования, книга или последовательность ответов на анкету.

В этом смысле МП25 можно рассматривать как *свободный макропроцессор* [23], который допускает максимальную свободу вызовов макросов.

Здесь, так же как и в МАКРОЯЗЫКе МГД, под *макросом* подразумевается средство замены одной последовательности символов другой. Свободная запись вызова макросов позволяет использовать их для замены в произвольном символьном файле одной последовательности литер на другую, что называется *контекстным редактированием.*

Наряду со свободной записью макроса в языке МП25 возможно использование макроса с *предупреждающим* символом, по которому макропроцессор определяет, что вслед за этим символом следует вызов макроса.

Поддержкой языка макропроцессора МП25 является *библиотека макроопределений для системного программирования* (# БМСП). Помимо естественности и достаточной эффективности программы, написанные с использованием # БМСП, легко переносимы на другие машины. Весь перенос выливается только в создание новой библиотеки макроопределений.

Названия макросов, составивших библиотеку, подобраны таким образом, чтобы пользователь мог использовать их как операторы некоторого языка структурного программирования:

 # DECL – оператор объявления переменных;
 # SET – оператор присваивания;
 # INC – оператор условного или безусловного прибавления 1;
 # DECR – оператор условного или безусловного вычитания 1;
 # CALL – оператор вызова подпрограммы;
 # SUBR – оператор определения начала подпрограммы;
 # RET – оператор выхода из подпрограммы;
 # SVC – вызов супервизора;
 # IF – условный оператор перехода;
 # IF – условный оператор IF–THEN–ELSE;
 # WHILE – оператор цикла типа WHILE;
 # CYCLE – оператор цикла типа пересчета;
 # CASE – оператор выбора;
 # SWP – оператор обмена содержимым двух ячеек;
 # ASC – задание распакованной символьной строки;

 # FOUND – поиск текста;
 # MOVE – пересылка данных в памяти;
 # PRINT – распечатка памяти;
 # FILE – оператор описания файла;
 # OPEN – открытие файла;
 # GET – чтение файла;
 # PUT – запись в файл;
 # CLOSE – закрытие файла.

Для написания сложных программных систем на базе языка макропроцессора МП25 может быть использован *отладчик макроопределений*, который подключается к макропроцессору в любое удобное время. Описание возможностей этого отладчика приведено в гл. 14.

Интерпретатор с языка БЕЙСИК. Вариант языка БЕЙСИК, реализованный в составе ПО СМ-2М, позволяет проводить арифметические операции с целыми и вещественными числами и массивами этих чисел, организовывать простейшие циклы и обращения к подпрограммам, а также форматы и правила ввода-вывода данных в ходе работы программы [24].

Интерпретатор БЕЙСИКа работает либо в режиме непосредственной интерпретации операторов языка, либо в режиме выполнения программы по специальной команде **RUN**.

Набор типов данных и правил работы с многомерными массивами БЕЙСИКа сравним с ФОРТРАНом, что облегчает изучение этих языков путем их сравнительного анализа. Ряд конструкций БЕЙСИКа обеспечивает большую, чем в ФОРТРАНе, компактность записи выражений. Это облегчает работу в диалоговом режиме и повышает надежность программ.

Естественное стремление использовать простоту и удобство языка БЕЙСИК для решения не только математических и инженерных задач, но и задач реального времени обусловило появление языка БЕЙСИК-РВ.

Язык БЕЙСИК-РВ является расширенной версией языка БЕЙСИК, совпадающего с BASIC ANSI X3.60–1978. Введены следующие расширения: операторы ввода-вывода информации от устройств связи с объектом; операции побитной обработки; расширенные операторы ввода-вывода информации от любых устройств, обозначенных логическими номерами; операторы для работы с файлами; оператор вызова внешней подпрограммы; операторы управления задачами по времени и внешним событиям.

Программы, написанные на языке БЕЙСИК-РВ, могут быть отлажены в режиме интерпретации, а затем обработаны компилятором. В результате получаются программы в объектном формате, которые могут быть выполнены под управлением операционной системы.

Интерпретирующая система для БЕЙСИК-РВ называется диалоговой многозадачной системой реального времени (ДМСРВ). Типичными за-

дачами, решаемыми с помощью ДМСРВ, являются сбор и обработка информации в системах управления технологическими процессами, а также управление технологическим оборудованием, сбор, обработка и накопление измеренных данных о ходе научного эксперимента, расчет и выдача управляющих воздействий для управления экспериментальными установками. Выполнение программы осуществляется в реальном масштабе времени.

Язык БЕЙСИК-РВ позволяет вызывать внешние подпрограммы, написанные на языках ФОРТРАН, МНЕМОКОД, АЛГОЛ и предварительно оттранслированные с помощью специального оператора вызова внешней подпрограммы. Это дает возможность пользователям самостоятельно развивать систему.

Диалоговая ДМСРВ обеспечивает многопультовую работу, т.е. несколько пользователей могут одновременно и независимо друг от друга готовить и выполнять свои программы.

Использование транслирующих систем, компиляторов и макросредств практически невозможно без применения средств отладки и редактирования, к рассмотрению которых и переходим.

Глава 14. СРЕДСТВА РЕДАКТИРОВАНИЯ, ДОКУМЕНТИРОВАНИЯ И ОТЛАДКИ

14.1. Общие сведения

Особое место во всей системе подготовки программ, для написания которых можно использовать любой из языков программирования, занимают средства редактирования и документирования. И это вполне закономерно, так как в процессе подготовки и отладки программ пользователю чаще всего приходится иметь дело с исходным текстом, который постоянно подвергается всевозможным корректировкам.

Средства редактирования системы подготовки программ включают различные по своим функциям и исполнению редакторы символьной информации.

Наиболее удобными проявили себя *интерактивные* средства редактирования, которые позволяют пользователю редактировать исходные тексты отдельными фрагментами и всегда видеть перед собой результат.

В качестве редактируемых фрагментов в интерактивных редакторах используют отдельные строки или кадры текста, высвечиваемые на экранах дисплеев, которые применяют для редактирования. При этом помимо команд оператора, управляющих процессом редактирования, в полной мере хороши "редакторские" возможности самой клавиатуры дисплеев, которая дает возможность сдвигать строки и символы в различных направлениях внутри редактируемых фрагментов, удалять и вставлять различные тексты и т.п.

Наряду с усовершенствованием средств редактирования появилась необходимость решения другой проблемы – *подготовки документации.* Для нее характерны частые корректировки и поиск текста, ориентация на определенный формат с множеством оформительных операций – нумерация страниц, глав, выделение абзацев, пунктов, оформление рисунков, таблиц и т.д.

В данной главе приведены описания двух редакторов из класса интерактивных, которые осуществляют *построчное* (интерактивный редактор *IET*) и *покадровое* (дисплейный редактор *SLIDE*) редактирование текстовых файлов. Кроме того, редактор *SLIDE* снабжен также *документатором,* позволяющим осуществлять наиболее часто используемые функции при форматировании текстовых документов.

В данной главе дано также краткое описание многофункциональной интерактивной системы обработи текстовой информации *ОЛИМП,* основное применение которой ориентировано на усовершенствование процесса подготовки и изготовления текстовых документов, включая обработку и корректировку текстовых файлов, хранение и многократное надежное воспроизведение текстовых материалов.

14.2. Интерактивный редактор IET

Интерактивный редактор IET предназначен для построчного редактирования небольших символьных дисковых файлов. Это могут быть исходные программы на языках ФОРТРАН, АЛГОЛ, МНЕМОКОД и других, а также текстовые документы произвольной природы.

Входной информацией для редактора являются, с одной стороны, исходный символьный файл, подлежащий редактированию, с другой – управляющие команды редактирования и корректирующие записи, которые обеспечивают редактирование исходных файлов.

Редактор начинает работу с чтения исходного файла, выводит на экран первую строку файла и непосредственно за ней разрешающий символ. Редактирование производится построчно, и та строка, на которой редактор приостанавливается перед вводом очередной команды оператора, в дальнейшем будет называться *редактируемой строкой.* В исходном состоянии, очевидно, редактируемой строкой является первая.

Команды оператора интерактивного редактора могут быть разделены на восемь основных групп: управления, поиска, посимвольного редактирования, изменения, перемещения, построчного редактирования, печати и завершения.

Команды управления позволяют осуществлять вставку файлов, установку и отмену нумераций строк, установку длины строки и ряд других функций.

Команды поиска осуществляют поиск от начала файла или от редактируемой строки первой встретившейся строки, содержащей заданный

текст (контекстный поиск), удаление строк до указанного текста, переход по тексту и другие функции.

Команды изменения используются для изменения текста в редактируемой строке или по всему файлу. По этим командам заменяется прежний текст, начинающийся в пределах "окна" редактирования, на новый, указанный в параметрах команды. Такое редактирование получило название контекстной замены. Для команд изменения можно указывать такой режим, который позволяет вывести на пульт оператора все строки, в которых произошли контекстные замены.

Команды посимвольного редактирования заменяют, вставляют и удаляют символы, а также усекают строки.

Команды этой группы осуществляют посимвольное редактирование с использованием спецсимволов клавиатуры дисплея. Такое редактирование получило название *экранного* редактирования. С помощью специальных разделителей курсор дисплея подводится к нужному месту редактирования в строке, и число разделителей в этом случае укажет на номер позиции в строке, с которой начинается редактирование.

Безусловно, экранное редактирование может принести наибольшую эффективность при работе с теми дисплеями, у которых имеется достаточно мощная аппаратная поддержка для выполнения этих функций.

Команды перемещения обеспечивают сдвиг фрагментов текста, перемещение по файлу с сохранением или удалением пропущенных строк.

Команды построчного редактирования осуществляют замену строк, вставку текста до или после редактируемой строки и другие функции.

Команды печати обеспечивают выполнение достаточно большого набора функций, связанных с печатью различных фрагментов текста, количественных характеристик файла, а также инструкций по пользованию отдельными командами и всем редактором IET.

К *командам завершения* относятся команды, обеспечивающие нормальное и аварийное завершение работы редактора.

14.3. Дисплейный редактор SLIDE

Дисплейный редактор текста *SLIDE* предназначен для автоматизации процесса подготовки и редактирования больших текстовых документов, оформленных в символьные файлы на дисках или носителях. Результирующими файлами этого редактора являются отредактированные файлы на диске либо файлы, выведенные на печать в заданном формате.

Логика работы этого редактора основана на представлении текста в виде одной длинной строки. Существенно деление этой строки на предложения, абзацы, пункты, разделы и т.п. В то же время деление на страницы и строки несущественно. Редактор текста *SLIDE* ориентирован на редактирование такой строки. Он может находить в ней определенные слова и заменять их, вставлять одни части и выбрасывать другие, менять порядок частей и т.д. Если затем появляется необходимость выдать эту

строку в виде печатного документа, то SLIDE делает это по указаниям оператора, в которых должны быть заданы, в частности, число строк на странице и число символов в строке.

Кроме того, в редакторе SLIDE реализован ряд других функций типографского редактирования, таких, как составление оглавления печатного документа, соблюдение правил грамматического переноса и др.

Структурно дисплейный редактор текста состоит из программы ведения файлов (собственно редактора) и программы-документатора.

Все функции редактирования текста возложены на клавиатуру дисплея, с помощью которой оператор осуществляет просмотр и *покадровое* редактирование в автономном режиме исходных файлов. При этом во время генерации ОС, использующей работу дисплея с данным редактором, необходимо осуществить деление экрана на две части: *неподвижную* (область просмотра и редактирования кадров) и *подвижную* (служебная область) для ввода команд оператора и вывода сообщений системы. При задании области редактирования рекомендуется резервировать запас строк для возможного расширения кадра исходного текста при его редактировании.

Кроме того, в программе генерации ОС рекомендуется произвести назначение технологическим клавишам дисплея часто используемых команд оператора (см. § 12.4).

Наиболее удобный режим работы с редактором – считывание кадра с исходного файла в область редактирования дисплея, редактирование этой информации с использованием клавиатуры дисплея в автономном режиме (экранное редактирование) и последующая передача отредактированной информации в результирующий файл и (или) печать в заданном формате.

Для осуществления всех перечисленных функций рекомендуется применять такие дисплеи, как ДМ-2000 и ВТА2000(500)-11(15). На рис. 14.1 приведена упрощенная схема алгоритма работы редактора *SLIDE*.

Для управления работой *документатора* в редакторе SLIDE реализована специальная команда задания формата печати выходного документа, в которой задаются необходимые атрибуты для формирования выходного текста; размер страницы, длина строки, расположение названий глав, пунктов и подпунктов и исходное значение их нумерации, расположение нумерации строк и ее исходное значение, смещение от левого края листа, формирование оглавлений. С помощью этой же директивы возможно динамическое изменение размера области редактирования на экране дисплея.

Задание функций форматирования выходного текста возможно также с помощью *спецсимволов*, расставляемых непосредственно в исходном тексте, предназначенном для печати в указанном формате. Распознавание этих спецсимволов и выполнение указанных ими функций осуще-

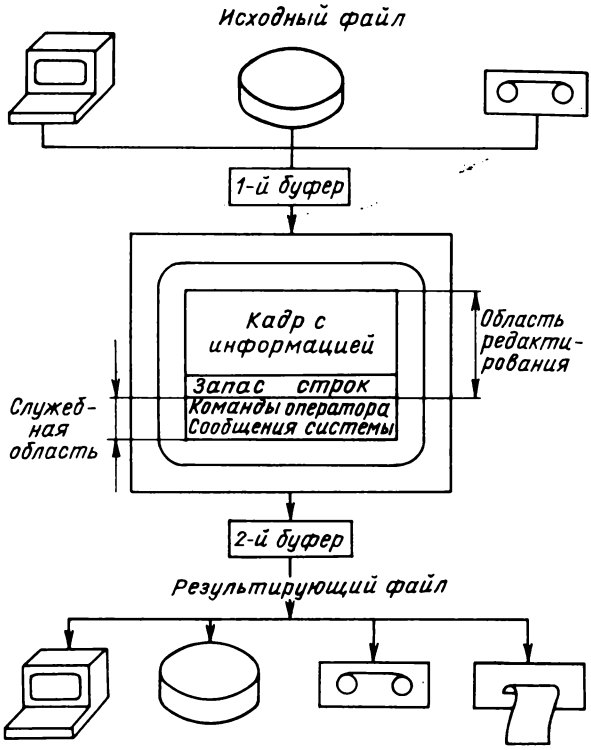


Рис. 14.1. Алгоритм работы редактора SLIDE

ствляет специальный документатор, который включается автоматически перед непосредственной выдачей документа на печать. Все спецсимволы помечаются спецмаркером и делятся на две группы.

К спецсимволам *первой* группы относятся спецсимволы, реализующие функции выделения глав, пунктов и подпунктов, выделения абзацев, пропуска строк, отмены форматирования по всему тексту, генерации текста, установки условных меток и т.п. Эти спецсимволы должны начинаться с первой позиции строки исходного файла.

Спецсимволы *второй* группы реализуют функции отмены форматирования внутри строки и замены условных меток необходимыми координатами. Они могут находиться и внутри строк.

В отредактированном согласно заданному формату документе *SLIDE* делает попытку переноса в словах, состоящих только из русских букв. Лишние пробелы в выходной строке документа распределяются по возможности равномерно. Под словом понимается любой набор символов, оканчивающихся пробелом, точкой, точкой с запятой, двоеточием, восклицательным или вопросительным знаком и запятой. В выходном до-

кументе *SLIDE* между выделенными словами вставляет хотя бы один пробел.

Таким образом, с помощью редактора *SLIDE* впервые появилась возможность готовить машинописные документы. Текстовую информацию большого объема целесообразно заготавливать и редактировать частями, а затем с помощью директив программы *SLIDE* собирать в единый файл для последующей выдачи документа.

14.4. Многофункциональная интерактивная система обработки текстовой информации ОЛИМП

Многофункциональная интерактивная система обработки текстовой информации ОЛИМП является модификацией и развитием ранее разработанных в рамках АСПО СМ ЭВМ средств подготовки и редактирования текстов (*ПОФ, IET, РСИД, SLIDE* и др.) [25].

В системе ОЛИМП каждой группе операторов входного языка поставлена в соответствие отдельная программная подсистема, в том числе подсистемы управления файлами, редактирования, документирования, управления печатью, управления макросредствами, вычислительной обработки и оказания помощи. Иерархическое дерево состава системы приведено на рис. 14.2.

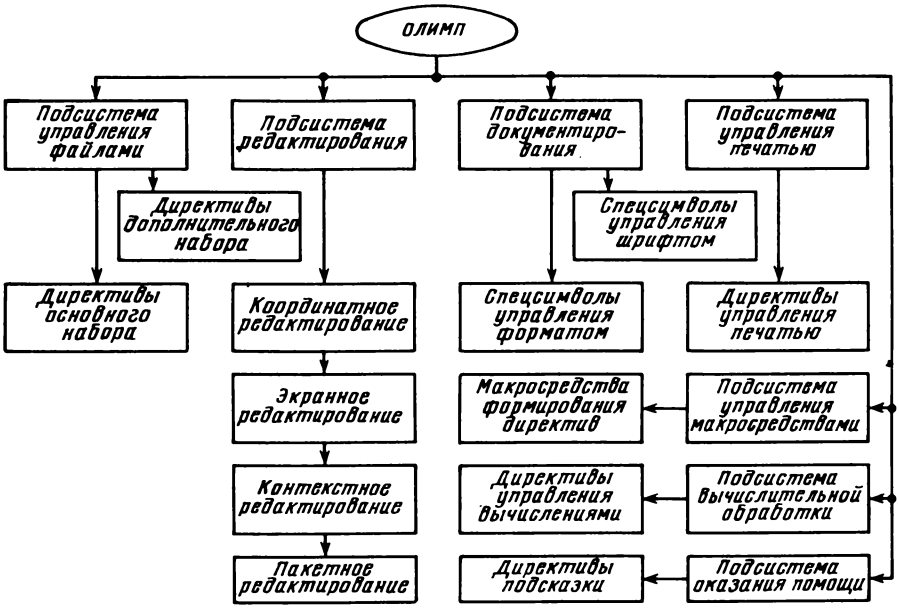


Рис. 14.2. Состав системы ОЛИМП

Процесс подготовки текстового документа состоит из этапов ввода, редактирования, форматирования и вывода отредактированного документа.

Общение пользователя с системой происходит в диалоговом режиме при вводе директив с клавиатуры пульта оператора. Каждая директива обрабатывается непосредственно после ее ввода.

Приведем краткое назначение каждой подсистемы.

Подсистема управления файлами. Эта подсистема предназначена для организации на устройствах внешней памяти входных, промежуточных и выходных файлов, необходимых для работы подсистемы редактирования, документирования и управления печатью. В нее вошли функции, обеспечивающие сервис, необходимый для предварительной обработки данных, как-то: копирование файлов, сравнение и удаление файлов, переименование файлов, нормальные и аварийные закрытия файлов, получение справочников, полная и выборочная распечатка файлов, сортировка файлов.

По желанию пользователя для регистрации его диалога с системой на любом носителе, файле или устройстве печати может быть создан файл с протоколом работы оператора.

Подсистема редактирования. Это основная часть системы *ОЛИМП* предназначена для осуществления функций редактирования текстовых файлов с помощью различных методов редактирования: *экранное, контекстное, координатное* и *пакетное редактирование.*

В этом смысле подсистема редактирования представляет собой дальнейшее усовершенствование и функциональное расширение возможностей редакторов *РСИД, IET, SLIDE.* Существенным добавлением здесь является координатное редактирование, которое также использует экран дисплея и представляет собой дополнительный набор директив, осуществляющих чтение, передвижение, копирование и выдачу информации не по текстовому содержанию записей, а по их номерам в файле или расположению на экране (координатам). При этом координатное редактирование обеспечивает выполнение таких операций, как копирование по номеру записи, перестановка строк на экране, форматирование фрагментов текста в заданных координатах и т.п.

Подсистема документирования. Она предназначена для подготовки к печати в заданном формате текстовых файлов. Эту подсистему можно также рассматривать как дальнейшую модификацию документатора редактора SLIDE. Из введенных расширенный интерес здесь могут вызвать такие функции, как центрирование текста, формирование таблиц, управление шрифтом печати, управление кодировкой символов, задание форматов для различных устройств печати.

Подсистема управления печатью. Подсистема предназначена для выдачи на устройство печати отформатированных текстовых файлов.

Подсистема обеспечивает распечатку отформатированного текста на любом устройстве печати с учетом допустимого шрифта, свойственного

данному устройству. Наиболее эффективным в этом смысле зарекомендовало себя устройство печати СМ 6403.

На этом устройстве с помощью встроенных программных знакогенераторов система ОЛИМП может распечатывать информацию четырьмя типами шрифта: прямым, прямым с подчеркиванием, курсивом (наклонный шрифт) и курсивом с подчеркиванием. Тексты могут содержать прописные и строчные буквы обоих алфавитов. Кроме того, обеспечивается возможность задания нестандартных символов различного начертания.

Подсистема вычислительной обработки (вычислитель). Она предназначена для выполнения некоторых элементарных вычислений десятичной арифметики. Для осуществления этих операций в вычислителе функционируют семь арифметических регистров, каждый из которых может содержать максимально допустимые значения для целых и вещественных чисел. Особенностью вычислителя является его способность выполнять *групповые* арифметические операции над данными, расположенными непосредственно в обрабатываемых текстовых файлах, что широко применяется при работе с бухгалтерской и канцелярской документацией.

Подсистема управления макросредствами. Подсистема основана на использовании собственного встроенного макропроцессора. Все языковые средства этого свободного макропроцессора могут быть применены для составления макросов, расширяющих состав входного языка системы *ОЛИМП.*

Подсистема оказания помощи. Это служба подсказки, предназначенная для получения кратких инструкций по пользованию всей системой *ОЛИМП* и каждой ее директивой в отдельности. Перед внедрением этой подсистемы можно распечатать полный список всех директив системы. С помощью специальных директив имеется возможность перед распечаткой текстов получить информацию о параметрах форматирования.

Обращаться к подсистеме оказания помощи рекомендуется на первых этапах работы с системой *ОЛИМП,* когда еще полностью не изучены все возможности системы и пока пользователь не определил для себя тот минимальный набор директив и спецсимволов, который необходим и удобен, с его точки зрения, для обработки текстов.

14.5. Многофункциональная отладочная программа

Еще раз возвращаясь к технологии создания программного продукта, отметим, что этапу редактирования символьных файлов логически предшествует этап отлади, на котором, собственно, и выясняется, что и как следует отредактировать.

Как показывают специальные исследования [26], на этап отладки в целом уходит до 70% общего времени создания программных систем. Поэтому от того, насколько эффективны существующие средства отлад-

ки, во многом зависит производительность труда программиста-разработчика.

В рамках АСПО созданы достаточно мощные диалоговые средства отладки, которые позволяют вести отладку программ как в режимах *интерпретации,* так и в режимах *реального функционирования.*

Многофункциональная отладочная программа *ДОТЛМ* (в дальнейшем – отладчик) предназначена для отладки в режиме интерпретации одной задачи, состоящей из программ, написанных на МНЕМОКОДе, на уровне адресов отлаживаемых программ внутри раздела. Для подготовки к работе этот отладчик должен быть скомпонован компоновщиком вместе с отлаживаемой задачей, включающей отлаживаемые программы, в один загрузочный модуль.

Загрузочный модуль с отладчиком загружается в раздел с помощью команды оператора **:ВД.** После ввода загрузочного модуля в память отладчик запускается на выполнение командой оператора:

:ВЫ [ЗОВ] ,*ДОТЛМ,p1,p2,p3,p4,p5*

Параметры, заданные в этой команде, принадлежат и передаются отлаживаемой задаче.

После запуска отладчика с устройства ввода, назначенного при компоновке, вводятся управляющие операторы для отладки, по которым отладчик выполняет операцию и выдает на пульт оператора разрешающий символ. По своему функциональному назначению управляющие операторы разделены на 12 основных групп: задания базы, чтения (изменения) содержимого регистров и областей памяти, останова в заданных контрольных точках, печати справочника контрольных точек, отмены остановов, пуска, задания устройства печати, корректировки программ на уровне МНЕМОКОДа, восстановления ветвления программы, слежения, прерывания, завершения отладки.

Ниже приводится краткое описание некоторых операторов отладки.

С помощью *оператора задания базы* пользователь указывает на начальный адрес отлаживаемой программы внутри раздела, который берется из распечатки протокола компоновки загрузочного модуля с отлаживаемой задачей и отладчиком. Значение этого базового адреса добавляется отладчиком ко всем адресным параметрам, которые указываются в управляющих операторах отладки, что позволяет вести отладку программы по адресам листинга, выработанного транслятором.

Операторы чтения (изменения) содержимого регистров и областей памяти позволяют выводить на пульт оператора указанные величины в следующих форматах: восьмеричном, символьном, десятичном, с плавающей запятой, ретранслированном.

Для приостанова отлаживаемой программы при заданных условиях используются *операторы останова в контрольных точках,* которые делятся на пять типов: главный останов, оперативный останов, регистровый останов, запретный останов и останов по счетчику. Условия, при

которых должны произойти остановы, программист указывает в параметрах этих операторов.

В момент останова на пульт оператора выводится стандартное сообщение, содержащее информацию об адресе приостанова, о выполняемой команде в момент останова и всех оперативных регистрах.

Главный останов применяется для приостанова программы по указанному адресу.

Оперативные остановы используются для приостанова выполняемой программы при выполнении одного из следующих условий: адрес операнда равен одному из указанных адресов; содержимое ячейки памяти равно (или не равно) значению, определяемому параметром; изменилось содержимое областей, определяемых начальными и конечными адресами; нарушилась последовательность принимаемых значений в указанных ячейках памяти.

Регистровые остановы необходимы для приостанова отлаживаемой программы, когда выполняется одно из следующих условий: регистр принимает заданное значение; регистр принимает значение, отличное от заданного; изменилось содержимое регистра; нарушилась последовательность принимаемых значений в указанном регистре.

Запретный останов вызывает приостанов выполнения программы при попытке записи или перехода в указанные области памяти.

Останов по счетчику вызывает приостанов выполнения программы, когда указанный участок этой программы будет пройден заданное число раз.

Операторы печати справочника используются для получения информации о контрольных точках, установленных операторами останова.

Операторы корректировки программ на уровне МНЕМОКОДа играют роль, аналогичную РСИД, редактируя программы непосредственно в памяти раздела. С их помощью можно вставлять, заменять и удалять участки программ, применяя адреса листинга исходной программы на МНЕМОКОДе.

Оператор прерывания приостанавливает отлаживаемую программу при ее зацикливании в процессе выполнения. На пульт оператора последует сообщение о точке приостанова в момент прерывания.

Оператор слежения выдает сообщения о состоянии отлаживаемой программы после каждой выполняемой команды.

14.6. Отладчик программ на языке ФОРТРАН

Отладчик программ на ФОРТРАНе (в дальнейшем – отладчик) предназначен для отладки программ на уровне входного языка. Отладчик представляет собой несегментированную программу в объектном формате, объединяемую компоновщиком в один загрузочный модуль с отлаживаемыми программными единицами.

Отладчик работает в режиме диалога с человеком, обеспечивая выполнение следующих функций с помощью управляющих команд оператора, вводимых с клавиатуры пульта оператора после выдачи очередного разрешающего символа:

RUN – запуск отлаживаемой программы с любого оператора. Если в качестве параметра в операторе **RUN** указано имя, то осуществляется выполнение программной единицы с заданным именем;

PAUSE – приостанов выполнения программы в контрольных точках. При выполнении этой команды происходит приостанов выполнения отлаживаемой программы перед оператором с указанным номером. Номер оператора выводится на экран пульта оператора;

GOTO – изменение последовательности выполнения операторов. При выполнении этой команды управление передается от одного оператора другому с указанным номером;

LET – присваивание значений указанным переменным. По этой команде простой переменной или переменной с индексами присваивается значение,указанное в параметрах команды. Значения переменным можно присваивать либо перед выполнением программы, либо когда программа находится в приостанове;

TRACE – выполнение программы в режиме слежения. По этой команде осуществляется пооператорное выполнение отлаживаемой программы с выводом на печать значений указанных переменных перед выполнением каждого оператора области слежения. Прослеживание указанной области остается в силе до выбора новой области слежения или до отмены данного выбора;

DUMP – распечатка значений указанных переменных. В процессе выполнения этой команды на пульт оператора по заданной спецификации выводятся номер оператора и значения переменных, перечисленных в параметрах команды;

DELEATE PAUSE (DP) – отмена остановов в контрольных точках;

ASSING (AS) – слежение за изменением значений переменных. При выполнении программы под управлением отладчика происходит постоянное наблюдение за значением переменных, перечисленных в параметрах команды **AS**. Как только значение какой-нибудь переменной из списка изменяется, на печать выдается идентификатор этой переменной и измененное значение; **STOP (ST)** – завершение выполнения отлаживаемой программы.

14.7. Система интерактивной отладки программ на уровне МАКРОАССЕМБЛЕРа (ТЕМП)

Система интерактивной отладки программ на уровне МАКРОАССЕМБЛЕРа является средством подготовки и отладки программ в *режиме реального функционирования.*

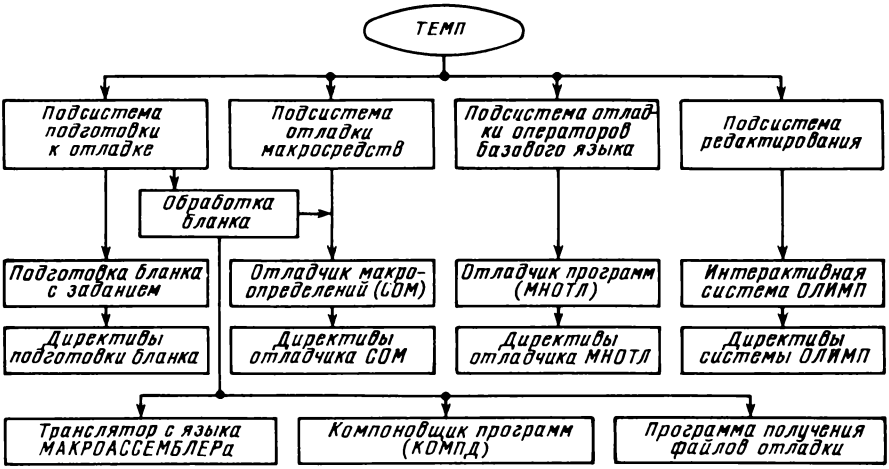


Рис. 14.3. Состав системы ТЕМП

По своему функциональному назначению ТЕМП представляет собой комплект программных модулей, объединенных в четыре основные подсистемы: подсистему подготовки к отладке, подсистему отладки макросредств, подсистему отладки операторов базового языка и подсистему редактирования [27].

В целом система представляет собой древовидную иерархию программных модулей, во главе которой стоит управляющая программа системы—монитор. Иерархическое дерево состава системы приведено на рис. 14.3.

Подсистема подготовки к отладке. Эта подсистема обеспечивает подготовку загрузочного модуля с отлаживаемыми задачами. Все задания на подготовку к отладке программ задаются на стандартном бланке, вызванном с помощью директивы монитора **БЛ** в область заполнения экрана пульта оператора.

Основная работа монитора ТЕМП заключается в чтении бланка, заполненного пользователем, и последовательном выполнении пунктов, указанных в этом бланке. При этом монитор предоставляет пользователю возможность с помощью специальных директив подготовки бланка заполнить чистые бланки, прочитать и модифицировать ранее заполненные бланки, распечатать и удалить любой бланк. Обрабатывая бланк, монитор последовательно передает управление соответствующему модулю системы и по окончании работы модуля заполняет отчетной информацией соответствующую строку бланка.

В бланке задания можно указать выполнение трансляции программ с возможной предварительной макрообработкой, компоновку этих программ в единый загрузочный модуль и получение файлов отладки, необ-

ходимых для осуществления отладки этих программ на уровне операторов базового языка. Пример заполненного бланка приведен ниже:

ЗАГР.МОДУЛЬ = КЗТЕМП ::14 ПН ДИСКА ПОЛЬЗОВАТЕЛЯ = 16 КЛЮЧ =			ПРОТОКОЛ.КОМП КОМПОНОВАТЬ – ПЕЧАТЬ-9 ФИКУС		
ОСНОВНЫЕ ПРОГРАММЫ	ИСПОЛЬЗУЕМЫЕ БИБЛИОТЕКИ	БИБЛИОТЕКА МАКРО	ОТЛАДКА		МН КД
			МО	МН	
SORT		#БМСП	+		
SORTC	#БПСП				+
ОБМАС				+	+
	СЛВЕК			+	+
	СЧБИТ			+	+

В результате обработки бланка будет сформирован загрузочный модуль, содержащий задачи, подготовленные к отладке.

Подсистема отладки макросредств. Эта подсистема обеспечивает выполнение отладочных операций при отладке макроопределений с помощью специально разработанного для этих целей отладчика макроопределений СОМ. Отладчик макроопределений представляет собой макропроцессор МП25, работающий в режиме отладки, который предоставляет в распоряжение пользователя набор отладочных директив, позволяющих выполнить в отлаживаемом макроопределении такие функции, как чтение и модификация переменных, останов по макровызову, останов и запуск с любой точки макроопределения, редактирование операторов макроопределений, прослеживание выполнения участков макроопределения, прослеживание очередности появления макровызова с соответствующими распечатками фактических параметров, выполнение макроопределений в автоматическом режиме с прерываниями.

Запуск подсистемы отладки макросредств осуществляется отметкой в графе макроотладки бланка задания.

Подсистема отладки операторов базового языка. Все функции подсистемы отладки выполняет отладчик программ МНОТЛ, который компонуется вместе с отлаживаемыми задачами в единый загрузочный модуль. Отладчик МНОТЛ является дальнейшим усовершенствованием средств отладки АСПО, который обеспечивает отладку до 64 задач в режиме реального функционирования с использованием листингов этих задач, полученных в результате трансляции с помощью транслятора МНКД.

Отладчик МНОТЛ позволяет производить чтение и изменение содержимого памяти в восьмеричном, десятичном, символьном и шестнадцатеричном формате, а также в формате чисел с плавающей запятой основной или двойной длины.

Для приостанова выполнения отлаживаемой программы используется аппарат регистрации останова в контрольной точке и аппарат регистрации условных остановов. Задание условных остановов позволяет зафиксировать запись или переходы в указанную область памяти.

```
-ПРКЗТ ::15
ДА
-26
```

РЕЗУЛЬТ.ФАЙЛ ТРАНС./МАКРО	ФАЙЛЫ ОТЛАДКИ
SORTC	
SORTД	
ОБМАСД	ОБМ/ФО
СЛВЕКД	СЛВ/ФО
СЧБИТД	СЧБ/ФО

Выполнение программы можно производить в *автоматическом* режиме, в *пошаговом* и в режиме *слежения*. В автоматическом режиме и в режиме слежения приостанов выполнения программы осуществляется при каждом выполнении условия останова, а также в любой момент отладки. Выполнение программы можно прервать, набрав с пульта оператора любой символ, инициируя тем самым приостанов выполнения программы. В каждом из этих случаев на пульт оператора выводятся стандартное сообщение о точке приостанова (с указанием имени задачи, имени программы, текущего адреса команды, на которой произошел останов), значение всех регистров общего назначения и причины приостанова.

Для корректировки программ на уровне МНЕМОКОДа в отладчике МНОТЛ имеется возможность вставлять, заменять и удалять участки программ, используя адреса или метки листинга исходной программы.

Если во время отладки накапливается большое число ошибок, то рекомендуется во избежание новых периодически производить редактирование программ. При этом нет необходимости каждый раз перетранслировать программу, получать загрузочный модуль и файлы отладки.

Все программные модули, входящие в систему ТЕМП, могут выполняться как автономно, вне системы ТЕМП, так и под управлением монитора подсистемы подготовки программ к отладке.

В заключение отметим, что, несмотря на развитые средства отладки и редактирования, создание больших программных продуктов представляет собой все же длительный и трудоемкий процесс. Поэтому пользователь должен стремиться по возможности полнее использовать уже имеющиеся проблемно-ориентированные ПО. Пятый, заключительный раздел книги посвящен рассмотрению пакетов и библиотек прикладных программ, поставляемых пользователю УВК СМ-2М.

РАЗДЕЛ ПЯТЫЙ

ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Глава 15. ДИСКОВЫЙ ПАКЕТ ПРОГРАММНЫХ МОДУЛЕЙ ГЕНЕРАЦИИ ЗАДАЧ СБОРА И ОБРАБОТКИ В АСУ ТП

15.1. Назначение и состав пакета

В предыдущих главах были рассмотрены языковые и системные средства, обеспечивающие создание, отладку и функционирование программных систем, реализующих задачи пользователя.

В то же время существуют классы задач, которые обладают общими функциями, и поэтому нет смысла для каждой задачи заново создавать программы, отвечающие этим функциям. Для решения таких классов задач в АСПО предлагаются комплекты программных модулей, при разработке которых, так же как и при разработке операционных систем, использован агрегатно-модульный принцип. Указанные комплекты составляют проблемно-ориентированное программное обеспечение АСПО, которое в свою очередь делится на две части: пакеты прикладных программ и библиотеки прикладных программ.

Характерной особенностью пакетов является наличие входного языка, на котором пользователь излагает свои требования к системе, генерируемой или компонуемой на основе пакета, и языка управления пакетом—набора команд, обеспечивающего диалог пользователя с системой.

Библиотеки в отличие от пакетов представляют собой просто набор подпрограмм в перемещаемом формате, допускающих вызов на языках МНЕМОКОД, ФОРТРАН и др.

В проблемно-ориентированное обеспечение входят:

дисковый ППМ генерации задач сбора и обработки информации в АСУ ТП (ДППМ СОИ);

ППМ для компоновки диалоговых многозадачных систем реального времени;

ППМ для организации банков данных;

ППМ системы выдачи акустической информации;

ППМ связи с ЕС ЭВМ;

ППМ компоновки систем для работы с аппаратурой КАМАК;

ППМ для компоновки ОС многомашинных комплексов;

библиотека численного анализа и статистики;

библиотека программ оптимизации и исследования операций;

библиотека программ графики;

библиотека подпрограмм для цветного графического терминала;

библиотека программ для полутоновых дисплеев и др.

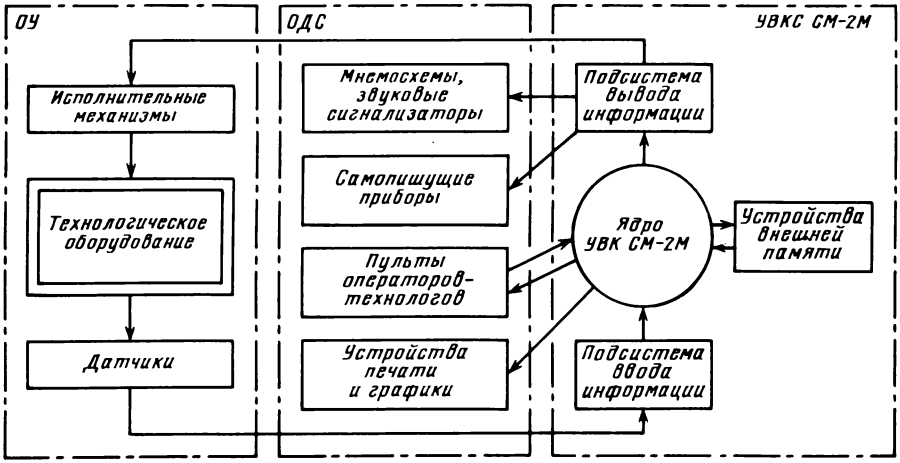


Рис. 15.1. Структурная схема АСУ ТП на базе УВКС СМ-2М

Дисковый ППМ СОИ является основным проблемно-ориентированным пакетом АСПО [29]. Как и пакет для генерации ОС, ДППМ СОИ включает в себя библиотеки макроопределений и библиотеки подпрограмм. Он состоит из трех частей: основная версия (файлы *БМ1, БМ2, БМ3, БМ4, ББЛ1, ББЛ2*), дополнения к пакету (файлы *БМ5, БМ6, ББЛД*) и средства отладки (файлы *БОТЛ, БМНСХ, ФАКТВ*).

Главное назначение пакета – создание программных систем, обеспечивающих функционирование АСУ ТП.

Рассмотрим структурную схему автономной АСУ ТП на базе УВКС СМ-2М (рис. 15.1). Технологический объект управления (ОУ) включает в себя технологическое оборудование (одну или несколько установок, реализующих протекание единого ТП), датчики, регистрирующие параметры состояния ТП, и исполнительные механизмы, позволяющие изменять режимы ТП.

Ввод информации от датчиков в ядро УВК СМ-2М осуществляется через подсистему ввода, включающую в себя соответствующие модули УСО и разветвители, вывод информации и управляющих воздействий – через подсистему вывода. Оперативно-диспетчерская служба (ОДС) может быть оборудована пультами операторов-технологов (на базе дисплеев типа ВТА2000-10), устройствами печати и вывода графической информации, самопишущими приборами, мнемосхемами, устройствами звуковой сигнализации.

Дисковый ППМ СОИ обеспечивает прежде всего информационные функции АСУ ТП – сбор и обработку информации, контроль состояния ТП. Эти функции являются достаточно общими для различных АСУ ТП, определяются в основном характеристиками датчиков и конфигу-

рацией технических средств и лишь в незначительной степени свойствами управляемого ТП.

Помимо информационных функций пакет позволяет частично обеспечить и функции диспетчерского управления. Задачи диспетчерского управления, которые целиком определяются спецификой ТП, должны быть разработаны самим пользователем, но пакет позволяет включить эти задачи в систему, обеспечить обмен данными с информационными задачами, а также предоставляет библиотеку подпрограмм, вызываемых на МНЕМОКОДе, ФОРТРАНе или АЛГОЛе, для облегчения программирования этих задач.

15.2. Типы задач, генерируемых пакетом

Дисковый ППМ СОИ позволяет сгенерировать шесть типов задач, из которых затем компонуется многозадачная система. Три типа задач являются основными (в скобках указаны имена первых макрокоманд описания задач); задача сбора и обработки (ЗАДАЧА), задача связи с пультом оператора-технолога (ПТО), задача периодической печати (ОТЧЕТ). Другие три типа задач являются вспомогательными (служебными): задача поиска информации (ПОИСК), задача загрузки сегментов (ЗАГРС), задача буферированного вывода на печать (ПЕЧАТЬ).

Задачи сбора и обработки, ПТО, ОТЧЕТ и ПОИСК могут быть как ОЗУ-резидентными, так и сегментированными. Задачи ЗАГРС и ПЕЧАТЬ являются несегментированными ОЗУ-резидентными.

В генерируемой системе может быть одна задача ПТО и по несколько задач других типов. Компоноваться все они могут как в один, так и в несколько загрузочных модулей.

Задача сбора и обработки осуществляет опрос датчиков с заданной цикличностью (от 100 мс до 8 ч) или прием сигналов по инициативе датчиков. Введенные значения параметров ТП обрабатываются по обязательным или выбираемым пользователем алгоритмам из числа реализованных в программах обработки пакета. В пакете реализованы перевод введенных значений из физической шкалы во внутреннюю (в виде целых чисел 0–4095) и обратно, контроль выхода параметров за заданные границы (уставки) или из заданного состояния, осреднение, сглаживание, вычисление поправок, запоминание предшествующих значений. Обработанные значения параметров и заданные пользователем константы алгоритмов обработки хранятся в информационных массивах задач сбора и обработки, образующих базу данных системы.

Задача ПТО обеспечивает вывод информации о состоянии параметров ТП на пульты операторов-технологов, устройства печати, мнемосхемы, самопишущие приборы. Она реализует также диалог оператора с системой по предлагаемому пакетом набору команд.

Задача ПТО обслуживает до восьми различным образом скомпонованных рабочих мест оператора, каждое из которых должно включать

один пульт оператора и может иметь свой набор устройств отображения информации. Сообщения о возникновении и устранении нарушений уставок параметров могут одновременно выдаваться на один, два или три пульта и устройства печати, закрепленные за рабочими местами (или на одно главное устройство печати). Диалог с системой может осуществляться одновременно со всех рабочих мест (обработка запросов при этом, естественно, выполняется последовательно).

Задача ОТЧЕТ осуществляет периодическую печать информации о параметрах ТП с задаваемым периодом на выделенное устройство печати. При генерации задачи ОТЧЕТ пользователь устанавливает как формат бланка, так и тип выводимой информации и ее расположение в строках бланка.

Служебная *задача ПОИСК* обеспечивает доступ к базе данных задач сбора и обработки со стороны задач ПТО и ОТЧЕТ по символьному идентификатору (шифру), который присваивается каждому параметру ТП при описании задач сбора и обработки.

Задача ПОИСК должна включаться в каждый загрузочный модуль, содержащий задачи сбора и обработки.

Поиск информации в базе данных по всем загрузочным модулям происходит последовательно, начиная с наиболее приоритетного, условно называемого *главным*. Если запрашиваемая информация не найдена задачей ПОИСК главного загрузочного модуля, то запрос передается задаче ПОИСК следующего по приоритету модуля и т.д.

Задача ЗАГРС обслуживает задачи с диск-резидентными сегментами (ЗАДАЧА, ПТО, ОТЧЕТ, ПОИСК), осуществляя ввод сегментов с диска и вывод отработанных сегментов на диск. В состав каждого загрузочного модуля системы, содержащего задачи с диск-резидентными сегментами, должна быть включена своя задача ЗАГРС.

Задача ПЕЧАТЬ обеспечивает буферирование вывода на печать и может применяться для повышения быстродействия задач, осуществляющих вывод сообщений на печать (ПТО, ОТЧЕТ), если устройства печати не обладают достаточным быстродействием.

15.3. Организация связей между задачами

Связи между задачами системы осуществляются для разных задач на разных уровнях: через общую область памяти, общие таблицы, аппарат обмена информацией между задачами.

Через общую область памяти организуется связь между задачами сбора и обработки или дополнительными задачами пользователя, размещенными в одном загрузочном модуле. Такая связь необходима в тех случаях, когда для вычисления поправок на значения параметров в одной задаче требуются значения параметров в той же или в другой задаче, например, значения давления и температуры для расчета поправок к расходам газовых потоков, поправок на температуру свободных концов

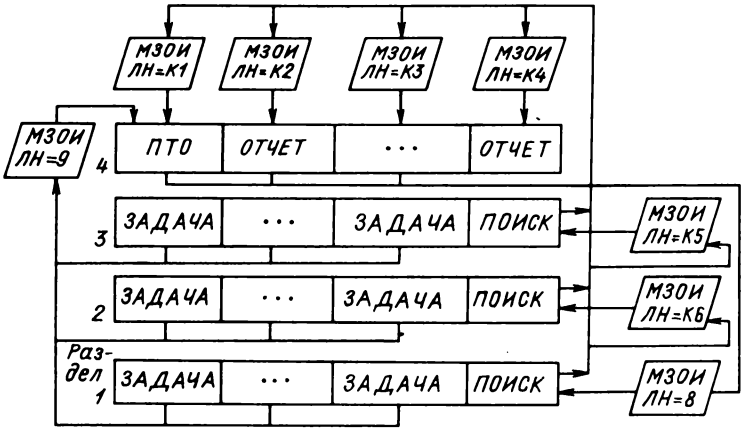


Рис. 15.2. Схема взаимодействия задач, генерируемых пакетом

термопар. Распределение общей памяти производится пользователем при генерации задач.

Общие таблицы автоматически генерируются для ОЗУ-резидентных задач сбора и обработки, описываемых в общей программе генерации. Такие задачи должны компоноваться в один загрузочный модуль. Тогда доступ к этим таблицам будет осуществляться одной задачей ПОИСК, скомпонованной в том же модуле. Задачи сбора и обработки с сегментами на диске независимы друг от друга и могут компоноваться в разные загрузочные модули.

Связи между задачами сбора и обработки, ПТО и ОТЧЕТ организованы через псевдоустройства межзадачного обмена информацией (МЗОИ), необходимое число которых должно быть включено в операционную систему при ее генерации. Схему связей между этими задачами рассмотрим на примере, представленном на рис. 15.2, где изображены четыре загрузочных модуля с ОЗУ-резидентными задачами: задачи сбора и обработки (ЗАДАЧА) включены в три загрузочных модуля вместе с задачами ПОИСК, задачи ПТО и ОТЧЕТ включены в отдельный модуль.

Запросы от задач ПТО и ОТЧЕТ на поиск информации в базе данных поступают в задачу ПОИСК главного загрузочного модуля через псевдоустройство МЗОИ, всегда имеющее логический номер, равный 8. Если ответ в главном модуле не найден, запрос через соответствующие псевдоустройства МЗОИ с логическими номерами *ЛН = К6, ЛН = К5* на схеме передается задачам ПОИСК других загрузочных модулей. Ответ от задач ПОИСК задачи ПТО и ОТЧЕТ получают через соответствующие для каждой задачи псевдоустройства МЗОИ.

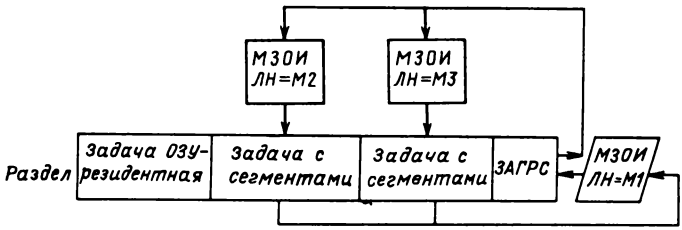


Рис. 15.3. Схема взаимодействия задач с сегментами на диске с задачей ЗАГРС

Через специально выделенное псевдоустройство МЗОИ с *ЛН = 9* от задач сбора и обработки поступает информация о нарушениях уставок ТП (границ технологического контроля) в задачу ПТО.

Рассмотрим теперь связь задач с диск-резидентными сегментами и задачей ЗАГРС. Заметим, что диск-резидентные сегменты могут быть двух типов: программные и табличные. Необходимость сегментирования той или иной задачи определяется пользователем, тип сегмента – спецификой задачи. Задача ПОИСК может иметь только программные сегменты на диске, задача сбора и обработки и ОТЧЕТ – только табличные, задача ПТО – как программные, так и табличные.

В один загрузочный модуль могут быть скомпонованы несколько задач с табличными сегментами, но только одна задача с программными сегментами. На рис. 15.3 изображен загрузочный модуль с двумя задачами, имеющими диск-резидентные сегменты, и задачей ЗАГРС, обслуживающей этот модуль.

Запросы на загрузку сегментов поступают на псевдоустройство МЗОИ задачи ЗАГРС (логический номер равен *М1* на схеме), ответ – на псевдоустройство МЗОИ задачи с сегментами, сделавшей запрос (логический номер равен *М2* или *М3*).

Задача ПЕЧАТЬ также требует одно псевдоустройство МЗОИ для приема запросов от задач, выполняющих ввод сообщений на печать (ПТО, ОТЧЕТ).

Отметим, что для задач ПТО и ОТЧЕТ можно совместить псевдоустройства МЗОИ, через которые эти задачи получают ответы от задач ЗАГРС или ПОИСК.

15.4. Структура программ генерации задач

Макрокоманды описания задач подробно изложены в руководстве по пользованию ДППМ СОИ. Рассмотрим лишь структуру программ генерации задач и назначение некоторых макрокоманд, следуя основной версии пакета.

Генерация задач сбора и обработки. Все параметры ТП, подлежащие обработке, распределяются пользователем по задачам сбора и обра-

ботки. Этих задач, как уже говорилось, может быть несколько, и они могут располагаться в разных загрузочных модулях.

В одну задачу объединяются параметры с одним периодом опроса в случае их обработки по времени либо параметры, обрабатываемые по инициативе датчиков. Соответственно задачи сбора и обработки делятся на неинициативные и инициативные. Каждой задаче присваивается имя, приоритет, период вызова и фаза начального запуска. Если задача ОЗУ-резидентная, то идентификатор имени должен начинаться с буквы *S*, для задачи с диск-резидентными сегментами – с буквы *D*.

Внутри задачи параметры распределяются по сегментам. Сегмент объединяет параметры с периодом опроса, кратным периоду вызова задачи. Каждый сегмент имеет свой номер. Неинициативные задачи могут содержать несколько сегментов, инициативные – только один.

Внутри сегмента параметры разбиваются на группы. В одну группу могут входить только параметры с одним и тем же типом датчика, со смежными адресами подключения датчиков в подсистеме ввода, с однотипной обработкой. Разбиение параметров на группы является обязательным. Группа может состоять из одного параметра.

Указанному разбиению параметров соответствует порядок чередования макрокоманд описания задач сбора и обработки: **ЗАДАЧА, СЕГМЕНТ, ГРУППА, ПАРАМЕТР.**

Структура программы генерации нескольких задач сбора и обработки, генерируемых совместно, имеет следующий вид:

```
      ЗАДАЧА                  начало описания первой задачи
      СЕГМЕНТ
      ГРУППА                  начало описания первой группы первого сегмента
      ПАРАМЕТР
{макрокоманды описания }
 параметра             }
      ПАРАМЕТР
{макрокоманды описания }
 параметра             }
      . . .
      ГРУППА                  начало описания второй группы
      ПАРАМЕТР                первого сегмента
      . . .
      ПАРАМЕТР
      . . .
      СЕГМЕНТ
      ГРУППА                  начало описания первой группы
      ПАРАМЕТР                второго сегмента
      . . .
      ЗАДАЧА                  начало описания второй задачи
      СЕГМЕНТ
      ГРУППА
      . . .
      КОНЕЦ
```

В макрокоманде **ЗАДАЧА** указываются период вызова и фаза начального запуска, приоритет, логические номера псевдоустройств ГИУ для инициативных задач или МЗОИ для связи с задачей ЗАГРС, размер общей области памяти, идентификатор входной точки при наличии подпрограммы внешних условий запуска. В макрокоманде **СЕГМЕНТ** указывается его номер и кратность периода вызова сегмента периоду вызова задачи.

В макрокоманде **ГРУППА** до шести операндов описывают способ подключения датчиков группы: число датчиков, логический номер подсистемы ввода, адреса подключения первого датчика, первого модуля группового подключения, число задействованных входов в модуле; до 30 операндов описывают способ обработки информации от датчиков группы: имя отдельно генерируемой программы обработки, перечень входящих в нее подпрограмм и некоторые их константы, указание о необходимости индикации состояния параметров на мнемосхеме.

В макрокоманде **ПАРАМЕТР** кроме шифра параметра, данного в метке (до 5 символов), можно указать полное имя параметра (до 30 символов), которое может фигурировать в сообщениях оператору.

Макрокоманда **ПАРАМЕТР** повторяется после макрокоманды **ГРУППА** столько раз, сколько параметров включено в группу. После каждой макрокоманды **ПАРАМЕТР** следуют макрокоманды описания параметра. В пакете предусмотрено 14 таких макрокоманд. Их имена и содержание их операндов приведены ниже:

ШКАЛА – тип градуировки, начало и конец шкалы в физических единицах, размерность;

РМО – номера рабочих мест для выдачи сообщений о нарушениях уставок;

ВГ – значения возможных границ измерений;

ТГ – номера элементов мнемосхемы для индикации нарушений границ;

ТК – значение зоны нечувствительности при вхождении параметра в норму, указания на таблицы соответствия номеров границ контроля их названиям;

ПДТ – адреса в общей области памяти (ООП) значений давления и температуры для вычисления поправок к значениям расходов газовых потоков;

СГЛ – значения коэффициентов сглаживания;

ОПЗС – адрес в ООП для записи среднего значения;

СКИ – значения допустимой скорости изменения;

ОП – адрес в ООП для записи текущего значения;

ИСТ – количество сохраняемых значений в массиве истории параметра;

РЕЗ – шифры попарно резервируемых параметров и значение допустимого рассогласования;

ОДП – код нормального состояния дискретного параметра, номер элемента мнемосхемы для индикации смены состояния;

ДОП – имя дополнительной подпрограммы пользователя и значения констант.

Некоторые макрокоманды действуют только в пределах описания параметра (например, запись текущих значений в общую область памяти), другие – в пределах описания группы, что позволяет указывать их только 1 раз для всей группы параметров (например, описание шкалы для одинаковых датчиков).

В макрокомандах описания параметров предусмотрена возможность включения в систему дополнительных подпрограмм, написанных пользователем, для обработки информации от датчиков с нестандартной (не предусмотренной пакетом) шкалой и (или) по нестандартному алгоритму обработки.

Остановимся теперь на понятиях *тип группы, программа обработки и шифр параметра.*

Тип группы параметров. В пакете тип группы трактуется шире, чем просто тип датчиков, в нем учитывается вид и характер ввода поступающей информации.

По виду информации различают датчики аналоговые (непрерывные сигналы – датчики давления, термопары и пр.), число-импульсные (подсчет событий – расходомер типа "турбоквант", счетчик расхода электроэнергии); дискретные (обнаружение событий, например, состояние оборудования: включен–выключен, открыто–закрыто); кодовые (цифровой ввод от различных преобразователей – вольтметров, частотомеров и пр.).

Аналоговые и кодовые сигналы могут вводиться только по времени, дискретные и число-импульсные – как по времени, так и по инициативе (соответственно они подключаются через разные модули УСО).

В перечень параметров кроме фактических, соответствующих каждому входу в подсистему ввода, могут быть включены фиктивные и фиктивные резервируемые, которые обрабатываются системой точно так же, как и фактические. К фиктивным относятся параметры, значения которых вводятся не через подсистему ввода, а с пульта оператора (например, значение температуры окружающей среды, результаты экспресс-анализа) или программным путем через псевдоустройства МЗОИ из дополнительных задач пользователя (например, результаты косвенных измерений).

Фиктивный резервируемый параметр обслуживает два входа в подсистему ввода. Он применяется при необходимости резервирования (дублирования) отдельных датчиков и (или) схем их подключения, т.е. подключаются либо два одинаковых датчика на разные входы, либо один датчик на два входа.

Таким образом, в ДППМ СОИ различают девять типов групп параметров. Шифры типов групп приведены в табл. 15.1.

Программа обработки. Для каждой группы параметров может быть сгенерирована программа обработки из набора подпро-

Таблица 15.1

Шифр группы	Шифр программы обработки	Тип группы параметров
АНС	**АНС, АНС1, АНС2, АНС3**	Аналоговые (по времени)
ДС	**ДС**	Дискретные (по времени)
ДСИ	**ДС1, ДС2**	Дискретные (по инициативе)
ЧИС	**ЧИС, ЧИС1, ЧИС2, ЧИС4**	Число-импульсные (по времени)
ЧИСИ	**ЧИС3**	Число-импульсные (по инициативе)
ЦВВ	**ЦВВ**	Кодовые (с цифровым вводом)
ФАНС	**ФАНС, ФАНС1**	Фиктивные аналоговые (фиктивные или резервируемые)
ФДС	**ФДС**	Фиктивные дискретные (резервируемые)
ФЧИС	**ФЧИС**	Фиктивные число-импульсные (резервируемые)

грамм, включенных в пакет, либо с привлечением дополнительных подпрограмм пользователя. Структура программы их генерации имеет вид

⟨ *имя* ⟩ **ПРОГР [АММА]** ⟨ *тип* ⟩
макрокоманды включения подпрограмм
⟨ *имя* ⟩ **ПРОГР [АММА]** ⟨ *тип* ⟩
{ макрокоманды включения подпрограмм }
...
КОНЕЦ

Имя программы обработки и перечень включаемых подпрограмм должна совпадать с указываемыми в операндах макрокоманды **ГРУППА.** Типов программ обработки 17, т. е. больше, чем типов групп. Это обусловлено тем, что в типах программ обработки учитывается дополнительно уровень сигналов и способ подключения датчиков. Шифры типов программ приведены в табл. 15.1.

Имена макрокоманд включения подпрограмм с указанием функций каждой подпрограммы приведены ниже:

ПЛИН – линеаризация измеренных значений аналоговых параметров;
ПКВГ – контроль измеренных значений на соответствие возможным границам (границам измерений);
ПКТГ – контроль измеренных значений на соответствие технологическим границам (до 60 границ);
ПКСКИ – контроль по допустимой скорости изменения;
ППДТ – введение поправок по давлению и температуре к расходам газовых потоков;
ППТХС – введение поправки на температуру холодного спая (свободных концов) термопар;
ПСГЛ – сглаживание измеренных значений параметров;
ПУСР – усреднение измеренных значений параметров на неограниченном или заданной длины интервале;
ПЗОП – запись обработанных значений в общую область;

ПЗИСТ – запоминание истории параметра (заданного количества значений);
ПДОП – дополнительная программа пользователя.

Параметры подпрограмм (константы алгоритмов обработки) указываются частично в операндах макрокоманд включения и макрокоманды **ГРУППА** (общие константы для группы параметров) и, главным образом, в макрокомандах описания параметров (индивидуальные константы).

Если программа обработки для группы параметров не описана, то выполняется минимальная обработка, объем которой определяется типом группы.

Шифр параметра. В основной версии пакета шифр параметра может состоять не более чем из пяти символов: первые два символа указывают тип установки (объекта, задачи), третий символ – тип параметра (Т – температура, Р – давление, F – расход газа и т. д.), последние два символа – номер параметра. Примеры шифров: КСТ12, НР43, FO4 (первые два символа можно опустить). При таком способе шифрации типов установок может быть не более 32, типов параметров – не более 16, номеров параметров – не более 100 (0–99).

Генерация задачи ПТО. Программа генерации ПТО имеет следующий вид (указаны только имена макрокоманд, точками обозначено повторение команды либо группы команд):

ПТО
РАБМО – определение состава оборудования на каждом рабочем месте оператора
...
ТАБЛС – присвоение имен границам технологического контроля и построение
НГ таблиц их соответствия
...
НГ
...
ФОРМАТ – определение формата сообщения о нарушениях границ
СИМВ – заполнение формата символьной информации
СОПУТ – описание групп сопутствующих параметров, вызываемых одновременно
СМП – описание самопишущих приборов
...
МНСХ – описание способа подключения группы элементов мнемосхемы
...
ТКО – определение состава команд оператора для каждого пульта
КОМОП
...
КОМОП
...
КОНЕЦ

В операндах команды **ПТО** указывается приоритет, число рабочих мест, обслуживаемых данной задачей, ЛН главного устройства печати

для вывода сообщений о нарушении уставок, ЛН псевдоустройства ГИУ.

Число макрокоманд **РАБМО** должно соответствовать количеству рабочих мест операторов, номер рабочего места определяется порядком следования этих макрокоманд. Операндами **РАБМО** являются ЛН пульта и устройства печати, номер дублирующего рабочего места, если это необходимо, количество параметров, сообщения о которых сохраняются для оператора (до 1000) в буфере (таблице нарушений) в течение всего времени их существования.

Для удобства работы оператора границам технологического контроля присваиваются имена (до 4 символов), например НГ27, ВГ15. Количество границ контроля (до 30 нижних и до 30 верхних) указывается в макрокоманде **ТАБЛС**, имена – в макрокомандах **НГ.**

Сообщение об изменении состояния параметра (для аналоговых параметров – нарушение возможных границ измерения, границ технологического контроля, допустимой скорости, для дискретных – переход в другое состояние) выдаются на пульт оператора по инициативе задач сбора и обработки либо по команде оператора. В пакете предусмотрен стандартный формат сообщений на экран:

⟨ *шифр параметра* ⟩ = ⟨ *текущее значение* ⟩ ⟨ *размерность параметра* ⟩

⟨ *название границы контроля* ⟩ = ⟨ *значение границы контроля* ⟩

Например, КСТ12 = 270 ГРАД ВГ12 = 250.

С помощью макрокоманд **ФОРМАТ** и **СИМВ** можно описать формат сообщения, отличный от стандартного, для любого параметра и любого номера его состояния.

В макрокоманде **СОПУТ** указываются шифры параметров, сопутствующих главному, описываемому первым. Например, пусть макрокоманда **СОПУТ** имеет вид

СОПУТ F36, T40, P32

Это означает, что одновременно с вызовом параметра F36 (расход газа) будут вызваны на индикацию параметры T40 (температура) и P32 (давление).

Описания способов подключения самопишущих приборов и элементов мнемосхем аналогичны друг другу. Они производятся макрокомандами **СМП** и **МНСХ** для групп приборов или элементов со смежными адресами подключения в подсистемах вывода информации. В каждой макрокоманде **СМП** или **МНСХ** указывается ЛН подсистемы вывода, адрес подключения первого прибора или элемента мнемосхемы и их количество. Сквозная нумерация самопишущих приборов или элементов мнемосхемы определяется по умолчанию порядком следования соответствующих макрокоманд.

Для каждого рабочего места с помощью макрокоманд **ТКО** и **КОМОП** составляется таблица команд оператора (она может быть общей для нескольких или всех рабочих мест). В макрокоманде **ТКО** указывает-

ся номер рабочего места и число команд, помещаемых в таблицу, в макрокоманде **КОМОП** – имена команд оператора (1–3 символа) и идентификатор входной точки программы обработки, если команда нестандартная.

Имена стандартных команд оператора и их функции приведены ниже:

ОПР – чтение-запись периода опроса параметра;
ТЗН – чтение-запись текущего значения параметра;
СРЗ – чтение среднего значения параметра;
СКИ – чтение-запись допустимой скорости изменения;
НВГ – чтение-запись нижней возможной границы контроля аналогового параметра;
ВВГ – чтение-запись верхней возможной границы аналогового или нормального состояния дискретного параметра;
КСГ – чтение-запись индивидуальных констант подпрограммы сглаживания;
СПТ – вызов группы сопутствующих параметров на экран и (или) печать;
ИМЯ – чтение-запись полного имени параметра;
ОБР – чтение-запись разрешения или запрещения на обработку параметра;
ТНД – регистрация текущего значения параметра на СМП с заданным номером;
ПРВ – прекращение регистрации на СМП с заданным номером;
ОБН – чтение-запись значения периода обновления;
КВС – квитирование сообщений на экране;
ВСЕ – просмотр сообщений о нарушениях, ушедших за пределы экрана;
НРШ – восстановление сообщений на экране.

Отметим, что в макрокомандах **КОМОП** может быть указана метка ЧТ, при наличии которой соответствующая команда оператора осуществляет только операцию чтения. Это позволяет для некоторых пультов оператора запретить операцию записи.

Формат набора первых 11 команд на пульте оператора следующий:

⟨ *имя команды* ⟩, ⟨ *шифр параметра* ⟩ [,⟨ *новое значение уставки* ⟩]

В командах чтения-записи технологических границ контроля, которые перечислены выше, но также являются стандартными, роль имени команды играет имя границы контроля.

При описании этих команд в макрокомандах **КОМОП** указывается

[ЧТ] **КОМОП** ⟨ *имя границы* ⟩, ТГР

где ТГР – идентификатор входной точки программы обработки данного типа команд.

На экране эта команда набирается в формате

⟨ *имя границы контроля* ⟩⟨ *шифр параметра* ⟩ [,⟨ *новое значение границы контроля* ⟩]

Указанный выше набор команд не является обязательным. Исходя из назначения генерируемой системы, можно описать только некоторые стандартные команды или включить дополнительные (нестандартные). Обязательными являются только последние четыре команды, назначение

которых станет ясным из рассмотрения процесса выдачи сообщений на экран.

Работа с экраном пульта-оператора. Экран пульта оператора на базе дисплея типа ВТА-2000 содержит 24 строки. Первые 3 строки отводятся для набора команд оператора и ответных сообщений, часть строк, начиная с четвертой (количество их указывается в макрокоманде **РАБМО**, но не более 21), – для индикации сообщений о нарушениях, оставшаяся часть строк может быть отведена для вывода информации от задач пользователя.

Под "нарушениями" в данном случае понимается выход текущего значения параметра за какую-либо границу контроля, превышение допустимой скорости изменения, изменение текущего или нарушение заданного состояния дискретного параметра, выход параметра из диапазона достоверности измерений (возможных границ).

Сообщения о нарушениях поступают на экран по инициативе задач сбора и обработки в хронологическом порядке, т.е. по мере их возникновения (индицируются со сдвигом на одну позицию вправо).

На каждое новое сообщение оператор должен реагировать командой **КВС** – квитирование сообщений. Рассмотрим понятия квитированных и неквитированных сообщений.

Каждое сообщение о нарушениях сохраняется на экране или при его переполнении в таблице нарушений, пока это нарушение не устранено. После устранения нарушения (с течением времени либо вследствие управляющего воздействия) вместо сообщения о нарушении появляется сообщение ВОШЕЛ В НОРМУ.

Сообщения типа ВОШЕЛ В НОРМУ, а также сообщения о текущем состоянии дискретных параметров и выходе параметра за границу возможных измерений относятся к типу квитируемых. По команде **КВС** они убираются с экрана и из таблицы нарушений, после чего экран и таблица уплотняются.

Сообщения о нарушениях границ контроля или заданного состояния являются неквитируемыми. По команде **КВС** эти сообщения только смещаются к началу строки, т.е. переводятся в разряд старых сообщений, сохраняясь в таблице нарушений.

По команде **ВСЕ** оператор может просмотреть все старые сообщения, которые вышли за пределы экрана, а по команде **НРШ** – восстановить последние сообщения на экране и обеспечить возможность принятия новых сообщений.

Команда **НРШ** необходима также после команды **СПТ** или дополнительных команд оператора, при которых занимается часть экрана, выделенная для сообщений о нарушениях.

Командой **ОБН** запрашивается или изменяется период обновления информации о текущих значениях параметров. С этим периодом в каждом сообщении, где фигурирует текущее значение параметра, старое значение автоматически заменяется на новое.

Генерация задачи ОТЧЕТ. Программа генерации задачи ОТЧЕТ имеет следующий вид (указаны только имена макрокоманд):

```
ОТЧЕТ
БЛАНК      – начало описания первого бланка
ТЕКСТ      – описание статической символьной информации первого бланка
...
ТЕКСТ
СТРАНИЦА   – начало первой страницы
ШИФР       – шифр первого параметра
КООРД      – описание информации о первом параметре
ШИФР       – шифр второго параметра
КООРД
...
СТРАНИЦА   – начало второй страницы первого бланка
ШИФР
КООРД
...
БЛАНК      – начало описания второго бланка
ТЕКСТ
...
СТРАНИЦА   – начало первой страницы второго бланка
...
КОНЕЦ
```

В макрокоманде **ОТЧЕТ** указываются имя, присваиваемое задаче, период вызова, его размерность и фаза начального запуска, приоритет, ЛН устройства печати (ЛНП), ЛН для запросов к задаче ЗАГРС, если задача ОТЧЕТ с диск-резидентными сегментами (ЛНЗ), ЛН для ответов от задач ЗАГРС и ПОИСК (ЛНО).

Пример. Задача ОТЧЕТ имеет диск-резидентные сегменты, ЛНП = = 15, ЛНЗ = 13, ЛНО = 10, приоритет 50, период вызова и начальная фаза 20 мин, имя – *ОТЧ15*. Макрокоманда **ОТЧЕТ** имеет вид

ОТЧЕТ (*ИМЯ,ОТЧ15*), (*ПВ,20,МН,20*), (*ПРТ,50*), (*ЛНП,15*), (*ЛНЗ,13*), (*ЛНО,10*)

В макрокоманде **БЛАНК** указывается количество строк в странице и количество символов в строке (до 128).

В одной задаче ОТЧЕТ может быть несколько бланков, а в каждом бланке – несколько страниц. Количество строк и символов в строке, указанное в макрокоманде **БЛАНК**, одинаково для всех страниц бланка. Общее число символов в странице, включая пробелы, не должно превышать 4096.

Каждая макрокоманда **ТЕКСТ** определяет последовательность символов (без пробелов), размещаемую в строках бланка с указываемого номера позиции. Эти макрокоманды описывают статическую информацию в бланке, которая сохраняется при печати каждой страницы бланка: заголовки, разметка строк и столбцов, наименования граф и т.д.

В макрокоманде **СТРАНИЦА** указывается кратность периода печати страницы бланка по отношению к периоду вызова задачи ОТЧЕТ. После этой макрокоманды идет описание динамической (изменяющейся) информации, которой будут заполняться различные строки бланка в процессе печати.

Описание динамической информации производится макрокомандами **ШИФР** и **КООРД**. Макрокоманда **ШИФР** указывает шифр параметра, информация о котором будет печататься, а макрокоманда **КООРД** – тип информации о параметре, номер строки и расположение информации в строке.

Макрокоманда **ШИФР** указывается столько раз, сколько параметров необходимо описать в странице. Полный набор информации о параметре может описываться несколькими макрокомандами **КООРД**, следующими непосредственно за макрокомандой **ШИФР**. Если набор информации для нескольких параметров совпадает, то макрокоманда **КООРД** может быть опущена.

К типам информации о параметре (операнд макрокоманды **КООРД**) относятся шифр и полное имя параметра, его размерность, текущее значение, среднее значение, значение границы технологического контроля с заданным номером, текущее время суток или дата, определяемые периодом вызова параметра, значения из общей области памяти.

Общая область памяти используется тогда, когда задача ОТЧЕТ является ОЗУ-резидентной и скомпонована в один загрузочный модуль с ОЗУ-резидентными задачами сбора и обработки информации. В этом случае информация о параметре может быть получена не через задачу ПОИСК, а непосредственно из общей памяти при соответствующем ее распределении.

Отметим, что задача ОТЧЕТ может обслуживать только одно устройство печати. Если устройств печати несколько, то для каждого из них должна быть сгенерирована своя задача ОТЧЕТ (или несколько задач для одного устройства). Все они могут быть описаны в одной программе генерации.

Генерация задач ПОИСК, ЗАГРС, ПЕЧАТЬ. Задача ПОИСК описывается одной макрокомандой **ПОИСК** и одной или несколькими макрокомандами **ФПОИСК.**

Задача ПОИСК осуществляет ряд стандартных функций поиска, которые обеспечиваются включением в задачу ПОИСК соответствующих программ обработки запросов на поиск. Кроме того, в задачу ПОИСК могут быть включены нестандартные программы, написанные пользователем и реализующие дополнительные функции поиска.

Полный набор функций задачи ПОИСК описывается одной или несколькими макрокомандами **ФПОИСК.** Этот набор функций поиска должен быть согласован с описанными в задаче ПТО командами оператора.

Задача ЗАГРС описывается одной макрокомандой **ЗАГРС**, операндами которой являются приоритет, ЛН псевдоустройства МЗОИ, по которому задача принимает запросы, и указания о типе сегментов [программные и (или) табличные].

Задача ПЕЧАТЬ также описывается одной макрокомандой **ПЕЧАТЬ**. В этой макрокоманде указываются имя, присваиваемое задаче, приоритет, ЛН устройства печати, ЛН псевдоустройства МЗОИ, по которому она принимает запросы, число буферов, длина буфера в словах, количество сообщений, которые могут быть напечатаны в одну строку.

Программы генерации служебных задач могут быть объединены с программами генерации программ обработки или других задач.

15.5. Возможности самостоятельного расширения пакета

Дисковый ППМ СОИ допускает возможность расширения пакета самим пользователем на уровне дополнительных подпрограмм или на уровне дополнительных задач. Дополнительные подпрограммы реализуют следующие функции: обработку информации от датчиков по нестандартным алгоритмам, обработку дополнительных команд оператора, обработку дополнительных запросов к задаче ПОИСК, реакцию на нарушение состояний параметров (подпрограммы действия), определение внешних условий запуска задач сбора и обработки.

Дополнительные задачи (будем называть их задачами диспетчерского управления) реализуют такие функции, как управление ТП по заданным алгоритмам, расчет технико-экономических показателей и значений параметров, не измеряемых непосредственно, и т.п. Эти задачи включаются в систему на общих основаниях. Они могут иметь свой приоритет, период вызова или внешние условия запуска.

Для облегчения программирования этих задач в состав пакета включен набор подпрограмм, осуществляющих запуск и приостанов задачи, ввод информации от датчиков, вывод управляющих воздействий, побитную обработку информации, связь с базой данных через задачу ПОИСК. Перечисленные подпрограммы могут вызываться на языках МНЕМОКОД, ФОРТРАН, АЛГОЛ.

Запрашивать данные о параметрах задача диспетчерского управления может либо самостоятельно (с помощью подпрограмм ввода информации от датчиков), либо из задач сбора и обработки (через общую область памяти или через задачу ПОИСК по шифру параметра). Передача данных в задачи сбора и обработки (и соответственно в задачу ПТО) осуществляется через фиктивные параметры.

15.6. Дополнения к основной версии пакета

Рассмотрим некоторые дополнения, связанные с дальнейшим развитием пакета и включенные в дополнительные библиотеки макроопределений и подпрограмм.

Двухформатный шифр параметра. При изложенном выше способе шифрации один и тот же шифр фигурирует как в программах генерации задач, так и в диалоге оператора с системой. В пакете предусмотрен другой способ шифрации, при котором шифр параметра может иметь два формата: внешний, которым он представляется в сообщениях оператору и в его запросах, и внутренний, который присваивается параметру в программах генерации системы и с помощью которого осуществляются связи между задачами системы и базой данных.

Внешний формат может включать до 20 символов и иметь элементную структуру, где каждый элемент (1 – 5 символов) может принимать различные значения. Это позволяет увеличить наглядность шифров, приблизить их к традиционно сложившимся на производствах. Например, *2ЦФ3Т15, КС5Р85.*

Внутренний формат состоит из 5 символов, где первый символ является буквой, а 4 остальных – числовым номером. При генерации системы в задачи ПТО и ОТЧЕТ должны быть включены специальные подпрограммы перекодировки шифров.

Блоки данных. Под блоками данных в пакете понимаются массивы числовой либо символьной информации, служащие для обмена информацией между задачами пользователя или между задачей пользователя и задачей ПТО. Эти массивы, например, могут содержать параметры настройки задач диспетчерского управления.

Блоки данных включаются в задачи сбора и обработки на правах сегментов и описываются специальными макрокомандами. При этом каждому блоку данных присваивается шифр, по структуре аналогичный шифру параметра. Так же как и параметр, блок данных может иметь полное имя.

При генерации системы элементам блоков данных могут быть либо присвоены первоначальные значения, либо только зарезервированы их число и размеры (в этом случае элементы массивов заменяются нулями).

Доступ к элементам блоков данных (чтение, запись, вывод на печать) со стороны задач ПТО, ОТЧЕТ и задач диспетчерского управления обеспечивается задачей ПОИСК по шифру блока и порядковому номеру элемента.

Файл режима. При переключении технологического оборудования на новый режим работы возникает необходимость изменения некоторых констант в программах обработки параметров (границ контроля, допустимой скорости изменения и т.д.) и в задачах диспетчерского управления. Одновременная замена констант может быть выполнена одной командой оператора **СМР** (сменить режим) путем ввода заранее заготовленного файла режима, содержащего новые значения констант.

Файлов режима может быть несколько. Они генерируются отдельно с помощью макрокоманд **РЕЖИМ** и **КРЕЖИМ**, в которых указываются шифры параметров или блоков данных, типы и значения заменяемых

констант, типы извещений о завершении смены режима для запуска задач.

В программе обработки **СМР** предусмотрены диагностические сообщения на экран и устройство печати об ошибках в файле режима.

Другие дополнения пакета. Кроме рассмотренных в ДППМ СОИ реализован ряд других дополнений, обеспечивающих следующие функции:

включение в задачу ПТО программ обработки дополнительных запросов (отсчет промежутка времени или инициатива какого-либо устройства, в частности печати);

вывод графиков на устройства быстрой печати;

вывод информации о группах параметров на экран или печать в задаваемом формате по команде оператора;

управление расстоянием между страницами бланков и количеством копий при периодической печати;

организация группового вывода на мнемосхемы, когда каждому элементу мнемосхемы соответствует несколько состояний одного или нескольких параметров;

аппроксимация нестандартных градуировок полиномом третьей степени;

чтение-запись констант обработки параметров задачами диспетчерского управления.

15.7. Программные средства отладки

Программные средства отладки включают в себя: библиотеку подпрограмм отладки (файл БОТЛ), подпрограммы имитации вывода на мнемосхему (БМНСХ) и программу подсчета числа фактических выполнений задач сбора и обработки (ФАКТВ).

Указанные программные средства предназначены для отладки сгенерированных задач в режиме реального времени на УКВ минимальной конфигурации (без подсистемы ввода информации).

С этой целью в библиотеку подпрограмм отладки включены программные имитаторы ввода информации, а также средства диалога оператора с системой и выдачи протокола в процессе отладки.

Непрерывные сигналы от датчиков (аналоговых и число-импульсных) имитируются пилообразными симметричными функциями времени с размахом, равным диапазону измерений датчиков, дискретные – циклическими функциями сдвига 16-разрядных счетчиков (после 16 сдвигов восстанавливается исходное состояние параметров). Инициативные сигналы имитируются командами оператора записи текущих значений параметров, вывод информации на мнемосхему подменяется выводом сообщений о состоянии элементов мнемосхемы на устройство печати. Кодовые сигналы не имитируются, и их обработка в процессе отладки не выполняется.

Отладочный вариант системы отличается от рабочего только тем, что на стадии компоновки в загрузочные модули с задачами сбора и обработки, ПТО и ПОИСК включаются отладочные подпрограммы. Соответственно после завершения отладки все эти модули должны быть скомпонованы заново.

Процесс отладки происходит в диалоговой форме на языке управления средствами отладки и сопровождается выводом на печать протокола отладки. Для отладки неинициативных задач сбора и обработки предусмотрены режимы: 1 – начальной настройки, 2 – полуавтоматического выполнения, 3 – автоматического выполнения.

В режиме 1 для каждой группы параметров вводятся начальные данные. В режиме 2 для каждой группы имитируется ввод информации, после чего автоматически выполняется ее обработка и вывод диагностических сообщений. В режиме 3 задача запускается и функционирует точно так же, как и в рабочем варианте.

Инициативная задача по команде оператора **ТЗН, ОТЛ** настраивается на режим отладки. Командой записи текущего значения, равного 0 или 1, для каждого параметра имитируется наличие инициативы от датчика, которая обрабатывается задачей.

Отладка задачи ПТО производится с одного пульта путем поочередного закрепления его за каждым рабочим местом командой:

НРШ,⟨ *номер РМО*⟩.

В процессе отладки можно оценить временные характеристики системы с помощью программы ФАКТВ. Эта программа, запускаемая по команде оператора, определяет, сколько раз на заданном интервале времени неинициативная задача сбора и обработки должна выполниться за этот интервал и сколько раз она фактически выполнилась.

Несовпадение этих чисел будет указывать на то, что данная задача не успевает провести обработку информации при заданных периодах опроса датчиков. Такую же проверку можно аналогичным образом выполнить в реально действующей системе.

Глава 16. ПАКЕТ ПРОГРАММНЫХ МОДУЛЕЙ ДЛЯ ОРГАНИЗАЦИИ БАНКОВ ДАННЫХ (ППМ БАНК-2)

16.1. Основные понятия и термины

Структура банка данных. Пакет ПМ БАНК-2 является расширенной версией разработанного ранее ППМ БАНК и предназначен для создания информационно-поисковых систем на базе УВК СМ-2М.

Информационно-поисковая система состоит из банка данных (БД) – совокупности наборов файлов на машинных дисках, в которых разме-

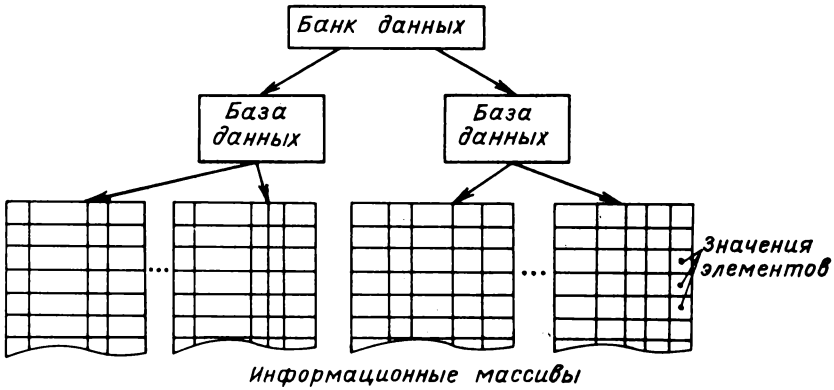


Рис. 16.1. Структура банка данных

щается рабочая информация, и системы управления банком данных (СУБД) – комплексом программ, компонуемых с помощью ППМ БАНК-2, для организации, хранения и поиска информации в банке данных.

Структура банка данных с точки зрения расположения рабочей информации представлена на рис. 16.1. Банк данных, создаваемый на основе пакета, состоит из одной или нескольких (до восьми) баз данных. База данных включает в себя информационные массивы. Массивы состоят из записей, которые автоматически нумеруются в порядке поступления в банк данных и в которых хранятся значения некоторого набора элементов. Каждый массив определяется своим набором элементов и порядком их следования в наборе, так что все записи в массиве одинаковой длины и структуры.

Если рассматривать массив как двумерную таблицу, то в каждом столбце располагаются значения одного и того же элемента, а каждая строка является записью. Значения элемента в различных записях могут повторяться, записи содержат отличающиеся друг от друга комбинации значений различных элементов (идентичные записи не несут новой информации, и их следует удалить из массива).

Элементы могут быть трех типов: целые числа в диапазоне от минус 32768 до плюс 32767 (тип Ц), вещественные числа с плавающей запятой от 10^{-18} до 10^{18} (тип Р), строка от 1 до 126 символов (тип С). Значения элементов занимают в записи соответственно типу одно, два или максимально 63 слова. Общий размер записи не должен превышать 256 слов (245 без служебной информации).

В базе данных может быть определено до 200 элементов. Полный набор элементов распределяется пользователем по массивам, исходя из логической связи между элементами и ограничения на размер записи.

Рассмотрим пример информационного массива, который будем называть рабочим.

Пример 1. Расписание вылета самолетов из Москвы

Запись	ПУНКТ	АЭРПРТ	ТИПСМТ	ВРВЫЛ	ВРПРИЛ	СТМСТ
1	Киев	Внуково	ТУ-154	19.30	20.10	22
2	Ворошиловград	Внуково	ТУ-134	14.10	16.40	26
3	Ворошиловград	Внуково	ТУ-134	9.40	12.10	26
4	Северодонецк	Быково	АН-24	9.05	12.25	26

В этом примере массив содержит 6 элементов: три типа С, два типа Р, один типа Ц. В записях для значений элементов следует отвести количество слов: 10, 6, 3, 2, 2, 1.

Каждому элементу, массиву, а также самой базе данных должны быть присвоены имена – наборы от 1 до 6 символов, начинающиеся с буквы. Все имена элементов должны быть различными, даже если один и тот же по смыслу элемент входит в разные массивы. Имя элемента однозначно определяет его принадлежность к массиву.

Во всех процедурах СУБД доступ к базе данных, массивам, значениям элементов осуществляется по именам. В этом смысле СУБД можно считать расширением системы управления файлами, когда прямой доступ реализован не только к файлу, но и к его сегментам.

Поиск и сортировка записей. Поиск и сортировка записей относятся к основным функциям СУБД. Поиск заключается в отыскании записей с заданным множеством значений одного или нескольких элементов, принадлежащих одному массиву. Множество значений определяется логическими отношениями: равно, не равно, больше, не больше, меньше, не меньше, и, или. Например, для массива в примере 1 возможны следующие директивы поиска: найти записи, в которых значение элемента ПУНКТ равно Ворошиловград и значение элемента ВРВЫЛ меньше 12,00, или найти записи, в которых значение элемента АЭРПРТ не равно Быково. В ответ на эту директиву в первом случае будет найдена запись 3, во втором – записи 1, 2, 3.

Сортировка заключается в упорядочении записей (точнее, их номеров) по значениям выбранных для сортировки элементов в алфавитном или возрастающем порядке.

Записи, найденные по директиве поиска, копируются в отдельный файл, и их номера сортируются перед выдачей на экран или устройство печати, т.е. выдача информации производится в упорядоченном виде.

Сортировка записей по нескольким элементам производится последовательно: вначале по одному элементу, затем те записи, у которых одинаковое значение этого элемента, сортируются по другому элементу,

и т.д. В примере 1 сортировка по элементам ПУНКТ и ВРВЫЛ приведет к следующему порядку записей: 3, 2, 1, 4.

Ключевые элементы. Для ускорения поиска записей вводится аппарат *ключевых* элементов. В каждом рабочем массиве пользователь может до пяти элементов объявлять ключевыми. Ключевому элементу присваивается собственное имя, значения ключевых элементов размещаются во вспомогательных массивах, называемых *управляющими.* В управляющем массиве хранятся различные (неповторяющиеся) значения только одного ключевого элемента. Ключевой элемент может совпадать по значениям с элементами, входящими в разные рабочие массивы (до пяти). Соответственно управляющий массив будет связан с этими массивами. Выделение ключевых элементов и описание управляющих массивов входит в функции пользователя.

По способу заполнения управляющие массивы делятся на неавтоматические и автоматические. Первые могут содержать кроме ключевого неключевые элементы и заполняются пользователем наравне с рабочими массивами, вторые содержат только ключевой элемент и заполняются автоматически по мере заполнения рабочих массивов.

Поиск может производиться как по ключевым, так и по неключевым элементам.

Рассмотрим использование ключевых элементов на примере.

Пример 2. База данных с именем ФИНТЕМ предназначена для хранения сведений о финансировании тем. Сведения можно разбить на две группы: об исполнителях тем (номер отдела, название темы, должность, стаж и оклад исполнителя) и о сроках их выполнения (название темы, номер этапа, год завершения).

Соответственно этому разбиению сформируем два рабочих массива.

Массив СМЕТА:

ОТДЕЛ	ТЕМ1	ИСПОЛН	СТАЖ	ОКЛАД
34	ТЕМА3	ИНЖЕНЕР	12	150
10	ТЕМА3	СТ.Н.С.	20	190
2	ТЕМА2	СТ.ИНЖ.	6	160

Массив СРОКИ:

ТЕМ2	ЭТАП	ГОД
ТЕМА2	1	1975
ТЕМА1	1	1972
ТЕМА2	2	1978
ТЕМА5	1	1980

Обратим внимание, что название темы в разных массивах имеет разные имена (ТЕМ1, ТЕМ2).

Объявим ключевыми элементами номер отдела, название темы и оклад исполнителя, присвоив им соответственно имена ОТД, ТЕМ, ОКЛ, и введем, присвоив имена, управляющие массивы АВОТД, АВТЕМ, АВОКЛ. При этом массивы АВОТД и АВОКЛ будут связаны только с массивом СМЕТА, а массив АВТЕМ – с массивами СМЕТА и СРОКИ. (Ключевой элемент ТЕМ имеет те же значения, что и элементы ТЕМ1,

ТЕМ2, – названия тем, но в массиве АВТЕМ все названия тем будут различные, а в массивах СМЕТА и СРОКИ они могут повторяться).

Таким образом, определена база данных, включающая 11 элементов (3 из них ключевые) и 5 массивов (2 рабочих, 3 управляющих).

Защита информации. Кроме обычной защиты файлов с помощью паролей (кодов защиты в виде целых чисел от 0 до 32000), которые устанавливаются для самой базы данных и всех массивов, в СУБД предусмотрена уровневая защита элементов.

С этой целью вводится понятие *категории* пользователей (абонентов) по их полномочиям при работе с базой данных. Каждая категория идентифицируется кодовым словом (набор символов от 1 до 6) и имеет свой уровень – номер от 1 до 15. Чем выше уровень, тем большими полномочиями обладает абонент данной категории.

В свою очередь каждому элементу назначается уровень защиты по чтению и записи также в виде целых чисел (1–15). Аппарат защиты построен таким образом, что абонент, имеющий уровень, меньший уровня защиты элемента по какой-либо операции, не имеет доступа к этому элементу для этой операции.

Так, в примере 2 можно ввести следующие категории абонентов: ЗАВЛАБ, РУКТЕМ, РАЗРАБ, присвоив им соответственно уровни 1, 4, 15. Если для какого-либо элемента установлены уровни защиты (4, 15), т.е. 4 – по чтению, 15 – по записи, то абонент РАЗРАБ будет иметь доступ к этому элементу для чтения и записи, РУКТЕМ – только для чтения, ЗАВЛАБ не будет иметь доступа к этому элементу.

Аварийное восстановление базы данных. В СУБД предусмотрена возможность восстановления базы данных при аварийных ситуациях: непредвиденного останова системы или разрушения какой-либо базы данных на диске. Для этого в процессе работы с банком данных директивы СУБД копируются в служебные файлы: аварийный и архивный.

Аварийный файл при обработке его соответствующей процедурой позволяет восстановить базы данных в случае останова процессора путем повторной модификации начального состояния по копиям директив, записанным в этот файл. После завершения работы с банком аварийный файл уничтожается.

Архивный файл позволяет таким же образом восстановить базы данных при их разрушении на диске. При этом предполагается, что копия начального состояния баз данных размещена на другом носителе (диске или магнитной ленте). После завершения работы с банком архивный файл только закрывается, т.е. в нем хранятся копии директив всех сеансов работы с банком, и в этом смысле он является архивом банка данных.

16.2. Состав и функции ППМ БАНК-2

Пакет ПМ БАНК-2 содержит более 30 подпрограмм и 2 библиотеки, все подпрограммы поставляются в перемещаемом формате. Из этих подпрограмм по общим правилам компоновки с учетом связей между подпрограммами компонуются 9 задач, которые по функциональному назначению можно разделить на 4 подсистемы: проектирования, установки, управления и аварийного запуска.

Подсистема проектирования состоит из одной задачи СХЕМА и предназначена для создания схемы базы данных -- файла, содержащего описание структуры конкретной базы данных.

Подсистема установки базы данных включает 6 задач, которые обеспечивают первичное заполнение записями созданной схемы базы данных (БАНК), сортировку записей в управляющих массивах (СОРТ), копирование базы данных на внешний носитель (КПБД), загрузку на диск с внешнего носителя (ЗГБД), реконфигурацию базы данных (РКБД), групповое добавление записей (ГДБВ).

Последовательность выполнения задач этой подсистемы может быть произвольной, за исключением того, что при начальном заполнении базы данных первой должна быть выполнена задача БАНК, а после задач БАНК и РКБД обязательным является выполнение задачи СОРТ.

Подсистема управления состоит из одной сегментированной задачи ДИСПТ. Эта задача является наибольшей по объему и определяет требуемый объем памяти для работы всей системы – 19 К.

Подсистема обеспечивает одновременную работу с 8 базами данных, при этом доступ к ним могут иметь до 100 абонентов.

Подсистема управления выполняет следующие функции: поиск записей по заданным условиям, сортировку записей перед выдачей на экран или устройство печати, единичное добавление новых записей в массивы, вычеркивание записи, замену значений ключевых или неключевых элементов, копирование директив в архивный и аварийный файлы.

При выдаче информации можно произвести разметку бланка, указать число найденных записей, сумму и среднее значение по элементам.

Подсистема аварийного запуска включает три задачи: АВР, АДИСП и АРХИВ.

Задача АВР обрабатывает аварийный файл и восстанавливает базы данных в случае сбоев во время сеанса работы с банком.

Задача АДИСП выполняет те же функции, что и задача ДИСПТ, кроме копирования директив в архивный файл, и работает параллельно с задачей АРХИВ. Задача АРХИВ выбирает копии заполненных по всей предшествующей работе с банком директив из архивного файла и обращается к задаче АДИСП через псевдоустройство МЗОИ с логическим номером восемь, а задача АДИСП производит модификацию копии первоначального состояния баз данных, восстанавливая базы данных в слу-

чае их разрушения на дисках. Задачи АРХИВ и АДИСП могут быть скомпонованы в один загрузочный модуль.

Функционирование задач, компонуемых на основе ППМ БАНК-2, можно проверить на девяти контрольных задачах, поставляемых вместе с пакетом.

16.3. Входные файлы для организации базы данных

При создании файлов с описанием схемы базы данных и информацией для ее первичного заполнения (входных файлов для задач СХЕМА и БАНК) необходимо соблюдать определенные требования к записям. (Здесь под *записью* будем понимать информацию, расположенную на одной экранной строке и заканчивающуюся символом *перевод строки.*)

Например, служебные слова, как-то: ПАКЕТ, УРОВНИ, ЭЛЕМЕНТЫ и другие, должны располагаться с первой позиции, комментарии должны отделяться хотя бы одним пробелом и т.д.

Описание схемы базы данных состоит из трех частей: уровневой, элементной и описания массивов. В уровневой части указываются в возрастающем порядке уровни и кодовые слова категорий абонентов. В элементной части каждая запись содержит информацию об одном элементе: имя, тип, для символьных элементов размер в словах, уровни защиты по чтению и записи.

При описании массивов указываются имя массива, тип (управляющий неавтоматический, автоматический, рабочий), пароль, имена элементов в порядке появления их значений в записях массива, резервируемое число записей. Для ключевых элементов в управляющих массивах указывается число связок с рабочими массивами, а для элементов рабочих массивов, объявленных ключевыми, – имена соответствующих управляющих массивов. В описательной части вначале следует описание управляющих массивов, затем рабочих.

Ниже приведен входной файл описания схемы базы данных ФИНТЕМ, рассмотренный в примере 2 (см. § 16.1):

```
ПАКЕТ ФИНТЕМ,10035;        ИМЯ БАЗЫ ДАННЫХ,ПАРОЛЬ
УРОВНИ:
      1  ЗАВЛАБ,
      4  РУКТЕМ,
      15 РАЗРАБ;
ЭЛЕМЕНТЫ:
      ОТД,       Ц(4,15),        КЛЮЧЕВЫЕ ЭЛЕМЕНТЫ
      ТЕМ,       С8(4,15),
      ОКЛ,       Ц(4,15),
      ОТДЕЛ,     Ц(4,15),        ЭЛЕМЕНТЫ МАССИВА СМЕТА
      ТЕМ1,      С8(4,15),
      ИСПОЛН,    С5(4,15),
      СТАЖ,      Ц(4,15),
      ОКЛАД,     Ц(4,15),
      ТЕМ2,      С8(4,15),       ЭЛЕМЕНТЫ МАССИВА СРОКИ
      ЭТАП,      Ц(1,15),
      ГОД,       Ц(1,15);
```

```
МАССИВ:                          АВТОМАТИЧЕСКИЕ МАССИВЫ
ИМЯ:АВОТД,А,00007;
ЗАПИСЬ:ОТД(1);                        КОЛИЧЕСТВО СВЯЗОК 1
РАЗМЕР:50;
МАССИВ:
ИМЯ:АВТЕМ,А,00007;
ЗАПИСЬ:ТЕМ(2);                        КОЛИЧЕСТВО СВЯЗОК 2
РАЗМЕР:50;
МАССИВ:
ИМЯ:АВОКЛ,А,00007;
ЗАПИСЬ:ОКЛ(1);                        КОЛИЧЕСТВО СВЯЗОК 1
РАЗМЕР:50;
МАССИВ:                               РАБОЧИЕ МАССИВЫ
ИМЯ:СМЕТА,Р,00100;
ЗАПИСЬ:ОТДЕЛ(АВОТД),
       ТЕМ1(АВТЕМ),
       ИСПОЛН,
       СТАЖ,
       ОКЛАД(АВОКЛ);
РАЗМЕР:200;
МАССИВ:
ИМЯ:СРОКИ,Р,00101;
ЗАПИСЬ:ТЕМ2(АВТЕМ),ЭТАП,ГОД;
РАЗМЕР:150;
КОНЕЦ
```

Во входном файле для задачи БАНК в первой записи должно быть указано имя базы данных, пароль и кодовое слово уровня 15. В описательной части для каждого массива указывается его имя, пароль и количество вводимых записей, после чего следуют записи массива, значения элементов в которых записываются без пробелов и левых полей и могут занимать несколько строк. Последовательность массивов должна совпадать с описанием схемы базы данных.

Входной файл начального заполнения базы данных ФИНТЕМ представлен ниже:

```
ФИНТЕМ,10035,РАЗРАБ;  ИМЯ БАЗЫ,ПАРОЛЬ,КОД СЛОВО 15
МАССИВ:АВОТД,7,0:        ИМЯ,ПАРОЛЬ,ЧИСЛО ЗАПИСЕЙ
МАССИВ:АВТЕМ,7,0;
МАССИВ:АВОКЛ,7,0;
МАССИВ:СМЕТА,100,3;
34,ТЕМА3,ИНЖЕНЕР,12,150, ОТДЕЛ,ТЕМА,ИСПОЛ,СТАЖ,ОКЛАД
10,ТЕМА3,СТ.Н.С.,20,190,
2,ТЕМА2,СТ.ИНЖ.,6,160;
МАССИВ:СРОКИ,101,4;
ТЕМА2,1,1975,                ТЕМА,ЭТАП,ГОД
ТЕМА1,1,1972,
ТЕМА2,2,1978,
ТЕМА5,1,1980;
КОНЕЦ
```

При первичном заполнении базы данных не следует стремиться вводить все записи сразу с помощью задачи БАНК, поскольку файл с большим количеством записей трудно проконтролировать. Целесообразно ввести лишь часть записей либо создать базу данных с пустыми массивами (число вводимых записей равно нулю), а остальные записи вводить по частям с помощью задачи ГДБВ. При этом сортировка записей в управляющих массивах производится автоматически.

Формат входного файла для задачи ГДБВ аналогичен формату файла для задачи БАНК, за исключением того, что после каждой записи на отдельной строке в первой позиции ставится косая черта. Пример входного файла показан ниже:

```
ФИНТЕМ,10035,РАЗРАБ;
/
МАССИВ:СМЕТА,100,2;
/
5,ТЕМА7,ИНЖЕНЕР,5,140,
/
2,ТЕМА7,СТ.Н.С.,10,170,
/
МАССИВ:СРОКИ,101,2;
/
ТЕМА3,1,1976,
/
ТЕМА3,2,1979,
/
КОНЕЦ
```

16.4. Директивы управления базами данных

Подсистема управления базами данных может одновременно обслуживать запросы от нескольких терминалов и нескольких программ пользователя. Один из терминалов, которому назначается логический номер 7, является пультом оператора. С этого пульта осуществляется запуск задачи ДИСПТ и начальный диалог для передачи следующих данных: указания о необходимости вести архив; если его необходимо вести, то номер устройства и имя или номер архивного файла; количество баз данных на обслуживании, их имена и пароли, число одновременно обслуживаемых абонентов, включая программы пользователя; логический номер группы инициативных устройств, если кроме пульта оператора предполагается использование других терминалов; логический номер псевдоустройства МЗОИ для приема запросов от задач пользователя; номера дисков для вспомогательных файлов: с таблицами абонентов, аварийного, записей и процедур.

Файлы с таблицами абонентов и аварийный (имена ТАБЕН и АВАРИЯ) являются общими и открываются задачей ДИСПТ, файлы записей и процедур являются собственностью абонентов и открываются ими с помощью специальных команд. В файл записей помещаются номера записей, найденные абонентом в результате поиска в базе данных. Файл процедур дает возможность хранить под собственными именами многострочные тексты часто повторяющихся команд.

Абонент может работать с базой данных в одном из трех режимов: 1 – чтения; 2 – чтения и замены значений неключевых элементов; 3 – чтения и модификации значений любых элементов. В режиме 2 запрещены операции добавления и вычеркивания записей. В режиме 3 абонент монопольно владеет массивом, к которому обращен запрос, в этом режиме допустимы замены значений ключевых элементов и операции добавления и вычеркивания записей.

Абонент, работающий с терминалом, объявляет о начале сеанса словом **БАНК**. В первой команде **БДН** (назначение базы данных) он должен указать имя и пароль базы данных, свое кодовое слово и режим, в котором он собирается работать.

Командой **ФЗП** (файл записей) абонент указывает имя файла записей, код защиты и размер в секторах, после чего должна следовать команда **ПСК** (поиск записей).

В команде **ПСК** задаются условия поиска в виде соотношений между именами элементов одного массива и их значениями. Например, команда поиска записей в массиве СРОКИ базы данных ФИНТЕМ, в которых значение элемента ГОД находится в пределах от 1978 до 1982, будет иметь вид

ПСК ГОД **НМ** " 1978 " **И** ГОД **НБ** " 1982 " **КНЦ;**

где **НМ** – не меньше; **НБ** – не больше.

Дальнейшие процедуры производятся с теми записями, которые найдены в результате поиска. Такими процедурами могут быть выдача значений элементов на экран или устройство печати (команды **ВЫД, ВЗП, ВРП**), вычеркивание найденных записей (команда **ВЧК**), добавление записей (команда **ДБВ**), замена значений элементов (команды **ЗМН, ЗКЛ**).

Команды выдачи информации имеют несколько форматов и могут включать операторы, позволяющие выводить данные страницами как на узкую, так и на широкую печать, снабжать заголовками, сортировать записи по значениям отдельных элементов, группировать записи с одинаковыми значениями элементов, указывать сумму, среднее значение и число записей в группе.

Пример выдачи данных по результатам поиска в массиве СМЕТА базы данных ФИНТЕМ приведен ниже:

СВЕДЕНИЯ ПО ОТДЕЛАМ

2	ТЕМА1	ТЕХНИК	100	
	ТЕМА2	СТ.ИНЖ.	160	
	ТЕМА3	СТ.ИНЖ.	160	
			СУММА = 420	ЗАПИСИ =3
10	ТЕМА2	ИНЖЕНЕР	140	
	ТЕМА2	СТ.Н.С.	180	
	ТЕМА3	ИНЖЕНЕР	130	
	ТЕМА3	СТ.Н.С.	170	
			СУММА = 620	ЗАПИСИ = 4
34	ТЕМА3	ИНЖЕНЕР	140	
	ТЕМА3	ИНЖЕНЕР	150	
	ТЕМА3	СТ.ИНЖ.	160	
			СУММА = 450	ЗАПИСИ = 3
		ОБЩАЯ	СУММА = 1490	ЗАПИСИ = 10

Текст команды выдачи информации, например команды **ВЗП**, с помощью которой создан отчет, может занимать несколько строк.

Команды для работы с процедурным файлом выполняют следующие функции: открытие процедурного файла, запись, корректировку и удаление команды, выдачу имен и текста команд, вызов команды на исполнение, копирование команд в операторский процедурный файл, открываемый с пульта оператора, и перезапись команд из операторского в абонентские файлы.

Завершается сеанс работы абонента с банком данных командой **ЗВР**. По этой команде уничтожаются файлы записей и процедур, созданные абонентом.

Останов подсистемы управления банком производится командой **ОС**, посылаемой с пульта оператора. По команде **ОС** уничтожаются все абонентские файлы записей и процедур, служебные файлы ТАБЕН и АВАРИЯ, закрываются архивный и операторский процедурный файлы. При этом в последнем файле сохраняются все тексты команд, которые были скопированы из абонентских файлов процедур.

В программах пользователя запросы в подсистему управления банком можно составлять как в терминальной, так и в программной форме. В первом случае запросы имеют тот же текстовый вид, что и запросы с терминала, во втором случае они содержат в основном двоичную информацию.

Глава 17. БИБЛИОТЕКИ ПРОГРАММ МАШИННОЙ ГРАФИКИ

17.1. Назначение библиотеки графики для устройства А521-4

В данной главе рассмотрены две библиотеки подпрограмм для вывода графической информации на наиболее распространенные в номенклатуре АСВТ-М графические устройства: устройство печати знакосинтезирующее (УПЗ) А521-4, которое входит во все базовые комплексы СМ-2М, и цветной графический терминал (ЦГТ) А543-11, который поставляется как самостоятельно, так и в составе субкомплекса РМОТ – рабочего места оператора-технолога.

Для формирования графической информации и ее вывода на УПЗ предназначена библиотека программ графики (БПГ) АСПО.

По составу и структуре БПГ является подмножеством известного комплекса ГРАФОР (ГРАфика на ФОРтране), реализованного на ряде ЭВМ для различных графических устройств.

При перенесении ГРАФОРа на УВК СМ-2М учтены особенности операционных систем АСПО. Специфика УПЗ как графического устройства отражена в нескольких подпрограммах, написанных на МНЕМОКОДе, к которым пользователь непосредственно не обращается. Общие принципы и структура базового ГРАФОРа сохранены вплоть до имен и форматов вызова подпрограмм.

Устройство А521-4 относится к типу комбинированных устройств, которые могут работать как в режиме печати, так и в режиме графического вывода. Рабочим органом УПЗ является печатающая головка, содержащая семь печатающих иголок, расположенных по вертикали. Печать производится на бумажную, перфорированную с двух сторон ленту шириной 420 мм.

Реализованные в БПГ подпрограммы позволяют строить графики изменения регистрируемых параметров от текущего времени, изображать зависимости параметров друг от друга, рисовать совмещенные графики различных процессов. Для различения графиков предусмотрены три типа линий – сплошная, штриховая, пунктирная, а также специальные маркеры. Характерные точки графиков можно снабдить поясняющими надписями. Графики могут быть построены как в декартовой, так и в полярной системе координат. Координатные оси размечаются программно основными и вспомогательными делениями и сопровождаются наименованиями осей.

Кроме графиков функций с помощью БПГ можно строить гистограммы (столбчатые диаграммы), использовать календарную ось, на которой пометками являются сокращенные названия месяцев.

Ряд подпрограмм БПГ предназначен для построения разнообразных геометрических фигур – прямоугольников, многоугольников, окружностей, эллипсов, спиралей, их дуг. С их помощью можно изобразить, например, схему технологического процесса в виде условных картинок, отражающих конструктивные особенности отдельных агрегатов, и показать состояние этих агрегатов в регистрируемые моменты времени (клапан открыт, часть емкости заполнена и т.п.).

Для вывода текста можно использовать четыре набора шрифта. Они включают в себя строчные и прописные буквы русского, латинского и греческого алфавитов, математические и специальные символы. Тип шрифта и размеры символов задаются программно. Строка текста может располагаться под любыми углами и в любом месте рабочего поля.

Программа графического вывода на языке ФОРТРАН-IV включает в себя как стандартные операторы, так и вызовы библиотечных подпрограмм. При компоновке программы необходим дополнительный режим:

```
БП, ОБПГ,
```

где ОБПГ – имя поставляемого файла с библиотечными подпрограммами в перемещаемом формате.

Загрузочный модуль состоит из общей области, программ пользователя, подпрограмм БПГ и подпрограмм основной библиотеки. Структура загрузочного модуля программы графического вывода на УПЗ имеет следующий вид:

Нулевая страница
Поименованная общая область
Программы пользователя
Подпрограммы библиотеки графики
Подпрограммы основной библиотеки
БУФЕР до 16 К

Оставшаяся часть раздела автоматически отводится под буфер, максимальный размер которого установлен 16 К слов.

В буфер выводится результат работы подпрограмм БПГ в виде управляющих слов для УПЗ, в каждом байте которого закодировано элементарное перемещение печатающей головки. Очистка буфера производится автоматически при его переполнении. Таким образом, если изображение превышает размер буфера, то оно выводится по частям. Окончательно очищается буфер специальной подпрограммой.

17.2. Организация графического вывода на УПЗ

Первой подпрограммой, вызов которой надо выполнить до обращения к какой-либо подпрограмме БПГ, является подпрограмма **GRINT**. Она вводит логический номер УПЗ для графического вывода и осуществляет начальные установки. Затем могут идти обращения к подпрограммам установки единицы измерений, типа линий и шрифта. После этого должен идти вызов подпрограммы **PAGE** - *открытие страницы.*

Страничная организация в ГРАФОРе обеспечивает первый уровень защиты графического материала: до открытия страницы всякое обращение к графическим подпрограммам игнорируется; после того как страница открыта, т.е. указаны ее размеры, игнорируется всякий выход за ее пределы.

Страница определяет то "физическое" пространство, на котором должны быть размещены все рисунки и графики. Со страницей жестко связана система координат. В БПГ принято, что ось x направлена вдоль бумажной ленты вниз, ось y – слева направо (рис. 17.1). Начало координат – точка (0; 0) – всегда совпадает с начальным положением печатающей головки (при подготовке к работе оператор должен перевести ее в левое крайнее положение).

Начало координат считается левым нижним углом страницы, т.е. страница всегда располагается вдоль бумажной полосы. Такое расположение страницы не явля-

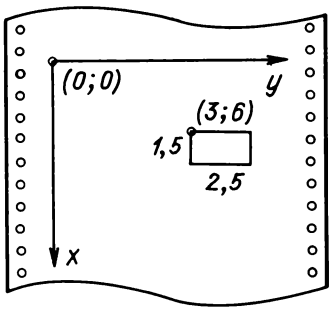


Рис. 17.1. Расположение осей страничных координат

ется существенным ограничением. Все надписи, рисунки и графики можно располагать слева направо, т. е. поперек бумажной полосы. Для этого надо в подпрограммах рисования геометрических фигур и текста указать поворот на 90°, а в подпрограммах вывода графиков поменять местами значения функции и аргумента.

После открытия страницы можно обращаться к графическим подпрограммам вывода текста, чисел, маркеров, рисования геометрических фигур. Расположение графических объектов на странице задается либо в абсолютных координатах, либо в приращениях к текущим координатам печатающей головки.

Для рисования графиков и гистограмм необходимо "открыть область" и указать масштаб. Область для рисования графиков задается своим расположением на странице и размерами (подпрограммы **REGION, POLREG**). Задание области обеспечивает второй уровень защиты графического материала: пока область не открыта, игнорируются вызовы подпрограмм для рисования графиков, после открытия области игнорируется всякий выход за ее пределы.

На странице могут быть заданы одна или несколько областей, каждое новое задание области отменяет предыдущее, поэтому области могут перекрываться и пользователь сам должен следить за их расположением на странице. Предполагается, что при открытии страницы открывается и область, совпадающая со страницей.

Масштаб для рисования графиков задается подпрограммой **LIMITS**, с ее помощью указываются значения математических координат на границе области. Здесь встречаемся с такими понятиями ГРАФОРа, как математическое и физическое пространство, или математические и физические координаты.

Под математическими координатами при изображении функций понимаются координаты точки (значения аргумента и функции) в "натуральных" единицах – с, град, кг и т.п., под физическими координатами — координаты той же точки на физическом носителе изображения (в нашем случае на бумажной ленте УПЗ) в выбранных единицах измерения, т.е. страничные координаты этой точки.

Таким образом, подпрограммы **REGION, POLREG** определяют физическое пространство для рисования графиков, а подпрограмма **LIMITS** – соответствие между математическим и физическим пространствами. Если в одной и той же области необходимо нарисовать несколько графиков, то для каждой функции подпрограммой **LIMITS** каждый раз устанавливают свой масштаб.

Для пересчета координат используются вспомогательные подпрограммы **TMF, TFM, TPF**. Их удобно применять, например, для вывода надписей, поясняющих характерные точки графика, поскольку точка задается математическими координатами, а расположение текста – страничными.

После завершения графического вывода в пределах страницы она должна быть закрыта. Закрытие страницы производится либо подпрограммой **ENDPG**, либо автоматически при открытии новой страницы. При этом печатающая головка будет выведена за пределы страницы в новое начальное положение. В одной программе может быть открыто любое число страниц.

Последней подпрограммой, завершающей программу графического вывода, должна быть подпрограмма **GRFIN**, по которой производится окончательная очистка буфера (см. § 17.1).

17.3. Пример построения графиков на УПЗ

В качестве примера рассмотрим программу для построения графиков двух функций $y_1(x)$ и $y_2(x)$ в одних и тех же осях. Пусть функции заданы массивом абсцисс и двумя массивами ординат с размерностью 100. Макет требуемого изображения приведен на рис. 17.2.

На макете показаны следующие данные для построения графиков. Направление оси абсцисс – вдоль оси x (вдоль бумажной полосы). Размеры страницы – x_s, y_s, наименование – СТРАНИЦА 1, границы должны быть очерчены пунктирной линией.

Над областью графика должен быть текст ГРАФИКИ Y (X) с удвоенным размером шрифта относительно начальной точки с координатами x_u, y_u, направление строки текста – вдоль оси x (угол поворота равен нулю).

Координаты левого нижнего угла области графика – x_0, y_0, размеры области вдоль осей – x_l, y_l, наименование области и очерчивание границ не требуется. Пределы изменения аргумента и функции в области графика – x_{m_0}, x_{m_l}, y_{m_0}, y_{m_l} (математические значения абсциссы и ординаты на границах области). Координатные оси должны идти по границам области, шаг основных делений – UX, UY (математические значения), число дополнительных делений между основными – KX, KY. График функции $y_1(x)$ должен быть очерчен сплошной линией, $y_2(x)$ – штриховой.

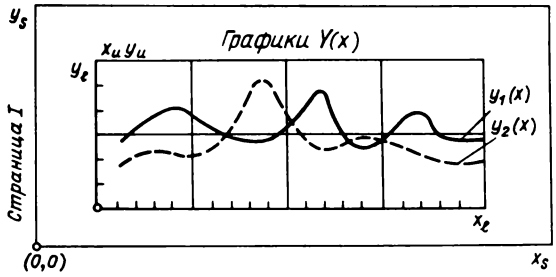


Рис. 17.2. Макет примера построения графиков функций

Программа графического вывода в соответствии с изложенным заданием будет иметь вид

```
FTN4,B,L
C
C   ПРОГРАММА РИСОВАНИЯ ГРАФИКОВ
C
      PROGRAM GRAF
      DIMENSION X(100),Y1(100),Y2(100),NAM(5),ITEXT(6)
C   ВВОД МАССИВОВ С ТЕКСТОМ
      DATA NAM/2HСТ,2HРА,2HНИ,2HЦА,2H 1/,
     *ITEXT/2HГР,2HАФ,2HИК,2HИ ,2HУ(,2HХ)/
C   ВВОД ДАННЫХ
      READ(5,*) XS,YS,XN,YN,X0,Y0,XL,YL,UX,KX,UY,KY,
     *XMO,XML,YMO,YML
      READ(5,*) X,Y1,Y2
C   УСТАНОВКА ЛОГИЧЕСКОГО НОМЕРА УПЗ
      CALL GRINT(7)
C   ТИП ЛИНИИ - ПУНКТИРНАЯ
      CALL SETLIN(-1)
C   ОТКРЫТИЕ СТРАНИЦЫ С УКАЗАНИЕМ ИМЕНИ
C   (10 СИМВОЛОВ) И ОЧЕРЧИВАНИЕМ ГРАНИЦ
      CALL PAGE(XS,YS,NAM,10,1)
C   ВЫВОД ТЕКСТА ИЗ МАССИВА ITEXT (12 СИМВОЛОВ)
      CALL SYMBOL(XN,YN,2.,ITEXT,12,0.)
C   ОТКРЫТИЕ ОБЛАСТИ БЕЗ УКАЗАНИЯ ИМЕНИ И ОЧЕРЧИВАНИЯ
      CALL REGION(XO,YO,XL,YL,0,0,0)
C   УСТАНОВКА ПРЕДЕЛОВ ИЗМЕНЕНИЯ АРГУМЕНТА И ФУНКЦИИ
      CALL LIMITS(XMO,XML,YMO,YML)
C   ПРОВЕДЕНИЕ И РАЗМЕТКА ОСЕЙ БЕЗ УКАЗАНИЯ НАИМЕНОВАНИЙ
C   С НАНЕСЕНИЕМ КООРДИНАТНОЙ СЕТКИ
      CALL AXES(0,0,UX,KX,0,0,UY,KY,3)
C   ТИП ЛИНИИ - СПЛОШНАЯ
      CALL SETLIN(0)
C   ПОСТРОЕНИЕ ГРАФИКА Y1(X)
      CALL LINEO(X,Y1,100)
C   ТИП ЛИНИИ - ШТРИХОВАЯ
      CALL SETLIN(3)
C   ПОСТРОЕНИЕ ГРАФИКА Y2(X)
      CALL LINEO(X,Y2,100)
C   ЗАКРЫТИЕ СТРАНИЦЫ БЕЗ ВЫВОДА ТЕКСТА
      CALL ENDRG(0)
C   ОЧИСТКА БУФЕРА
      CALL GRFIN
      END
      END¤
```

17.4. Структура библиотеки подпрограмм для ЦГТ

Библиотека подпрограмм для ЦГТ (БП ЦГТ) позволяет создавать программы на языках МНЕМОКОД или ФОРТРАН для вывода алфавитно-цифровой и графической информации на экран цветного телевизионного приемника.

Библиотечные подпрограммы позволяют формировать следующие графические элементы: алфавитно-цифровые и псевдографические символы, точку, вектор, заштрихованную площадку. Каждый графический элемент определяется на экране абсолютными или относительными координатами и содержит признаки цвета, мерцания и типа линии. Началом отсчета абсолютных координат является нижний левый угол рабочего поля экрана.

Подпрограммы БП ЦГТ можно разделить на следующие группы: 1 – организующие и вспомогательные; 2 – формирования графических

элементов; 3 – работы с графическими объектами; 4 – преобразования графического изображения; 5 – работы с курсором.

Подпрограммы первой группы распределяют память для хранения графических слов (открывают текущий буфер в разделе памяти УВК), производят выдачу диагностических сообщений, установку типа линии и цвета, признака мерцания.

Подпрограммы второй группы являются основными рабочими подпрограммами. С их помощью можно построить отдельные точки, векторы, графики функций, гистограммы. Результатом работы этих подпрограмм является последовательность графических слов, соответствующих типам графических элементов. Сформированные графические слова заносятся в текущий буфер, образуя *дисплейный файл*, который в свою очередь делится на сегменты.

Каждый сегмент соответствует *графическому объекту*, т.е. законченному изображению, реализованному с помощью БП ЦГТ. В текущем буфере сегмент представляет собой часть буфера от начала до первой свободной ячейки. Специальной подпрограммой сегмент закрывается, и ему присваивается двухсимвольное имя – имя графического объекта, при этом начало текущего буфера переносится в первую свободную ячейку.

Третья группа подпрограмм предназначена для выполнения операций с графическими объектами: формирование объектов (записи в таблицу объектов имени, адреса и длины), вывода на экран или убирания с экрана, изменения цвета или мерцания, добавления новой графической информации к объекту, уничтожения и копирования объекта, перемещения его по экрану.

Подпрограммы четвертой группы обеспечивают преобразование графического изображения: изменение масштаба, поворот вокруг начала координат или любой заданной точки экрана, сдвиг изображения в заданном направлении.

К пятой группе относятся подпрограммы для работы с курсором: перемещение курсора в заданном направлении на один шаг или в заданную точку с абсолютными координатами. Для работы с курсором в интерактивном режиме необходимо использовать клавиатуру от другого устройства.

По сравнению с БПГ библиотека подпрограмм для ЦГТ предоставляет меньше возможностей для рисования геометрических фигур, в ней также нет подпрограмм для построения и разметки осей графиков. Все такие подпрограммы пользователь должен составлять сам на основе имеющихся подпрограмм рисования отдельных элементов.

В то же время в БП ЦГТ развиты возможности преобразования изображений, перемещения их по экрану, вывода нескольких графических объектов, оперирования как двумерными, так и трехмерными изображениями, выделения мерцанием отдельных фрагментов изображений.

ВНИМАНИЮ ЧИТАТЕЛЕЙ!

Научно-производственное и проектно-технологическое объединение "Атлант" Всесоюзного общества "Знание" и научно-внедренческий кооператив "Эклип" по разработке и внедрению проблемно-ориентированных систем предлагают:

многоканальные АЦП, ЦАП, устройства дискретного ввода-вывода и другие программно-аппаратные средства связи с объектом (модули УСО) для мини- и микроЭВМ типа СМ, ДВК, ЕС, IBM PC;

спецпроцессоры быстрого преобразования Фурье, производительностью до 1 млн. базовых операций/с совместимые с ЭВМ указанных типов;

программные средства расширения ДОС АСПО СМ-2М и ВК ПС-2000, в том числе средства автоматизации учета пользователей, дополнительные средства отладки программ на МНЕМОКОДе и ФОРТРАНЕ IV;

программно-аппаратные средства для подключения к каналу данных ВК ПС-2000 дополнительных субкомплексов внешней памяти и дисплейных станций типа ДГП-3, ДГП-7, а также заключают договора на проведение работ по сопряжению ЭВМ общего назначения с лабораторным, испытательным и технологическим оборудованием.

Заявки направляйте по адресу: 113461 Москва, ул. Каховка, 21 НВК "Эклип". Справки по телефону 193-25-46.

Ответы (указания) к упражнениям

К главе 2

1. а) 100 000, 150 000, 250 000;
(102 000–147 777), (152 000–247 777), (252 000–277 777);

б)

Адреса ячеек	Содержимое
151 100	100 000
151 101	000 377

в)

0 0 1 0 1 1 0 1	0 0 1 1 0 0 0 0	= 026 460
0 0 1 0 1 1 0 0	0 0 1 1 0 1 0 1	= 026 065
,	5	

К главе 3

1. Программа должна включать:

выделение нулевого, пятого и шестого разрядов приемом, аналогичным (3.47), с маской 103 000;

сравнение выделенного кода с признаками из табл. 3.1 (совпадение с 0 или 1000 будет означать прямую адресацию к нулевой странице, с 2000 – прямую адресацию к текущей странице, с 100 000 или 101 000 – косвенную через нулевую страницу и т.д.);

в зависимости от установленного способа адресации выделение адреса в нулевой странице (маска 1777) либо смещения (маска 777);

в случае адресации с использованием индексных регистров, вызов на аккумулятор содержимого *РИ1* или *РИ2* и анализ нулевого разряда;

построение искомого адреса сложением базового адреса (содержимого *РНК* или *РИ1/2* и смещения со знаком, если установлена автоиндексация или адресация к текущей странице, или сложение с положительным смещением, если установлена индексация;

в случае косвенной адресации анализ нулевого разряда адресуемой ячейки.

2. Это возможно потому, что в подпрограмме (3.61) имеется единственная ячейка, через которую организована косвенная адресация (ячейка *АДРМ*) и, кроме того, не используется аккумулятор *РБ*.

3. В подпрограмме можно будет использовать лишь один индексный регистр *РИ1* (в *РИ2* – база области реентабельности); всего же требуется организовать три переменных индекса: два для исходных массивов и один для результирующего. Поэтому в подпрограмме дважды следует применить прием (3.59).

К главе 4

1.

```
УС      DEC  – 10000        –10000₁₀ задает интервал
В       NOP                 Δτ = 10 мкс и ΔГТ – Т = 0,1 с
*
        LDA  УС             Обнуление нулевого разряда (УС)
        ELA,CLE,ERA         увеличивает Δτ и, следовательно,
*                           ΔГТ–Т в 10 раз.
        RI0  20B            Код выборки ТМР равен 20₈.
        OTA  20B,C          ТМР запущен.
*       ...
        LTA  20B            РА := текущее значение счетчика Сч интервалов Δτ
```

2. Команда **STC** *КВ* не только разблокирует запросы на прерывания меньших приоритетов, но и сбрасывает маску устройства с кодом выборки *КВ* и, следовательно, делает возможным прерывание по запросам этого УВВ. Чтобы предотвратить повторное прерывание, в драйвер устройства непосредственно вслед за командой **STC** *КВ* следует включить команду **CLC** *КВ* или команду сброса сигнала готовности (для многих УВВ, например, для таймера это команда **STF** *КВ*).

К главе 5

1. в)

2. а) 36_{10}; б) (*РНК*) + 10_8.

3. В соответствии с ⟨5.3.1.⟩ обязательным является *список параметров*, который в данном случае отсутствует.

К главе 6

1. Первая команда устанавливает режим индексации и поэтому 9-разрядный дополнительный код смещения −1 будет восприниматься как положительное число 777_8. Таким образом, в данном случае **LDA** *−1*,**X** ⟨=⟩ *РА*:=(1177).

2. Пропуск команды **CLB** будет действительно происходить при условии, что (РА) ≠ 0. Однако это условие будет нарушаться с периодичностью 1 раз на $200\,000_8$ повторений команды **INA** за счет переноса из старших разрядов *РА*.

3. Освобождающиеся разряды должны заполняться битом из знакового разряда.

4. Под списком понимается совокупность слов, в которой в качестве разделителя используется запятая, признак конца списка – пробел. Программу удобно построить на двух основных операциях: **CBT** – операции сравнения массивов байтов (один из этих массивов – символьная константа **TASK**, другой – четыре байта заданного массива) и операции **SFB** – сканирования байтов с РА = = ␣ , что обеспечивается командой **LDA = A** ,. Если **CBT** обнаружит несовпадение, то **SFB** найдет запятую и начало очередного слова, а если сравнение с константой **TASK** пройдет успешно, **SFB** проверит, является ли следующий символ запятой или пробелом. В противном случае поиск должен быть продолжен.

К главе 7

1. а) 142 075; б) **М** = 10_8 (абсолютное), **М1** = 0 (перемещаемый в программе, **М2** = 13_8 (перемещаемое и программное).

2. В соответствии с определениями ⟨6.1.8.⟩ и ⟨7.2.7.⟩, мнемоника **CBS** и мнемоника **DEF** могут использоваться с литералами. Литералы будут построены транслятором вне основной памяти программы, а во втором и третьем словах команды **CBS** транслятор разместит их адреса.

3. При программировании тела цикла (3.57) в программе (3.58) перед оператором **SSA,RSS** добавить три оператора:

```
IFN
CMA,INA
XIF
```

К главе 8

```
MACRO
ВВОДМ   &ЛН,&0Ш,&БФ,&N
EXIT    .DIO  .,. IAY.
LDA     =B&ЛН
CLB,INB
JSB     .DIO.
OCT     0
```

```
        DFF     & ОШ
        JSB     JAY
        DEF     & БФ
        DEC     & N
        MEND
```

К главе 9

1. NAM *E*,3,,13,*45*,*45*.
2. а) Составление программы генерации ОС и задач пользователя;
   б) макрогенерация программы генерации ОС и задач пользователя;
   в) трансляция программы генерации ОС и задач пользователя;
   г) компоновка загрузочного модуля.
3. :*2*
   :ВД,*ДДПРВ*
4. WP = 00

К главе 10

1. Если какое-нибудь событие, заказанное в одной макрооперации, полностью завершается при переходе к следующей макрооперации, то в каждой последующей макрооперации можно использовать ячейки отработанного БУСа или БЗ из предыдущей макрооперации. Поэтому в задачах А и В излишне резервировать ячейки БУС и БУС1, когда вполне можно использовать рабочие ячейки БЗ.

2. Задача М

```
ASMB,R,B,L
        NAM   M,1
        RUN   N,БУС1, =L-16,=L234
        STIME БУС2,=L1,=L82
        WAITM   БУС0,БУС1,БУС2
        LDA   БУС0
        CPA   =L2
        JMP   M1
        EXIT
M1      EXIO  =L6,=L2,БЗ,БУФ,=L12
        WAITE БЗ
        EXIT
*
БУС1    BSS   1
БУС2    BSS   1
БУС0    BSS   2
БЗ      BSS   2
БУФ     ASC   12,ВРЕМЯ ВЫПОЛНЕНИЯ ИСТЕКЛО
        END
        ENDM
```

3. При использовании общего ресурса обе задачи должны зарезервировать общую ячейку БУСа с исходным содержимым WP, равным 01. Тогда при занятии ресурса любая из задач выполняет макрооперацию WAITE, а при освобождении – POSTE.

4. Задача L

```
ASMB,R,B,L
        NAM   L,1
        TURN  N,БУС,=L2,=L360,=L-1334
        PAUSE
        LDB   1,I
```

```
        SZB,RSS
        EXIT
        EXIO   =L1,=L2,БЗ,БУФ,=L6
        WAITE  БЗ
        EXIT
*
БУС     BSS    1
БЗ      BSS    2
БУФ     ASC    6,КОНЕЦ РАБОТЫ
        END
        ENDM
```

К главе 11

1. **а)** Назначение свободному логическому номеру имени файла;
   б) открытие файла; **в)** выполнение операции чтения/записи;
   г) закрытие файла; **д)** освобождение логического номера.
2.
```
        EXIO   =L0,=L20002B,БЗ,БУФ,=L9
        WAITE  БЗ
```
Здесь
```
БУФ     OCT    0
        ASC    4,ОЛИМП
        OCT    0,0,0
        DEC    21
```
3.
```
ASMB,R,B,L,T
        NAM    D,3
```
Остальные операторы идентичны со следующими исключениями:

а) для опроса давления используются
```
        EXIO   =L8,=L101B,БЗ,БУФ,=L1,=L0
```
б) в буфере вывода БУФВ произвести замену:
```
ОС      NOP
        ASC 8,СЕК.ДАВЛЕНИЕ
```
в) для вывода очередной записи из буфера БУФВ:
```
ВЫВОД   PUT,БЗ,БУФВ,=L23
```

СПИСОК ЛИТЕРАТУРЫ

1. **Костелянский В.М., Резанов В.В.** Управляющий вычислительный комплекс М-6000// Измерения, контроль, автоматизация. М.: ЦНИИТЭИ приборостроения, 1974. Вып. 1.

2. **Управляющие** вычислительные машины в АСУ технологическими процессами/ Под ред. Т. Харрисона. М.: Мир, 1975. Т.1, 2.

3. **Резанов В.В., Костелянский В.М.** Программное обеспечение УВК СМ-1 и СМ-2// Приборы и системы управления, 1977. № 10. С. 12–15.

4. **СМ ЭВМ, АСВТ-ПС.** Технические и программные средства. Каталоги, ч. I–IV. Внешторгиздат, 1984.

5. **Агрегатные** комплексы технических средств в АСУ ТП/ Н.А. Боборыкин, А.А. Андреев, В.П. Теленков и др.// Под ред. Н.А. Боборыкина. Л.: Машиностроение, 1985.

6. **Каган Б.М., Каневский М.М.** Цифровые вычислительные машины и системы. М.: Энергия, 1974.

7. **Вигдорчик Г.В., Воробьев А.Ю., Праченко В.Д.** Основы программирования на АССЕМБЛЕРе СМ ЭВМ. М.: Финансы и статистика, 1983.

8. **Дал У., Дейксра Э., Хоор К.** Структурное программирование. М.: Мир, 1975.

9. **Хьюз Дж., Мичтом Дж.** Структурный подход к программированию. М.: Мир, 1980.

10. **Брукс Ф.П. мл.** Как проектируются и создаются программные комплексы. М.: Наука, 1979.

11. **Майерс Г.** Надежность программного обеспечения. М.: Мир, 1980.

12. **Бруснецов Н.П.** Миникомпьютеры. М.: Наука, 1979.

13. **Экхауз Р., Моррис Л.** Мини-ЭВМ: организация и программирование. М.: Финансы и статистика, 1983.

14. **Пратт Т.** Языки программирования: разработка и реализация. М.: Мир, 1979.

15. **Алгоритмический** язык АЛГОЛ-60/ Под ред. П. Наура. М.: Мир, 1965.

16. **Резанов В.В., Костелянский В.М.** Средства вычислительной техники для управления технологическими процессами и обработки геофизической информации. Состояние и перспективы развития// Приборы и системы управления, 1983. № 1. С.7.

17. **Донован Дж.** Системное программирование. М.: Мир, 1975.

18. **ФОРТРАН ЕС ЭВМ.** М.: Статистика, 1978.

19. **Айнберг В.Д.** Основы программирования на алгоритмическом языке АЛГОЛ-60. М.: Машиностроение, 1978.

20. **КОБОЛ ЕС ЭВМ.** М.: Статистика, 1978.

21. **Непомнящий В.А.** Верификация программ обработки файлов на языке ПАСКАЛЬ. Программирование, 1981. № 2. С.34–43.

22. **Вирт Н.** Язык программирования ПАСКАЛЬ (Пересмотренное сообщение): Пер. с англ.// Алгоритмы и организация решения экономических задач. М.: Статистика, 1977. Вып. 9. С.52–86.

23. **Браун П.** Макропроцессоры и мобильность программного обеспечения. М.: Мир, 1977.

24. **Изаксон П.** Обзор перспектив применения языка БЕЙСИК// Электроника, 1980. Т.37, № 4.

25. **Бекетов А.С., Смолкин В.М.** Автоматизированная система изготовления текстовых документов машинным способом// Управляющие системы и машины, 1985. Т.5. С.116–118.

26. **Отладка** систем управляющих алгоритмов/ В.В. Липаев, Л.А. Фидловский и др. М.: Советское радио, 1974. С.27–35.

27. **Система** интерактивной отладки программ на уровне МАКРОАССЕМБЛЕРА (ТЕМП)/ В.М. Смолкин, Л.П. Волох, В.П. Клименко и др.// Управляющие системы и машины, 1986. Т.4.

28. **Бекетов А.С., Смолкин В.М.** Макропроцессор МП-25// Управляющие системы и машины, 1988. Т. 3. С. 43–44.

29. **Пакет** программ для генерации задач сбора и обработки информации в АСУ ТП/ М.М. Галочкина, В.М. Костелянский, В.И. Макарова и др.// Приборы и системы управления, 1981. № 1 С.9–11.

30. **Кетков Ю.Л.** Программирование на БЕЙСИКе. М.: Статистика, 1978.

31. **Мартин Дж.** Организация баз данных в вычислительных системах: Пер. с англ./ Под ред. М.М. Стогния и А.Л. Щерса. М.: Мир, 1980.

32. **Баяковский Ю.М., Галактионов В.А., Михайлов Т.Н.** ГРАФОР. Графическое расширение ФОРТРАНа. М.: Наука, 1985.

ОГЛАВЛЕНИЕ