

**БИБЛИОТЕЧКА
ПРОГРАММИСТА**

В. И. ЗВЕРЕВ
Ю. А. КЕТКОВ
В. С. МАКСИМОВ

**Алфавитно-цифровые
дисплеи ЕС-7920
в диалоговых
системах**



БИБЛИОТЕЧКА ПРОГРАММИСТА

В. И. ЗВЕРЕВ, Ю. Л. КЕТКОВ
В. С. МАКСИМОВ

АЛФАВИТНО-ЦИФРОВЫЕ ДИСПЛЕИ ЕС-7920 В ДИАЛОГОВЫХ СИСТЕМАХ



МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1986

Зверев В. И., Кетков Ю. Л., Максимов В. С.
Алфавитно-цифровые дисплеи ЕС-7920 в диалоговых системах.—
М.: Наука, Гл. ред. физ.-мат. лит., 1986. — 240 с. — (Библиотечка программиста.)

Описываются функциональные возможности одной из наиболее распространенных дисплейных станций ЕС-7920 локального типа, на базе которых функционируют и разрабатываются различные диалоговые системы. Приводятся сведения об аппаратуре и программных средствах обеспечения обмена с локальными дисплеями, представляемыми штатными операционными системами ЕС ЭВМ. Обсуждается технология разработки программы обмена информацией с использованием различных уровней доступа. Излагаются принципы построения и реализации конкретных диалоговых систем, обеспечивающие коллективный доступ к ЕС ЭВМ, редактирование текстовых данных, обмен информацией между пользователем и прикладной программой.

Для пользователей ЕС ЭВМ, инженеров и студентов вузов.
Табл. 19, Ил. 33, Библиогр. 15 назв.

Рецензент

член-корреспондент АН СССР Л. Н. Королев

ОГЛАВЛЕНИЕ

Предисловие	5
Г л а в а 1. Введение	7
1.1. Основные сферы применения диалоговых систем	7
1.2. Дисплейная техника — шаг на пути к безбумажной информатике	9
1.3. Почему мы решили написать эту книгу?	10
Г л а в а 2. Характеристика аппаратных средств	12
2.1. Экран и клавиатура дисплея	13
2.2. Форматизованный экран	19
2.3. Метка конца поля	25
2.4. Клавиша ВЫЗОВ ТЕСТА	25
2.5. Изменение информации в буферной памяти дисплея по командам из ЭВМ	25
2.6. Считывание информации с экрана дисплея	27
2.7. Команды управления	29
2.8. Основные отличия между локальными и удаленными станциями	31
Г л а в а 3. Программирование обмена в среде БТМД	33
3.1. Что такое БТМД	33
3.2. Описание и инициализация терминала	34
3.3. Формат макрокоманд обмена с локальным дисплеем	36
3.4. Анализ результатов выполнения операции	39
3.5. Специфика ввода по инициативе пользователя	43
3.6. Дополнительные возможности по многотерминальному обслуживанию	45
Г л а в а 4. Особенности программирования под управлением ОТМД	47
4.1. Что такое ОТМД	47
4.2. Потоки данных в системе ОТМД	49
4.3. Структура программы управления сообщениями	50
4.4. Функциональные возможности прикладной программы в среде ОТМД	55
Г л а в а 5. Элементы программирования на физическом уровне с использованием средств БТМД	60
5.1. Описание и инициализация терминала в ДОС ЕС	61
5.2. Описание и инициализация терминала в ОС ЕС	62
5.3. Обработка сигнала ВНИМАНИЕ В ДОС ЕС	65
5.4. Обработка сигнала ВНИМАНИЕ в ОС ЕС	68
5.5. Структура программ канала для основных процедур обмена	69

Глава 6. Проектирование подпрограммы обмена с терминалом	71
6.1. Логическая схема подпрограммы	72
6.2. Соглашения об использовании регистров	77
6.3. Инициализация терминала	78
6.4. Обработка параметров	79
6.5. Реализация обменов с дисплеем	80
6.6. Сравнение вариантов	87
Глава 7. Основные принципы построения дисплейного редактора текстовой информации	88
7.1. Что такое естественное редактирование	88
7.2. Структура экрана	91
7.3. Входной и выходной рабочие файлы	92
7.4. Язык общения с редактором и программные средства синтаксического контроля	94
7.5. Работа с библиотеками исходных текстов	97
7.6. Формирование исходного текста во входном файле	100
7.7. Основные процедуры коррекции исходного текста и их реализация	101
7.8. Экранный сервис	108
7.9. Работа с внешними устройствами	114
7.10. Внутренние подпрограммы редактора	116
7.11. Основные проблемы многотерминального обслуживания	125
Глава 8. Диалоговые системы коллективного доступа	134
8.1. Основные принципы построения систем	135
8.2. Основные функциональные возможности систем	138
8.3. Специальные возможности системы JEC	141
8.4. Дополнительные функции в системе PRIMUS	142
Глава 9. Концепции, архитектура и возможности системы БЕС7920	143
9.1. Назначение и возможности системы	143
9.2. Особенности использования дисплейных станций	144
9.3. Языковые средства системы	148
9.4. Структура программного обеспечения	159
9.5. HELP-функция как средство обучения и информационного обслуживания пользователей	188
9.6. Перспективы развития системы БЕС7920	194
Глава 10. Диалог с прикладной программой на базе ППП ДИАФОР	197
10.1. Организация обмена с дисплеем	198
10.2. Использование функциональной клавиатуры для работы в режиме «меню»	204
10.3. Обработка символьных и числовых данных	205
10.4. Изменение параметров прикладной программы в режиме диалога	215
10.5. Отображение выходных параметров по запросу пользователя	224
Заключение	226
Приложение	228
Список литературы	236

ПРЕДИСЛОВИЕ

Почти полтора десятилетия отделяют нас от момента появления первых моделей ЭВМ третьего поколения — электронных вычислительных машин Единой серии (ЕС ЭВМ). Их аппаратная реализация и программное обеспечение были ориентированы на параллельное выполнение заданий, обеспечивающее максимальную загрузку вычислительной системы. На машинах третьего поколения достигли наивысшего развития идеи и принципы режима пакетной обработки, основы которых закладывались в системах программирования на ЭВМ предшествующих поколений.

Пакетная обработка обладает высокой эффективностью в технологической цепочке, связанной с эксплуатацией на ЭВМ готовых программ. Однако пакетный режим неприемлем для разработки новых программ, поиска оптимальных решений в сложных научно-исследовательских и проектных задачах, автоматизированного обучения и т. п. Многочасовые ожидания результатов очередного запуска задания на ЭВМ резко затягивают календарные сроки программных разработок, уменьшают эффективность научных исследований.

Поэтому через несколько лет после появления первых моделей ЕС ЭВМ и освоения их достаточно сложного математического обеспечения в ряде организаций были развернуты работы по созданию различных диалоговых систем, повышению их оперативности и увеличению сервисных функций, предоставляемых пользователям ЭВМ в режиме диалога.

Развитие и внедрение подобных систем немыслимо без современной дисплейной техники с ее многофункциональной клавиатурой и встроенным оборудованием, позволяющими обрабатывать не только алфавитно-цифровую, но и графическую информацию. Дисплеи находят все большее применение в автоматизированных системах управления производством и технологическими процессами, в системах автоматизированного проектирования и конструирования, в учебном процессе и научно-исследовательской работе, в сфере обслуживания и в быту.

В рамках данной книги мы остановимся на проблемах разработки программного обеспечения для обмена между ЕС ЭВМ и алфавитно-цифровыми дисплеями ЕС-7920 локального типа, а также на технологии использования этих дисплеев в различных диалоговых системах. Сделаем некоторое пояснение по поводу выбора тематики и структуры излагаемого материала.

Алфавитно-цифровая дисплейная техника, которая эксплуатируется в составе вычислительных комплексов Единой серии, представлена тремя основными моделями. К первому поколению относится дисплейная станция типа ЕС-7066 с устройствами отображения ЕС-7906. Эти дисплеи (уже снятые с производства) пред-

ставляют собой разновидность электронного варианта пишущей машинки. У них нет собственной памяти и для регенерации изображения на экране используется общая память, расположенная в устройстве управления. Поэтому число строк текстовой информации, видимых на экране, зависит от количества дисплеев, подключенных к устройству управления. Для установления связи с ЭВМ на клавиатуре ЕС-7906 существует единственная функциональная клавиша.

Второе поколение алфавитно-цифровых дисплеев представляют собой дисплейные станции типа ЕС-7920 с устройствами отображения ЕС-7927. Их функциональные возможности подробно описаны в гл. 2. Акцент на станции локального типа нами сделан по двум причинам. Во-первых, некоторую информацию об удаленных станциях типа ЕС-7920 можно найти в книге [4]. Во-вторых, мы ориентировались на диалоговые системы, в которых время реакции не превышает 1—2 с, тогда как для удаленных дисплеев только время заполнения экрана исчисляется уже десятками секунд.

В последнее время начали появляться «интеллектуальные» терминальные станции типа ЕС-7970, устройство управления которых допускает автономное использование их под управлением встроенного микропроцессора. Одной из характерных особенностей новых станций является возможность работы в режиме ЕС-7920. Поэтому программное обеспечение диалоговых систем, ориентированных на ЕС-7920, может быть легко адаптировано для работы с новыми дисплеями.

Материал первых пяти глав носит учебный характер. Сюда включены сведения о функциональных возможностях аппаратных средств и об управлении локальными дисплейными станциями в объеме, необходимом для понимания канальных программ. Описаны процедуры обмена информацией между ЭВМ и дисплеями, обеспечиваемые штатными средствами операционных систем. Глава 6 посвящена реализации и обсуждению различных вариантов подпрограммы обмена с одиночным терминалом. Все они представляют собой законченный программный продукт, который может быть использован при проектировании конкретных диалоговых систем.

Остальная часть книги (гл. 7—10) посвящена изложению принципов и методов построения конкретных диалоговых систем на базе дисплейных станций типа ЕС-7920. Выбор излагаемого материала носит субъективный характер — авторы описывают системы, в разработке которых они принимали непосредственное участие. Однако диапазон представленных здесь систем достаточно широк; среди них — дисплейный редактор текстовой информации (гл. 7), система коллективного доступа с широкими функциональными возможностями (гл. 9), пакет программ для организации диалога между пользователем и прикладной программой (гл. 10). В каждой из этих глав делалась попытка показать внутреннюю структуру, обратить внимание на проблемы, возникающие при разработке подобного рода систем, привести конкретную реализацию отдельных блоков.

Главы 1—4, 7 и 10 написаны Ю. Л. Кетковым, гл. 5 и 6 — В. С. Максимовым, гл. 8 и 9 — В. И. Зверевым.

В. И. Зверев, Ю. Л. Кетков, В. С. Максимов

ГЛАВА 1 ВВЕДЕНИЕ

1.1. Основные сферы применения диалоговых систем

На любом этапе развития вычислительной техники общение пользователя с ЭВМ представляет собой более или менее оперативную форму диалога. Основные фазы этого взаимодействия, с которыми программистам приходится сталкиваться ежедневно, отображены на циклограмме (рис. 1.1).

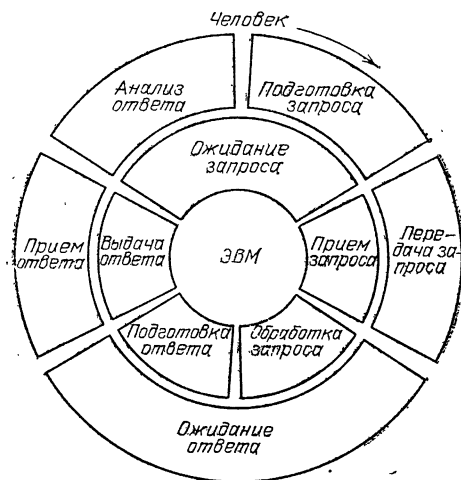


Рис. 1.1. Циклограмма человек — ЭВМ

Отладка программ на машинах первого поколения напоминала диалог по заранее подготовленному сценарию. Программист набирал на пульте останов по адресу в интересующей его точке программы, пресматривал на пультовом «дисплее» содержимое ячеек оперативной памяти, выдавал отдельные фрагменты на печать, исправлял замеченные ошибки. И эти процедуры в течение одного сеанса отладки выполнялись многократно.

Термин «дисплей» мы заключили в кавычки только потому, что 30 лет назад им еще не пользовались. Однако электронно-лучевые трубки в те времена применялись и в качестве оперативных запо-

минающих устройств (например, ЭВМ «Стрела»), и как пассивные пультовые устройства отображения содержимого ячеек оперативной памяти, и как средства индикации результатов работы тестовых программ.

Появление первых алфавитно-цифровых терминалов (телетайпов, электрифицированных пишущих машинок) послужило стимулом к разработке более совершенных диалоговых отладочных систем, включавших развитые средства оперативного редактирования текстов исходных программ. Одновременно диалоговые средства начинают проникать и в прикладные программы.

Появляются первые информационно-справочные системы, работающие по принципу вопрос — ответ. Первые образцы такого рода систем использовались для привлечения посетителей крупных промышленных выставок, для информационного обслуживания судей и журналистов на международных соревнованиях.

Середина 60-х годов знаменуется появлением первых диалоговых систем программирования, построенных на базе интерактивных алгоритмических языков джосс и бейсик. Техника пошаговой трансляции и пошагового выполнения рабочих программ в этих системах оказала большое влияние на развитие систем программирования коллективного пользования. При их разработке приходилось сталкиваться с многочисленными проблемами организационного характера — выбором дисциплины обслуживания, организацией много-терминальной работы без тиражирования транслирующих систем в оперативной памяти, защитой ресурсов от несанкционированного доступа и т. п.

Конец 60-х и начало 70-х годов характеризуются появлением многочисленных обучающих систем, нуждающихся в совершенствовании методов представления знаний и технологии общения с не очень опытным пользователем. В это же время возникает и активно развивается новое направление, связанное с интерактивным использованием средств вычислительной техники в задачах автоматизированного проектирования и конструирования различных изделий. Позднее оно приводит к созданию специализированных автоматизированных рабочих мест (АРМ) конструктора, в которых основным режимом общения с ЭВМ является диалог с помощью алфавитно-цифрового или графического дисплея. Оба эти направления стимулировали работы по созданию банков и баз данных, а также по совершенствованию программных средств информационного обслуживания прикладных программ и терминальных пользователей.

Объединение вычислительных машин в сети ЭВМ, характерное для середины 70-х годов, еще больше расширило сферу применения диалоговых средств. Наряду с простейшим терминальным оборудованием появились абонентские пункты, сочетавшие в себе мощные средства ввода и вывода информации и аппаратуру передачи данных

по различным каналам связи. Эта техника обеспечивала доступ широкого круга пользователей к вычислительным мощностям — к централизованным базам данных, накапливающим и обрабатывающим с помощью ЭВМ огромные массивы информации по различным отраслям науки и техники. Количество терминалов, обслуживаемых такими системами, исчисляется тысячами, но оперативность доступа к вычислительным ресурсам или информационным запасам оставляет желать лучшего.

Наряду с гигантскими вычислительными сетями продолжают совершенствоваться и локальные вычислительные центры коллективного пользования (ВЦКП). В таких центрах одна или несколько ЭВМ (обычно однотипных и совместно эксплуатирующих общее поле внешней памяти) оперативно обслуживают несколько десятков пользователей. В качестве терминального оборудования в ВЦКП, как правило, применяются алфавитно-цифровые дисплеи, расположенные на небольшом удалении от ЭВМ.

Наконец, тенденцией 80-х годов во всем мире стало проникновение микропроцессоров и микро-ЭВМ во все сферы человеческой деятельности и, как следствие, появление персональных ЭВМ, ориентированных на широкий диапазон пользователей — от детей до людей пожилого возраста, от домохозяек до профессиональных исследователей. Здесь и развлекательные телеигры, и рецепты кулинарных изделий, и возможность подключиться к сети ЭВМ с целью получения свежей научной информации, а при необходимости и воспользоваться более мощными вычислительными или программными средствами. Все эти процедуры строятся на базе, как правило, достаточно оперативного диалога.

1.2. Дисплейная техника — шаг на пути к безбумажной информатике

Термин «безбумажная информатика» в широком смысле, введенный в литературу академиком В. М. Глушковым, рассматривался им в первую очередь в связи с автоматизацией организационного управления, систем управления технологическими процессами, информационно-справочных служб, экспериментальных исследований, проектно-конструкторских работ и т. п. Этот термин импонирует нам и в более узком смысле — в свете тематики настоящей книги. Дело в том, что бумага — это традиционное средство общения программиста с ЭВМ ранних поколений. Сначала были бланки с текстом исходных программ, потом перфокарты, наконец результаты отладочных выходов — рулоны распечаток, которые через день-два оказываются в корзине.

В вычислительных центрах с интенсивным отладочным режимом темпы потребления бумажных носителей держат хозяйственные

службы в постоянном напряжении. За год такие организации расходуют десятки тысяч бланков для программ, тонны дефицитной фальцованной бумаги для АЦПУ, миллионы перфокарт.

Дисплеи устраняют лишние звенья в цепочке пользователь — ЭВМ, ускоряют оборачиваемость в процессе разработки и отладки программного продукта и в конечном счете приводят к повышению производительности труда программистов.

При установившемся на ЕС ЭВМ режиме пакетной обработки программа от момента ее составления или очередной коррекции до получения отладочного результата переключается из рук в руки десять раз: сначала четыре пересадки на пути программист — диспетчер — перфораторная, затем шесть аналогичных этапов в цепочке программист — диспетчер — оператор — ЭВМ. При хорошем взаимодействии указанных служб программисту удастся в течение рабочего дня совершить один, максимум два выхода на ЭВМ. А это означает, что за день больше одной-двух ошибок в своей программе он не обнаружит.

1.3. Почему мы решили написать эту книгу?

В начале 70-х годов двоим из авторов довелось участвовать в разработке диалоговой системы программирования на базе алгоритмического языка бейсик для ЭВМ типа М-222. Она была введена в эксплуатацию в 1973 г. и в течение 10 лет эксплуатировалась в Горьковском государственном университете им. Н. И. Лобачевского, обслуживая одновременно восемь — десять пользователей. Эксперимент оказался довольно удачным, и с появлением ЭВМ ЕС-1020 мы решили повторить систему в улучшенном исполнении на базе новой техники. К моменту завершения разработки варианта бейсик — ЕС дисплеев у нас еще не было, но использование их в проекте предусматривалось.

Пакетная эксплуатация показала, что новая система обладает более высокой пропускной способностью на задачах учебного характера по сравнению со штатными средствами ДОС ЕС (трансляция с фортрана + загрузка + выполнение). Наконец проблема с терминалами была решена, и в нашем распоряжении оказались две дисплейные станции ЕС-7920 локального типа. Но штатным математическим обеспечением ДОС-2.0 и ОС-4.1 эти дисплеи в то время не поддерживались, так как выпуск соответствующей аппаратуры несколько опередил работы по централизованной доработке версий операционных систем. Поэтому в первую очередь пришлось обратить внимание на то, как стандартные средства базисного телекоммуникационного метода доступа обрабатывают сигнал прерывания (сигнал ВНИМАНИЕ) от других активных устройств. Некоторая информация по обработке ошибочных ситуаций была извлечена при

расшифровке программ автономного тестирования. Затем были написаны простейшие программы обмена с одиночным дисплеем на физическом уровне для ДОС ЕС. Наконец, было налажено обслуживание одновременно восьми терминалов с использованием механизма подзадач. И диалоговый вариант системы бейсик — ЕС заработал.

Следующим шагом по освоению дисплеев ЕС-7920 была довольно робкая попытка использовать функциональную клавиатуру: за несколькими клавишами были закреплены функции стандартных директив бейсика, а одна из клавиш использовалась для синхронизации процесса вывода информации на экран.

Следующим пробным камнем, на котором мы уже более серьезно осваивали функциональные возможности дисплеев, явилось создание многотерминального редактора текстовой информации для ДОС ЕС. Здесь состоялась окончательная отработка отдельной подпрограммы обмена, которая подробно обсуждается в гл. 6. Авторы сознательно приводят несколько реализаций этой программы для разных операционных систем и методов доступа, так как она может быть использована при создании различных диалоговых систем.

Наконец, на конференции «ДИАЛОГ-83» состоялась встреча, заложившая основу авторского коллектива. Один из авторов книги (В. И. Зверев) является руководителем работы по созданию системы коллективного доступа БЕС7920, выполнявшейся совместно Белорусским государственным институтом народного хозяйства им. В. В. Куйбышева и Институтом технической кибернетики АН БССР.

Среди мотивов, побудивших нас взяться за перо, можно выделить следующие. Во-первых, пока доступной литературы справочного или учебного характера по локальным дисплейным станциям просто нет. Во-вторых, техническая документация по операционным системам ЕС ЭВМ также не предлагает отдельного документа по программированию обмена с локальными станциями. Более того, в ней не содержится никаких рекомендаций и по выбору метода доступа. Наконец, мы хотели поделиться опытом по созданию конкретных диалоговых систем и использованию в них богатых функциональных возможностей дисплейных станций ЕС-7920.

ГЛАВА 2

ХАРАКТЕРИСТИКА АППАРАТНЫХ СРЕДСТВ

Дисплейная станция ЕС-7920 локального типа состоит из устройства управления ЕС-7922, к которому может быть подключено до 32 терминалов. В качестве терминалов этой станции используются дисплеи типа ЕС-7927 и активные АЦПУ типа ЕС-7934 или ЕС-7936. Термином «активный» авторы хотели подчеркнуть тот факт, что эти АЦПУ снабжены клавиатурой, поэтому они могут использоваться и как устройства ввода. Среди терминалов, обслуживаемых устройством управления, дисплеи и АЦПУ могут находиться в любой пропорции. Непосредственной связи между различными терминалами нет, поэтому печатающее устройство нельзя использовать автономно от ЭВМ с целью получения твердой копии экрана дисплея. Точно так же передача сообщения с одного терминала на другой может быть обеспечена только программным путем через ЭВМ.

Устройство управления ЕС-7922 находится в непосредственной близости от ЭВМ и подключается к любому из ее каналов — мультиплексному, байт-мультиплексному, блок-мультиплексному или селекторному. Устройство управления располагает собственной памятью емкостью 1920 байт и обменивается с ЭВМ информацией со скоростью не менее 180 кбайт/с. Терминалы соединяются с устройством управления с помощью одножильного телевизионного кабеля, длина которого может достигать 1200—2000 м.

Каждый терминал располагает индивидуальным буферным запоминающим устройством (ЗУ), емкость которого зависит от типа терминала. Для наиболее распространенных дисплеев объем их ЗУ составляет 1920 ячеек. Ячейка буферного ЗУ емкостью 10 двоичных разрядов предназначена для хранения одного информационного (10-й разряд равен 0) или управляющего (10-й разряд равен 1) символа. Девятый разряд в ячейках буферного ЗУ используется для контроля; он формируется аппаратно таким образом, чтобы число единиц в ячейке было нечетно. Разряды ячейки с первого по восьмой соответствуют внутреннему коду информационного или управляющего символа.

Обмен между терминалом и устройством управления производится с помощью 14-разрядных слов, передаваемых по телевизи-

онному кабелю последовательно со скоростью примерно 700 кбит/с, т. е. около $5 \cdot 10^4$ символ/с.

Следует обратить внимание на то, что кодировка информационных и управляющих символов в буферном ЗУ дисплея не совпадает с кодировкой ДКОИ, принятой в ЕС ЭВМ. Однако эти внутренние коды участвуют в передаче данных только по линии между терминалом и устройством управления. В канале между ЭВМ и устройством управления информация передается в коде ДКОИ; нужные преобразования обеспечивает само устройство управления.

2.1. Экран и клавиатура дисплея

Дисплей типа ЕС-7927 (рис. 2.1) состоит из основного блока с электронно-лучевой трубкой и из клавиатуры в виде отдельного блока; последний выпускается в нескольких модификациях. На экране большинства дисплеев размещаются 24 строки емкостью по 80 символов, хотя раньше выпускались и модели, отображавшие 12 строк по 40 символов.

Под экраном дисплея на панели основного блока расположено шесть индикаторов 1—6, ручки регулировки контрастности 7 и яркости 8 изображения и замок 9. Индикатор СЕТЬ 2 горит все время, пока на блок питания терминала подано напряжение 220 В. Для приведения терминала в рабочее состояние необходимо вставить ключ в замок, слегка утопить ключ и повернуть его направо. При первом щелчке (положение ВКЛ соответствует повороту ключа примерно на 45°) должен загореться индикатор включения ВКЛ 3. Загорание аварийного красного индикатора АВАР 1 свидетельствует либо о неисправности дисплея, либо о нарушении режима питания. Следующее фиксируемое положение ключа (поворот примерно на 90°) соответствует режиму РАБОТА, в котором дисплей может обмениваться информацией с ЭВМ. Спустя 1,5—2 с после перевода терминала в рабочее состояние загорается индикатор СИСТ ДОСТ 4 — с этого момента разрешен доступ к ЭВМ. Индикатор ВВОД ЗАПР 6 предупреждает пользователя о том, что либо не завершена предыдущая операция обмена между дисплеем и ЭВМ, либо обнаружена попытка выполнить недопустимую операцию. При нажатии управляющей кнопки ВСТВ загорается индикатор РЕЖИМ ВСТАВКИ 5, используемый для раздвижки информации на экране и вставки пропущенных символов.

Индикаторы 6 и 7 могут быть сброшены нажатием кнопки СБРОС.

Справа от экрана расположен кронштейн, в который вставляется *световое перо*, именуемое в технических описаниях *фотоселектором*. Наконечник светового пера представляет собой светочувствительный элемент, снабженный микровыключателем. Если свето-

вое перо поднести вплотную к экрану и слегка прижать до щелчка микровыключателя, то в блок управления дисплея поступят координаты — номер строки и номер позиции, определяющие местоположение светочувствительного элемента. Таким образом, пользователь может «указать» на тот или иной фрагмент информации, находящейся на экране. При определенных условиях координаты отмеченного таким образом участка экрана могут быть переданы в ЭВМ.

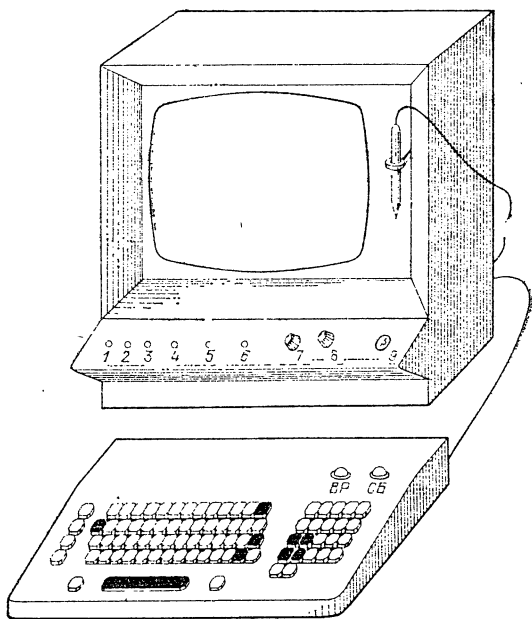
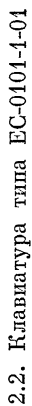


Рис. 2.1. Схема расположения органов управления на дисплее ЕС-7927: 1 — авария; 2 — сеть; 3 — включено; 4 — система доступна; 5 — режим вставки; 6 — ввод запрещен; 7 — ручка регулировки контрастности; 8 — ручка регулировки яркости; 9 — замок на три положения

Клавиатуры дисплеев ЕС-7927 выпускаются в четырех модификациях: клавиатура пульта оператора (ЕС-0101-1-01), клавиатура подготовки данных (ЕС-0101-1-02), клавиатуры пишущих машинок с русским алфавитом (ЕС-0101-1-03) и латинским алфавитом (ЕС-0101-1-04). Наиболее распространены клавиатуры первых двух типов. Состав и расположение их клавиш представлены соответственно на рис. 2.2 и 2.3.

С целью уменьшения общего количества клавиш, обеспечивающих на экране набор 84 различных символов, большинство из них используется для воспроизведения двух символов. Символ, выгра-



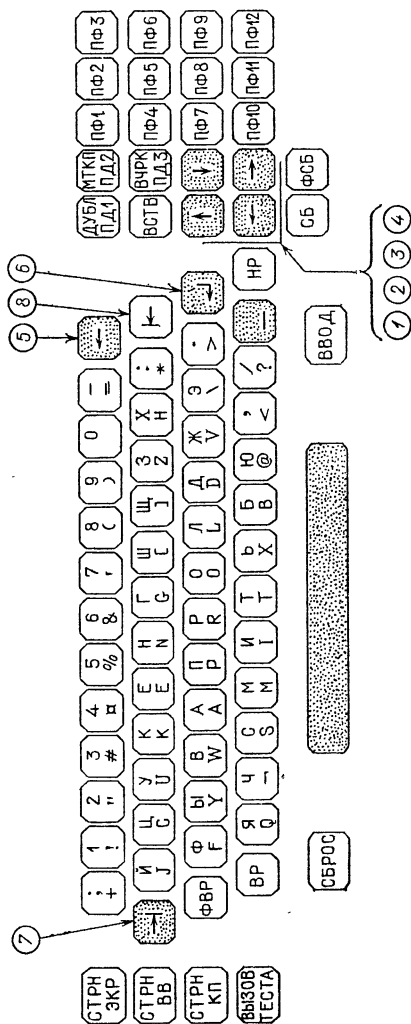


Рис. 2.3. Клавиатура типа ЕС-0101-1-02

вированный в верхней части клавиши, появится на экране в том случае, если нажать наборную клавишу при нажатой одновременно клавише ВР (верхний регистр). Для набора нижнего символа необходимо производить ввод при нажатой одновременно клавише НР (нижний регистр). Для того чтобы зафиксировать соответствующий регистр, существует клавиша ФВР (фиксация положения регистра). При включении дисплея автоматически устанавливается режим нижнего регистра (при этом красный индикатор ВР на клавиатуре не горит). Нажатие клавиши ФВР устанавливает регистровый режим, противоположный текущему.

На нижнем регистре удобно вести набор латинских текстов, на верхнем — русских и цифровых. Клавиатура типа ЕС-0101-1-02 позволяет вести набор цифровой информации (белые клавиши справа) при любом положении регистра. Эта же привилегия распространяется на пробел (на обеих клавиатурах — длинная черная клавиша внизу) и подчеркивание (черная клавиша в нижнем ряду, справа).

Перед конструкторами клавиатуры стояла довольно сложная задача — совместить в пределах сравнительно небольшого числа клавиш русский и латинский алфавиты, унифицировать расположение символов с клавиатурой пишущих машинок, удобно расположить управляющие и функциональные клавиши. Не все, конечно, получилось удачно. Так, ориентация на пишущую машинку привела к тому, что цифры и латинские буквы оказались на разных регистрах. Это вызывает многочисленные переключения регистров при наборе программ на алгоритмических языках.

Большое неудобство приходится испытывать при наборе формул, так как одни знаки арифметических операций (+, *) находятся на нижнем регистре, а другие (—, /) — на верхнем. Некоторая путаница происходит и при выборе нужного регистра, когда набираются сходные по написанию символы русского и латинского алфавита (Р, Н, Х, В, С). Не всегда созвучны между собой буквы русского и латинского алфавита, объединяемые одной клавишей (Ы—У, Ж—V, Я—Q, Ъ—X). Расположение функциональных клавиш ПФ1, . . . , ПФ5 не идентично на разных типах клавиатур.

Вообще говоря, с каждой наборной клавишей связаны не две, а четыре кодовые комбинации, заносимые в буферную память дисплея. Для каждого символа предусмотрены две кодовые комбинации, соответствующие строчному (малому) и прописному (заглавному) изображениям символа. Если горит красный индикатор СБ на клавиатуре дисплея, то в буферную память заносится код, соответствующий строчной букве, в противном случае — код заглавной буквы. Режим **СТРОЧНЫЕ БУКВЫ** можно установить временно, зажав клавишу СБ, или постоянно — с помощью клавиши ФСБ. Несмотря на то что коды строчных и заглавных букв в буферной памяти различны, изображения их на экране одинаковы, поэтому проконт-

ролировать набор текста с разновысокими символами практически невозможно. И лучше к режиму СБ без особой необходимости не прибегать, тем более что периферийные устройства ЕС ЭВМ (ввод с перфокарт, АЦПУ) ориентированы только на работу с заглавными буквами.

2.1.1. Управление курсором. Место на экране, в котором должен появиться набираемый символ, подсвечено небольшим прямоугольником, называемым *курсором*, или *маркером*. После набора очередного символа курсор автоматически перемещается на одну позицию вправо, а из последней (80-й) позиции строки — в первую позицию следующей строки. Достигнув конца экрана, маркер переходит в первую позицию первой строки. На этом же месте он находится и после включения дисплея.

На клавиатуре терминала расположено восемь управляющих клавиш, воздействующих тем или иным образом на перемещения курсора. Пять из них — клавиши 1, . . . , 5 — позволяют сдвинуть маркер в горизонтальном или вертикальном направлении на одну позицию. Направление перемещения маркера показано на клавишах в виде стрелок. Смысл дублирования клавиш, вызывающей возврат курсора на одну позицию назад, объясняется скорее всего попыткой приблизить клавиатуру дисплея к клавиатуре пишущей машинки.

Клавиша 6 перемещает курсор в начало следующей строки. Если курсор при этом находился в самой нижней строке, то он переводится в первую позицию первой строки. Сведения о работе клавиш 7 и 8, реализующих табуляторные сдвиги вправо или влево, приводятся в п. 2.2.1. При работе с неформатизованным экраном обе эти клавиши переводят маркер в его начальное положение — в верхний левый угол экрана.

Клавиши, управляющие перемещениями курсора, работают абсолютно одинаково в режимах НР (нижний регистр) и ВР (верхний регистр). Символы, находящиеся в буферной памяти и расположенные на пути движения курсора, не изменяются.

2.1.2. Клавиши с автоматическим повтором. Клавиши черного цвета отличаются от других наборных клавиш тем, что при продолжительном их нажатии соответствующая функция начинает повторяться с частотой примерно 10 раз в секунду. Поэтому такие клавиши называются автоклавишами. К их числу относятся семь клавиш (1, . . . , 7), управляющих перемещениями курсора, а также символы пробела и подчеркивания.

Несмотря на то что действие клавиши пробела похоже на сдвиг курсора вправо, однако не надо забывать и о существенной разнице: перемещая курсор с помощью клавиши пробела, мы заносим в каждую позицию буферной памяти и код символа пробела, стирая тем самым предыдущий символ.

2.1.3. Функциональная клавиатура. К функциональной клавиатуре относятся следующие клавиши:

— клавиши, выполняющие только автономные функции при любом положении регистра (СТРН ВВ, СТРН КП, ВСТВ, клавиша обратной табуляции);

— клавиши, вызывающие прерывание (сигнал ВНИМАНИЕ), обработка которого должна быть предусмотрена программным путем (ПФ1, . . . , ПФ12; ВВОД; ВЫЗОВ ТЕСТА);

— клавиши, выполняющие автономные функции в положении «верхний регистр» (ДУБЛ, МТКП, ВЧРК) и вызывающие прерывание в положении «нижний регистр» (ПД1, ПД2, ПД3);

— клавиша СТРН ЭКР, выполняющая автономную функцию стирания буферной памяти дисплея и посылающая в ЭВМ сигнал прерывания при любом положении регистра.

Информации, считываемой в ЭВМ с экрана, всегда предшествует управляющий символ, который носит название *идентификатор внимания* (ИВ). Числовое значение байта ИВ позволяет определить функциональную клавишу, нажатую пользователем в момент, предшествующий чтению. Информация об этом соответствии приведена в табл. 2.1.

Т а б л и ц а 2.1

Функциональная клавиша	Код ИВ	Функциональная клавиша	Код ИВ
ПФ1	F 1	ПФ10	7A
ПФ2	F 2	ПФ11	7B
ПФ3	F 3	ПФ12	7C
ПФ4	F 4	ВВОД	7D
ПФ5	F 5	ПД1	6C
ПФ6	F 6	ПД2	6E
ПФ7	F 7	ПД3	6B
ПФ8	F 8	СТРН ЭКР	6D
ПФ9	F 9	ВЫЗОВ ТЕСТА	F0

Информация с экрана может быть прочитана по инициативе ЭВМ и без ожидания сигнала ВНИМАНИЕ. В этом случае в качестве байта ИВ выдается код X'60'.

Автономные операции, выполняемые с помощью функциональных клавиш, не отвлекают ЭВМ и осуществляются аппаратно. Однако в большинстве случаев их действие связано с форматизованным экраном; более подробная информация будет приведена в разд. 2.2.

2.2. Форматизованный экран

То, что мы видим на экране дисплея, является отображением содержимого буферной памяти. Каждой позиции на экране соответствует одна 10-разрядная ячейка, в которой хранится либо ин-

формационный, либо управляющий символ. Информационным символам, за исключением пробела, соответствует какое-то графическое изображение на дисплее. Управляющие символы на экране отображаются пробелами. Их основное назначение состоит в выделении некоторых полей, обладающих определенными свойствами. *Поле* — это ячейки буферной памяти с последовательными адресами. Начинается поле с позиции, следующей за управляющим символом, который в руководствах принято называть *символом-атрибутом*. Концом поля считается символ, предшествующий новому атрибуту. Поле может быть расположено с любой позиции и даже переходить с конца экрана в его начало.

Символ-атрибут можно записать в буферную память дисплея только программным путем, выполнив соответствующую процедуру вывода на экран. Разряды управляющего символа содержат следующую информацию:

1	2	3	4	5	6	7	8	9	10
МД	0	t_3	t_4	t_5	t_6	0	0	K	1

Разряд МД (модификация данных) устанавливается в «1», если пользователь заменил в поле хотя бы один символ или переместил курсор правее символа-атрибута. Разряды t_3t_4 определяют следующие характеристики поля:

$$t_3t_4 = \begin{cases} 00 — \text{поле отображаемое, невыбираемое,} \\ 01 — \text{поле отображаемое, выбираемое,} \\ 10 — \text{поле повышенной яркости, выбираемое,} \\ 11 — \text{поле неотображаемое, невыбираемое.} \end{cases}$$

Запрет на отображение содержимого поля позволяет вводить в ЭВМ *секретную* информацию типа паролей, наименований системных наборов и т. п.; одним словом — тех данных, значение которых надо утаить от постороннего взгляда.

Повышенная яркость способствует выделению на экране информации, на которую пользователь должен обратить внимание.

Термин *выбираемое* или *невыбираемое* связан с использованием светового пера. Если перо подвести к позиции, расположенной вблизи от начала выбираемого поля, и нажать микровыключатель, то в ЭВМ поступит сигнал прерывания с кодом ИВ = 7Е. Если поле невыбираемое, то сигнал ВНИМАНИЕ в машину не поступит. Указанное свойство может быть использовано для организации работы в режиме *меню*.

Разряд t_5 устанавливается в «1», если на данном поле можно набирать только цифровую информацию (т. е. цифры от 0 до 9, за-

пятую, точку или знак минус). При $t_b = 0$ поле считается алфавитно-цифровым; в его позициях можно набирать любые информационные символы. Объявление поля цифровым позволяет аппаратным путем предупредить синтаксические ошибки оператора, либо нажимающего не ту клавишу, либо ошибочно не установившего ВР.

Разряд t_b принимает значение «1», если требуется защитить поле от вмешательства пользователя. В защищенном поле можно только отобразить информацию, выдаваемую из ЭВМ. Ее нельзя изменить путем ввода с клавиатуры.

Неформатизованный экран можно рассматривать как частный случай форматизации, при которой имеют дело с одним незащищенным алфавитно-цифровым полем, не определяемым с помощью светового пера. Все символы такого поля имеют нормальную яркость свечения.

Следует обратить внимание на два момента. Во-первых, использование атрибутов уменьшает информационную емкость строк. Во-вторых, любая запись на экран со стороны ЭВМ может «затереть» предыдущую разметку. Стирание экрана с помощью клавиши СТРН ЭКР также уничтожает все атрибуты, хранившиеся в буферной памяти.

2.2.1. Сдвиг курсора на границу поля. Если экран форматизован, т. е. на нем размечено хотя бы одно поле, то имеется возможность перемещать курсор в начало незащищенного поля путем нажатия одной клавиши. Клавиша обратной табуляции (8 на рис. 2.2) перемещает курсор влево так, что он оказывается в первой позиции текущего поля. Если при этом курсор находился в позиции, соответствующей символу-атрибуту, то он перемещается влево в начало предшествующего поля. Клавиша прямой табуляции (7 на рис. 2.2) осуществляет аналогичные перемещения вправо. Прямой сдвиг на последнем поле экрана может привести к переводу маркера в начало первого поля на экране. Обратный табуляционный сдвиг в начале экрана может переместить курсор в конец экрана. При табуляционных сдвигах курсор «проскакивает» защищенные поля. Защищенные поля пропускаются и при обычных горизонтальных сдвигах курсора на одну позицию.

При форматизованном экране несколько иначе работает и клавиша перевода курсора на новую строку. Оказавшись в следующей строке, курсор продолжает двигаться вправо до тех пор, пока не будет найдена первая позиция ближайшего незащищенного поля. В частности, таковой может оказаться и первая позиция следующей строки. Если под текущей строкой экрана окажутся только защищенные строки, то курсор возвратится в начальное положение и продолжит поиск ближайшего незащищенного поля в верхней части экрана. При полностью защищенном экране клавиша «новая строка» (6 на рис. 2.2) возвращает курсор в левый верхний угол.

2.2.2. Автономное стирание информации на экране. Как мы уже упоминали, клавиша СТРН ЭКР производит стирание всех ячеек буферной памяти. При этом в каждую ячейку заносится нулевой восьмиразрядный код — символ «пусто». Символ «пусто» так же, как и пробел, не имеет графического изображения. Поэтому, увидев пробел в той или иной позиции экрана, еще нельзя с уверенностью сказать, какому символу этот пробел соответствует: то ли фактическому пробелу (код X'20' во внутреннем коде дисплея), то ли символу «пусто» (код X'00'), то ли символу-атрибуту.

Стирание экрана уничтожает все атрибуты, находившиеся в буферной памяти, и делает тем самым дисплей неформатизованным. Дополнительным действием клавиши СТРН ЭКР является перевод курсора в его начальное состояние.

Клавиши СТРН ВВ и СТРН КП используются для стирания части форматизованного экрана. Первая из них стирает содержимое всех незащищенных полей и переводит курсор в первую позицию первого незащищенного поля. Поскольку незащищенные поля форматизованного экрана преимущественно предназначены для ввода информации с клавиатуры, то разумным применением клавиши СТРН ВВ можно считать очистку вводных участков для набора по «чистому» полю. Клавиша СТРН КП производит стирание буферной памяти от текущего положения курсора до конца поля. Однако курсор при этом никуда не перемещается. Чаще всего эту клавишу используют для перенабора ошибочно набранной информации. Сначала с помощью обратной табуляции курсор возвращают в начальную позицию поля, а затем нажатием клавиши СТРН КП осуществляют роспись нулевым кодом (символ «пусто») всех позиций поля.

В отличие от клавиши СТРН ЭКР стирание вводных полей (СТРН ВВ) и стирание до конца поля (СТРН КП) не вызывают сигнала прерывания. Клавишами СТРН ВВ и СТРН КП можно пользоваться и при работе с неформатизованным экраном. При этом нажатие клавиши СТРН ВВ приводит к стиранию всего экрана и переводу курсора в начальное положение, а нажатие клавиши СТРН КП очищает экран от позиции курсора до конца экрана.

2.2.3. Автономная коррекция информации на вводном поле. Аппаратные возможности клавиатуры позволяют произвести исправление информации на незащищенном поле путем вставки пропущенных символов или вычеркивания лишних.

Вставка символов производится следующим образом. Нажимается клавиша ВСТВ. При этом на панели дисплея загорается индикатор режима вставки 5 (рис. 2.1). Затем курсор подводится к символу, который должен быть расположен правее вставляемого знака. После этого набирается один или несколько недостающих символов. Во время набора очередного символа предыдущий текст от курсора и до конца поля сдвигается на одну позицию вправо, освобож-

дая место для нового символа. При вставке каждого нового символа вместе с текстом сдвигается и курсор, так что если нужно вставить целое слово, то набирать его надо в естественном порядке — с первой буквы. Для отмены режима вставки необходимо нажать клавишу СБРОС.;

Работая в режиме вставки вы можете столкнуться с двумя необычными ситуациями. Во-первых, сдвигается не весь текст, расположенный на поле правее маркера. Это означает, что на месте видимых на экране пробелов в буферной памяти находятся нулевые коды (символы «пусто»). Такие символы во время сдвига «выдавливаются» в отличие от пробелов, которые считаются «значащими» символами. Как только последний значащий символ одной порции текста придвинется вплотную к первому значащему символу следующей порции, они начнут сдвигаться вместе как сцепленные железнодорожные вагоны. Во-вторых, во время вставки может загореться индикатор запрещения ввода б (рис. 2.1). Это означает, что последний значащий символ сдвигаемого текста достиг самой правой позиции поля и дальше двигаться некуда. Для продолжения работы после такого запрета придется нажать клавишу СБРОС.

Вставка на неформатизованном экране приводит к сдвигу всего содержимого экрана, расположенного правее курсора. Однако нулевые коды при этом вытесняются так же, как описано выше.

Для вычеркивания одного или нескольких лишних символов курсор подводится к самому левому из удаляемых знаков и (в режиме «верхний регистр») нажимается клавиша ВЧРК. При каждом нажатии часть текста правее курсора сдвигается на одну позицию влево, а символ из помеченной позиции пропадает. В освободившуюся ячейку буферной памяти, соответствующую последней позиции поля, заносится нулевой код (символ «пусто»). Так происходит, если конец поля находится в той же строке, что и курсор. Если поле переходит на следующую строку, то в сдвиге влево участвует только остаток текущей строки. И именно ее позиции с конца будут заполняться нулевыми кодами.

При работе с неформатизованным экраном действие клавиши ВЧРК распространяется на символы, расположенные от помеченной позиции до конца строки.

Внесение изменений путем вычеркивания или вставки фиксируется в разряде МД: = 1 символа-атрибута.

2.2.4. Функции фотоселектора. Световое перо используется для того, чтобы ЭВМ получала информацию о координатах поля, выбранного пользователем на экране дисплея. Фотоселектор подносится вплотную к экрану в районе позиции, соответствующей началу «определяемого» поля. Ошибка в установке пера по горизонтали не должна превышать семи позиций. Поэтому на определяемое

поле желательно выдавать текст соответствующего меню и ориентировать перо на первый символ текста. В момент нажатия микровыключателя все определяемые поля на экране начинают светиться чуть ярче. Эту разницу в освещении фиксирует чувствительный элемент, и по количеству шагов, которые проходит луч развертки дисплея от первой позиции первой строки до встречи с фотоселектором, устанавливается местоположение кончика пера.

Далее с помощью электронных схем находится местоположение атрибута выбираемого поля и регистрируется ситуация «выбор состоялся». В ЭВМ при этом поступит сигнал ВНИМАНИЕ с соответствующим кодом ИВ и координатами выбранного поля. Для индикации этого факта на экране дисплея используется первый символ определяемого поля — так называемый *символ-определитель*. Его записывают из ЭВМ при разметке определяемых полей или вносят с клавиатуры. В качестве символа-определителя может использоваться либо знак «?», либо знак «>». В момент выбора определяемого поля символ-определитель автоматически заменяется на противоположный, т. е. вопросительный знак на знак «больше» или наоборот. Употребление в качестве определителя символа «пусто» (или пробела) приводит к аппаратно регистрируемой ошибке в момент выбора такого поля. Ситуация «поле выбрано» приводит и к изменению разряда МД в атрибуте поля на противоположное значение.

Если фотоселектор подносится к неопределяемому полю или используется на неформатизованном экране, то никаких прерываний в ЭВМ не поступает и никаких изменений с информацией на экране не происходит.

2.2.5. Дублирование информации. При наборе документов стандартной структуры бывает удобно формировать содержимое некоторых полей нажатием одной функциональной клавиши. Процедура копирования информации на соответствующее поле реализуется, конечно, программным путем. Однако для ее упрощения можно воспользоваться клавишей ДУБЛ. Нажатие этой клавиши в режиме «нижний регистр» вызывает появление на месте курсора символа «меньше» (<), а курсор перемещается вправо до первой позиции следующего незащищенного поля. Одновременно устанавливается в «1» разряд МД атрибута помеченного поля. Несмотря на то что на экране появился символ «<», внутренний код, занесенный в соответствующую ячейку буферной памяти, отличается от внутреннего кода настоящего знака «меньше». Прикладная программа, которая прочитает эту информацию, получит в соответствующей позиции код X'1C'. Для нее указанный код и должен послужить сигналом к заполнению поля требуемой информацией. Клавишей ДУБЛ можно пользоваться и для заполнения цифрового поля.

2.3. Метка конца поля

Клавиша МТКП (в режиме «верхний регистр») используется для занесения в текущую ячейку буферной памяти специального кода — КОНЕЦ ПОЛЯ. Его внутренний код совпадает с кодом аналогичного управляющего символа КП в коде ДКОИ (X'1E'). Однако на экране дисплея метка поля отображается знаком «больше» (\rangle). Вообще говоря, символ КП к форматизации экрана никакого отношения не имеет. И его не надо путать с признаком конца размеченного поля. Клавиша СТРН КП также никак не относится к этой метке.

Знак КП может быть считан из буферной памяти в ЭВМ и использоваться для запрограммированных процедур.

2.4. Клавиша ВЫЗОВ ТЕСТА

К числу функциональных клавиш, передающих в ЭВМ сигнал прерывания, относится клавиша ВЫЗОВ ТЕСТА. Такое название объясняется тем, что эта клавиша задействована в работе тестовой программы, проверяющей работу оборудования. Режим ее использования, вообще говоря, зависит от операционной системы.

2.5. Изменение информации в буферной памяти дисплея по командам из ЭВМ

Для того чтобы изменить информацию в буферной памяти дисплея, необходимо с помощью логических или физических средств управления вводом-выводом выполнить одну из трех канальных команд: запись (КОД=X'01'), стирание—запись (КОД=X'05') или стирание незащищенных полей (КОД=X'0F').

Первые две команды сопровождаются передачей из ЭВМ цепочки байтов $S_1 S_2 \dots S_k$. Первый байт в этой цепочке (S_1) называется символом управления записью (СУЗ). Для дисплея в этом байте имеют смысл только разряды с пятого по седьмой. При $t_5 = 1$ операция записи завершается звуковым сигналом, при $t_6 = 1$ после завершения операции отменяется блокировка клавиатуры и сбрасывается код ИВ, единица в седьмом разряде СУЗ приводит к сбросу разрядов модификации данных (МД=0) во всех атрибутах, хранящихся в буферной памяти.

Среди последующих байтов, передаваемых из ЭВМ по командам записи, могут присутствовать как управляющая информация, не заносимая в буферную память дисплея, так и запоминаемые символы. Управляющая информация всегда начинается с управляющего байта, вслед за которым может быть расположено до 2—3 байт с дополнительными сведениями.

Установка адреса буферной памяти. Эта процедура связана с необходимостью разместить выводимый текст, начиная с определенной позиции экрана. Вслед за управляющим символом УАБ (код X'11') следует двухбайтовый адрес ячейки буферной памяти, имеющий следующий формат:



Разряды 3—7 первого байта и 2—7 второго байта составляют 11-разрядный адрес ячейки буферной памяти, который должен принадлежать диапазону [0, 1919]. В противном случае будет зарегистрирована сбойная ситуация.

Счетчик адресов ячеек буферной памяти, установленный таким образом в начальное состояние, затем продвигается автоматически по мере записи символов из передаваемой цепочки. Его содержимое участвует и при обработке большинства управляющих комбинаций. Так, например, в позицию, определяемую текущим значением указанного счетчика, можно переместить курсор на экране. Выполняется эта процедура по управляющему символу УСТ КР (код X'13').

Запись символа-атрибута. Символу-атрибуту в цепочке данных предшествует управляющий символ «начало поля» (НП, код X'1D'). Записится атрибут в ячейку, определяемую текущим содержимым счетчика адреса буферной памяти.

Размножение заданного символа. Любой графически отображаемый символ можно повторить на заданном участке экрана, если включить в цепочку выдаваемых данных указание «повторить до адреса» (ПА, код X'3C'). Вслед за символом ПА должны следовать двухбайтовый конечный адрес буферной памяти и размножаемый символ. Если конечный адрес (КА) меньше текущего адреса, то буферная память расписывается заданным символом циклически — от текущего адреса до конца экрана и от начала экрана до КА. Если КА больше 1919, то запись в буфер не происходит и регистрируется сбойная ситуация.

Стирание незащищенных позиций экрана. Если среди управляющей информации присутствует символ «стирание незащищенного до адреса». (СНА, код X'12'), а вслед за ним указан двухбайтовый конечный адрес, то все незащищенные поля буферной памяти между текущим и конечным адресами заполняются символом «пусто».

Программная табуляция. Управляющий символ (ПТ, код X'05') устанавливает в счетчике ячеек буферной памяти новое значение, совпадающее с первой позицией ближайшего справа незащищенного поля. Это позволяет в одной цепочке данных разместить несколько порций, предназначенных для вывода на разные поля экрана.

2.5.1. Запись на экран. Команда записи на экран дисплея имеет следующий формат:

CCW X'01', АДРЕС, ПР, ДЛИНА

Здесь поле АДРЕС соответствует цепочке управляющих и информационных байтов, подготовленных с учетом описанных выше правил. Количество передаваемых данных, включая СУЗ и другую управляющую информацию, задается параметром ДЛИНА. Команда записи может быть одиночной или находиться в цепочке канальных команд, поэтому в ее состав включен признак ПР.

Начальный адрес в буферной памяти дисплея, с которого располагается переписываемая из ЭВМ информация, определяется следующим образом. Это — либо позиция, определяемая положением курсора, если команда записи была одиночной или находилась в цепочке команд после команды управления; либо вслед за СУЗ передается символ УАБ, устанавливающий требуемое значение счетчика адреса; либо, наконец, запись происходит с текущего адреса буфера, если команда находилась в цепочке команд канала вслед за командой чтения или записи.

Физически процедура вывода информации на экран производится в три этапа. На первом этапе устройство управления переписывает содержимое буферной памяти дисплея в свою оперативную память. Затем в памяти устройства управления производятся все операции, указанные в потоке данных, поступающих из ЭВМ. И наконец, последний этап заключается в переписи содержимого оперативной памяти устройства в буферную память дисплея.

2.5.2. Запись на экран со стиранием. Запись со стиранием отличается от предыдущей команды только тем, что предварительно чистится вся буферная память дисплея и устанавливаются в начальные значения адреса в счетчиках буфера и курсора. После этого выполняется описанная ранее процедура.

2.5.3. Стирание незащищенных полей. Формат соответствующей программы канала таков:

CCW X'0F', 0, ПР, 4

Действие команды эквивалентно нажатию клавиши СТРН ВВ, т. е. после ее выполнения все незащищенные поля на экране или весь экран, если он не форматизован, заполняются символами «пусто».

2.6. Считывание информации с экрана дисплея

Локальная дисплейная станция обеспечивает выполнение двух процедур чтения — чтение буферной памяти (КОП=02) и чтение модифицированных полей (КОП=06). Формат данных, принимаемых в ЭВМ, имеет различную структуру в зависимости от выпол-

няемой команды и функциональной клавиши, нажатой пользователем. Общими являются лишь первые 3 байта, в составе которых передаются идентификатор внимания (ИВ) и двухбайтовый адрес курсора. Байт ИВ определяет, какая из функциональных клавиш была нажата перед операцией чтения (см. табл. 2.1). Структура адреса курсора совпадает с 11-разрядным адресом буферной памяти, описанным в разд. 2.3.

2.6.1. Чтение буферной памяти. Если команда чтения является первой командой обмена в программе канала или ей предшествуют только команды управления («холостой ход», «уточнить состояние», «выбрать»), то считывание начинается с самого начала буфера. Если содержимое первых строк экрана вас не интересует, то их все равно придется прочитать, хотя бы и фиктивно. Например,

CCW X'02',0,PP+10,3*80+5

По этой команде данные из трех первых строк читаются фиктивно, с тем чтобы последующая в цепочке команда чтения могла передать в ЭВМ информацию, начиная с шестого символа четвертой строки

Разметка (форматизация) экрана сказывается только на том, что в информацию, передаваемую в ЭВМ, вставляются дополнительные символы. Каждому прочитанному символу-атрибуту предшествует управляющий байт НП.

Если команде чтения в программе канала предшествует любая команда чтения или записи, то информация с экрана читается, начиная с текущего адреса буфера. Каждая команда чтения, выполняющаяся в канальной цепочке, перед массивом данных выдает упоминавшиеся ранее 3 байта (код ИВ и адрес курсора).

2.6.2. Считывание модифицированных полей (команда СЧТ М). Основное отличие этой команды от предыдущей состоит в том, что читаются не все поля экрана, а только те, у которых разряд модификации данных в байте-атрибуте установлен в «1». Две другие, менее существенные особенности команды СЧТ М заключаются в том, что в ЭВМ не передаются символы-атрибуты и пустые символы. Структура информации, прочитанной с экрана, имеет вид

1 байт		2 байт			
ИВ		Адрес курсора		УАБ	Адрес поля ₁
					Содержимое модифицированного поля ₁
				УАБ	Адрес поля ₂
					Содержимое модифицированного поля ₂

.....

В качестве адреса поля в этом потоке фигурирует адрес первой значащей позиции, т.е. адрес соответствующего атрибута плюс 1. Если на форматизованном экране в момент выполнения команды СЧТМ не было ни одного модифицированного поля, то в ЭВМ передается только заголовок — байт ИВ и адрес курсора. И еще одна особенность команды — использование фотоселектора. В этом случае поток считываемых данных состоит из заголовка и цепочки адресов полей, подвергнувшихся к этому моменту модификации. Содержимое модифицированных полей в ЭВМ не передается.

Команда СЧТМ может прочитать и содержимое неформатизованного экрана. В этом случае в потоке данных, поступающих в ЭВМ, вслед за первыми тремя байтами заголовка находятся все значащие символы, прочитанные начиная с нулевой ячейки буферной памяти.

В зависимости от кода функциональной клавиши, нажатой непосредственно перед командой СЧТМ, существует еще два варианта исполнения команды. Нажатие клавиш ПД1, ПД2, ПД3 и СТРНЭКР приводит к передаче в ЭВМ одного-единственного байта ИВ. Это так называемая разновидность команды — «считывание короткое». Вторая модификация команды СЧТМ связана с нажатием клавиши ВЫЗОВ ТЕСТА и носит название «считывание запроса на тест». При ее исполнении в ЭВМ передается всего 4 байта, содержащие управляющий символ «начало заголовка» (НЗ, код X'01'), знаки %, / и управляющий символ «начало текста» (НТ, код X'02'). Используется этот режим в макрокомандах телекоммуникационного доступа для тестирования терминалов.

2.7. Команды управления

К командам управления дисплеями относятся процедура выбора, процедура уточнения состояния устройства и команда «холостой ход».

2.7.1. Повышение эффективности работы канала. Команда «выбрать» (код X'0B') используется с целью оптимизации загрузки оборудования, подключенного к тому же каналу, где работает дисплейная станция. Работая в монопольном режиме в мультимплексном канале или в блок-мультимплексном подканале селекторного канала, устройство управления ЕС-7922 захватывает канал на время выполнения операции обмена. Однако мы уже упоминали о том, что большинство процедур обмена имеет многоступенчатый характер. Сначала устройство управления переписывает в свою память содержимое буферной памяти обслуживаемого в данный момент дисплея. Затем происходит обмен между устройством управления и каналом. И на завершающей стадии оперативная память устройства управления переписывается в буферную память дисплея.

Смысл команды «выбрать» заключается в том, что устройство управления начинает первую фазу выполнения обмена (перепись в свою память) и сразу же освобождает канал для работы других устройств. Если в программе канала вслед за командой «выбрать» находятся в цепочке команды чтения или записи, то при их выполнении первая фаза уже не повторяется. Одним словом, команда «выбрать» реализует аппаратное совмещение операций подобно тому, как мы подводим головки к нужному участку магнитного диска или подгоняем удаленный блок на магнитной ленте.

2.7.2. Анализ завершения процедуры обмена. Общие сведения по анализу результатов произведенного обмена на физическом уровне можно найти в работе [2]. Дополнительная информация для определения характера сбоя при обмене может быть получена из байта уточненного состояния, который запрашивается с помощью следующей команды:

CCW X'04', BUC, PR, 1

Дисплейная станция выдает единственный байт уточненного состояния (BUC), разряды которого имеют стандартный смысл. Единица в нулевом разряде BUC означает, что в программе канала была выдана команда с неверным кодом операции. В частности, устройство управления отвергает команду с кодом X'00' — «проверить ввод-вывод», если такая команда включена в цепочку программы канала. Проверка такого рода может быть проведена только в результате выполнения команды TIO, запрещенной для использования в прикладных программах.

Значение $t_1 = 1$ свидетельствует о том, что обращение проведено к недоступному или неготовому терминалу. Единица во втором разряде означает, что устройство управления обнаружило ошибку по четности в команде или данных, поступивших из канала. Третий и четвертый разряды BUC устанавливаются в «1», если ошибка по четности обнаружена в схемах обмена между устройством управления и терминалом или в оперативной памяти периферийных устройств. Разряд t_5 свидетельствует о том, что ошибка в передаче данных обнаружена схемами контроля дисплея.

Ситуация, когда терминал за установленное время не справился с заданием от устройства управления, регистрируется в шестом разряде BUC. И наконец, последний разряд BUC свидетельствует об ошибках в командах установки адреса буферной памяти, повторения или стирания до адреса и др.

К числу такого рода ошибок относятся неверное указание адреса ($A > 1919$) и отсутствие дополнительной информации после управляющих символов УАБ, ПА, СНА или НП.

2.7.3. Команда «холостой ход». Как и любое другое периферийное устройство ЕС ЭВМ, дисплей ЕС-7927 «умеют» выполнять пу-

стую команду — «холостой ход». Она не производит почти никаких действий и исполняется немедленно, так что канал занимается на очень небольшой промежуток времени. Применяется команда «холостой ход» для того, чтобы сбросить ситуацию «висячие данные», которая может возникнуть при использовании фотоселектора. Другим возможным вариантом употребления команды «холостой ход» является организация «пустых» циклов в канальной программе с целью придержать терминал до наступления какого-то события.

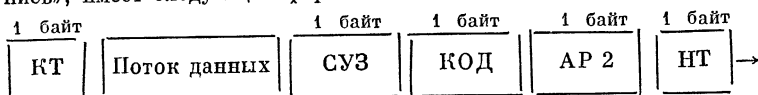
2.8. Основные отличия между локальными и удаленными станциями

Удаленные дисплейные станции типа ЕС-7920 выпускаются в двух модификациях: многотерминальные (с устройством управления ЕС-7921) и одностерминальные (с функциями устройства управления, совмещенными с аппаратурой дисплея). Многотерминальные станции комплектуются такими же дисплеями и печатающими устройствами, как и локальные станции, и выполняют эти терминалы точно такие же операции — запись, запись со стиранием, стирание незащищенных полей, чтение буфера, чтение модифицированных полей и т. п. Однако технология общения между ЭВМ и терминалом совершенно другая.

К каналу ЭВМ подключается не устройство управления, а мультиплексор передачи данных (обычно типа ЕС-8403; его же иногда называют МПД-3). Один мультиплексор может обеспечить работу четырех многотерминальных станций. Соединяется МПД с устройством управления через коммутируемый или выделенный телефонный канал связи. В первом случае канал имеет на обоих концах два телефонных аппарата, и для установления между ними связи надо произвести набор номера абонента. Во втором случае номер набирать не приходится — достаточно просто снять трубку. Связь между мультиплексором и устройством управления устанавливается и поддерживается аппаратными средствами по командам, поступающим из ЭВМ.

Если устройство управления локальной дисплейной станции получало из ЭВМ соответствующее указание по командным шинам канала, то программа канала для обмена с удаленной станцией обращается непосредственно к мультиплексору. Команды мультиплексора передачи данных (МПД) не совпадают с командами локальной дисплейной станции, так как ему приходится выполнять такие операции, как включение и отключение удаленного устройства управления, набор телефонных номеров, анализ результатов приема информации абонентом и т. п. Поэтому команды, адресуемые непосредственно дисплею, включаются в поток данных, которые передаются мультиплексором в канал связи. Структура данных, например

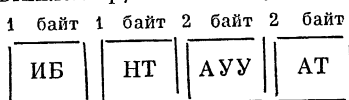
передаваемая удаленному устройству управления по команде «запись», имеет следующий формат:



Управляющие байты НТ («начало текста») и КТ («конец текста») всегда окаймляют передаваемое сообщение. Байт АР2 («авторегистр 2») сообщает устройству управления, что следом за ним идет код операции канальной команды. Остальная информация — СУЗ и поток данных — имеет ту же структуру, что и для локальных дисплеев.

Для того чтобы не загружать линию связи, по которой передается информация всем обслуживаемым дисплеям, данные разбиваются на короткие порции — блоки. МПД при работе с удаленными дисплейными станциями позволяет передавать блоки длиной 16, 32, 64, 128 или 256 байт. Размер блока устанавливается на мультиплексоре с помощью переключателя. При передаче блоками каждый блок завершается управляющим символом КБ («конец блока»), а последний блок — символом КТ.

Информация, которая поступает от удаленного устройства управления в мультиплексор, имеет шапку из 6 байт:



Здесь ИБ — *индексный байт*, значение которого определяет порядковый номер терминала в абонентском списке у мультиплексора, НТ — *начало текста*, АУУ — *физический адрес устройства управления*, АТ — *физический адрес терминала*, посылающего сообщения. Вслед за шапкой располагается поток данных с индикатором внимания, адресом курсора, адресами и содержимым прочитанных полей. Внутренняя структура этих данных точно такая же, как и в случае локальных дисплеев. Разница заключается лишь в том, что весь поток может быть разбит на блоки установленной длины.

Скорость передачи данных по телефонным каналам невысока. МПД-3 работает на основной скорости 1200 бод и допускает удвоенную скорость 2400 бод, что соответствует 150 и 300 байт в секунду. Это означает, что время полного заполнения экрана данными из ЭВМ составляет примерно 10—15 с. Поэтому на удаленной дисплейной станции невозможно организовать диалоговую систему с приемлемым (1,5—2 с) временем реакции. Для таких дисплеев особое значение приобретает форматизация экрана, позволяющая передавать выборочные поля с небольшим объемом данных.

3.1. Что такое БТМД

Базисный телекоммуникационный метод доступа (БТМД) представляет собой компоненту операционных систем ДОС и ОС ЕС, в которую, начиная с версий ДОС-2.2 и ОС-6.1, включены средства обслуживания дисплейных станций ЕС-7920. В состав программного обеспечения БТМД, относящегося к дисплейным станциям локального типа, входят 14 макроопределений и довольно большое количество служебных программ, расположенных в системных библиотеках (в ДОС ЕС — библиотека исходных модулей ассемблера и библиотека абсолютных модулей, в ОС ЕС — библиотеки MACLIB и SVCLIB, библиотека загрузочных модулей TELCMLIB). Почти все макрокоманды БТМД в ДОС ЕС и ОС ЕС, связанные с обслуживанием локальных комплексов, имеют одинаковый формат. Поэтому программирование процедур обмена в среде БТМД от операционной системы практически не зависит. На те отклонения, которые имеют место в описательных макрокомандах и операторах управления заданием, обязательно следует обратить внимание пользователя.

С точки зрения прикладного программиста, создающего индивидуальные средства диалогового общения со своей задачей, БТМД представляет собой совокупность небольшого количества процедур низшего уровня. С помощью макрокоманд БТМД можно описывать структуру областей и управляющих блоков, необходимых при обмене, производить двусторонний обмен с одним или несколькими терминалами, изменять конфигурацию терминального оборудования во время выполнения прикладной программы. Однако анализ ситуаций, возникающих из-за неправильного обращения к внешнему устройству или из-за сбоя оборудования, пользователь должен делать сам. А для этого приходится заниматься проверкой байтов состояния и уточненного состояния, анализировать отдельные поля управляющих блоков БТМД и т. д.

В составе БТМД имеется возможность организовать тестирование дисплейного оборудования по инициативе пользователя, вести машинный журнал сбойных ситуаций и работать с дисплейными станциями типа ЕС-7920. Однако на этих средствах БТМД останавливаться здесь не желательно, чтобы не утомлять читателя

техническими деталями. Для более подробного знакомства с возможностями БТМД могут быть рекомендованы работа [15] и техническая документация по соответствующим версиям операционных систем.

3.2. Описание и инициализация терминала

С точки зрения операционной системы любое внешнее устройство эквивалентно файлу данных (терминология ДОС ЕС) или набору данных (терминология ОС ЕС). Поэтому описание терминала должно включать сведения по организации данных, характеристики, необходимые для процедур обмена, и информацию о привязке к конкретному физическому устройству. По такому описанию операционная система формирует необходимую табличную информацию как на своих полях, так и на поле прикладной программы, настраивает служебные программы БТМД и присоединяет их к программе пользователя.

Следующий шаг, предшествующий сеансу обмена, сводится к открытию набора (файла) данных. Мы будем называть этот шаг *инициализацией терминала*. Он заключается в том, что операционная система заполняет определенные поля в сформированных таблицах. Попытка произвести обмен без инициализации дисплея приведет к аварийному завершению задания.

Основные отличия между ДОС ЕС и ОС ЕС заключаются в описании терминала. Пусть, например, мы собираемся написать программу, общающуюся с одним дисплеем ЕС-7927 локального типа. В ДОС ЕС это будет выглядеть следующим образом. Если в программе терминал будет выступать под логическим устройством, например, с номером SYS012, то в состав управляющих карт придется включить директиву назначения соответствующего физического устройства. Например,

```
//, ASSGN SYS012, X'0C1'
```

Описание этого терминала в программе осуществляется с помощью макрокоманды DTFBT:

```
DCB7920 DTFBT CU=7920, DEVICE=7927, LINELST=(012)
```

Функции описания терминала в ДОС ЕС продолжает еще одна макрокоманда — BTMOD, описывающая характеристики обслуживаемых программ. В самом минимальном объеме она может выглядеть так:

```
BTMOD L7927=YES
```

В ОС ЕС привязка терминала к физическому дисплею задается в директиве DD:

```
// SYS012 DD UNIT=0C1
```

Описание дисплея сосредоточено в стандартной макрокоманде DCB, имеющей в минимальном объеме следующий формат:

```
DCB7920 DCB DSORG=CX, DDNAME=SYS012,  
MACRF=(R,W)
```

На необходимость обслуживания дисплея средствами БТМД указывают приставка ВТ (макрокоманды DTFBT и BTMOD) в ДОС ЕС и способ организации набора данных (DSORG=CX) в ОС ЕС.

Может оказаться, что одна задача должна общаться с несколькими терминалами. Пусть их логические обозначения имеют вид SYS006, SYS012, SYS008. Тогда с помощью макрокоманд DTFBT в ДОС ЕС или DCB в ОС ЕС придется описать группу дисплеев. Это может выглядеть следующим образом:

```
// ASSGN SYS006,X'0C1'  
// ASSGN SYS008,X'0C3'  
// ASSGN SYS012,X'0D1'
```

```
.....  
DCB7920 DTFBT CU=7920,DEVICE=7927,  
LINELST=(006,012,008)
```

или

```
//DD7920 DD UNIT=0C1  
// DD UNIT=0D1  
// DD UNIT=0C3  
.....
```

```
DCB7920 DCB DSORG=CX,MACRF=(R,W),  
DDNAME=DD7920
```

Единственное отличие между этими описаниями в ДОС ЕС и ОС ЕС состоит в том, что дисплеям в группе присваиваются порядковые номера, начиная с 0 в ДОС ЕС и с 1 в ОС ЕС. Таким образом, дисплей с физическим номером X'0D1' в программе ДОС будет выступать под номером 1, а в ОС — под номером 2. Эти номера в дальнейшем заменяют адреса дисплея в макрокомандах обмена.

В приведенных выше примерах использовался минимальный набор обязательных параметров, допускаемых средствами БТМД. Остальные параметры, набор которых в ДОС ЕС и ОС ЕС немного не совпадает, управляют режимами буферизации, способами исправления ошибок, техникой тестирования устройств и т. п. В первом приближении для работы с локальными дисплеями (особенно в задачах, общающихся с единственным терминалом) всеми остальными параметрами макрокоманд BTMOD, DTFBT и DCB можно пренебречь.

Инициализация терминалов в ДОС ЕС и ОС ЕС формально выглядит очень похоже. В обоих случаях используется та или иная модификация макрокоманды OPEN со ссылкой на метку управляющего блока, сконструированного с помощью DTFBT или DCB.

Например,

	OPEN	DCB7920	}	для ДОС ЕС
	OPENR	DCB7920		
метка	OPEN	(DCB7920)	}	для ОС ЕС
	OPEN	(DCB7920),MF=L		
	OPEN	(DCB7920),MF=(E,метка)		

Модификация OPENR или OPEN с MF=L используются при создании повторновходимых (реентерабельных) программ.

Действия макрокоманд OPEN в ДОС ЕС и ОС ЕС различаются. Управляющие таблицы, формируемые на месте макрокоманд DTBVT и DCB, тоже различны. В ОС ЕС функции открытия связаны с извлечением из системных библиотек соответствующих программ обмена. В ДОС ЕС программа обмена является результатом трансляции модуля BTMOD. Ее можно сформировать автономно от прикладной программы, а подключить только на стадии редактирования. А можно протранслировать и вместе с прикладной программой, если параметры библиотечного модуля IJLBTM нас не устраивают. Среди других функций макрокоманды OPEN в любой операционной системе отметим контроль за принадлежностью назначенного физического устройства к сфере обслуживания БТМД.

3.3. Формат макрокоманд обмена с локальным дисплеем

Макрокоманды обмена в БТМД включают восемь модификаций двух основных процедур — считывания данных с экрана (READ) и вывода данных на экран (WRITE). Сразу следует оговориться, что в отличие от соответствующих процедур алгоритмических языков высокого уровня обмен в БТМД выполняется без каких-либо преобразований информации. Вводимые и выводимые сообщения представляют собой цепочку символов в коде ДКОИ. Каждый байт такой цепочки — это либо отображаемый на экране знак алфавита ДКОИ, либо символ, управляющий работой дисплейного оборудования.

Формат макрокоманд READ и WRITE в ДОС и ОС одинаков, за исключением нумерации дисплеев ($n = 0, 1, 2, \dots$ в ДОС ЕС; $n = 1, 2, 3, \dots$ в ОС ЕС).

Основная стандартная форма макрокоманд обмена, с которой должен работать начинающий пользователь, должна содержать семь позиционных аргументов:

{	READ	}	$P_1, P_2, P_3, P_4, P_5, P_6, P_7$
{	WRITE	}	

Первый аргумент P_1 задает имя расширенного блока управления событиями (DECB), который создается внутри тела макрокоманды. Блок DECB занимает 48 байт и используется как средство

для обмена информацией между компонентами БТМД. Отдельные поля DECB содержат исходную информацию для обмена (адреса областей в оперативной памяти и на дисплее, длину передаваемых сообщений, тип операции обмена и т.п.). В другие поля заносятся результирующие данные (характер завершения операции, байты состояния устройств, код ошибки и т.п.). После окончания операции приходится анализировать содержимое некоторых полей DECB. Более подробные сведения о расположении нужных областей этого блока будут приведены в разд. 3.4.

Второй аргумент p_2 определяет модификацию команды READ или WRITE. Допустимые значения этого параметра для локальных дисплеев приведены в табл. 3.1.

Т а б л и ц а 3.1

Код обмена (значение p_2)	Пояснение
READ-TI	Чтение содержимого экрана с начальной позиции после нажатия функциональной клавиши
READ-TM	Чтение содержимого всех модифицированных полей экрана
READ-TB	Чтение содержимого буферной памяти дисплея с начальной позиции
READ-TMP	Чтение содержимого модифицированных полей, начиная с заданного адреса буферной памяти дисплея
READ-TBP	Чтение содержимого экрана, начиная с заданного адреса буферной памяти дисплея
WRITE-TI	Вывод на экран с начальной позиции буферной памяти
WRITE-TS	Вывод на экран с начальной позиции с предварительной очисткой буферной памяти
WRITE-TUS	Стирание незащищенных полей

Третий аргумент p_3 — метка блока управления данными (DCB), который был описан с помощью макрокоманды DTFBT или DCB.

Четвертый и пятый параметры определяют начало (p_4) и длину (p_5) в байтах области оперативной памяти, с которой происходит обмен. При их назначении не нужно забывать о некоторых деталях, на которые мы уже обращали внимание в разд. 2.3 и 2.4. Дело в том, что кроме текста, отображенного на экран или считываемого с экрана, здесь должны присутствовать управляющие символы и адреса буферной памяти дисплея. Так, например, при считывании информации с неформатизованного экрана по макрокоманде READ-TI мы получаем три дополнительных байта, предшествующие основному тексту: это — байт ИВ, определяющий номер нажатой функциональной клавиши, и двухбайтовый адрес, соответствующий позиции марке-

ра на экране. Поэтому для считывания самой верхней строки экрана может быть использована следующая группа команд и констант:

READ DECB1,TI,ECB7920,IW,83,,1

.....
IW DS 3X поле для управляющих байтов
СТРОКА1 DS CL80 поле для информации

Если экран форматизован, то по команде READ-TI считывается содержимое модифицированных полей. Это означает, что помимо первых трех управляющих байтов мы будем получать по дополнительной тройке байтов (УАБ — 1] байт, адрес начала поля — 2 байта) перед содержимым каждого модифицированного поля.

Шестой аргумент r_6 задает метку четырехбайтового поля с информацией, которая передается на дисплей в цепочке команд канала для предварительной установки счетчика адреса буферной памяти. Байт с адресом r_6 содержит символ управления записью (СУЗ), который не изменяет состояние дисплея (код X'40'). В байте $r_6 + 1$ находится код команды УАБ (X'41'), а оставшиеся два байта поля r_6 задают номер требуемой позиции на экране дисплея в формате, описанном в разд. 2.3. Параметр r_6 имеет смысл только для макрокоманд READ-TBP и READ-TMP.

Наконец, последний параметр r_7 определяет номер терминала, с которым производится обмен. Если у задачи терминал один, то $r_7 = 0$ в ДОС ЕС и $r_7 = 1$ в ОС ЕС.

Наряду с описанным форматом макрокоманд READ и WRITE в БТМД существуют еще две формы этих же процедур — описательная и исполнительная. Признаком этих форм является наличие восьмого параметра в списке аргументов, имеющего ключевой способ задания $MF = L$ или $MF = E$. Первый вариант соответствует описательной форме, второй — исполнительной. Описательная форма в процессе трансляции преобразуется в расширенный управляющий блок DECB, некоторые поля которого заполняются соответствующими параметрами макрокоманды. Однако никаких операций по обмену описательная макрокоманда не производит. Фактически она порождает набор формируемых констант.

Исполнительных макрокоманд в программе может быть несколько. Они могут ссылаться на один и тот же уже описанный блок DECB или на разные блоки, если было использовано несколько описательных форм. На месте исполнительных макрокоманд блок DECB уже не порождается, но перед выполнением процедуры обмена дополнительные значения параметров замещают содержимое соответствующих полей в описанном ранее блоке DECB. Исполнительный формат процедуры обмена может отличаться от описательного любой комбинацией своих параметров, в том числе и кодом операции обмена. Более того, исполнительная макрокоманда

да READ может ссылаться на блок DECB описательной процедуры WRITE.

При задании параметров макроманд READ или WRITE можно пользоваться явными и неявными способами. Так, поле DECB может быть задано либо меткой блока, создаваемого в теле стандартной или описательной формы процедуры, либо меткой блока, расположенного в теле другой макрокоманды, либо адресом такого блока, задаваемого содержимым общего регистра. Например,

READ	DECB1,...,MF=L	} блок DECB1 формируется в теле этой макрокоманды
.....	
READ	DECB1,...,MF=E	} явная ссылка на существующий блок DECB1
WRITE	DECB1,...,MF=E	
.....	
LA	5,DECB1	} косвенная ссылка на существующий блок DECB1
READ	(5) ,..., MF=E	

Другие параметры — такие, как R_3 , R_4 , R_6 , — могут задаваться либо меткой соответствующего поля оперативной памяти, либо адресом этого поля, предварительно загруженным в любой регистр общего назначения из диапазона номеров от 2 до 12. Длина сообщения (параметр R_6) или номер терминала (параметр R_7) могут быть заданы либо в виде явного десятичного числа, либо косвенным образом, если соответствующее число занесено в один из общих регистров.

Что дает использование описательной и исполнительной форм макрокоманд обмена? Во-первых, это дает экономию объема памяти за счет предотвращения дублирования последовательно эксплуатируемых блоков DECB. Во-вторых, вынос блока DECB в поле констант упрощает построение реентерабельных программ. Наконец, исполнительная форма удобна еще и тем, что она не требует обязательного перечисления всех восьми параметров. В минимальной конфигурации процедура обмена может содержать только три параметра:

WRITE DECB7920,TI,MF = E

Значение всех остальных параметров процедуры обмена предполагаются уже сформированными в блоке DECB7920, полное описание которого было выполнено в другой макрокоманде.

3.4. Анализ результатов выполнения операции

Чтобы убедиться в том, что процедура обмена составлена правильно и принята к исполнению, необходимо после выдачи макрокоманды READ или WRITE проанализировать содержимое 15-го регистра. Именно в этот регистр помещают результаты своего рас-

следования обслуживающие программы БТМД. Этот результат принято называть *кодом возврата*, и располагается он в младшем байте 15-го регистра. Нулевой код возврата свидетельствует о том, что заявка на обмен принята к исполнению. Причин отказа может быть несколько, и основные из них приведены в табл. 3.2.

Таблица 3.2

Код возврата	Причина отказа	Что делать
X'04'	Дисплей не успел закончить предыдущую операцию обмена	Повторить процедуру
X'08'	Номер устройства в макрокоманде не входит в список описанных терминалов	Исправить программу
X'0C'	Ошибка в задании параметра p_3 — шифра процедуры обмена	» »
X'14'	Во время выполнения процедуры инициализации (OPEN) дисплей был выключен	Выдать макрокоманду LOPEN и попытаться включить дисплей с помощью оператора ЭВМ
X'24'	Байт $p_6 + 1$ в макрокоманде READ-TMP или READ-TVP не есть X'11' (код УАВ)	Исправить программу
X'28'	Содержимое управляющих полей блока DCB испорчено из-за возможной ошибки в прикладной программе	» »

3.4.1. Специфика в организации начала сеанса под управлением ДОС ЕС и ОС ЕС. Неприятности могут начаться с момента инициализации терминала. При выполнении макрокоманды OPEN имеют место следующие непредвиденные ситуации:

- отсутствует назначение устройства, связанного с открываемым блоком DCB;
- назначение выполнено правильно, но устройство управления или дисплей не приведены в состояние готовности;
- назначение не соответствует устройству, обслуживаемому средствами БТМД.

В первой ситуации ДОС ЕС снимает задание, а ОС ЕС этого не делает, но устанавливает в «1» один из разрядов блока DCB. Проверка последнего может быть реализована следующим образом:

```

OPEN  (DCB7920)
TM     DCB7920+48,X'10'  проба разряда на аварийное завершение, если назначение отсутствует
BNO    ABEND

```

Неготовность устройства в любой операционной системе во время выполнения оператора OPEN приводит к выдаче на операторскую консоль соответствующего сообщения. Реакция на это у оператора ЭВМ стандартная, как и на готовность любого другого устройства. Он должен перевести устройство в состояние готовности и ответить CONT. По такому ответу система попытается повторить процедуру опроса готовности. Если будут иметь место повторные сообщения о неготовности, то оператор вправе снять задание и прибегнуть к услугам системного программиста или дежурного инженера для выяснения ситуации. Однако предложенная схема поведения оператора не всегда может оказаться разумной. Например, если локальный дисплей отнесен от машинного зала на значительное расстояние (до 1200 м), то на сообщение о неготовности дисплея оператор ответит POST, в результате операционная система завершит открытие устройства фиктивно, но отметит эту ситуацию в одном из управляющих блоков. Первая же заявка на обмен с таким терминалом будет отвергнута с кодом возврата X'14'.

Выйти из создавшегося положения можно, например, следующим образом. Прикладная программа делает повторную попытку открыть терминал с помощью макрокоманды LOPEN. Воспользоваться макрокомандой OPEN для этой цели уже нельзя, так как операционная система снимет задание. Если повторное открытие не проходит, то программа может вступить в связь с оператором для принятия последующего решения. Например, если дисплей расположен в непосредственной близости от ЭВМ, то оператору можно передать такое сообщение: ВКЛЮЧИТЕ ДИСПЛЕЙ ОС2. ЕСЛИ ВСЕ В ПОРЯДКЕ, ОТВЕЬТЕ G, В ПРОТИВНОМ СЛУЧАЕ — E.

Если дисплей расположен на значительном расстоянии от оператора, то сообщение может звучать так: ПОЗВОНИТЕ ПО ТЕЛЕФОНУ 5-40, И Я СКАЖУ, ЧТО ДЕЛАТЬ ДАЛЬШЕ. В любом случае прикладная программа ждет заранее предусмотренный отрицательный или положительный ответ и в зависимости от этого либо аварийно завершает свое выполнение, либо пытается продвинуться дальше. Приведем фрагмент программы проверки дисплея в начале сеанса (т. е. одну из возможных реализаций начала сеанса):

```
BEGIN OPEN DCB7920
*** ПОПЫТКА ПРОВЕРИТЬ ДИСПЛЕЙ ПУТЕМ СТИРАНИЯ
*** ЭКРАНА МОЖЕТ ОБНАРУЖИТЬ, ЧТО ТЕРМИНАЛ
*** НЕ ВКЛЮЧЕН
RESTART WRITE DECB7920,TUS,DCB7920,*,1,NTERM
*** ДЛЯ ОС ЕС NTERM=1, ДЛЯ ДОС ЕС NTERM=0
      LTR      15,15
      BZ       START
      CH       15,=H'20'
```

```

BNE      ABEND
LOPEN    DECB7920
LTR      15,15
BZ       START
*** ОБМЕН СООБЩЕНИЯМИ С ОПЕРАТОРОМ (ДЛЯ ОС ЕС)
WTOR     'ВКЛЮЧИТЕ ОС2. ЕСЛИ ПОЛУЧИТСЯ,
        ОТВЕТ=G, ЕСЛИ НЕТ, ОТВЕТ=E',
        ОТВЕТ,1,DCBWTOR
WAIT     ECB=DCBWTOR
CLI      ОТВЕТ,C'E'
BNE      RESTART
ABEND    ABEND 1986
START    WAIT  ECB=DCB7920

```

Здесь после процедуры OPEN не делается проверка блока DCB, связанная с анализом назначений в ОС ЕС. Дело в том, что следующая за ней попытка проверки дисплея с помощью макрокоманды стирания экрана сразу же приведет к аварийному снятию задания, если назначения для терминала не было. Поэтому логика приведенного фрагмента одинаково хороша и для ДОС ЕС, и для ОС ЕС с той лишь разницей, что общение с оператором в одном случае организуется по макрокоманде WTOR (ОС ЕС), а в другом — по макрокомандам OUTLOG и INLOG (ДОС ЕС).

В ОС ЕС существует и более удобное средство для передачи управления в заданную точку прикладной программы в момент перевода неготового терминала в состояние готовности. Обеспечивается такая возможность с помощью необязательного параметра READYQ в описании терминала. Например,

```

DCB7920  DCB    DSORG=CX,MACRF=(R,W),
          DDNAME=DD7920,READYQ=START
. . . . .
BEGIN    OPEN  (DCB7920)
          READ  DECB7920,TL,DCB7920,СТРОКА,3,1
START    . . . . .

```

И в том и в другом случае диалоговая программа оккупирует ресурсы ЭВМ, а ожидание готовности может продлиться достаточно долго. Поэтому лучше сначала включить терминал, а уже потом запускать программу на исполнение.

Ситуация с неправильным назначением в любой операционной системе приводит к снятию задания.

3.4.2. Нормальное и аварийное завершение обмена. Предположим, что мы выдали макрокоманду READ или WRITE и убедились в том, что наша заявка на обмен принята к исполнению. В этой ситуации прикладная программа должна выдать макрокоманду

WAIT:

WAIT ECB = P_i

Ссылка на первый параметр макрокоманды обмена означает, что мы ждем записи так называемого кода завершения в первый байт расширенного блока DECB.

БТМД предусматривает выдачу четырех различных кодов завершения. Однако если мы работаем с одиночным дисплеем, не пользуемся средствами автоматического тестирования оборудования и не отменяем ранее выданную заявку на READ-TI, то разбираться приходится всего с двумя кодами завершения и соответствующими им тремя ситуациями.

Код завершения X'7F' означает, что процедура обмена выполнена успешно. Сбой оборудования или отключение дисплея в процессе диалога приводит к появлению в байте DECB кода X'41'. При сбое можно попытаться повторить неудавшуюся операцию, организовав программный счетчик сбойных ситуаций. Когда его значение превысит установленный порог, задачу придется снять. Такого же рода контроль может быть реализован и средствами БТМД. В случае неготовности терминала, возникшей в процессе работы, надо выполнить те же операции, что и после процедуры OPEN. Для определения ситуации, имевшей место при коде завершения X'41', можно проанализировать разряды байта уточненного состояния. Этот байт после выполнения операции заносится по адресу DECB + 16. Смысл всех разрядов был описан в п. 2.7.2.

3.5. Специфика ввода по инициативе пользователя

Ввод по инициативе пользователя связан с так называемым сигналом ВНИМАНИЕ, который вырабатывается аппаратурой дисплея по нажатию функциональной кнопки, вызывающей прерывание работы ЭВМ. В этот момент средства БТМД регистрируют появление сигнала ВНИМАНИЕ в одном из своих управляющих блоков. Но если со стороны прикладной программы запроса на чтение не поступало, то БТМД ограничивается только констатацией факта появления сигнала прерывания и подсчетом общего количества обработанных сигналов такого рода. Почему может накопиться несколько сигналов ВНИМАНИЕ? Дело в том, что нажатие функциональной клавиши блокирует клавиатуру дисплея. Однако эту блокировку пользователь может отменить клавишей СБРОС, и таким образом возникает потенциальная возможность многократной выдачи сигнала прерывания со стороны дисплея.

Если команда READ-TI выдана до появления сигнала ВНИМАНИЕ, то обслуживающие программы БТМД формируют заявку на обмен, но выполнение программы канала откладывается до разрешения со стороны пользователя. После выполнения отложенной

заявки на чтение БТМД сбрасывает регистрационный байт и счетчик сигналов ВНИМАНИЕ, т. е. приходит в состояние готовности к приему следующей команды READ-TI.

Однако если между двумя последовательно выполняющимися макрокомандами READ-TI пользователь успел несколько раз нажать функциональную клавишу, то повторное чтение выполнится уже без ожидания и в ЭВМ поступит необновленная информация. Это обстоятельство должен учитывать проектировщик диалоговой системы, в которой можно запрограммировать соответствующие защитные действия (фиксацию фазы диалога на экране, быстрое подтверждение принятого сообщения и т.п.). Но и партнер по диалогу должен проявлять достаточную выдержку и очень аккуратно пользоваться клавишей СБРОС.

Неконтролируемое нажатие функциональных клавиш вызывает и еще одну неприятную ситуацию, связанную с анализом байта ИВ. Своевременное нажатие функциональной клавиши позволяет аппаратуре дисплея нормально запомнить цифровой код, связанный с этой клавишей (см. табл. 2.1). Повторное нажатие другой функциональной клавиши до выполнения процедуры READ-TI вызовет логическое наложение кода новой клавиши на предыдущий код ИВ. А если с командой чтения связана последующая обработка, определяемая кодом ИВ, то разобраться с возникшей ситуацией будет невозможно. Придется прибегнуть к повторному вводу с предварительной выдачей соответствующего предупреждения.

Средства БТМД располагают возможностью пресечь неконтролируемые прерывания с помощью макрокоманды RESETPL. Эта макрокоманда выполняет две функции. Если в момент ее выдачи не было начато исполнение заявки на чтение по ранее выданной команде READ-TI, то отложенная заявка уничтожается, т. е. отменяется приказ на ранее выданное чтение. Признаком отмены является код завершения X'48', занесенный программами БТМД в начало блока DECB после выполнения цепочки READ-TI — WAIT. Помимо уничтожения заявки на обмен по команде RESETPL сбрасываются регистрационный байт и счетчик сигналов ВНИМАНИЕ. Таким образом можно обезопасить себя перед выдачей новой команды READ-TI. Схематично эту процедуру можно разбить на три следующих шага. На первом шаге отменяется блокировка клавиатуры, сбрасываются разряды ИВ и издается звуковой сигнал для привлечения внимания пользователя.

Выполнить это можно, выдав, например, на дисплей по команде WRITE-TI один-единственный байт — СУЗ:

```
WRITE DECB7920,TI,DCB7920,SUZ,1,,1
WAIT  DCB=DECB7920
```

```
.....
SUZ  DC      X'07' СУЗ для установок дисплея
```

С целью упрощения здесь исключены анализ кодов возврата и байта завершения обмена на первом шаге. Затем выдается команда RESETPL:

RESETPL DECB7920, ATTENT

Второй аргумент здесь указывает на необходимость отмены ранее выданной команды READ-TI. Но поскольку мы все равно дождемся ее окончания, то эта функция имеет смысл только для уменьшения объема генерируемого макрорасширения. Наконец, на третьем шаге выдается застрахованная команда READ-TI, после которой остается либо ждать ее завершения, либо отменить отсроченную заявку на обмен с помощью макрокоманды RESETPL.

После выдачи команды READ-TI в БТМД нельзя запустить какую-либо другую операцию обмена — READ или WRITE, ссылающуюся на тот же самый управляющий блок DCB. Это означает, что можно описать в одной группе несколько дисплеев, но обмен с ними придется вести последовательно. Более того, команда READ-TI, выданная для одного из дисплеев группы, может прочитать информацию с другого дисплея этой же группы. Это зависит от того, на каком терминале будет раньше нажата функциональная клавиша. БТМД устроен так, что заявка на READ-TI адресуется не к конкретному дисплею, а к группе целиком. Прикладная программа может узнать, с какого именно терминала в ЭВМ поступили данные. Номер этого дисплея заносится в байт DECB + 36.

3.6. Дополнительные возможности по многотерминальному обслуживанию

Создание диалоговой системы, общающейся одновременно с несколькими терминалами, представляет собой довольно сложную задачу. При ее реализации возникает много вопросов. Здесь указаны некоторые из них. Как организовать раздельное хранение и средства защиты индивидуальной информации и придумать такой порядок обслуживания пользователей, который не вызывает длительных ожиданий? Как описывать терминалы — в виде одной группы (одна макрокоманда DTFBT или DCB на все терминалы) или каждый терминал должен быть связан со своим управляющим блоком и соответственно иметь индивидуальное описание? Как быть, если число обслуживаемых пользователей может варьироваться? Что делать, если тот или иной терминал вышел из строя уже после запуска системы? Если его успели отремонтировать во время сеанса, то как его снова подключить к системе?

Некоторые из этих проблем связаны с возможностями операционных систем; они обсуждаются при разборе соответствующих систем коллективного пользования в гл. 7—9. Здесь же следует

обратить внимание читателей на специальные средства БТМД, которые могут оказаться полезными при многотерминальном обслуживании. В первую очередь интерес представляет макрокоманда CNGNTRY, которая позволяет динамически отключаться от уже назначенного дисплея или снова принимать его в сферу обслуживания. Формат этой макрокоманды таков:

CNGNTRY DCB7920, ATTTLST, r7,, SKIP

или

CNGNTRY DCB7920, ATTTLST, r7,, ACTIVATE

В первом случае терминал с номером r7, привязанный к управляющему блоку с меткой DCB7920, временно отключается от прикладной программы, а во втором случае (параметр ACTIVATE) этот терминал вновь становится активным. Макрокомандой CNGNTRY можно пользоваться не только в случае неисправности терминала, но и для маскировки сигналов прерывания от дисплея, на который система временно не желает обращать внимания. В частности, с ее помощью можно реализовать защиту от многократных нажатий функциональных клавиш между двумя командами READ-TI.

Другой полезной макрокомандой БТМД является команда TWAIT, представляющая собой более удобную модификацию обычной процедуры ожидания событий (WAIT). Макрокоманда WAIT позволяет зафиксировать появление заданного числа событий:

WAIT N, ECBLIST = адрес списка ECB

Так можно определить момент завершения всех событий, связанных с указанными блоками ECB. Однако при многотерминальном обслуживании более важно определить момент завершения одного из событий. Макрокоманда TWAIT имеет следующий формат:

TWAIT (r), ECBLIST = адрес списка ECB

Список адресов блоков ECB должен содержать нулевые байты в каждом адресе ECB, за исключением последнего, в котором старший байт должен быть равен X'80'. Макрокоманда TWAIT удобна тем, что она сокращает до минимума процедуру выяснения, какое именно событие наступило первым. В регистре r находится адрес блока DECB (или ECB, если это событие не связано с макрокомандами READ/WRITE), а в 15-м регистре хранится смещение этого адреса в списке адресов ECBLIST.

ГЛАВА 4

ОСОБЕННОСТИ ПРОГРАММИРОВАНИЯ ПОД УПРАВЛЕНИЕМ ОТМД

4.1. Что такое ОТМД

Общий телекоммуникационный метод доступа (ОТМД) представляет собой компоненту операционных систем типа ОС-4, ОС-6, предназначенную для организации обмена между удаленными абонентскими пунктами (АП) и прикладными программами. В состав программного обеспечения ОТМД входят более 80 макроопределений, расположенных в системных библиотеках MACLIB, SVCLIB, TELCMLIB, LINKLIB и несколько сервисных программ.

С помощью макроопределений ОТМД составляются служебные программы, управляющие процессами обработки и передачи сообщений по каналам связи; реализуются процедуры обмена информацией между абонентами телекоммуникационной сети; выполняются директивы оператора, связанные с обслуживанием удаленного терминального оборудования.

ОТМД с точки зрения прикладной программы представляет собой служебную программу-посредник, через который можно отправлять сообщения на один или несколько терминалов и получить оттуда ответные сообщения. Программа эта носит название *программа управления сообщениями (ПУС)*. ПУС не есть нечто фиксированное, придающееся к операционной системе. Эту программу должен написать квалифицированный системный программист применительно к местным условиям, т. е. с учетом имеющегося терминального оборудования и специфики его использования. В рамках одной вычислительной установки могут эксплуатироваться несколько различных ПУС.

Существует два варианта работы с ПУС. В первом случае ПУС запускается в самостоятельном разделе с достаточно высоким приоритетом и может обслуживать одновременно несколько заданий, выполняющихся в других разделах. Это, так сказать, *ПУС коллективного пользования*. Во втором варианте ПУС запускается как подзадача, выполняющаяся в одном разделе с прикладной программой и обслуживающая только породившую ее задачу. Такую ПУС будем называть *индивидуальной*. Схемы взаимодействия между абонентами телекоммуникационной сети в обоих вариантах использования ПУС представлены соответственно на рис. 4.1 и 4.2.

В каком-то смысле работа ПУС напоминает действия почтового отделения, в котором есть приемный пункт — ящик для отсылаемой корреспонденции — и окно выдачи, работающее в режиме «до востребования». ПУС «умеет» делить длинные сообщения на порции, чтобы предотвратить монополизацию канала связи одним абонентом. Она может установить связь с абонентом в заданное время,

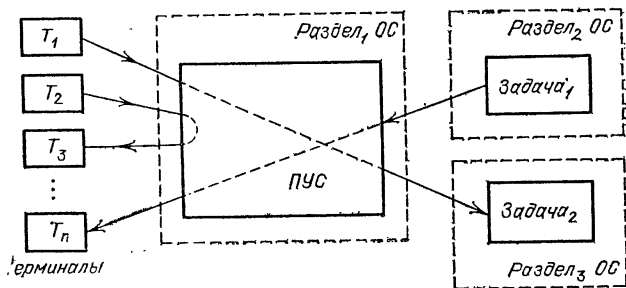


Рис. 4.1. Схема взаимодействия прикладных задач с ПУС коллективного пользования

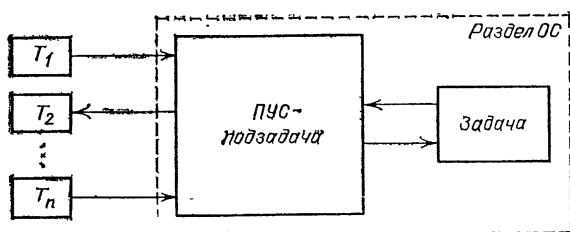


Рис. 4.2. Схема взаимодействия прикладной задачи с индивидуальной ПУС

проконтролировать правильность передачи сообщений, переправить корреспонденцию по запасному каналу, если неисправен основной, и т. п.

Все эти возможности программируются на ассемблере с использованием макрокоманд ОТМД. Общее количество последних превышает 80, а количество параметров, управляющих режимами обмена, достигает 300. Если к этому добавить, что только руководства программиста по средствам ОТМД составляют семь томов общим объемом около 1300 страниц, то нетрудно понять, почему большинство пользователей не испытывают особого желания взять ОТМД на свое вооружение.

Несмотря на то что ОТМД ориентирован главным образом на обслуживание разнообразных удаленных АП, имеется возможность

организовать через ОТМД и работу с локальными дисплейными станциями ЕС-7920. Правда, введена такая возможность только в состав операционных систем типа ОС-6. Для локальных терминалов используется далеко не весь аппарат ОТМД. Однако даже на тех примерах, которые приведены ниже, можно разобраться в идеологии работы с основными компонентами ОТМД.

4.2. Поток данных в системе ОТМД

Основные контрольные пункты, через которые проходит поток данных между терминалом (Т) и прикладной программой (ПП), отражены на рис. 4.3.

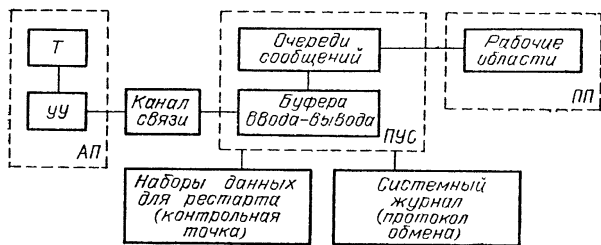


Рис. 4.3. Схема потоков данных в системе ОТМД

Сообщения, циркулирующие по каналам связи под управлением ОТМД, состоят, как правило, из двух компонент — заголовка и текстовой части сообщения. Структура заголовка зависит от его описания и может варьироваться по желанию пользователя. В состав заголовка входят некоторые обязательные параметры (шифр передающей станции или ЭВМ, шифр станции назначения, приоритет сообщения) и параметры, которые ОТМД может сформировать по дополнительному заказу, — порядковый номер сообщения, дата и время передачи. Расположение полей с указанной информацией и их длины задаются в соответствующих описаниях ПУС. Часть информации, входящей в заголовок, носит служебный характер и до прикладной программы не доходит. Формированием и обработкой этих данных занимается ПУС.

Заголовку предшествует управляющий символ НЗ («начало заголовка»). Вслед за заголовком располагается текст, который содержит управляющие и информационные данные, зависящие от направления обмена и типа используемого терминала. Формированием и обработкой текста занимается прикладная программа. Завершается сообщение управляющим символом КП («конец передачи»).

В зависимости от режима работы канала связи ПУС может передавать сообщения либо целиком, либо порциями. В последнем случае каждая внутренняя порция сообщения завершается управляющим символом КБ («конец блока»). В некоторых ситуациях ОТМД может передавать только заголовок или только текст.

Сообщения, поступающие из каналов связи, размещаются на входном буферном поле, а сообщения, выдаваемые из ЭВМ, — на выходном буферном поле. В зависимости от длины сообщения и размера буферов одно сообщение может либо занимать один буфер, либо располагаться на нескольких буферах. В буферных полях ввода-вывода к тексту сообщения добавляется служебная информация, облегчающая поиск частей сообщения и их последующую обработку.

Помимо буферного поля ввода-вывода ПУС располагает еще одним хранилищем, в котором находятся сообщения, подлежащие системной обработке, — так называемые очереди сообщений. Если прикладная программа ведет двусторонний обмен с одним терминалом, то в ПУС необходимо описать, как минимум, две очереди — одну для вводимых с терминала сообщений, другую для сообщений, выдаваемых на терминал. Каждой очереди присваиваются имя и приоритет, задаются размеры буферов, на которых должны размещаться ждущие сообщения, указываются правила их обработки. Очереди от разных задач объединяются ПУС в одну общую область, которая может либо целиком помещаться в оперативной памяти, либо частично располагаться на магнитном диске.

Обработка сообщений, находящихся в очереди, сводится к заполнению или проверке содержимого полей заголовка, к перекодировке текстовой части сообщения из кодов ДКОИ в коды, принятые в соответствующем канале связи, или к обратной перекодировке, к разбиению сообщений на требуемые порции, к вставке или удалению управляющих символов и т. п.

При необходимости ПУС может протоколировать в системном журнале сведения о всех сообщениях, поступающих с терминалов или выдаваемых прикладными программами. Периодическая запись состояния ПУС и содержимого всех его полей в набор данных, зарезервированный на магнитном диске или магнитной ленте, позволяет при сбоях организовать повтор сеансов обмена, начиная с последней контрольной точки.

4.3. Структура программы управления сообщениями

По своей форме ПУС напоминает программу на коболе. Начальный раздел ПУС составляет всего четыре макрокоманды — INTRO, OPEN, READY и CLOSE. Первая из них приводит в состояние готовности средства ОТМД, настраивает их на исполь-

зубое оборудование, устанавливает количество и размеры различных буферов, параметры очередей и т.п. Макрокоманда OPEN открывает наборы данных, используемые в ПУС,— очереди сообщений на магнитных дисках, области для контрольных точек и системного журнала, наборы данных для групп каналов связи. Инициализация ПУС завершается макрокомандой READY, по которой в случае успешного старта на терминалы выдается приветственное сообщение. Текст этого сообщения заготавливается во внешней программе, включаемой в качестве секции в необязательный раздел ПУС. Выход на макрокоманду READY происходит и при каждом рестарте ОТМД. По этому поводу на терминалы выдается сообщение, текст которого также заготавливается во внешней программе. Макрокоманда CLOSE используется для закрытия (всех наборов данных, если старт ПУС по тем или иным причинам не состоялся.

Два следующих раздела ПУС имеют прямую аналогию с разделами оборудования и данных в коболовской программе. Разделу оборудования соответствует секция управления терминалами, которая представляет собой описание абонентской таблицы. Начинается это описание с макрокоманды TTABLE. В ней же, подобно фортрановскому оператору цикла, задается и метка последней макрокоманды описаний. Основное содержание абонентской таблицы представлено строками двух типов. Первую группу составляют элементы описания терминального оборудования. Формируются такие строки с помощью макрокоманды TERMINAL, включающей от 5 до 20 операндов. Каждому терминалу, к которому может адресоваться прикладная программа, должна соответствовать своя макрокоманда TERMINAL. Ее метка играет роль шифра абонента, который фигурирует в заголовке сообщения в качестве адресата-отправителя или адресата-получателя. Обязательный операнд TERM задает тип терминала (для локальных дисплеев типа EC-7920 значение этого параметра равно 792L). Два других обязательных параметра — DCB и RLN — в конечном итоге определяют физический номер терминала, связанного с данной макрокомандой. Первый из них ссылается на метку управляющего блока DCB, который в свою очередь с помощью операнда DDNAME выводит нас на оператор DD в задании на запуск ПУС. Параметр RLN указывает на относительный номер в группе терминалов, описанных найденной строкой DD. Два последних обязательных операнда в команде TERMINAL — QBY и QUEUES — определяют характер постановки выходных сообщений в очереди и место расположения всех очередей сообщений. Остальные параметры задают режимы работы канала связи (время связи с абонентским пунктом, временные интервалы между сеансами связи), размеры буферов и блоков, допустимые приоритеты сообщений для данного АП и т. п.

Вторую группу строк абонентской таблицы составляют указания о программах обработки сообщений. Формируются они с помощью макрокоманд TPROCESS. Метки этих макрокоманд выполняют роль имен очередей сообщений, к которым обращаются прикладные программы. Единственный обязательный параметр макрокоманды TPROCESS ссылается на метку блока DCB, в описании которого содержится ссылка на программу обработки сообщений (MH).

Кроме строк указанных типов в состав абонентской таблицы могут быть включены сведения о структуре очередей сообщений, заносимых в системный журнал (макрокоманды LOGTYPE), информация о списках приглашений (макрокоманды INVLIST), сведения об абонентах, работающих в каскадном или распределительном режимах (макрокоманды TLIST). В последнем случае несколько абонентов (терминалов или прикладных программ) объединяются в группу, которой присваивается имя. Если пунктом назначения является это групповое имя, то в распределительном режиме сообщение последовательно передается каждому абоненту группы, а в каскадном помещается в самую короткую очередь.

В разделе терминального оборудования можно встретить группу макрокоманд OPTION, с помощью которых для конкретного терминала можно зарезервировать несколько служебных полей и затем в макрокомандах TERMINAL, TPROCESS или TLIST задать начальные значения этих полей. Содержимое этих полей можно использовать в процессе обработки сообщений.

Третий раздел ПУС в основном состоит из макрокоманд DCB с описанием всех используемых наборов данных. Первую обязательную группу наборов данных для любого ПУС составляет информация, поступающая с терминалов или адресованная им. Для того чтобы сократить объем описаний, однотипное терминальное оборудование объединяется в так называемые группы каналов связи. В пределах одной группы терминалы должны использовать одинаковый способ передачи данных, их сообщения обрабатываются одной и той же программой, каждому каналу в группе выделяется одинаковое количество буферов для приема или выдачи сообщений. Любой терминал может быть причислен только к одной группе каналов связи.

Макрокоманда DCB, описывающая набор данных для группы каналов связи, имеет от 6 до 16 параметров. К числу обязательных относятся: способ организации данных (DSORG = TX); метод доступа к сообщениям из программы обработки (MACRF = (G,P)); ссылка на физические адреса группы устройств (DDNAME = метка оператора DD); имя программы обработки сообщений (MH = адрес входа); имя таблицы управляющих сигналов (для локальных комплексов EC-7920 параметр SCT должен принимать значение DKOI); имена списков приглашений (INVLIST = (имя 1,,, имя

2,,, ...)). Остальные параметры макрокоманды DCB задают: число входных (BUFIN = ...) и выходных (BUFOUT = ...) буферов; максимальное количество буферов (BUFMAX = ...), используемых каждым каналом одновременно; размер буфера (BUFSIZE = ...); количество дополнительных байтов в буфере для хранения служебной информации (дату и время передачи; порядковый номер сообщения заносится на буфер, если в макрокоманде DCB предусмотрено задание параметра RESERVE); соотношение между приоритетами принимаемых и передаваемых сообщений (CPRI = ...); способ перекодировки текста сообщения (TRANS = DKOI), адрес списка дополнительных программ обработки блока DCB (EXLST = метка); режим использования программноуправляемого прерывания (PCI = ...); частоту опроса терминалов (INTVL = ...).

Вторую обязательную группу наборов данных составляют очереди сообщений. Они могут размещаться либо в оперативной памяти, либо на магнитном диске, либо комбинированным способом — в оперативной памяти с перезаписью на диск. Первый вариант обеспечивает максимальное быстродействие, однако обладает и некоторыми отрицательными характеристиками. Так, например, переданные в канал связи сообщения из очереди удаляются и восстановить ошибочное сообщение уже невозможно. Далее, в такой очереди нельзя придержать передачу сообщения на временно бездействующий терминал. Наконец, организация очереди в оперативной памяти отбирает этот ресурс у прикладных программ.

Для организации очереди сообщений на магнитном диске соответствующие наборы данных должны быть предварительно созданы с помощью сервисной программы IEDQXA. Использование диска уменьшает оперативность общения с терминалами, однако имеет и свои плюсы. Так, например, получив информацию о временной неработоспособности терминального оборудования, обработчик может задержать передачу адресованных ему посланий и послать их позже. Прикладная программа получает возможность восстановить переданное ранее сообщение и повторно передать его в случае утери. Наиболее эффективным является размещение части очереди в оперативной памяти с перезаписью сообщений на магнитный диск. Однако выбор рационального соотношения объемов оперативной и внешней памяти, определение режимов перезаписи и периодической реорганизации очереди сообщений представляют собой довольно сложную задачу для системного программиста.

Наборы данных для очередей сообщений, расположенных полностью или частично в дисковой памяти, описываются с помощью макрокоманды DCB требующей задания четырех обязательных и двух вспомогательных параметров. Аргумент DSORG = TX определяет способ организации набора, аргумент MACRF = (G,P) — способ доступа к сообщениям из программ OTMD, а аргументы

DDNAME и OPTCD — соответственно расположение набора и технологию его использования. Отсутствие оператора DCB означает, что очередь сообщений расположена только в оперативной памяти.

Два дополнительных набора данных ПУС предназначены для хранения контрольной точки, используемой при рестарте средств ОТМД, и системного журнала, в котором регистрируется протокол обмена. Оба этих набора не являются обязательными, и мы рассматривать их не будем.

В третьем разделе ПУС описывается еще один управляющий блок, через который осуществляется связь с прикладной программой. Это — блок PCB, формируемый с помощью одноименной макрокоманды. Для каждой прикладной программы должен быть описан один блок PCB, на который ссылаются все макрокоманды TPROCESS, обслуживающие данную прикладную программу. Макрокоманда PCB использует два обязательных параметра — MH и BUFSIZE.

Первый из них ссылается на имя программы — обработчика сообщений. Второй определяет размер буферов, через которые ПУС обменивается сообщениями с прикладной программой. Основные необязательные параметры определяют количество буферов, используемых в процессе выполнения макрокоманд PUT/WRITE (BUFIN = ...) или GET/READ (BUFOUT = ...), а также число дополнительных байтов в буфере для размещения даты, времени и порядкового номера сообщения (RESERVE = ...).

Четвертый раздел ПУС — аналог коболовского раздела процедур — носит название «обработчик сообщений» и обозначается в руководствах через MH (от *Message Handler*). Вообще говоря, в этот раздел могут входить несколько разных обработчиков сообщений, если ПУС ориентирован на обслуживание неоднородного терминального оборудования. Любой обработчик сообщений начинается с макрокоманды STARTMH, метка которой идентифицирует имя обработчика. Именно на него ссылаются макрокоманды TPROCESS, DCB и PCB.

В каждом обработчике имеется две группы процедур. Первая группа связана с обработкой сообщений, поступающих в ЭВМ с терминального оборудования. Ее называют входной группой; заканчивается она макрокомандой INEND. Вторая группа — выходная — связана с обработкой сообщений, выдаваемых из ЭВМ; ее завершает макрокоманда OUTEND. В пределах каждой группы процедуры обработки делятся на три подгруппы, связанные соответственно с обработкой заголовков, текстов сообщений и управлением буферами. Указанные подгруппы начинаются с макрокоманд INHDR или OUTHDR, INMSG или OUTMSG, INBUF или OUTBUF.

Для типовых процедур обработки в ОТМД предусмотрено довольно много макрокоманд. Например, макрокоманда DATATIME заносит дату и время в зарезервированные ранее байты заголовка

сообщения. Макрокоманда **SEQUENCE** вставляет порядковый номер в выдаваемое сообщение или проверяет, не пропущено ли очередное сообщение, поступившее с терминала. С помощью макрокоманды **CODE** можно произвести прямую или обратную перекодировку текста сообщения (ДКОИ \rightleftharpoons кодировка канала связи). Обработчик может направить сообщение в ту или иную очередь (**FORWARD**, **REDIRECT**) или задержать его отсылку (**HOLD**), проверить адресат-отправитель (**ORIGIN**), изменить приоритет (**PRIORITY**), записать сообщения в системный журнал (**LOG**) и т. д. К процедурам, связанным с управлением буферами, относятся макрокоманды подсчета числа сегментов в сообщении (**COUNTER**), редактирование текста (**MSGEDIT**), ограничение приема слишком длинных сообщений (**CUTOFF**) и др.

Помимо типовых процедур обработки заголовков управления буферами, обеспечиваемых макрокомандами **ОТМД**, в программу **МН** можно включить и любые фрагменты на ассемблере, выполняющие нестандартные операции над содержимым зарезервированных полей. В этих вставках допускается использование любых макрокоманд **ОТМД**. Кроме того, из **МН** по оператору **CALL** можно обратиться к внешней подпрограмме. Однако в таких подпрограммах макрокомандами **ОТМД** уже пользоваться нельзя.

В обработке сообщений могут быть использованы условные переходы в зависимости от содержания некоторых байтов заголовка сообщения. Для этой цели применяют либо машинные команды перехода по условию, либо макрокоманды **PATH** и **MSGTYPE**. Многие функциональные макрокоманды **МН** тоже допускают условный режим выполнения, если соответствующее условие закодировано в необязательном операнде «контрольные символы». Все это позволяет в рамках одного **МН** проводить обработку сообщений различного типа.

В пятый, необязательный раздел **ПУС** включаются все внешние подпрограммы, оформленные в качестве секций с одним или несколькими входами. К этим программам обращаются различные макрокоманды **ОТМД** в связи с выдачей тех или иных сообщений, из-за аварийного завершения процедур открытия наборов данных, по случаю нестандартного способа обработки сообщений.

Все разделы **ПУС**, кроме начального, могут формироваться в любом порядке.

4.4. Функциональные возможности прикладной программы в среде ОТМД

Обмен сообщениями между **ПУС** и прикладной программой производится через буферную область, которая описывается в прикладной программе с помощью макрокоманды **DCB**. Вообще говоря

таких областей две. Одна из них называется входной, в ней размещаются целиком или порциями сообщения, адресованные прикладной программе и уже прошедшие обработку в МН. Прикладная программа выбирает очередное сообщение из входного набора данных по макрокомандам GET или READ. Вторую буферную область образует выходной набор данных, в который прикладная программа передает по макрокомандам PUT или WRITE сообщения, адресованные терминалам или другим прикладным программам.

Кроме входного и выходного набора данных прикладная программа может иметь так называемую рабочую область. Сообщения из входного набора переписываются средствами ОТМД в рабочую область, или содержимое рабочей области переносится в выходной набор, если задача пользуется макрокомандами GET или PUT в режиме переноса (MACRF = GM или MACRF = PM). В отличие от этого возможна еще и работа в режиме ссылки, когда задаче передают не текст сообщения, а его адрес во входном (MACRF = GL) или выходном (MACRF = PL) наборе данных. В этом случае оперативную память для рабочей области ПУС сама запрашивает у операционной системы.

В обоих вариантах структуры поступившего и передаваемого сообщений одинаковы. Если буферная область состоит из записей фиксированного (RECFM = F) или неопределенного (RECFM = U) формата, то тексту сообщения предшествуют девять байт со следующей информацией. Первый байт содержит указание о том, разместилось ли все сообщение на буфере или же в нем находится первая/промежуточная/последняя порция сообщения. Следующие восемь байт отводятся для идентификатора абонента, которому адресовано или от которого получено сообщение; это — метка макрокоманды TERMINAL, определяющая в конечном итоге физический адрес соответствующего внешнего устройства. Если записи буферной области имеют переменный формат или заблокированы, то указанным девяти байтам предшествует четырех или восьмибайтовый префикс, используемый соответственно методами доступа QSAM (GET—PUT) или BSAM (READ—WRITE). Структура собственно текста сообщения, поступившего с терминала или адресованного терминалу, уже рассматривалась в гл. 2 достаточно подробно.

В задании на запуск прикладной программы, общающейся с активным терминалом, должно быть два управляющих оператора DD со ссылкой на имя соответствующей очереди сообщений в ПУС, т. е. на метки макрокоманд TPROCESS:

```
INDD DD QNAME=метка входной очереди [,DCB=...]  
OUTDD DD QNAME=метка выходной очереди [,DCB=...]
```

В тексте прикладной программы с помощью макрокоманд DSB описываются управляющие блоки, связанные соответственно с

входным и выходным наборами:

INDCB DCB DDNAME=INDD,DSORG=PS,...
OUTDCB DCB DDNAME=OUTDD,DSORG=PS,...

Набор параметров у входного и выходного блоков DCB примерно одинаков. Обязательными аргументами являются DDNAME (ссылка на метку DD), DSORG (способ организации набора) и MACRF (способ доступа к набору). Для входного набора MACRF может принимать одно из шести значений — GM, GMT, GL, GLT, R, RP. Первая буква определяет тип используемых макрокоманд для выборки сообщения: G — GET, R — READ. Буквы M и L соответствуют режиму пересылки сообщения (M — *Move*) из буферной области в рабочую или режиму ссылки на адрес сообщения (L — *Link*). Буквы T и P означают, что соответствующие макрокоманды ввода могут использоваться в сочетании с макрокомандой POINT. В выходном наборе параметр MACRF может принимать одно из трех значений — PM (PUT — *Move*), PL (PUT — *Link*) или W (WRITE).

Следующая группа параметров специфицирует размеры рабочей области (BLKSIZE = ...), длину блока в рабочей области (LRECL = ...), емкость буферов для обмена с ПУС (BUFL = ...) и формат записей (RECFM = ...). Параметр OPTCD, задаваемый в виде комбинации, содержащей от одного до трех символов из букв W, U и C, управляет заполнением служебных байтов рабочей области. Все эти параметры в макрокоманде DCB могут быть опущены, если их значения указаны с помощью параметра DCB в операторе DD. Кроме них в управляющем блоке DCB могут быть заданы адреса вспомогательных подпрограмм, работающих в экстренных случаях, — сбой при обмене (SYNAD), обнаружение признака конца файла при чтении (EODAD) и т. п.

Перед началом сеанса прикладная программа должна открыть входной и выходной наборы с помощью макрокоманды OPEN:

OPEN (INDCB, (INPUT), OUTDCB, (OUTPUT))

Обмен между прикладной программой и буферной областью с ожиданием конца обмена выполняется с помощью макрокоманд GET (прием сообщения) и PUT (выдача сообщения). Обе команды имеют одинаковый формат и в режиме пересылки содержат два операнда — адрес управляющего блока и адрес рабочей области. Например,

GET INDCB,СТРОКА

После выполнения этой программы прикладная программа получит управление только тогда, когда текст принимаемого сообщения уже перенесен в массив СТРОКА.

Комбинация команд POINT и GET используется в том случае, если нужно вернуть ранее посылавшееся сообщение, которое хранится в очереди на магнитном диске. Операндами макрокоманды POINT являются метка управляющего блока DCB, с которым связана последующая процедура GET, и адрес области, определяющей координаты возвращаемого сообщения. Среди последних сведений задаются шифр абонента и порядковый номер сообщения.

Несколько более сложным является вариант обмена без ожидания, выполняемый с помощью макрокоманд READ и WRITE. Число обязательных операндов в обеих процедурах — четыре. Например,

READ DECB,SF,INDCB,СТРОКА

Здесь первый аргумент задает адрес блока управления событиями, в котором ОТМД сделает отметку после завершения операции обмена. Для того чтобы не занимать процессорное время на ожидание конца обмена, прикладная программа может выдать после одной или нескольких команд READ макрокоманду WAIT. Например,

READ DECB1,SF,INTV1,СТРОКА1
 READ DECB2,SF,INTV2,СТРОКА2
 READ DECB3,SF,INTV3,СТРОКА3
 WAIT 1,ECBLIST=ADECB

.
 ADECB DS 0F
 DC A(DECB1,DECB2),XL1'80',AL3(DECB3)

В этом примере выдаются команды чтения с трех терминалов и задача переводится в режим ожидания. Как только придет сообщение от любого из этих абонентов, в старший байт соответствующего блока управления событиями будет занесен код X'40' и задача будет «разбужена». Для определения номера абонента, разбудившего задачу, можно организовать цикл проверки байтов DECB1, DECB2, DECB3. Эта процедура может быть упрощена, если вместо макрокоманды WAIT использовать ее модификацию — TWAIT. После такой команды в 15-м регистре будет находиться порядковый номер адреса DECB в списке ADECB, соответствующий сработавшему терминалу.

Вслед за макрокомандой READ может быть выдана макрокоманда CHECK, которая переведет прикладную программу в режим ожидания конца обмена и проверит правильность выполнения операции чтения.

После завершения сеанса связи прикладная программа должна выдать макрокоманду CLOSE и закрыть входные и выходные наборы. Вместо нее можно воспользоваться макрокомандой MPCLOSE, выполнение которой приведет и к снятию ПУС.

Кроме описанных выше макрокоманд ОТМД предоставляет прикладной программе довольно большой набор средств по обработке сообщений, анализу ошибок, выборке и изменению элементов управляющих таблиц ПУС, координации между контрольными точками ПУС и прикладной программы и т. п. Однако описание этого материала выходит за рамки настоящего пособия.

По сравнению с БТМД работа в среде ОТМД требует существенно больших ресурсов памяти и сопряжена с серьезным ухудшением временных характеристик диалоговой системы. Сравнение на частном примере, которое мы приведем в следующих главах, дает не очень полную картину. Однако в работе [5] приводится ПУС, обслуживающая 16 локальных дисплеев. Цифра 90 кбайт, в которую заведомо не включена память под буфера и копии экранов, должна насторожить разработчиков небольших диалоговых систем. В работе [15] утверждается, что для реализации двух эквивалентных программ в среде БТМД потребовалось 26 кбайт, а в среде ОТМД — 60 кбайт оперативной памяти.

ГЛАВА 5

ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ НА ФИЗИЧЕСКОМ УРОВНЕ С ИСПОЛЬЗОВАНИЕМ СРЕДСТВ БТМД

Несмотря на то что стандартные методы доступа БТМД и ОТМД обеспечивают достаточно широкие возможности по организации обмена с дисплеями ЕС-7920, пользоваться ими не очень просто. Во-первых, программист должен хорошо знать структуру передаваемых данных, в которой наряду с отображаемой информацией присутствуют различные управляющие символы. Ошибки в кодировке выдаваемой на экран информации могут быть восприняты как коды управления дисплеем, и тогда результат обмена трудно предсказать. Специфика в задании и расшифровке управляющей информации (СУЗ, ИВ, адреса буферной памяти) усложняет кодировку и логический анализ. Стандартные процедуры штатных методов доступа в некоторых случаях могут и дезинформировать прикладную программу. Так, например, после выдачи макрокоманды READ-TI в БТМД мы можем прочесть не новое сообщение пользователя, а предыдущее, если в промежутке между двумя обходами человек по ошибке лишний раз нажал функциональную клавишу; или, обратившись с макрокомандой READ-TI к терминалу с номером 3, мы вдруг получаем сообщение с первого терминала, так как его владелец нажал функциональную клавишу раньше.

Во-вторых, представление отображаемой информации в виде одномерного массива оперативной памяти плохо увязывается с естественным восприятием экрана дисплея как двумерной таблицы, с которой легче оперировать в терминах строк и столбцов. Наконец, менее заметным недостатком стандартных модулей обмена является их универсализм. Телекоммуникационные программы ориентированы на обслуживание широкого спектра абонентских пунктов и включают полный анализ многочисленных ситуаций, которые принципиально возможны. Но в реальных задачах могут использоваться далеко не все из имеющихся возможностей. Поэтому, обращаясь к БТМД или ОТМД, пользователь жертвует оперативной памятью и теряет время на выполнение ненужных ему проверок. Кроме того, прикладные программисты испытывают массу неудобств от необходимости использования ассемблера.

Все это заставляет искать более удобные и быстрые средства общения с дисплеями. Казалось бы, что самый простой выход —

написать каналные программы для обмена и добавить к ним процедуры обработки ошибочных ситуаций. Однако специфика общения с активным внешним устройством упирается в проблему синхронизации работы программы с поведением ее абонента. По сути дела, мы должны сообщить операционной системе, что ждем сигнал прерывания от пользователя. И как только такой сигнал в ЭВМ поступит, операционная система должна передать управление в указанную нами точку для считывания сообщения, набранного человеком. Именно такого рода механизм ожидания события предусмотрен в операционных системах ДОС ЕС и ОС ЕС, сгенерированных с возможностью обслуживания телекоммуникационных устройств. По сути дела, подготовка к обработке сигнала ВНИМАНИЕ от терминала сводится к записи в соответствующие таблицы операционной системы нужных признаков и адресов. Трудностей здесь две. Во-первых, записывать приходится в защищенную от задачи область памяти супервизора. Во-вторых, надо точно знать, что и куда записывать. Проще всего, если бы эту информацию можно было найти в справочной литературе. К сожалению, исчерпывающих сведений там нет, и наш опыт основан на расшифровке соответствующих модулей БТМД.

5.1. Описание и инициализация терминала в ДОС ЕС

В ДОС ЕС мы должны выбрать логический номер для терминала, например SYS012, назначить для него соответствующее физическое устройство с помощью директивы ASSGN и описать блок управления программой канала. Структура управляющего блока CCB подробно описана в [2], а формат соответствующей макрокоманды для нашего терминала предусматривает два обязательных и два необязательных параметра:

```
CCB7920 CCB SYS012,ACCW[, [X'nnnn'][,ABYS]]
```

Здесь ACCW — адрес первой команды CCW в программе канала. Третий параметр позволяет установить первоначальные значения в байтах признаков блока CCB, с помощью которых можно повлиять на работу супервизора ввода-вывода. Четвертый параметр ссылается на поле, зарезервированное для байтов уточненного состояния, используемых при анализе сбойных ситуаций. Для каждой программы обмена с дисплеем может быть описан свой управляющий блок CCB. Однако с целью уменьшения общего объема программы можно завести один блок CCB, а перед выполнением соответствующей программы канала заносить адрес нужной команды CCW в байты 9—11 общего блока.

Программы канала, которые мы собираемся построить, должны заменить функции модуля BTMOD, Отказавшись от услуг BTMOD

и связанных с этим модулем управляющих таблиц DTFBT, мы вынуждены найти способ замены оператора OPEN, одной из функций которого является формирование в таблице задач признака работы под управлением БТМД. Наличие этого признака обеспечит передачу управления нашим программам для обработки сигнала прерывания во время активного или пассивного обмена с терминалом. О технологии записи в защищенную от пользователя таблицу задач (PIB) с помощью В-транзита мы уже писали в [2], однако в приведенном там примере по вине авторов была пропущена одна команда. Поэтому в приведенной ниже программе моделирования макрокоманды OPEN в ДОС ЕС усовершенствованный вариант В-транзита имеет более удачное наименование $\square\square$ ВТОРЕН.

$\square\square$ ВТОРЕН CSECT

DC CL8' $\square\square$ ВТОРЕН'

LH 4,22 *адрес области связи*

LH 5,124(4)

LH 4,16(5)

LH 5,136(4)

LH 1,18(5)

AN 1,90(4)

OI 12(1),X'40'

SVC 11

END $\square\square$ ВТОРЕН

вычисление адреса и нужного поля в PIB

*запись признака БТМД
выход из В-транзита*

Предлагаемый В-транзит должен быть протранспилирован, отредактирован и помещен в системную библиотеку абсолютных модулей. Теперь вместо макрокоманды OPEN мы должны выполнить программу $\square\square$ ВТОРЕН. Делается это с помощью команды SVC с кодом 2:

LA 1,=CL8' $\square\square$ ВТОРЕН'

SVC 2

После этого мы имеем право выполнять любые каналные программы для обмена с дисплеем.

5.2. Описание и инициализация терминала в ОС ЕС

Назначение, описание и инициализация дисплея выполняются с помощью тех же самых директив DD, макрокоманд DCB и OPEN. Мы можем связать свою программу с конкретным терминалом или получить в свое распоряжение первый свободный дисплей из числа обслуживаемых одним устройством управления. В первом случае нужно указать физический номер устройства: //SYS012 DD UNIT=0C1

Во втором случае параметр UNIT задается значением группового имени, предусмотренного при генерации операционной системы: //SYS012 DD UNIT=EC7920

В макрокоманде DCB уже не надо указывать способ организации набора данных (DSORG = CX), так как обмен мы собираемся выполнять своими средствами: DCB7920 DCB MACRF = (E), DDNAME = SYS012

Параметр MACRF задает режим обмена на уровне EXCP. Поэтому макрокоманда OPEN не присоединит к нашей задаче модули БТМД, реализующие процедуры READ и WRITE. Вместе с тем не будет выполнен и ряд дополнительных мероприятий, обеспечивающих обработку сигнала ВНИМАНИЕ. Эту работу нам придется выполнить самостоятельно, и включает она следующие шаги: создание блока управления обменом (IOB) и блока запросов для обработки прерываний (IRB), формирование заявки на обработку прерывания (элемент очереди IQE), формирование указателей, связывающих управляющие блоки в соответствии с требованиями ОС ЕС.¹

Для формирования блока управления вводом-выводом мы предлагаем макрокоманду IOB.

MACRO		
&M	IOB	&DCB,&CCW,&ECB
&M	DS	0F
	DC	X'C000'
		<i>цепочка данных и команд</i>
	DC	H'0'
		<i>байты уточненно-го состояния</i>
	AIF	(T'&ECB NE '0').ECB
	MNOTE	0,'НЕ ЗАДАН АДРЕС ECB'
	DC	A(*)
	AGO	.CSW
.ECB	DC	A(&ECB)
		<i>под ECB и адрес ECB</i>
.CSW	DC	A(0)
		<i>счетчик адреса для CSW</i>
	DC	X'00'
		<i>байт состояния устройства</i>
	DC	X'00'
		<i>байт состояния канала</i>
	DC	H'00'
		<i>остаточный счетчик</i>
	AIF	(T'&CCW NE '0').CCW
	MNOTE	0,'НЕ ЗАДАН АДРЕС CCW'
	DC	A(*)
	AGO	.CCW1
.CCW	DC	A(&CCW)
		<i>нач. адрес программы канала</i>
.CCW1	AIF	(T'&DCB NE '0').DCB
	MNOTE	0,'НЕ ЗАДАН АДРЕС DCB'

	DC	A(*)	
	AGO	.DCB1	
.DCB	DC	A(&DCB)	<i>адрес DCB</i>
.DCB1	DC	2A(0)	<i>рабочие поля ОС</i>
	DC	X'01'	<i>номер терминала (резерв)</i>
	DC	7X'00'	<i>резерв</i>
	MEND		

В ней предусмотрена выдача сообщений об ошибках в случае отсутствия любого из обязательных аргументов — адреса блока управления данными (DCB), адреса программы канала (CCW) или адреса блока управления событиями (ECB). Поле, соответствующее отсутствующему операнду, заполняется случайным кодом для того, чтобы не сместить поля IOB относительно друг друга и иметь возможность их динамического формирования. Блок IRB создается с помощью системной макрокоманды CIRB, описанной в работах [1, 3]. Параметрами этой макрокоманды являются точка входа в программу обработки прерывания (INTERIO) и режимы ее выполнения (нулевой ключ защиты памяти, блокировка внешних прерываний).

Фрагмент программы (имитация функций OPEN в ОС ЕС) формирует 16-байтовый элемент очереди IQE, который будет использован операционной системой для активизации программы INTERIO.

	CIRB	EP=INTERIO, RETIQE=YES, KEY=SUPR, MODE=SUPR	
	ST	1, AIRB	<i>адрес IRB в поле заявки</i>
	L	1, 16	<i>адрес CVT</i>
	L	1, 0(1)	
	MVC	ATCB, 4(1)	<i>адрес TCB в поле заявки</i>
	L	1, DCB7920+44	<i>адрес DEB</i>
	L	9, 32(1)	<i>адрес UCB</i>
	L	14, 28(1)	<i>адрес таблицы ап-пендиксов</i>
	MODESET	KEY=ZERO	
	MVC	28(4,9), AIRB	<i>запись адреса IRB в UCB</i>
	MVI	36(9), 1	<i>признак головного UCB</i>
	MVC	5(3,14), =AL3(FORSIO)	<i>запись в DEB</i>
	MODESET	KEY=NZERO	
		
IQE	DC	2A(0)	} <i>заявка для обработки сигнала ВНИМАНИЕ</i>
AIRB	DC	A(0)	
AICB	DC	A(0)	

Для установления нужных связей адрес IRB заносится в блок управления дисплеем (поле UCB + 28), а в блоке расширения данных (DEB) формируется адрес программы-аппендикса FORSIO, исполняемой операционной системой перед любым обменом с дисплеем. На программу FORSIO мы возлагаем задачу имитации некоторых функций макрокоманды READ-TI, необходимых для обработки сигнала ВНИМАНИЕ. Во-первых, это формирование управляющего признака, предупреждающего операционную систему о готовности прикладной программы к обработке прерывания от дисплея (код X'04' в байте UCB + 27). Во-вторых, это запись в управляющий блок IRB ссылки на элемент очереди IQE. Одновременное выполнение этих условий приведет к тому, что в момент нажатия функциональной клавиши ОС передаст управление программе INTERIO. После обработки очередного сигнала ВНИМАНИЕ ссылка на заявку IQE из блока IRB удаляется операционной системой. Поэтому связь блока IRB с заявкой на обработку прерывания приходится каждый раз восстанавливать. Для операций обмена, не связанных с ожиданием сигнала ВНИМАНИЕ, программа-аппендикс заносит в байт UCB + 27 нулевой код.

Признаком головного блока UCB (код X'01' по адресу UCB + 36) средства БТМД пользуются при объединении нескольких дисплеев в группу. Блоки UCB для соответствующих устройств в этом случае хранят порядковый номер терминала и ссылку на головной блок UCB, привязанный к первому дисплею группы.

Выполнение некоторых из описанных выше действий связано с записью в защищенную от задачи область памяти (IRB, UCB, DEB). Для этой цели используется макрокоманда MODESET, которая входит в состав ОС ЕС, начиная с версии ОС-6.1. Описание ее в технической документации отсутствует. Мы можем оправдать это опасностью макрокоманды MODESET. Дело в том, что она позволяет прикладной программе выполнить действия, эквивалентные привилегированным командам, например установить режим «супервизора», изменить ключ защиты памяти, замаскироваться от любых сигналов прерывания. Неосторожное использование этих возможностей может нарушить режим функционирования ОС. Мы использовали макрокоманду MODESET для установки нулевого ключа защиты памяти только на время записи в UCB и DEB.

5.3. Обработка сигнала ВНИМАНИЕ в ДОС ЕС

Технология построения программы, ожидающей события, связанного с нажатием функциональной кнопки на клавиатуре дисплея, обсуждалась в работе [2]. Напомним кратко основные шаги этой процедуры. Записью признака в PIB мы уведомили ДОС ЕС, что наша задача может взять на себя функцию обработки сигнала

ВНИМАНИЕ. Это вовсе не означает, что любой обмен с дисплеем будет сопровождаться использованием этой возможности. Например, если задача собирается что-то выдать на терминал, то ни о каком ожидании не может быть и речи. Ждать сигнала от пользователя приходится в двух случаях. На первоначальном этапе общения задача может перейти в режим ожидания, если терминал был выключен или от пользователя не поступило изначальное уведомление о его готовности к работе. В процессе диалога режим ожидания необходим при вводе сообщения, набираемого пользователем.

Для того чтобы уведомить ДОС о желании взять на себя функции обработки прерывания от терминала, необходимо запустить программу канала, ссылающуюся на специальным образом построенный блок CCB. Признаком того, что у задачи имеется собственная программа обработки прерывания, является код X'40' в 13-м байте CCB. Адрес входа в эту программу задается содержимым байтов 14—16 в CCB.

В качестве команды канала, предшествующей переводу задачи в режим ожидания, удобно воспользоваться командой «уточнить состояние» (код X'04'). Ее преимущество перед командой «холостой ход», предлагавшейся в [2], заключается в том, что она избавляет нас от обработки аварийного завершения в случае отключенного дисплея.

Программы обработки сигнала **ВНИМАНИЕ** в ДОС ЕС:

	MVI	BHMH1+2,X'00'	<i>сброс байта уведомления</i>
	EXCP	BHMH	
	WAIT	BHMH1	<i>ожидание ВНИМАНИЯ</i>
	
BHMH	CCB	SYS012,CCWBHMH,X'0400'	
	ORG	*—4	
	DC	X'40',AL3(INTERIO)	
BHMH1	DC	A(0)	
CCWBHMH	CCW	4,0,X'30',1	
	
	USING	*,8	
INTERIO	TM	68,X'80'	<i>проба на ВНИМАНИЕ</i>
	BO	I01	<i>обход, если ВНИМАНИЕ</i>
	CLI	68,X'0C'	<i>проба на КК и ВУК</i>
	BNE	I02	

*** EXCP ВЫПОЛНЕНА УСПЕШНО. ОСТАВЛЯЕМ ЗАЯВКУ В ОЧЕРЕДИ

	OI	6(3),X'84'	<i>запись признаков в PUB</i>
	MVC	68(2),X'0080'	<i>запись PCI в CSW</i>
	B	4(7)	<i>возврат в СУВВ</i>
I01	TM	68,X'7F'	<i>только ли ВНИМАНИЕ?</i>
	BNZ	I02	<i>обход, если что-то еще</i>

*** УВЕДОМЛЕНИЕ ЗАДАЧИ О СИГНАЛЕ ВНИМАНИЕ

MVI	18(1), X'80'
MVC	68(2), =X'0C00'
NI	6(3), X'7B'
B	4(7)

*** ПОВТОРНАЯ ВЫДАЧА SIO

I02	OI	6(3), X'04'
	XC	68(2), 68
	BR	7

Приведены тексты программ, реализующие запуск процедуры фиктивного обмена, ожидание сигнала ВНИМАНИЕ и его обработку с помощью программы INTERIO. Признак X'0400' в макрокоманде CCB означает, что супервизор должен передать управление команде, следующей за EXCP, по сигналу УСТРОЙСТВО КОНЧИЛО. Таким образом, сразу же после выполнения программы канала задача будет переведена в режим ожидания.

В момент появления сигнала ВНИМАНИЕ ДОС передает управление программе INTERIO со следующей информацией в регистрах общего назначения: адрес блока CCB (регистр 1), адрес строки, соответствующей нашему терминалу в таблице PUB (регистр 3), адрес возврата в супервизор ввода-вывода (регистр 7) и адрес входа в программу INTERIO (регистр 8). Последнее обстоятельство определяет выбор базового регистра в программе обработки прерывания.

Основные возможности, предусмотренные в программе INTERIO, сводятся к следующему. Во-первых, появляется возможность повторить работу программы канала. Для этого необходимо занести код X'04' в шестой байт строки PUB, погасить *байты состояния канала* (БСК) и *устройства* (БСУ) в слове состояния канала (CSW) и возвратиться в супервизор ввода-вывода. Во-вторых, мы можем сохранить заявку на ожидание сигнала ВНИМАНИЕ в очереди супервизора ввода-вывода. Для этого необходимо занести код X'84' в шестой байт строки PUB, погасить байт БСУ, занести в БСК признак программно-управляемого прерывания (PCI) и возвратиться в супервизор по адресу 4 (7). Наконец, программа обработки прерывания может уведомить супервизор о нормальном завершении работы по данной заявке на обмен. Для этого необходимо погасить нулевой и пятый разряды в шестом байте строки PUB, занести в байты состояния БСК и БСУ код X'0C00' и возвратиться в супервизор по адресу 4 (7). Программа INTERIO выполняется с нулевым ключом защиты, поэтому она имеет право производить запись в защищенную от задачи область памяти.

5.4. Обработка сигнала ВНИМАНИЕ в ОС ЕС

Логика обработки сигнала ВНИМАНИЕ в ОС ЕС мало чем отличается от описанной выше схемы. Здесь сохранены обозначения основных программных единиц (ВНМН, INTERIO), чтобы подчеркнуть это сходство. Однако функциональное оформление этих программ несколько иное.

MVI	ВНМН1, X'00'	<i>сброс байта уведомления</i>
MVI	ВНМН+33, X'04'	<i>признак для ВНИМАНИЯ</i>
EXCP	ВНМН	
WAIT	ЕСВ=ЕСВВНМН	
WAIT	ЕСВ=ВНМН1	<i>ожидание ВНИМАНИЯ</i>
.		

*** ПОСТРОЕНИЕ БЛОКА УПРАВЛЕНИЯ ОБМЕНОМ

ВНМН	IOB	DCB7920, CCWBНМН, ЕСВВНМН
CCWBНМН	CCW	4, 0, X'30', 1
ЕСВВНМН	DC	F'0'
ВНМН1	DC	F'0'
.		
INTERIO	BALR	15, 0
	USING	*, 15

*** ИМИТАЦИЯ МАКРОКОМАНДЫ POST

LA	11, ВНМН1	<i>адрес байта уведомления</i>
L	12, ATCB	<i>адрес TCB</i>
SR	10, 10	
BCTR	10, 0	<i>аналог кода завершения</i>
L	1, 16	<i>адрес CVT</i>
L	15, 152(1)	<i>адрес входа в n/n POST</i>
BR	15	

*** ПРОГРАММА-АППЕНДИКС SIO

	USING	FORSIO, 15	
FORSIO	NOP	*	
	MVC	27(1, 7), 33(2)	<i>запись признака в UCSB</i>
	L	10, 28(7)	<i>адрес IRB</i>
	MVC	96(4, 10), =A(IQE)	<i>запись адреса IQE в IRB</i>
	BR	14	

Здесь, во-первых, появилась программа-аппендикс FORSIO, которая срабатывает при любой команде EXCP. Для того чтобы ждать сигнал ВНИМАНИЕ, мы должны перед выдачей EXCP записать в 33-й байт таблицы IOB код X'04'. А если процедура обмена в ожидании ВНИМАНИЯ не нуждается, то этот байт в IOB должен быть погашен. Программа-аппендикс выполняется операционной

системой' с нулевым ключом защиты. Так что запись в запретную область памяти не вызовет прерывания задачи.

Во-вторых, часть функций по предварительной обработке сигнала ВНИМАНИЕ берет на себя ОС ЕС, поэтому функция программы INTERIO свелась только к уведомлению ожидающей задачи.

5.5. Структура программ канала для основных процедур обмена

Минимальный набор средств, необходимый для создания диалоговой системы на базе локальных дисплейных станций, перекрывается тремя основными процедурами. Среди них — одна процедура вывода информации на экран дисплея с указанного адреса буферной памяти и две процедуры чтения содержимого экрана. О технологии включения или отключения режима ожидания сигнала ВНИМАНИЕ при чтении с дисплея уже говорилось в разд. 5.3 и 5.4. Сейчас мы остановимся на некоторых деталях построения программ канала, выполняющих указанные выше процедуры.

Вывод на экран можно реализовать либо с помощью команды записи (КОП = 01), либо с помощью команды записи с предварительным стиранием (КОП = 05). Операция чтения для информационного экрана также может быть реализована с помощью двух команд — чтения буферной памяти (КОП = 02) или чтения модифицированных полей (КОП = 06). Поэтому мы условимся, что в командах CCW код KR будет соответствовать одной из команд чтения, а код KW — одной из команд записи.

Для вывода на экран текстовой информации, содержащей L байтов и расположенной на поле ТЕКСТ, можно воспользоваться следующей парой команд CCW:

```
CCW KW, YSTADR, X'A0', 4
CCW KW, ТЕКСТ, X'20', L
```

```
YSTADR DC X'40', X'11', X'bbbb'
```

Первая команда выдает на дисплей четырехбайтовый заголовок, содержащий *символ управления записью* (СУЗ, код X'40'), код установки адреса буферной памяти (X'11') и двухбайтовый адрес начальной позиции текста на экране. В обеих командах CCW используется признак блокировки неправильной длины (X'20') для того, чтобы не обращаться к системной программе обработки соответствующей ошибки. С точки зрения программиста, никакой ошибки в указании длины нет, если только объем выводимого текста не выходит за пределы буферной памяти. Однако специфика дисплея такова, что на любой обмен устройство управления выдает признак неправильной длины. Первая команда CCW содержит дополнитель-

ный признак цепочки данных ($X'20' + X'80' = X'A0'$). Это удобнее, чем воспользоваться признаком цепочки команд, так как тогда пришлось бы в начале поля ТЕКСТ расположить еще один байт с кодом СУЗ. Код $X'40'$, выбранный в качестве символа управления записью, не изменяет состояния дисплея, т. е. сохраняет режим работы клавиатуры, не перемещает курсора и не выдает звукового сигнала. Однако можно было бы подобрать и другую комбинацию битов в СУЗ.

Операцию чтения данных с экрана можно выполнить по-разному. Например, сначала выдается на экран команда записи, устанавливающая счетчик адреса буферной памяти в нужное положение, а затем в цепочке команд выдается команда чтения:

```
CCW X'01',YCTADR,X'60',4
CCW KR,KYDA,X'20',L + 3
```

При этом на поле с меткой KYDA будут прочитаны сначала три управляющих байта (код ИВ и адрес маркера), а затем уже L информационных символов, отображенных на экране. Поэтому удобнее было бы прочитать информационные и управляющие байты в разные поля:

```
CCW X'01',YCTADR,X'60',4
CCW KR,IWADRM,X'A0',3
CCW KR,ТЕКСТ,X'20',L
```

Здесь IWADRM — трехбайтовое поле для управляющей информации. Другой вариант чтения с заданной точки экрана включает два этапа, на первом из которых фиктивно считывается ненужная часть информации с начала буферной памяти, а на втором читается интересующая нас информация. Это может быть выполнено, например, с помощью такой цепочки команд:

```
CCW X'02',IWADRM,X'A0',3
CCW X'02',ТЕКСТ,X'B0',LFIKT
CCW X'02',ТЕКСТ,X'20',L
```

Здесь LFIKT — число байтов, предшествующих нужному тексту. Фиктивное чтение информации обеспечивается наличием кода $X'10'$ в байте признаков второй команды CCW. С точки зрения времени работы программы канала первый вариант связан с передачей меньшего объема информации между ЭВМ и устройством управления. Однако у второго варианта есть другое преимущество — не требуется формировать внутреннее представление начального адреса буферной памяти.

ГЛАВА 6

ПРОЕКТИРОВАНИЕ ПОДПРОГРАММЫ ОБМЕНА С ТЕРМИНАЛОМ

Цель настоящей главы — познакомить читателя с конкретной реализацией подпрограммы обмена с локальным дисплеем, которую можно использовать для организации диалога в прикладных программах, составленных на фортране или ассемблере. В качестве базового примера выбран реальный модуль RWO7920, эксплуатируемый в текстовом редакторе (гл. 7) и пакете ДИАФОР (гл. 10). Не изменяя функциональных возможностей этого модуля, мы продемонстрируем технологию его построения для ДОС ЕС и ОС ЕС с использованием различных методов доступа, а затем сравним варианты получившихся программ. Для максимального приближения к производственным условиям мы начнем с формулировки пунктов технического задания на создаваемый программный продукт.

Назначение. Подпрограмма должна обеспечить двусторонний обмен с локальной дисплейной станцией и допускать обращение из программ, составленных на фортране.

Технические требования.

A1. Обмен между ЭВМ и терминалом должен производиться порциями, кратными целому числу строк. Для вызывающей программы в качестве указателя расположения данных на экране должен служить номер строки — целое число из диапазона [1, 24].

A2. Процедура вывода на экран дисплея должна обеспечивать дополнительные возможности по управлению позицией расположения курсора. В качестве указателя позиции курсора необходимо использовать целое число из диапазона [0, 1919].

A3. Процедура ввода должна начинаться с выдачи заявки операционной системе на чтение и завершаться после нажатия пользователем функциональной клавиши. В дополнение к информации, считываемой с экрана, процедура ввода должна сообщать позицию курсора и условный номер функциональной клавиши.

A4. Использование средств форматизации в программе обмена не предусматривается.

A5. Программа обмена должна обеспечивать соответствующую перекодировку символьной информации с целью исключения из передаваемого текста управляющих кодов и устранения двусмысленности из-за различия в кодировке больших и малых букв.

Обращение и описание параметров. Модуль RWO7920 должен иметь два входа — с метками RDDISP и WRDISP, обеспечивающими соответственно выполнение процедур чтения 'с экрана по разрешению пользователя и вывода на экран. Операторы вызова должны включать следующие параметры:

CALL RDDISP (ТЕКСТ, NCTP, KCTP, MARKER, NFK)
CALL WRDISP (ТЕКСТ, NCTP, KCTP, MARKER)

Здесь ТЕКСТ — массив в оперативной памяти, содержащий 80 * KCTP байтов и предназначенный для приема или передачи информации. Параметры NCTP и KCTP соответственно определяют номер начальной строки и количество строк, участвующих в процедуре обмена. Переменные NFK и MARKER предназначены для хранения условного номера функциональной клавиши (табл. 6.1) и указателя позиции курсора на экране. На поле ТЕКСТ располагается символьная информация в коде ДКОИ, все остальные параметры — переменные целого типа.

Т а б л и ц а 6.1

Код NFK	Функцио- нальная клавиша	Код NFK	Функцио- нальная клавиша	Код NFK	Функцио- нальная клавиша
0	ВВОД	6	ПФ6	12	ПФ12
1	ПФ1	7	ПФ7	13	ПД1
2	ПФ2	8	ПФ8	14	ПД2
3	ПФ3	9	ПФ9	15	ПД3
4	ПФ4	10	ПФ10	16	ВЫЗОВ
5	ПФ5	11	ПФ11	17	ТЕСТА
					СТРН ЭКР

Значение аргумента MARKER в процедуре вывода, не принадлежащее диапазону [0, 1919], означает, что курсор должен сохранить свое положение.

6.1. Логическая схема подпрограммы

Прежде чем приступить к составлению подпрограммы обмена, попробуем обсудить последовательность шагов, необходимых для выполнения процедур ввода-вывода. Это позволит выделить отдельные блоки, которые могут оказаться общими для обеих процедур. Реализация одних блоков может зависеть от операционной системы и выбранного метода доступа, для других блоков такая зависимость может отсутствовать. Так или иначе, расчленение программы на функциональные фрагменты окажет существенную помощь в ее последующей реализации и модификации,

Первый функциональный блок, который надо выделить, связан с инициализацией терминала. В этой операции, выполняемой при первом обращении к терминалу, может нуждаться и процедура RDDISP, и процедура WRDISP. Чтобы не дублировать анализ состояния терминала (открыт — закрыт) в каждой из процедур обмена, эту проверку целесообразно включить в блок инициализации, который назовем INIT. Этот блок будет выполнять все операции, связанные с открытием соответствующего набора данных, если обращение к RWO7920 было первым, и срабатывать как пустой оператор при всех последующих обращениях.

Второй функциональный блок, используемый в обеих процедурах, связан с предварительной обработкой и анализом правильности задания исходных параметров. В задачи этого блока, который назовем PARAM, следует включить преобразование номера начальной строки (NCTP) во внутреннее представление адреса буферной памяти, проверку на принадлежность значение NCTP интервалу [1, 24], выделение из фрагмента [NCTP, NCTP + KCTP — 1] части, принадлежащей экрану.

Последующие шаги в процедурах RDDISP и WRDISP расходятся. Для вывода эта цепочка включает всего две операции:

WR3. Перекодировка выдаваемой информации на предмет замены пробелами управляющих символов, которые могут быть по ошибке включены в состав массива ТЕКСТ, и замена значения МАРКЕР соответствующим адресом буферной памяти.

WR4. Вывод на экран.

В процедуре ввода приходится проделать гораздо больше шагов:

RD3. Вывод управляющей информации для разблокировки клавиатуры.

RD4. Ожидание сигнала ВНИМАНИЕ.

RD5. Чтение информации с экрана.

RD6. Перекодировка считанного текста с целью замены кодов малых букв и символов «пусто».

RD7. Преобразование адреса курсора и байта ИВ в более естественный формат.

Некоторые из перечисленных шагов связаны с выполнением соответствующих операций обмена. Поскольку их реализация зависит от операционной системы и выбранного метода доступа, выделим все операции обмена в группу исполнителей ввода (IO), к которым можно обращаться по номеру исполнителя: 0, 4, 8, 12. Такой подход несколько снижает эффективность нашей программы, так как для запуска исполнителя потребуются дополнительные команды перехода и возврата. Однако модифицировать такую программу при переходе на другие методы доступа будет значительно проще.

В дополнение к выделенным шагам введем две стандартные операции (макрокоманды), связанные с мероприятиями по базированию,

запоминанию регистров общего назначения при входе во внешнюю подпрограмму и восстановлению их содержимого при возврате в вызывающую программу. Назовем их соответственно PROLOG и

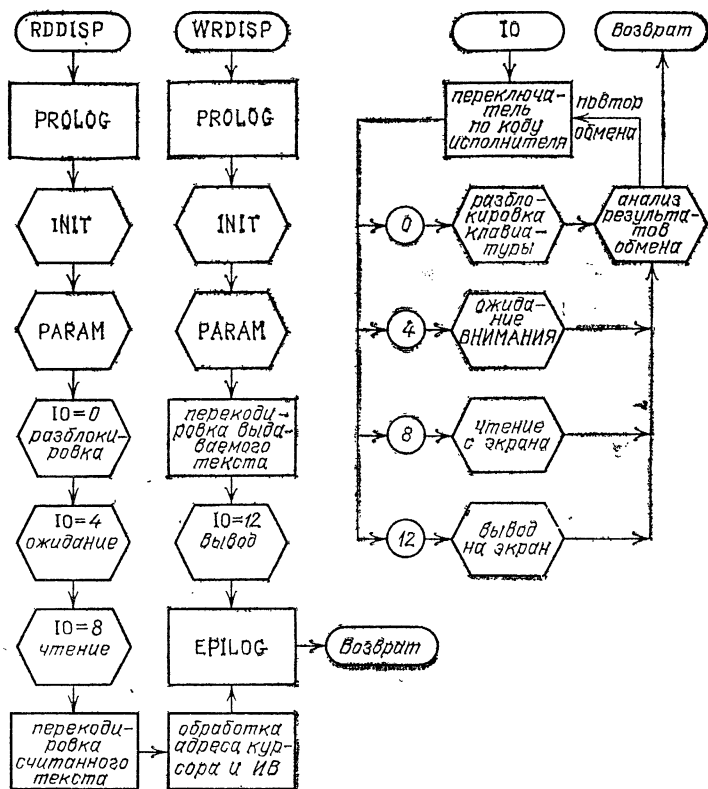


Рис. 6.1. Логическая схема модуля RWO7920

EPILOG. После всего сказанного логическая схема модуля RWO7920, приведенная на рис. 6,1, в дополнительных комментариях не нуждается.

Макрокоманды оформления входа и вывода:

```
MACRO
&M    PROLOG
      USING      &M,12
&M    SAVE      (14,12)
      LR         12,15
      LR         2,13
```

*запоминание регистров
перезагрузка базы
прежняя область сохра-
нения*

	LA	13,SAVRWO	<i>новая область сохранения</i>
	ST	2,4(13)	<i>связь областей</i>
	ST	13,8(2)	<i>связь областей</i>
	LM	2,6,0(1)	<i>адреса параметров</i>
	USING	SERVIS,11	
	LA	11,SERVIS	<i>база сервисных программ</i>
	MEND		
	MACRO		
&M	EPILOG		
&M	L	13,4(13)	
	RETURN	(14,12)	
	MEND		

Инвариантный текст процедуры чтения:

RDDISP	PROLOG		
	BAL	10,INIT	<i>инициализация терминала</i>
RD3	SR	7,7	
	BAL	10,IO	<i>на разблокировку клавиатуры</i>
RD4	LA	7,4	
	BAL	10,IO	<i>на ожидание ВНИМАНИЯ</i>
RD5	LA	7,8	
	BAL	10,IO	<i>на чтение с экрана</i>
RD6	BAL	10,TRТЕКСТ	<i>на перекодировку</i>

*** ПРЕОБРАЗОВАНИЕ ИДЕНТИФИКАТОРА ВНИМАНИЯ

RD7	SR	2,2	
	NI	BYTIW,X'1F'	<i>выделение 5 мл. разрядов</i>
	TRT	BYTIW(1), IWSLWR	
	ST	2,0(6)	<i>запись в KEY</i>

*** ПРЕОБРАЗОВАНИЕ АДРЕСА КУРСОРА ИЗ ФОРМАТА ДИСПЛЕЯ

IC	0,MARKER	<i>старшая цифра</i>
N	0,=F'31'	<i>выделение 6 мл. разрядов</i>
SLB	0,6	<i>сдвиг влево</i>
IC	1,MARKER+1	<i>младшая цифра</i>
N	1,=F'63'	<i>выделение 7 мл. разрядов</i>
OR	1,0	<i>объединение</i>
ST	1,0(5)	<i>запись адреса</i>
EPILOG		

*** СЛОВАРЬ ДЛЯ ПЕРЕКОДИРОВКИ БАЙТА ИВ

UWSLW	DC	X'0F0D100E1211'	ПД3, ПД1, СТРН ЭКР, ПД2,, ВЫЗОВ ТЕСТА
	DC	X'0102030405060708090A0B0C'	ПФ1—ПФ12
	DC	X'001212'	ВВОД

Инвариантный текст процедуры вывода:

WRDISP	PROLOG		
	BAL	10, INIT	инициализация терминала
WR3	L	5, 0(5)	новый адрес маркера
	LTR	5, 5	
	BM	WR32	обход, если адрес < 0
	CH	5, =H'1919'	
	BNH	WR31	обход, если адрес ≤ 1919
	LNR	5, 5	смена знака для IO
	B	WR32	

*** ПРЕОБРАЗОВАНИЕ АДРЕСА МАККЕРА В ФОРМАТ ДИСПЛЕЯ

WR31	LR	0, 5	
	SRL	0, 6	сдвиг вправо на 6 разрядов
	STC	0, МАККЕР	старшая цифра адреса
	N	5, =F'63'	выделение 7 мл. разрядов
	STC	5, МАККЕР+1	младшая цифра адреса

*** ПЕРЕКОДИРОВКА СТРОК ВЫДАВАЕМОГО ТЕКСТА

WR32	BAL	10, TRТЕКСТ	
WR4	LA	7, 12	
	BAL	10, IO	на программу вывода
	EPILOG		

*** ОБЛАСТЬ СОХРАНЕНИЯ ДЛЯ МОДУЛЯ RWO7920

SAVRWO	DS	18F
--------	----	-----

Подпрограмма перекодировки текста:

*** ПЕРЕКОДИРОВКА ДИСПЛЕЙНОГО ТЕКСТА

TRТЕКСТ	LTR	0, 4	задан ли текст
	BCR	8, 10	возврат, если не задан
	LR	1, 2	адрес текста в задаче
TR	TR	0(80, 1), DISPSLW	перекодировка строки
	LA	1, 80(1)	
	BCT	0, TR	

```
DISPSLW DC CL64'_'
DC CL10'_' ,C' [. < (+ '
DC C' && , CL9'_' C' ] Q * ) ; '
DC C' - / , CL9'_' C' , % - > ? '
DC CL6'_' , C' ЮБ _ : # @ ' = "
DC C' Ц A B C D E F G H I Д _ Ф Г _ И '
DC C' Й I K L M N O P Q R K Л М Н _ П '
DC C' Я _ S T U V W X Y Z _ _ Т _ Ж В '
DC C' Ъ Ы З Ш Э Щ Ч _ Ю _ Б Ц Д _ Ф Г '
DC C' _ A B C D E F G H I _ И Й _ Л _ '
DC C' _ J K L M N O P Q R _ _ П Я _ _ '
DC C' \ _ S T U V W X Y Z _ У Ж _ Ъ Ы '
DC C' 0 1 2 3 4 5 6 7 8 9 З Ш Э Щ Ч _ '

```

6.2. Соглашения об использовании регистров

Из-за технических причин, связанных с типографским набором текстов программ, было решено отказаться от мнемонического обозначения регистров общего назначения (РОН). Во всех последующих фрагментах программ за большинством регистров закреплены определенные функции, перечень которых приведен в табл. 6.2.

Т а б л и ц а 6.2

Номер РОН	Выполняемая функция
2	Адрес массива ТЕКСТ в вызывающей программе
3	Число символов информации, участвующей в обмене
4	Число строк информации, участвующей в обмене
5	Адрес слова с адресом курсора
6	Адрес слова с кодом функциональной клавиши
7	Номер исполнителя обмена (код IO)
10	Адрес возврата из внутренних подпрограмм
11	Адрес базы для внутренних подпрограмм
12	Базовый регистр для модуля RWO7920
13	Адрес области сохранения

Остальные регистры общего назначения используются в качестве рабочих.

6.3. Инициализация терминала

Приводимые ниже варианты блока INIT в какой-то мере повторяют рассматривавшиеся в предыдущей главе фрагменты соответствующих программ. Авторы пошли на этот повтор по двум причинам. Во-первых, соглашения об использовании общих регистров требуют внесения некоторых поправок. Во-вторых, сборка разрозненной информации позволяет увидеть работу блока в целом.

6.3.1. Блок INIT для ДОС ЕС (уровень EXCP). Текст блока INIT для ДОС ЕС состоит из четырех следующих команд:

INIT	NOP	PARAM	<i>входной переключатель</i>
	OI	INIT+1, X'F0'	<i>формирование обхода</i>
	LA	1, =CL8'QQBTOPEN'	
	SVC	2	<i>имитация OPEN</i>
PARAM		

Первая команда выполняет функцию переключателя, который обеспечивает переход на В-транзит при первом входе в блок INIT и обход процедуры открытия при последующих входах.

6.3.2. Блок INIT для ОС ЕС (уровень EXCP). Текст блока INIT отличается от соответствующего варианта из разд. 5.2 только употреблением регистров.

Инициализация терминала в ОС ЕС на физическом уровне

INIT	NOP	PARAM	<i>переключатель</i>
	OI	INIT+1, X'F0'	
	OPEN	(DCB7920)	<i>открываем набор</i>
	TM	DCB7920+48, X'10'	<i>открылся ли?</i>
	BO	INIT1	<i>обход, если «да»</i>
	ABEND	1111	
INIT1	CIRB	EP=INTERIO, RETIQC=YES, KEY=SUPR, MODE=SUPR	
	ST	1, AIRB	
	L	1, 16	
	L	1, 0(1)	
	MVC	ATCB, 4(1)	
	L	1, DCB7920+44	
	L	9, 32(1)	
	L	14, 28(1)	
	MODESET	KEY=ZERO	
	MVC	28(4, 9), AIRB	
	MVI	36(9), 1	
	MVC	5(3, 14), =AL3(FORSIO)	
	MODESET	KEY=NZERO	
PARAM		

6.3.3. Блок INIT для ОС ЕС (уровень БТМД). Начальный фрагмент этого блока полностью совпадает с предыдущим вариантом; отличие начинается с метки INIT1.

Модификация программы INIT для БТМД:

```
INIT1  L      1,DCB7920+44
        SR      8,8
        IC      8,36(1)      длина IOB
        B       14,28(1)
        LA      1,0(8,14)    адрес IOB
        NI      28(1),X'7F'  удаление признака
PARAM  . . . . .
```

В отличие от технологии проверки дисплея на готовность при открытии, описанной в п. 3.4.1, здесь применена тактическая хитрость. Макрокоманда OPEN, обнаружив, что дисплей не включен, занесет в 28-й байт блока IOB признак X'80' и сформирует в 15-м регистре код возврата X'14'. А мы удаляем признак неготовности из IOB, хотя терминал мог оказаться и выключенным. Это позволяет одинаково реагировать на отключение дисплея в процессе работы и упрощает логику обработки последствий обмена.

6.4. Обработка параметров

Подпрограмма обработки параметров:

```
PARAM  L      4,0(4)      КСТР — число строк
        LPR     4,4        | КСТР |
        BZ      PAR2      обход при КСТР=0
        L       3,0(3)     NCTP—номер первой строки
        BCTR    3,0        NCTP—1
        LPR     0,3        | NCTP—1 |
        CH      0,=H'23'   строка на экране?
        BNH     PAR1      обход, если «да»
        LA      0,23       ограничитель сверху
PAR1    MH      0,=H'80'   начальный адрес
        LR      1,0
        SRL     1,6
        STC     1,ADRBUFF  ст. цифра в коде дисплея
        LR      1,0
        N       1,=F'63'
        STC     1,ADRBUFF+1 мл. цифра в коде дисплея
        LPR     1,3
        LA      1,0(1,4)   номер последней строки
        CH      1,=H'24'   принадлежит ли экрану?
        BNH     PAR2      обход, если «да»
        LA      3,1920
```


	SR	3,0	<i>число символов в тексте</i>
	BR	10	
PAR2	LR	3,4	
	MH	3,=H'80'	<i>число символов в тексте</i>
	BR	10	

Приведенный блок обработки параметров не зависит ни от операционной системы, ни от метода доступа. Его задачи довольно просты. Если число строк (NCTP) равно нулю, то блок PAR2AM никаких действий не совершает. В противном случае номер первой выводимой строки NCTP преобразуется в адрес буферной памяти дисплея. В процессе этого перевода значение NCTP проверяется на принадлежность интервалу [1, 24], и в случае ошибочного задания NCTP принудительно загоняется в этот интервал. Двухбайтовый адрес начала информации на экране, сформированный в коде дисплея, заносится в поле ADRBUF. Затем вычисляется номер последней строки текстовой информации, который в случае необходимости обрезается по концу экрана. Число символов в определенном таким образом фрагменте заносится в третий регистр.

6.5. Реализация обменов с дисплеем

Подпрограмма IO состоит из входного переключателя и четырех исполнителей, реализующих соответственно операции по передаче инициативы пользователю (FORREAD), ожиданию сигнала ВНИМАНИЕ (WAITBHMH), чтению (READ) и записи (WRITE). Номер исполнителя задается в седьмом регистре; возврат из исполнителей организуется по содержимому 10-го регистра.

Текст входного переключателя, который приводится ниже, одинаков для всех вариантов программы RWO7920:

```

IO  B  *+4(7)
      B  FORREAD      Nисп=0
      B  WAITBHMH     Nисп=4
      B  READ          Nисп=8
      B  WRITE         Nисп=12

```

Ниже рассматриваются особенности реализации исполнителей обмена в зависимости от принятого метода доступа.

6.5.1. Исполнитель разблокировки клавиатуры. Задача этого исполнителя сводится к выдаче на дисплей одного-единственного байта SUZ06, содержащего признаки снятия блокировки клавиатуры и генерации звукового сигнала. В любом случае выполняется та или иная команда канала, после которой производится стандартный анализ результатов обмена. Тексты трех вариантов программы разблокировки приводятся ниже:

*** РАЗБЛОКИРОВКА КЛАВИАТУРЫ (УРОВЕНЬ EXCP, ДОС ЕС)

```
FORREAD MVC ACCW(4),=A(CCWFOR)
          B EXCP
CCWFOR CCW 1,SUZ06,X'20',1
SUZ06 DC X'06'
```

*** РАЗБЛОКИРОВКА КЛАВИАТУРЫ (УРОВЕНЬ EXCP, ОС ЕС)

```
FORREAD MVC ACCW(4),=A(CCWFOR)
          MVI ECBVNMH,0 для ожидания ВНИМАНИЯ
          MVI IOB7920+33,4 имитация READ-TI
          B EXCP1
```

*** РАЗБЛОКИРОВКА КЛАВИАТУРЫ (УРОВЕНЬ БТМД, ОС ЕС)

```
FORREAD WRITE DECB7920,TI,,SUZ06,1,MF=E
          LTR 15,15
          BNZ ABEND
          WAIT ECB=DECB7920
          B ANALIZ
```

6.5.2. Ожидание сигнала ВНИМАНИЕ. Техника обработки сигнала ВНИМАНИЕ с помощью программы INTERIO в ДОС ЕС подробно рассматривалась в 5.3. Простота исполнителя для уровня EXCP в ОС ЕС объясняется тем, что мы связали в общую цепочку работу двух последовательных исполнителей.

Исполнители ожидания сигнала ВНИМАНИЕ:

*** ОЖИДАНИЕ ВНИМАНИЯ (УРОВЕНЬ EXCP, ДОС ЕС)

```
WAITVNMH MVI ECBVNMH+2,0
          EXCP CCBVNMH
          WAIT ECBVNMH
          BR 10
```

*** ПОСТРОЕНИЕ РАСШИРЕННОГО БЛОКА CCB

```
CCBVNMH CCB SYS012,CCWBVNMH,X'0400'
          ORG *-4
          DC X'40',AL3(INTERIO)
ECBVNMH DC A(0)
CCWBVNMH CCW 4,*,X'10',1
```

*** ОЖИДАНИЕ ВНИМАНИЯ (УРОВЕНЬ EXCP, ОС ЕС)

```
WAITVNMH WAIT ECB=ECBVNMH
          BR 10
```

*** ОЖИДАНИЕ ВНИМАНИЯ (УРОВЕНЬ БТМД, ОС ЕС)

```
WAITVNMH READ DECB7920,TI,,BYTIW,1,MF=E
          WAIT ECB=DECB7920
          B ANALIZ
```

При разблокировке клавиатуры перед выдачей команды EXCP в байт с адресом IOB7920 + 33 заносится признак X'04', имитирующий работу макрокоманды READ-TI. Поэтому в момент нажатия функциональной клавиши сработает вариант программы INTERIO в ОС (см. 5.4), который занесет признак пробуждения в байт ESBVNMH.

В вариантах исполнителя, рассчитанных на физический уровень доступа, анализ работы канала по приему сигнала ВНИМАНИЕ не делается. Для варианта с БТМД выдача команды READ-TI связана с работой нескольких команд канала. Поэтому для ожидания на уровне БТМД анализ работы программ канала предусмотрен.

6.5.3. Обмен с дисплеем на физическом уровне. Исполнители обмена READ и WRITE в ДОС ЕС и ОС ЕС устроены идентично и приведены ниже.

Чтение с экрана на физическом уровне:

READ	MVC	ACCW(4),=A(CCWRD)	адрес CCW
	MVI	RD MARK+4,X'20'	разрыв цепочки
	LTR	3,3	
	BZ	EXCP	
	MVI	RD MARK+4,X'A0'	восстановление цепочки
	ST	2,RDTEKCT	адрес текста
	STH	3,RDTEKCT+6	длина текста
	B	EXCP	
.....			
CCWRD	CCW	1,SUZUAB,X'60',4	установка адреса
TDIW	CCW	2,BYTIW,X'A0',1	чтение ИВ
RD MARK	CCW	2,MARKER,X'A0',2	чтение маркера
RDTEKCT	CCW	2,*,X'20',1	чтение текста
.....			
BYTIW	DS	X	байт для ИВ
SUZUAB	DC	X'00'	нейтральный СУЗ
	DC	X'11'	код команды УАБ
ADRBUF	DC	X'0000'	формируемый адрес
.....			
UST MARK	DC	X'00'	нейтральный СУЗ
	DC	X'11'	код команды УАБ
MARKER	DC	X'0000'	формируемый адрес
	DC	X'13'	код команды УСТ КР

Вывод на экран на физическом уровне:

WRITE	MVC	ACCW(4),=A(CCWWRM)	заготовка для маркера
	LTR	3,3	есть ли текст?
	BNZ	WRITE1	

	LTR	5,5	<i>двигать ли маркер?</i>
	BP	EXCP	<i>на сдвиг маркера</i>
	BR	10	
WRITE1	MVC	ACCW(4),=A(CCWW R)	<i>заготовка для текста</i>
	ST	2,WRTEKCT	<i>адрес текста</i>
	STH	3,WRTEKCT+6	<i>длина текста</i>
	MVI	WRTEKCT+4,X'A0'	
	LTR	5,5	<i>текст+маркер?</i>
	BP	EXCP	
	MVI	WRTEKCT+4,X'20'	<i>разрыв цепочки</i>
	B	EXCP	
.			
CCWW R	CCW	1,SUZUAB,X'A0',4	<i>установка адреса</i>
WRTEKCT	CCW	1,*,X'A0',1	<i>вывод текста</i>
WRMARK	CCW	1,USTMARK,X'20',4	<i>сдвиг маркера</i>
CCWWRM	CCW	1,USTMARK,X'20',5	<i>сдвиг маркера</i>

В исполнителе чтения первые три команды канала CCWRD, RDIW и RDMARK выполняются в любом случае. Последняя команда присоединяется к цепочке данных только в том случае, если число строк, считываемых с дисплея, задано не равным нулю. Читателя не должен смущать тот факт, что при формировании адреса текста в команде RDTEKCT «затирается» код операции в команде CCW. Дело в том, что она выполняется с признаком цепочки данных, и ее код уже установлен в команде RDIW.

В исполнителе вывода в зависимости от заданных аргументов может сработать одна из трех программ канала. Это — либо пара команд CCWW R — WRTEKCT (выводится только текст), либо тройка команд CCWW R — WRTEKCT — WRMARK (выводится текст и производится установка маркера), либо одна команда CCWWRM (производится только установка маркера).

6.5.4. Исполнители обмена на базе БТМД. Исполнители обмена и структура данных на базе БТМД приводятся ниже:

*** ИСПОЛНИТЕЛИ ОБМЕНА (УРОВЕНЬ БТМД, ОС ЕС)

READ	LA	9,3(3)	<i>длина текста+3</i>
	READ	DECB7920,TBP,.,BYTIW,(9),SUZUAB,MF=E	
	WAIT	ECB=DECB7920	
	LA	1,BUFER	<i>начало прочитанного текста</i>
	LR	8,2	<i>начало текста в задаче</i>
	BAE	14,MOVE	<i>перенос текста</i>
	MVC	MARKER(2),ADRMARK	
	B	ANALIZ	

WRITE	LTR	9,3	есть ли текст?
	BNZ	WRITE1	обход, если «да»
	LTR	5,5	есть ли маркер?
	BCR	4,10	выход, если «нет»
	MVC	BUFWR(5), USTMARK	5 байт для маркера
	LA	9,5	длина информации
	B	WRITE2	
WRITE1	MVC	BUFWR(4), SUZUAB	упр. инф. для текста
	LA	9,4(9)	длина+4
	LA	8,BUFWR+4	адрес начала текста
	LR	1,2	начало текста в задаче
	BAL	14,MOVE	перенос текста
	LTR	5,5	есть ли маркер?
	BM	WRITE2	обход, если «нет»
	MVC	0(4,8), USTMARK+1	4 байта для маркера
	LA	9,4(9)	длина+4
WRITE2	WRITE	DECB7920,TI,,BUFWR,(9),MF=E	
	WAIT	ECB=DECB7920	
	B	ANALIZ	

*** РАЗМЕЩЕНИЕ ПОЛЕЙ ВВОДА-ВЫВОДА ДЛЯ БТМД

BUFWR	DS	X	начало поля вывода
BYTIW	DS	X	начало поля ввода
ADRMARK	DC	X'0000'	поле для адреса маркера
BUFER	DS	CL1920	поле для буфера экрана
RESERW	DS	CL4	запас для вывода
SUZUAB	DC	X'0011'	СУЗ и код УАБ
ADRBUF	DC	X'0000'	формируемый адрес
USTMARK	DC	X'00'	нейтральный СУЗ
	DC	X'11'	код команды УАБ
MARKER	DC	X'0000'	формируемый адрес
	DC	X'13'	код команды УСТ КР

Для реализации обмена средствами БТМД приходится вводить буфер емкостью 1928 байт. Дополнительные 8 байт используются для формирования в них управляющих кодов (СУЗ, УАБ, адрес буфера, адрес маркера, УСТ КР), о которых пользователь прикладной программы заботиться не должен. При считывании информации с экрана помимо текста заданной длины приходится принимать три дополнительных байта — байт ИВ (метка BYTIW) и адрес маркера (поле ADRMARK). Максимальный объем считываемой информации может составить 1923 байт. Выбор макрокоманды READ-TBP для реализации исполнителя чтения объясняется следующими

соображениями. По команде READ-TI в системе БТМД формируется команда канала «чтение модифицированных полей», по которой незначительные (шестнадцатеричные) пули с экрана не читаются. Кроме того, по этой команде не читается текст при нажатии некоторых функциональных клавиш. Команда READ-TBP свободна от этих недостатков.

При выводе информации на экран заданному тексту всегда предшествуют четыре управляющих байта — СУЗ, УАБ и двухбайтовый адрес. Вслед за текстом, если он присутствует, располагаются еще четыре управляющих байта — УАБ, двухбайтовый адрес маркера и код команды установки курсора (УСТ КР). Формирование этой информации на поле вывода особой трудности не представляет.

Для переписи текстовой информации с поля задачи на поле вывода или обратной переписи при чтении используется подпрограмма MOVE:

*** ПОДПРОГРАММА ПЕРЕПИСИ СТРОК

```
MOVE   LTR  0,4           число строк=0?
        BCR  8,14         выход, если текст не нужен
MOVE1  MVC  0(80,8),0(1)
        LA   1,80(1)
        LA   8,80(8)
        BCT  0,MOVE1
        BR   14
```

6.5.5. Выполнение программ канала и анализ завершения обмена. Выполнение программ обмена на физическом уровне, по существу одинаковое для ДОС и ОС ЕС, немного отличается по форме. В данном разделе мы приведем тексты наиболее важных фрагментов.

В ДОС ЕС блок управления обменом формируется по макрокоманде CCB и включает следующие наиболее важные поля:

```
CCB7920 CCB  SYS012,*,X'1500',BYS
ACCW    EQU  CCB7920+8
BSC     EQU  CCB7920+2
BYS     DS    X
```

Здесь BSC — байт состояния канала, BYS — байт уточненного состояния терминала. По их содержанию мы будем судить о характере выполнения обмена. Работа большинства исполнителей IO оканчивалась формированием в CCB адреса первой команды программы канала и передачей на метку EXCP. Запуск подготовленной программы в ДОС ЕС производится следующим образом:

```
EXCP    EXCP CCB7920
        WAIT (1)
ANALIZ  . . . . .
```

Признак X'1500' в ССВ7920 означает, что ДОС передаст управление по сигналу «КАНАЛ КОНЧИЛ & УСТРОЙСТВО КОНЧИЛО», а также предоставит право анализировать результат завершения с помощью нестандартной программы ANALIZ.

В ОС ЕС формирование управляющих блоков происходит несколько иначе:

```
DCB7920 DCB DDNAME=SYS012,MACRF=E
IOB7920 IOB DCB7920,*,ECB7920
ACCW EQU IOB7920+16
BSC EQU IOB7920+13
ECB7920 DS F
```

Здесь мы воспользовались макрокомандой IOB, описанной в гл. 5. Запуск канальной программы, подготовленной исполнителем IO, производится с помощью следующих команд:

```
EXCP MVI IOB7920+33,0 сброс признака TI
EXCP1 EXCP IOB7920
WAIT ECB=ECB7920
ANALIZ . . . . .
```

Анализ результатов выполнения программы канала в ДОС ЕС и ОС ЕС происходит по одинаковой схеме. Если имеет место особый случай (седьмой бит в BSC равен 1), то это означает, что произошел случайный сбой в линии между устройством управления (EC-7922) и терминалом (EC-7927). В такой ситуации достаточно повторить процедуру обмена:

```
ANALIZ TM BSC,1 проба на особый случай
BO IO
```

Так как содержимое седьмого регистра мы не испортили, то переход на IO означает передачу управления в начало того же самого исполнителя обмена. Если случай не особый и отсутствует признак сбоя устройства (шестой бит в BSC равен 0), то анализ можно оканчить:

```
TM BSC,2
BCR 8,10
```

При ненулевом шестом разряде байта состояния канала возможны две ситуации: либо терминал выключен, о чем свидетельствуют единицы в первом или шестом разряде байта уточненного состояния (TM BYS,X'42'), либо произошел сбой, который мог оказаться как случайным, так и неустранимым. При выключенном дисплее самое разумное — выдать сообщение оператору о ситуации и без ожидания от него ответа передать управление исполнителю WAITBHMH.

Если пользователь включит дисплей и нажмет любую функциональную кнопку, то работа будет продолжена. В случае сбоя оборудования (единица в одном из разрядов 2, 3, 4 или 5 в байте уточнен-

ного состояния) необходимо обратиться за решением к оператору ЭВМ. Например, выдать ему сообщение: «СБОЙ НА ДИСПЛЕЕ. ЧТО ДЕЛАТЬ? П — ПОВТОРИТЬ, С — СНЯТЬ». В зависимости от ответа оператора управление надо передать либо вновь на исполнитель IO, либо организовать аварийное завершение задачи. Мы не приводим полный текст программы анализа только потому, что общение с оператором в ДОС и ОС ЕС организуется с помощью разных макрокоманд. В ДОС ЕС — это макрокоманды OUTLOG и INLOG, в ОС ЕС — это макрокоманды WTO или WTOR. Можно организовать такой обмен и на уровне EXCP.

6.6. Сравнение вариантов

С точки зрения пользователя все приведенные варианты обладают равными функциональными возможностями и быстродействием, так что сравнивать их разумно только по объему оперативной памяти. Вариант для ДОС ЕС занимает примерно 1,5 кбайт, а обмен на физическом уровне в ОС ЕС — немногим больше (около 2 кбайт). С использованием средств БТМД объем программы возрастает примерно до 9 кбайт, из которых около 5 кбайт занимают программы обработки макрокоманд READ и WRITE. Еще 2 кбайт мы выделили под промежуточный буфер, чтобы снять с пользователя заботу о специфике обмена с дисплеем. Все приведенные варианты модуля RWO7920 апробированы в диалоговых системах, описанных в гл. 7 и 10. К ним можно обращаться из программ, написанных на ассемблере или фортране. Для прикладных программистов, использующих ПЛ/1, мы рекомендуем программу PL7920, описанную в работе [16]. Она базируется на средствах БТМД и предлагает пользователю практически все возможности по работе с форматизованным и неформатизованным экраном. В руководствах по языку ПЛ/1 для ОС ЕС описаны возможности работы с терминалами на базе ОТМД. Однако в состав штатных версий операционных систем ОС-4.1 и ОС-6.1 соответствующая ПУС не включена. В работе [7] описан комплекс PLTCAM, базирующийся на расширенном варианте ОТМД, эксплуатируемом в системе разделения времени (CPB).

ГЛАВА 7

ОСНОВНЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ ДИСПЛЕЙНОГО РЕДАКТОРА ТЕКСТОВОЙ ИНФОРМАЦИИ

Дисплейный редактор — одна из неперенных компонент системного математического обеспечения современной вычислительной установки. С его помощью мы вводим и корректируем программы, формируем задание для операционной системы, просматриваем сообщения системных программ и результаты выполнения прикладных программ и т. п. Вычислительным машинам Единой серии, ориентированным на пакетный режим работы, в этом смысле не повезло. Первые годы ЕС ЭВМ вообще не комплектовались дисплейным оборудованием. Поэтому естественно, что в централизованно поставляющихся версиях операционных систем никаких дисплейных редакторов не было. Это вынуждало пользователей заниматься адаптацией различных зарубежных систем или вести оригинальные разработки. Некоторые из этих разработок оказались настолько удачными, что получили массовое признание, хотя до сих пор в состав централизованной поставки не входят. И их распространение ведется только благодаря энтузиазму авторов и заинтересованных пользователей. В гл. 7—9 мы рассмотрим основные вопросы, связанные с разработкой и основными функциональными возможностями наиболее популярных систем.

7.1. Что такое естественное редактирование

Каждому из нас в той или иной мере приходится заниматься составлением и редактированием различных текстовых документов — отчетов по НИР, статей и докладов, деловых писем и служебных записок, характеристик и отзывов, программ на алгоритмических языках и т. п. Существует большое количество методических материалов, отраслевых и государственных стандартов, регламентирующих форму и содержание разделов тех или иных документов. Но вряд ли кому-нибудь удастся составить требуемый документ сразу начисто без каких-либо вставок или исправлений. И основные орудия труда, которыми при этом пользуются, стары как мир — бумага, пишущий инструмент, ножницы и клей. С их помощью мы формируем фрагменты будущего документа и многократно перекраиваем их — заменяем отдельные слова, предложения или целые абзацы, удаляем

неудачные места, дописываем новые формулировки. Все эти многократно повторяемые процедуры и составляют существо процесса редактирования.

Для определения термина «естественное редактирование» воспользуемся следующей моделью. Разделим письменный стол на три части. Пусть слева располагается текущая версия нашего документа, подлежащая очередной правке. Правую часть стола отведем для размещения страниц с исправленным текстом. И наконец, средняя часть стола представляет собой нечто вроде «операционной», оснащенной упомянутым выше инструментом для редактирования. Каждая итерация по правке текста сводится к тому, что мы переносим очередную страницу с левой части стола в «операционную» и начинаем корректировать текст, используя три основные процедуры — вставку, замену и удаление. Применять их можно по отношению к отдельным символам, словам, предложениям и целым абзацам. При этом не накладывается никаких ограничений на размеры вставок или на соотношения между размерами заменяемого и замещающего фрагментов. В результате длина страницы может существенно измениться как в ту, так и в другую сторону, и на ней могут появиться многочисленные исправления, так что в дальнейшем текст лучше перепечатать.

В процессе редактирования весьма полезными могут оказаться такие процедуры, как перестановка фрагментов исходного текста — включение в отредактированный текст разделов из других документов или стандартных заготовок. Кроме того, очень важно сохранять копию исходного материала, так как очередная правка может оказаться и не очень удачной. Это особенно полезно в программировании, ибо исправление обнаруженных ошибок зачастую влечет за собой появление новых ошибок в программе. Процедура редактирования может быть прервана до исчерпания материала на левой части стола. Например, нам захотелось сложить уже исправленные страницы с остатками еще не просмотренного материала и начать все сначала.

Теперь переложим сказанное на язык вычислительной техники. Представим себе, что содержимое левой части стола каким-то образом введено в ЭВМ и хранится во внешней памяти наряду с другими архивными документами. Для просмотра и корректировки исходного текста лучше всего воспользоваться экраном дисплея, который заменит нам среднюю часть стола. Условно можно считать, что исходный текст находится на длинной киноленте с кадрами-страницами, а экран дисплея представляет собой окно, которое можно передвигать по этому тексту в прямом и обратном направлениях на любое число строк или кадров. Представим себе, что функциональная клавиатура дисплея позволяет вносить в содержимое окна любые исправления, о которых упоминалось выше. Тогда исходный текст как бы

«выполняет» в окно снизу, подвергается там правке, а все, что «выполняет» из окна сверху, представляет собой отредактированный текст. Его тоже можно хранить в архиве и использовать для повторных

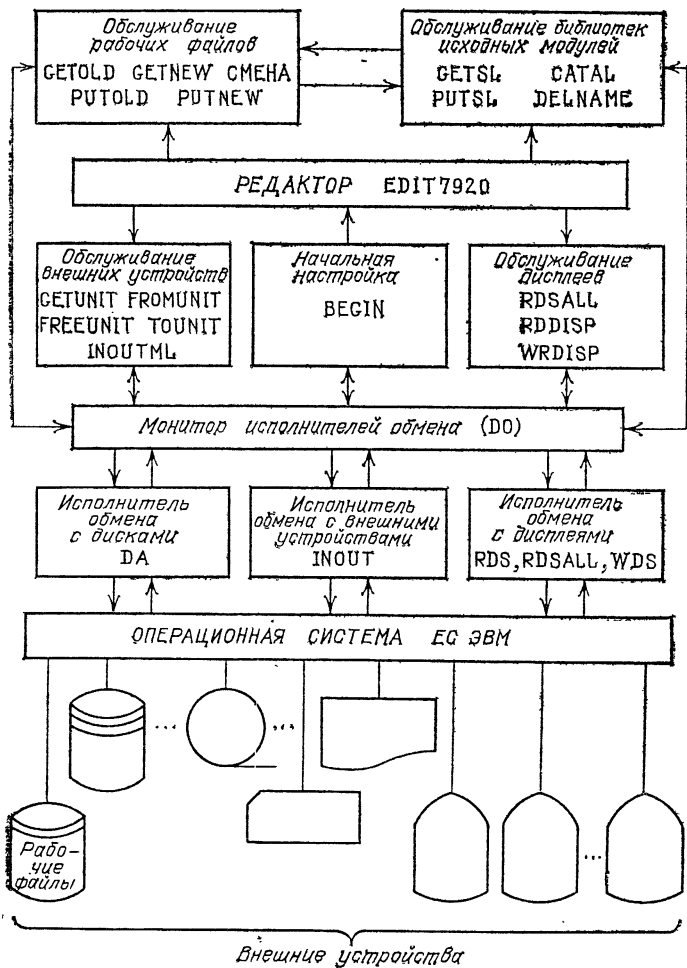


Рис. 7.1. Функциональная структура редактора

исправлений. Такого рода технологию мы считаем естественной и попытались ее реализовать в текстовом редакторе [10], о котором и пойдет речь в данной главе. Хотя описываемая реализация выполнена в рамках операционной системы ДОС ЕС, ее проектные решения применимы в любой операционной системе, в том числе и в ОС ЕС,

Описываемый редактор ориентирован в основном на работу с текстами исходных программ, т. е. с документами, представляющими набор строк по 80 символов каждая.

Функциональная структура редактора показана на рис. 7.1.

Прежде чем перейти к разговору об особенностях реализации редактора, рассмотрим формат экрана и организацию рабочих файлов.

7.2. Структура экрана

Экран дисплея ЕС-7927 вмещает 24 строки по 80 символов каждая. Исходя из соображений «естественности», высказанных ранее, поле экрана было распределено так, как это показано на рис. 7.2.

В самом верху (строка 1) располагается строка, в которой находится информация, только что «вытолкнутая» с экрана в файл с отредактированным текстом — новый файл. Следующие 20 строк (строки 2 — 21) используются для отображения очередной информации, предназначенной для записи в новый файл. Это — то, что мы называли окном для просмотра и исправлений. По мере выполнения редакторских процедур содержимое указанных строк поднимается вверх, а предложение, попадающее в первую строку, переписывается в новый файл. В окне экрана можно производить любые исправления — отменять ненужные строки, изменять их текст частично или полностью, вставлять новые строки.

Строка 22 предназначена для набора директив или корректируемых данных. В момент передачи управления пользователю наборная строка чистится, а маркер устанавливается в ее начало.

Строка 23 используется для индикации выполняемых процедур и информационных сообщений редактора. Устроена она следующим образом. Первые 28 позиций отведены под поле MESSAGE, которое используется для выдачи сообщений редактора. Собственно сообщение (информация об ошибках, разрешение продолжить работу и т. п.) занимает 23 позиции этого поля. Оставшиеся пять позиций используются для хранения количества строк в исходном тексте. Позиции 29 — 37 занимает поле REGIM, в котором отображается текущая операция, выполняемая редактором. Следующее поле REGIM1 занимает позиции 38 — 50 и используется для индикации режимов табуляции и нумерации. Далее располагается двухбайтовый служебный признак расположения рабочих файлов (поле K), часы с указанием текущего времени в виде T = ЧЧ.ММ и два счетчика с порядковыми номерами строк в новом (NOB = XXXX) и исходном (ISX = XXXX) файлах.

Самая нижняя строка экрана (строка 24) называется просмотровой. В ней находится очередное предложение из исходного файла, а его порядковый номер занесен в счетчик ISX.

Содержимое строк 1, 23 и 24 защищено от пользователя самым примитивным способом. В оперативной памяти хранится копия экрана, а редактор реагирует только на изменения, производимые пользователем в строках 2 — 22. Такая разметка экрана, на наш

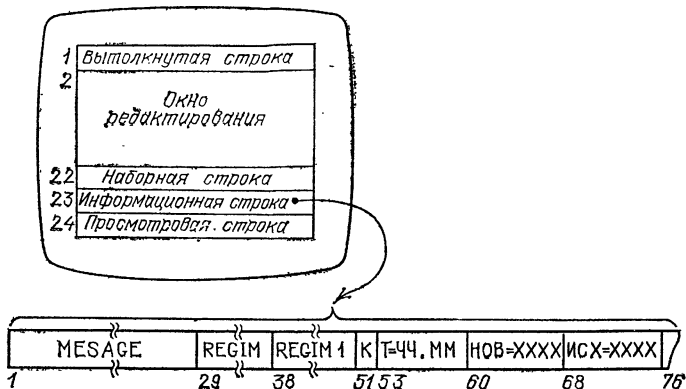


Рис. 7.2. Структура экрана

взгляд, близка к естественному представлению. Однако в других системах формат экрана может существенно отличаться от описанного. Прекрасный обзор по этому вопросу можно найти в работе [13].

7.3. Входной и выходной рабочие файлы

Размещение исходного и отредактированного текстов целиком в оперативной памяти не так уж и эффективно, хотя время реакции редактора в этом случае можно было бы свести к минимуму. Если ввести разумные ограничения на размеры редактируемого текста, например, до 1000 строк, то в оперативной памяти пришлось бы выделить порядка 160 кбайт только на обслуживание одного пользователя. А вместе с программами самого редактора объем занимаемой оперативной памяти мог бы достигнуть 200 кбайт. Поэтому файлы с исходным и отредактированным текстами разумно вынести на магнитный диск, а в оперативной памяти хранить только буферы для обмена с файлами. Рабочий файл с исходным текстом мы будем называть исходным или старым файлом, а файл с отредактированной информацией — новым файлом.

Для уменьшения времени перемещения механизма головок дискового оба файла разумно поместить на общих цилиндрах. Например, на дисках типа ЕС-5061 (емкость 29 Мбайт) можно 10 верхних дорожек одного цилиндра отвести под старый файл, а 10 нижних дорожек — под новый файл. Количество цилиндров, выделяемое для

рабочих файлов, зависит от размера блоков на дорожке и максимально допустимого объема редактируемых текстов. При выборе размера блока приходится считаться с довольно противоречивыми условиями. Во-первых, чем больше размер блока, тем эффективнее используется емкость дорожки. Но тогда приходится выделять слишком много места в оперативной памяти под буферы обмена. Во-вторых, информационная емкость дорожек на малых (7,25 Мбайт) и средних (29 Мбайт) дисках различна. А редактор желательно построить таким образом, чтобы он легко перенастраивался на требуемую конфигурацию. Иногда ведь даже на одной вычислительной установке могут эксплуатироваться разные дисковые устройства. И наконец, размер блока должен хорошо согласовываться с емкостью окна на экране дисплея. Это позволит организовать страничную работу с редактируемым текстом. С учетом вышеизложенных соображений размер блока был установлен равным 1680 байт, что соответствует странице текста емкостью 21 строка. На одной дорожке дисков типа ЕС-5051 размещается два таких блока, т. е. 42 строки, а на одной дорожке дисков типа ЕС-5061 — четыре блока, т. е. 84 строки текста. Таким образом, один цилиндр малых дисков позволяет завести рабочие файлы объемом до 210 строк, а один цилиндр средних дисков — до 840 строк. Отсюда и выбирается требуемое число цилиндров под рабочие файлы с учетом количества одновременно обслуживаемых пользователей и максимального объема редактируемого материала.

И еще одно соображение, связанное с размещением рабочих файлов: их следует располагать на дисковом пакете, не содержащем пользовательских библиотек; в противном случае механизм подвода головок будет совершать непроизводительные перемещения между библиотеками и рабочими файлами.

Рабочие файлы должны обеспечивать прямой доступ к любой строке. Для их обслуживания в редакторе был написан модуль IOFILE, обеспечивающий выполнение следующих операций: чтение строки из входного или выходного файла, запись, строки во входной или выходной файл, смену местами расположения входного и выходного файлов. Обращения к соответствующим входным точкам этого модуля имеют вид:

```
CALL GETOLD,(СТРОКА,НОМЕР,END)
CALL GETNEW,(СТРОКА,НОМЕР,END)
CALL PUTOLD,(СТРОКА,НОМЕР)
CALL PUTNEW,(СТРОКА,НОМЕР)
CALL СМЕНА
```

Параметр СТРОКА определяет 80-байтовое поле оперативной памяти, участвующее в приеме (GET) или передаче (PUT) информации. Номер строки в файле, с которой производится обмен, задается па-

аметром HOMER. Аргумент END является выходным. Он принимает нулевое значение, если считывание завершено удачно, и ненулевое ($END = 255$), если рабочий файл пуст.

С целью экономии объема программы и повышения ее быстродействия модуль IOFILE был написан на ассемблере с использованием доступа к дискам на физическом уровне. Для работы с каждым из файлов связаны два буфера BUFOLD и BUFNEW емкостью по 1680 байт и несколько указателей. К ним относятся указатели общего количества строк в файлах (NOLD и NNEW), дисковые адреса (CDOLD и CDNEW) в форме CC-HH-R, определяющие соответствие между блоком файла и его отображением на буфере, счетчики номеров текущих строк (IOLD и INEW). Строка исходного текста, номер которой указан в счетчике IOLD, отображается в нижней строке экрана. Строка отредактированного текста, находящаяся в самой верхней строке экрана, уже записана в буфер BUFNEW, и ее порядковый номер хранится в счетчике INEW. В информационной строке экрана отображено содержимое некоторых из перечисленных указателей: число строк в исходном тексте (NOLD), номер текущей строки в исходном (ISX = ...) и отредактированном (NOB = ...) файлах.

Все процедуры обмена с файлами автоматически продвигают на один номер строки (параметр HOMER) после удачного выполнения операции. Это упрощает работу редактора при последовательном чтении исходных или записи отредактированных строк. Процедура смены рабочих файлов сводится к обмену содержимого указателей, гашению указателя NNEW и начальной установке счетчиков строк ($IOLD = 1$, $INEW = 0$). После нее отредактированный текст начинает выступать в роли исходного и готов к повторной правке. Признак взаимного расположения старого и нового файлов на цилиндрах отображен в информационной строке: $K = 0$ или $K = 1$. Он позволяет организовать продолжение работы редактора после сбоях ЭВМ без потери информации.

Большинство из описанных процедур можно реализовать и с помощью макрокоманд операционной системы — READ и WRITE для прямого доступа. Однако использование обмена на физическом уровне не только позволило сократить объем программы, но и повысить их быстродействие. Составленные таким образом программы удалось очень экономично обобщить на случай многотерминального обслуживания.

7.4. Язык общения с редактором и программные средства синтаксического контроля

Запуск любой процедуры редактирования осуществляется путем набора пользователем соответствующей директивы в строке 22 и нажатия функциональной клавиши. Содержимое наборной директивы

формулируется на языке, близком к естественному: НАЙДИ такой-то фрагмент, ПЕРЕПИШИ столько-то строк и т. д. С целью сокращения времени набора наиболее употребительные директивы допускают односимвольные сокращения: К вместо КАТАЛОГ, З вместо ЗАПИШИ и т. д. Для выполнения ряда директив ничего набирать не приходится — достаточно нажать одну из функциональных клавиш. Чтобы различать директивы этих двух типов, будем называть их соответственно наборными и кнопчными. Наборные директивы, как правило, завершаются нажатием функциональной клавиши ПФ5. На клавиатурах типа ЕС-0101-01 наряду с клавишей ПФ5 аналогичную функцию выполняют и все функциональные клавиши нижнего ряда — ПФ10, ПФ11, ПФ12.

Рассмотрим технику анализа директивы, которая реализована в редакторе и которую с успехом можно применить в любой другой диалоговой системе. Предоставляя инициативу пользователю, редактор фактически обращается к подпрограмме RDDISP за вводом информации и переходит в режим ожидания. В тот момент, когда пользователь нажимает функциональную клавишу, редактору передают содержимое строк 2 — 22, код нажатой клавиши и адрес курсора. Код функциональной клавиши — это преобразованное значение байта ИДЕНТИФИКАТОР ВНИМАНИЯ. Нулевое значение кода соответствует клавише ВВОД, значения от 1 до 12 — клавишам ПФ1 — ПФ12. Клавишам ПД1 — ПД3 соответствуют коды 13 — 15, значения 16 и 17 — клавишам СТРН ЭКР и ВЫЗОВ ТЕСТА. При сбое оборудования программа чтения выдает в качестве кода клавиши число 18. Поступившие значения кода клавиши и адреса курсора хранятся в ячейках с метками KODFK и МАРКЕР. Анализ начинается с передачи управления по переключателю, построенному в соответствии с кодами функциональных клавиш:

	L	2,KODFK	код клавиши
	SLL	2,2	умножение на 4
	B	SWITCH(2)	переход на переключатель
SWITCH	B	WWOD	переход по кл.ВВОД
	B	PF1	переход по кл.ПФ1
	B	PF2	переход по кл.ПФ2

.....

Большинство функциональных клавиш закреплено за кнопчными директивами, не нуждающимися в дополнительной информации. Поэтому соответствующая команда переключателя произведет передачу управления на нужный исполнитель редактора. Однако с нажатием некоторых клавиш связаны две и более операций. Поэтому после нажатия, например, клавиш ПФ5, ПФ10 — ПФ12 мы попадаем на блок анализа наборной директивы. Его работа построена на проверке первого слова директивы. Для этого указатель на-

борной строки (регистр 1) перемещается на первый, отличный от пробела символ, и ключевое слово директивы в цикле сравнивается с табличным набором допустимых слов. Приведем фрагмент анализа директивы по ключевому слову:

*** АНАЛИЗ ДИРЕКТИВЫ

DIRAN	LA	1,СТРОКА22	<i>установка указателя в начало</i>
	BAL	11,ОВРВ	<i>обход пробелов в начале дирек-</i>
			<i>тивы</i>
	SR	2,2	
	LM	3,5,КОНСТ	
РОИСК	IC	2,0(3)	<i>длина ключевого слова—1</i>
	L	6,0(3)	<i>адрес ключевого слова</i>
	EX	2,СПАВН	<i>косвенное сравнение</i>
	BE	GOTODIR	<i>уход при совпадении</i>
	BXLE	3,4,РОИСК	
	B	OSHIB	<i>директива не опознана</i>
СПАВН	CLC	0(0,1),0(6)	

*** УХОД НА ОБРАБОТКУ ДИРЕКТИВЫ

GOTODIR EQU *

.....

*** КОНСТАНТА ДЛЯ ОРГАНИЗАЦИИ ЦИКЛА

КОНСТ DC A(DIRTAB,8,DIRTABK)

*** ТАБЛИЦА КЛЮЧЕВЫХ СЛОВ И ИХ ХАРАКТЕРИСТИКИ

DIRTAB	DC	0F
	DC	AL1(L'C#OUTLIB—1),AL3(C#OUTLIB), A(OUTLIB)
	DC	AL1(L'C#INLIB—1),AL3(C#INLIB), A(INLIB)

.....

DIRTABK EQU *—8

*** МАССИВ КЛЮЧЕВЫХ СЛОВ

C#OUTLIB	DC	C'НАЙДИ'
C#INLIB	DC	C'ЗАПИШИ'

Основную роль здесь играет таблица ключевых слов DIRTAB, организованная следующим образом. Каждая ее строка занимает 8 байт и содержит информацию об одном ключевом слове. Первый байт — длина ключевого слова, уменьшенная на 1. Три следующих байта — адрес константы с текстом ключевого слова. Пятый байт — признак обязательного наличия (X'00') или возможного отсутствия (X'80') параметров вслед за ключевым словом. Наконец, последние три байта хранят адрес программы обработки директивы. Полному и сокращенному наименованию директивы в таблице DIRTAB соот-

ветствуют две разные строки, так как длины ключевых слов при этом различны. На составление таблицы DIRTAB надо обратить внимание, так как ее строки нельзя переставлять произвольным образом. Например, строка с сокращенным вариантом директивы ПЕРЕПИШИ, ссылающаяся на однобуквенную константу П, должна быть расположена ниже строки, соответствующей директиве ПОВТОРИ. Если это не так, то директива ПОВТОРИ неправильно опознается по первой букве.

Использование таблицы DIRTAB очень удобно, так как ее легко модифицировать. Она не зависит от содержания обрабатываемых директив и может быть легко настроена на любую другую диалоговую систему с языком директивного типа. Байт, контролирующий возможность опускания параметров, используется для общей проверки перед выходом на блок обработки опознанной директивы. Иначе такого рода проверку пришлось бы многократно повторять в обрабатывающих программах.

7.5. Работа с библиотеками исходных текстов

Редактор начинает сеанс работы с очередным пользователем с приглашения произвести назначение диска и библиотеки. При этом в информационную строку заносится текст приглашения, а в наборной строке появляется заготовка для соответствующей директивы:

Д = _ _ _ _ _ , Б =

Это приглашение можно проигнорировать, нажав сразу же клавишу ПФ5, т. е. предложив редактору пустые имена. В этом случае будет обеспечено выполнение любых функций, кроме обращения к библиотеке.

Если указанный диск установлен и библиотека с указанным именем на этом диске найдена, то в информационной строке появится разрешение на продолжение работы. Сообщение ТАКОГО ДИСКА НЕТ означает либо ошибку в задании имени пакета, либо то, что диск с указанным именем не установлен, не успел раскрутиться или не приведен в состояние готовности. Ошибка в задании имени библиотеки приводит к появлению сообщения ТАКОЙ БИБЛИОТЕКИ НЕТ. Сообщения об ошибках можно либо проигнорировать, перейдя к набору других директив, либо исправить, подкорректировав то или иное имя.

В любой момент сеанса пользователь может просмотреть действующие назначения диска и библиотеки, нажав клавишу ВЫЗОВ ТЕСТА. При этом в наборной строке появится готовая к исполнению директива:

Д = имя1, Б = имя2

До нажатия клавиши ПФ5 любое из этих имен можно изменить, и тем самым произвести переназначение диска и/или библиотеки.

После назначения библиотеки целесообразно убедиться в правильности своего выбора. Для этой цели можно набрать директиву КАТАЛОГ, или К. При ее выполнении на экране высвечивается либо полное оглавление библиотеки, либо первая его страница, содержащая не более 120 наименований (20 строк по шесть имен). Запасные имена (только в ОС ЕС) могут быть помечены специальным символом, например звездочкой. Если в библиотеке находится достаточно много модулей с исходными текстами и каталог на экране целиком не помещается, то редактор сообщит: «КАТАЛОГ НЕ ВЕСЬ. ДАЛЬШЕ?». Чтобы увидеть продолжение, необходимо набрать директиву ДАЛЬШЕ. Для больших библиотек может потребоваться несколько таких директив, чтобы досмотреть оглавление до конца. Пользователь может в любой момент отказаться от просмотра продолжения каталога и перейти к выполнению любых других директив.

Еще одна операция, которая может быть выполнена на любой стадии после назначения библиотеки, — это удаление из библиотеки одного или нескольких модулей. Выполняется такая процедура с помощью директивы ИСКЛЮЧИ, или И, в которой после ключевого слова задается список имен удаляемых модулей. Выполняя удаление, редактор вносит коррекцию только в оглавление библиотеки, но не производит физического стирания самого модуля. Это обеспечивает возможность сохранения ошибочно уничтоженного текста до сжатия библиотеки.

Обслуживание библиотек с исходными текстами в редакторе выполняет модуль LIBRARY, имеющий четыре входа. По входу GETSL производится выборка из библиотеки. Исходной информацией для этой процедуры являются имена диска (NAMEDISK), библиотеки (NAMELIB) и программы (NAMEPROG), режим выборки (байт REG) и граничные номера строк извлекаемого фрагмента (NMIN и NMAX). Считываемая из библиотеки информация может быть передана редактору целиком (NMIN = 1, NMAX = 9999) или частично. Ее можно либо присоединить к содержимому старого файла (REG = 0 в директивах НАЙДИ или ДОБАВЬ), либо вставить в новый файл вслед за содержимым экрана (REG = 1 в директиве ВСТАВЬ). Код возврата, засылаемый в выходной параметр KEY, формируется кратно 4, с тем чтобы по нему можно было организовать переключатель для обработки ситуации. При выборке из библиотеки таких ситуаций может быть пять: выборка завершена успешно (KEY = 0), не найдены требуемый модуль (KEY = 4), библиотека (KEY = 8) или диск (KEY = 12), размер запрашиваемого текста превышает запасы свободного места в старом файле (KEY = 24).

Второй вход PUTSL используется для записи отредактированного текста в библиотеку. Эта процедура использует четыре параметра из упомянутых выше — имена диска, библиотеки и программы, а также значение управляющего байта KEY. При первом обращении к процедуре PUTSL байт KEY должен иметь нулевое значение. Если в библиотеке уже находится модуль с таким же именем, что и в поле NAMEPROG, то возможно повторное обращение к подпрограмме PUTSL с ненулевым значением байта KEY (KEY = 16). Это будет означать, что пользователь разрешил заменить прежний модуль новым, выдав директиву ДА. Работа программы PUTSL начинается с просмотра оглавления. Если в нем отсутствует имя новой программы, то делается попытка записать содержимое отредактированного файла. В случае, когда длина свободного места библиотеки недостаточна для размещения нового материала, формируется соответствующий код возврата (KEY=20) и управление передается редактору. Возврат со значением KEY=16 означает, что в библиотеке уже хранится модуль с таким же именем. Алгоритм записи в библиотеку таков. Материал с новым именем всегда записывается на свободное место вслед за находящимися в библиотеке модулями. Затем производится коррекция оглавления библиотеки и всех ее указателей (ссылка на свободное место, объем свободного места и т. п.). Программа, которая замещает старую программу, может быть записана на место своей предшественницы. Это происходит в случае, когда прежняя длина была не меньше длины нового модуля. Если замещаемая программа была в библиотеке последней, то новая программа всегда записывается на ее место.

Процедура удаления из библиотеки инициируется по директиве ИСКЛЮЧИ и выполняется подпрограммой DELNAME. Исходной информацией для ее работы являются имена диска, библиотеки и исключаемого модуля. Если указанный модуль найден, то в оглавление библиотеки вносится соответствующая корректура. Ошибка в задании параметров игнорируется, хотя, может быть, и было бы полезным сообщать об этом пользователю. Для удаления нескольких модулей, перечисленных в одной директиве, редактор обращается к подпрограмме DELNAME в цикле до тех пор, пока не будет исчерпан список имен.

Последняя процедура (вход CATAL) осуществляет выборку из оглавления библиотеки очередного имени. Исходную информацию в данном случае составляют имена диска и библиотеки, а также указатель JCTLG. При обращении за первым именем оглавления указатель должен быть нулевым. Подпрограмма CATAL выбирает из оглавления первое имя и помещает в поле NAMEPROG, а в указателе JCTLG запоминает всю информацию, необходимую для выборки следующего имени. Когда из оглавления будет извлечено имя последнего модуля, указатель JCTLG снова сбрасывается в нуль

Формированием и выдачей на экран очередной страницы оглавления занимается сам редактор.

Программная реализация модуля, обслуживающего библиотеки с исходными текстами, существенно зависит от операционной системы. В ДОС ЕС и ОС ЕС используются совершенно различные форматы оглавления библиотеки. Собственно тексты библиотечных программ хранятся в библиотеках тоже по-разному. Например, в ДОС ЕС исходный текст предварительно сжимается путем замены подряд идущих пробелов и хранится блоками по 160 байт. А в ОС ЕС библиотечные блоки с исходными текстами хранят информацию в первоначальном виде, но зато могут содержать различное количество строк. Операционная система ОС ЕС включает группу макрокоманд для обработки наборов данных с библиотечной организацией, а ДОС ЕС для этого никаких средств пользователю не предоставляет. Поэтому в данном разделе мы ограничились логикой функционирования модуля LIBRARY, а не комментированием соответствующей программной реализации.

7.6. Формирование исходного текста во входном файле

В начале сеанса редактирования входной файл пуст, и попытка прочесть что-либо из него редактором игнорируется. Исходный текст в старом файле может появиться в результате выполнения одной из трех процедур: изначальное формирование путем сборки из библиотечных фрагментов (директива НАЙДИ), дополнение уже существующего исходного текста библиотечными фрагментами (директива ДОБАВЬ) и переименование файла с отредактированным текстом во входной файл (кнопочная директива СМЕНА). Ссылка на библиотечный фрагмент представляет собой имя модуля с последующим заданием номеров граничных строк NMIN и NMAX в круглых скобках, например АНКЕТА (10,95). Номера граничных строк — это порядковые номера строк в модуле, отсчитываемые от 1 и не имеющие никакого отношения к нумерации операторов в текстах исходных программ (позиции 73—80 каждой строки). Если граничные номера отсутствуют, то указанный модуль извлекается из библиотеки целиком. Из одного и того же модуля можно выбрать несколько фрагментов и никаких ограничений на их порядок при этом не накладывается, например НАЙДИ АНКЕТА(50,100), АНКЕТА (1,8), АНКЕТА(50,120).

Значение максимального номера строки в фрагменте может превышать общее количество строк в модуле. Редактор не посчитает это за ошибку, но больше строк, чем есть в модуле, не выберет. После выполнения процедуры сборки или дополнения на экране дисплея появляется сообщение об общем количестве строк исходного текста

в старом файле. Изначальное формирование исходного текста заканчивается выборкой первых 80 байтов в просмотровую строку дисплея.

Ошибки при сборке исходного текста могут быть вызваны тремя причинами. Во-первых, очередной элемент списка библиотечных фрагментов может содержать ошибочное имя, либо модуля с таким именем в библиотеке нет, либо написание имени не соответствует общим правилам операционной системы. Во-вторых, возможна ошибка при задании начального и конечного номеров строк фрагмента (например, $NMIN \leq 0$, или $NMIN > NMAX$). Наконец, очередной фрагмент может просто не поместиться на свободном месте во входном файле. По каждому из этих поводов в информационной строке появится сообщение с указанием точных координат и причин ошибки. После любой ошибки двух первых типов сборку исходного текста не надо повторять заново, так как все предшествующие фрагменты в старый файл уже выбраны. Надо воспользоваться директивой ДОБАВЬ и начать пополнение исходного текста только с ошибочного модуля.

Процедура смены входного и выходного файлов позволяет начать повторное редактирование только что отредактированного текста. Запускается она по нажатию клавиши ПД2 в состоянии «нижний регистр» и выполняется следующим образом. Сначала в отредактированный файл выталкивается содержимое экрана. Потом к информации, уже находящейся в новом файле, присоединяются оставшиеся строки из исходного файла и отредактированный файл объявляется входным. На базе описанной процедуры легко удалось реализовать возврат по отредактированному тексту на страницу назад (кнопочная директива ПФ2).

7.7. Основные процедуры коррекции исходного текста и их реализация

Исходный текст, находящийся в старом файле, можно подвергать любым изменениям — удалять строки, заменять строки или вставлять новые. При этом старый текст без какого-либо изменения сохраняется во входном файле, все исправления включаются только в выходной файл и в ту его часть, которую пользователь еще видит на экране дисплея.

Перемещениями по исходному и отредактированному текстам управляют два счетчика строк — IOLD и INEW. Первый из них хранит номер текущей строки старого файла; его значение высвечивается в информационной строке в виде ИСХ = NNNN. Содержимое текущей строки исходного текста всегда дублируется в просмотрной строке экрана. По мере выборки строк исходного текста значение счетчика IOLD увеличивается. Иногда его содержимое «замирает», так как не все выполняемые редактором процедуры требу-

ют продвижения счетчика. Пользователь имеет возможность вернуть содержимое счетчика IOLD назад от текущего положения или перешагнуть на несколько позиций вперед. Счетчик INEW хранит номер текущей строки в отредактированном файле, т. е. номер той строки, в которую была произведена последняя запись. Содержимое этой строки можно увидеть в самой верхней строке экрана. Значение счетчика INEW отображено в информационной строке в виде НОВ = ММММ. Его можно уменьшить, и тогда строки, «выталкиваемые» с экрана, будут затирать прежнее содержимое отредактированного файла.

7.7.1. Перепись строк исходного текста. Перепись строк из входного файла означает, что мы собираемся включить строки исходного текста в отредактированный файл без каких-либо изменений. Выполнять эту перепись можно различными способами — по одной строке (кнопочная директива ВВОД), по 20 строк (кнопочная директива ПФ5), по произвольному числу строк (до строки с заданным номером или до строки, содержащей указанный контекст). Последние три варианта реализуются наборными директивами, начинающимися с ключевого слова ПЕРЕПИШИ, или просто П. Например,

ПЕРЕПИШИ 240 — перепись 240 строк

П ДО 280 — перепись до строки с номером 280

ПЕРЕПИШИ ДО — перепись до строки, содержащей ранее указанный контекст

Контекст или поисковый образ — это произвольная цепочка символов $S_1 S_2 \dots S_k$, которая должна отсутствовать в переписываемых строках. Первая же строка исходного текста, включающая установленный образец, прекращает перепись и попадает только в просмотрную строку. Поиск образа в переписываемых строках может вестись глобально — по всей строке от 1-й позиции до 80-й или в заранее установленном диапазоне — от позиции N_1 до позиции N_2 . Поисковый образ и границы поиска устанавливаются с помощью наборной директивы ОБРАЗ:

ОБРАЗ(7,72) = SUBROUTINE

В процедурах переписи можно задать параметр N (число переписываемых строк или номер строки, до которой происходит перепись), существенно превышающий объем исходного текста. Редактор не считает это ошибкой и по исчерпанию строк входного файла завершит перепись.

Выполнение кнопочных директив переписи всегда связано с «прокачкой» информации через экран дисплея. Все остальные директивы производят перепись напрямую — через буферы в оперативной памяти. На экране фиксируется только конечное состояние

процедуры переписи — последние 20 строк, еще не записанные в отредактированный файл.

Технология присоединения очередной строки исходного текста к содержимому окна экрана в любом случае выполняется следующим образом. Содержимое просмотровой строки (СТРОКА24) переносится в наборную строку (СТРОКА22), а затем содержимое строк 2 — 22 сдвигается на одну строку вверх. Информация, попавшая в первую строку, проверяется на наличие в ее первой позиции символа отмены (\backslash). Если его там нет, то содержимое поля СТРОКА1 с помощью подпрограммы PUTNEW заносится в новый файл и содержимое счетчика INEW увеличивается на 1. В противном случае в первой строке восстанавливается содержимое последней строки нового файла, которое хранится в поле СТРОКА0. Затем из исходного файла считывается очередная строка и заносится на экран в просмотровую строку. Описанная процедура оформлена в виде внутренней подпрограммы REWRITE. Она в свою очередь пользуется сервисными подпрограммами редактора ВВЕРХ (подъем содержимого окна), PUTNEW (запись строки в новый файл) и GETOLD (чтение строки из старого файла).

Для переписи заданного числа строк аргумент директивы заносится в счетчик повторений (RPT) и подпрограмма REWRITE выполняется в цикле N раз. Перепись до строки с номером N выполняется почти так же. Предварительно вычисляется разность между текущим значением счетчика iOLD и значением N. Если эта величина не отрицательна, то она и определяет количество повторений цикла. Для переписи с поиском по образцу используется еще одна сервисная подпрограмма FIND, в задачу которой входит циклический посимвольный просмотр очередной строки старого файла с целью обнаружить поисковый образ. Входной информацией для подпрограммы FIND являются: адрес подпрограммы переписи (REWRITE) или пропуска (RDOLD) очередной строки старого файла, заданный в восьмом регистре; поле IMAGE, содержащее однобайтовый указатель длины образца и 80-байтовый образец; поле КОНСТIM, содержащее три константы для организации цикла поиска — адрес самой левой позиции, 1, адрес самой правой позиции. Текст подпрограммы FIND приведен ниже.

Подпрограмма поиска строки с заданным контекстом:

*** ПЕРЕПИСЬ ИЛИ ПРОПУСК ОЧЕРЕДНОЙ СТРОКИ

FIND	BALR	11,8	
	CLC	СТРОКА24(80),PUSTO	<i>не конец ли</i>
	BE	READY	<i>уход, если файл исчерпан</i>
	LM	3,5,КОНСТIM	<i>границы поиска</i>
	MVC	FIND1+1(1),IMAGE	<i>длина—1</i>

*** ЦИКЛ ПОИСКА ОБРАЗА ВНУТРИ СТРОКИ

```
FIND1      CLC      0(1,3),IMAGE+3
            BE       FIND2
            BXLE     3,4,FIND1
            B        FIND      за следующей строкой
```

*** ВЫЧИСЛЕНИЕ ОТНОСИТЕЛЬНОГО АДРЕСА МАРКЕРА

```
FIND2      LA       4,СТРОКА1
            SR       3,4
            ST       3,MARKER
```

*** ВОССТАНОВЛЕНИЕ ДИРЕКТИВЫ

```
MVC      СТРОКА22(80),DIRLINE
B        READY3
```

.....

*** ГРАНИЦЫ ПОИСКА; ИЗМЕНЯЮТСЯ ПО ДИРЕКТИВЕ ОБРАЗ

```
CONSTIM DC      A(СТРОКА21,1,СТРОКА21+79)
```

*** ПАРАМЕТРЫ ПОИСКА; ИЗМЕНЯЮТСЯ ПО ДИРЕКТИВЕ ОБРАЗ

```
IMAGE      DC      X'00'      длина образа—1
            DC      C'O==',    стандартная заго-
                                товка
            DC      X'E0',CL77', стандартный образец
```

Выход на метку READY означает, что ни в одной из считанных строк старого файла и присоединенных к новому файлу не был обнаружен заданный контекст. Если поисковый образ найден, то происходит возврат в точку READY3. Текст директивы в наборной строке при этом восстанавливается, так как процедура поиска обычно выполняется несколько раз подряд. Строка с найденным контекстом располагается в нижней части экрана (СТРОКА21), а маркер установлен под первым символом контекста.

7.7.2. Удаление строк исходного текста. Удаление (точнее сказать, пропуск) строк исходного текста — это такая процедура, когда из входного файла информацию читают, а в новый файл не передают. Происходит нечто вроде фиктивного чтения «в воздух». Содержимое отредактированного файла и его продолжение на экране дисплея никакому изменению не подвергаются.

Удаление строк по одной выполняется с помощью кнопочной директивы ПФ1. При этом очередная удаляемая строка видна в просмотровой строке экрана. Очередное нажатие клавиши ПФ1 приводит к тому, что счетчик строк IOLD продвигается на 1, и уже следующая строка исходного текста появляется в нижней части экрана.

При удалении заданного числа строк, выполняемого по директиве ПРОПУСТИ N, содержимое счетчика IOLD увеличивается сразу на N. Затем на экране отображается очередная после «удаленных» строка из входного файла. Выполнение директивы ПРОПУСТИ ДО N приводит к тому, что в счетчик IOLD заносится значение N и строка с указанным номером вызывается на экран. Директива ПРОПУСТИ ДО набираемая без параметров, организует пропуск строк исходного текста до тех пор, пока не встретится строка с требуемым контекстом. Выполняется эта процедура с помощью ранее описанной подпрограммы FIND.

Любую строку, находящуюся в окне экрана (строки 2 — 21), можно удалить путем набора в первой позиции этой строки символа отмены — обратной косой черты (код X'E0'). Строка с таким символом при выталкивании информации в отредактированный файл не пишется.

7.7.3. Замена строк исходного текста. Для замены содержимого просмотрной строки, т. е. очередной строки исходного текста, надо набрать замещающий текст в наборной строке и нажать клавишу ПФ1. Вновь набранный текст присоединится к окну экрана, а счетчик IOLD будет продвинут на 1. Можно произвести замену любой строки, которую мы видим в окне экрана. Для этого достаточно переместить маркер в нужную строку и произвести перенбор текста. При ближайшем прерывании редактор прочтает измененное содержимое окна и в дальнейшем будет оперировать с новым текстом.

7.7.4. Включение новых строк в отредактированный текст. В операциях замены каждая строка исходного текста заменяется на одну новую. В некоторых случаях количество замещающих строк может превышать число замещаемых или просто приходится включать новый фрагмент в отредактированный текст. Для выполнения этих процедур в состав редактора включены следующие возможности: включение новых строк путем их набора на клавиатуре дисплея, ввод заранее заготовленных фрагментов с внешних носителей (перфокарты, магнитная лента), вставка библиотечных фрагментов или фрагментов из входного файла.

Вставка через наборную строку напоминает печатание текста на машинке с той лишь разницей, что вместо перевода бумаги и возврата каретки нажимается клавиша ВВОД. Редактор принимает новую строку и присоединяет ее к содержимому окна экрана, поднимая его предварительно на одну строку вверх. Входной файл в этот момент может быть и пустым, так как после вставки новой строки редактор не производит обращения к исходному тексту. Так мы можем сформировать новый файл и без наличия исходного текста.

Только что введенную вставку можно многократно повторить с помощью директивы ПОВТОРИ N. В результате ее выполнения

в отредактированном тексте появится несколько $(N + 1)$ подряд идущих одинаковых строк. К такому приему часто прибегают для формирования шаблонов, упрощающих ввод таблиц с одинаковой структурой строк.

Процедуру формирования текста с помощью редактора можно ускорить за счет заранее заготовленной информации на перфокартах или магнитной ленте. В последнем случае данные должны быть расположены блоками по 80 байт. Каждая карта или запись на МЛ рассматривается редактором как очередная строка вводимого текста. Для ввода информации с внешнего носителя необходимо предварительно запросить нужное устройство и передать на пульт оператора ЭВМ просьбу с установкой соответствующего носителя. Выполняется это с помощью одной из директив: $ПК = \text{имя}$ или $МЛ = \text{имя}$. По таблице устройств TABUNIT, в которой редактор фиксирует сведения о предоставляемых ему устройствах, определяется, свободно ли устройство указанного типа. Если таковое обнаружено, то оно закрепляется за данным терминалом, а на операторскую консоль выдается сообщение «УСТАНОВИТЕ имя НА CUU» (CUU — физический номер устройства, на который оператор должен поместить носитель — колоду перфокарт или кассету МЛ с указанным именем). После ответа оператора на эту просьбу устройство считается доступным, и с него можно вводить данные. Для ввода набирается директива ЧК или ЧЛ с указанием числа N вводимых строк.

Редактор может прочитать и меньшее число строк, если данные будут досрочно исчерпаны или произойдет ввод строки, содержащей в первых позициях установленный признак конца. Обычно в роли признака конца данных выступают символы $/*$, но с помощью директивы $KB = S_1 S_2 \dots S_n$ можно установить любую ограничивающую цепочку знаков. Строка с признаком конца вводится и присоединяется к тексту на экране дисплея. При вводе небольших порций ($N \leq 20$) все введенные данные можно увидеть на экране. Если же вводимая порция имеет большую длину, то передача данных производится в отредактированный файл напрямую. После завершения ввода на экране высвечиваются только 20 последних строк.

Подобно тому как используются библиотечные фрагменты при формировании исходного текста, можно производить библиотечные вставки непосредственно в отредактированный файл. Выполняется такая процедура с помощью директивы ВСТАВЬ:

ВСТАВЬ имя₁(N_1, N_2), имя₂(M_1, M_2), ...

Если в списке директивы ВСТАВЬ присутствуют фрагменты без указания имени, то соответствующая информация извлекается из входного файла. Выборка фрагментов исходного текста может производиться в любой последовательности и не зависит от той точки, которая определяется счетчиком строк IO.

Все предыдущие процедуры вставки включали новые строки в разрез между нижней строкой окна экрана и очередным предложением исходного текста. В редакторе предусмотрена возможность вставки между любыми строками, уже находящимися в окне экрана. Для этого надо освободить нужную строку в окне за счет подъема вверх соответствующей части экрана. Делается это следующим образом. Маркер подводится в любую позицию строки, начиная с которой все должно быть сдвинуто вверх, и нажимается клавиша ПФЗ. В результате верхние строки, включая и «помеченную», поднимаются на одну строку, маркер переводится в первую позицию отмеченной строки, а сама строка чистится. Если набрать в ней какой-нибудь текст и снова нажать ПФЗ, то освободится место для ввода следующей вставки и т. д. Количество вставляемых таким образом строк ничем не ограничено.

7.7.5. Вывод отредактированного текста. Отредактированный текст включает все то, что находится в выходном файле и на окне экрана. Его можно записать в библиотеку, выдать на магнитную ленту или отпечатать на АЦПУ. Возможность использовать отредактированный текст в качестве исходного для повторной правки нами уже рассматривалась (кнопочная директива СМЕНА).

Для записи в библиотеку используется наборная директива ЗАПИШИ с единственным параметром — именем записываемого модуля. Если места в библиотеке хватает и в ней отсутствует модуль с таким же именем, то редактор сразу же выполнит процедуру записи. При нехватке места в информационной строке появится сообщение: «НОВЫЙ МОДУЛЬ СЛИШКОМ ВЕЛИК». Если в библиотеке обнаружен модуль с таким же именем, то редактор попросит у пользователя подтверждения: — «ПРЕЖНИЙ МОЖНО СТЕРЕТЬ?». И только по утвердительному ответу «ДА» будет выполнена заказанная запись. К проблеме захвата библиотеки на время записи модуля мы еще вернемся при обсуждении многотерминального варианта редактора.

Выдача отредактированного текста на МЛ может преследовать различные цели — временное хранение, передачу другим пользователям и т. п. Если МЛ уже закреплена за данным терминалом, то процедура вывода выполняется по директиве «ПЛ N_1, N_2 ». (Сокращение ПЛ появилось от сочетания «печатать на ленту»). Необязательные аргументы N_1 и N_2 определяют порядковые номера начальной и конечной строк выводимого фрагмента ($1 \leq N_1 \leq N_2 < 9999$). Если ни один из них не задан, то файл с отредактированным текстом выводится на ленту целиком. Аналогичным образом осуществляется и печать, выполняемая по директиве «ПЧ N_1, N_2 ». Если то или иное внешнее устройство за терминалом не закреплено, то в информационной строке появляется сообщение: «У ВАС НЕТ ЭТОГО УСТР-ВА». Для захвата устройства и передачи оператору ЭВМ просьбы уста-

новить нужную кассету МЛ используется директива «МЛ = имя» или «АЦПУ = имя». В последнем случае имя используется для идентификации выводимого текста.

7.8. Экранный сервис

Под этим названием объединены функции редактора, упрощающие набор и исправление информации, уже находящейся на экране. Некоторые из них, такие как удаление строк или вставка новых, мы уже обсуждали.

7.8.1. Нумерация строк в отредактированном тексте. Режим нумерации строк, поступающих в новый файл, устанавливается по директиве ИМЯ = $S_1S_2S_3S_4$. Если хотя бы один из символов S_i отличен от пробела, то заданное имя переносится в позиции 47 — 50 информационной строки. С этого момента каждая строка редактируемого текста, выталкиваемая в верхнюю строку экрана, снабжается номером. Этот номер состоит из имени $S_1S_2S_3S_4$, расположенного в позициях 73 — 76, и содержимого счетчика INEW.

Отключение режима нумерации производится путем задания пустого имени ИМЯ = . В начальной стадии сеанса режим нумерации отключен. В процессе сеанса нумерацию можно включать или отключать много раз. Если требуется изменить имя, вписываемое в позиции 73 — 76, то при установленном режиме нумерации можно повторить директиву включения, но уже с другим именем.

7.8.2. Разметка экрана и табуляция. Большая часть информации, которую приходится набирать с клавиатуры дисплея, имеет фиксированный формат строки. Это — либо тексты исходных программ на том или ином алгоритмическом языке, либо исходные данные с фиксированным расположением полей в строке и т. п. При таком наборе содержимое некоторых полей может оказаться пустым (отсутствует метка оператора, нет комментария и т. д.) или частично заполненным. Тогда для перехода в начало следующего поля этой же строки приходится несколько раз нажимать клавишу горизонтального смещения маркера или пробел. Ускорить эту процедуру можно следующим образом. Зафиксируем номера позиций, с которых начинаются выделенные поля. Отмеченные позиции редактор должен запомнить, а потом по нажатию специально выделенной функциональной клавиши быстро переводить маркер в ближайшую справа границу очередного поля. Такой механизм широко применяется на обыкновенных пишущих машинках для печати таблиц и носит название «табуляционного пропуска».

Для запоминания выделенных позиций в редакторе предусмотрена директива вида $P = T, N_1, N_2, \dots, N_k$. Имя директивы происходит от слова *разметка*, а буква Т справа от знака равенства ассоциируется со словом *таблица*. Числовые аргументы N_1, N_2, \dots, N_k задают

номера выделенных позиций. Их необходимо перечислять в порядке возрастания и задавать в диапазоне от 1 до 80. Приведенный формат директивы разметки является самым общим, так как он позволяет описать таблицы любой структуры. Но при наборе наиболее употребительных текстов — программ на алгоритмических языках — фиксируемые позиции предусмотрены форматом языка. Поэтому директива разметки допускает массу модификаций:

$R=A$	эквивалентно $R=T, 10, 16, 40$	(ассемблер)
$R=\Phi$ $R=F$	} эквивалентно $R=T, 7$	(фортран)
$R=\Pi$ $R=P$		
$R=K$ $R=C$	} эквивалентно $R=T, 8, 12$	(кобол)
$R=T$		
	эквивалентно $R=T, 1$	(произвольный текст)

Технология набора программ, например, на ассемблере такова. В начальный момент после разметки маркер находится в первой позиции наборной строки. В этой же позиции находится символ отмены (\backslash). Если у оператора есть метка, то набираем ее и нажимаем клавишу ПФ4 — маркер «перескакивает» в 10-ю позицию. Набираем шифр операции и снова нажимаем ПФ4 — маркер перемещается в 16-ю позицию. Набираем операнды, а дальше могут быть вариации. Если комментарий в набираемой строке нет, то можно нажать клавишу ВВОД. Тогда произойдет подъем содержимого окна, наборная строка очистится и маркер установится в ее начало. При отсутствии комментария можно вместо клавиши ВВОД нажать ПФ4. При этом маркер переместится в 40-ю позицию. Повторное нажатие ПФ4 заставит редактор принять набранную информацию и установить маркер в первой выделенной позиции следующей строки. В нашем случае это будет 10-я позиция наборной строки. Такая технология хороша, если у следующей строки отсутствует метка оператора. Если маркер находится в первой позиции наборной строки, а метки у оператора все равно нет, то мы нажимаем ПФ4. При этом символ отмены из первой позиции автоматически удаляется, а маркер смещается в 10-ю позицию. Аналогичным образом выполняется набор программ на любом другом языке.

Для просмотра действующей разметки используется директива R без параметров ($R =$). При ее выполнении в наборной строке появляется директива, эквивалентная установленной разметке. Кроме того, о разметке можно догадаться по содержимому позиций 38—45 информационной строки на экране. Там высвечивается одно из слов: АССЕМБЛР, ФОРТРАН, ПЛ/1, КОБОЛ, ТЕКСТ или ТАБЛИЦА.

Режим табуляции действует не только в пределах наборной строки, но и тогда, когда маркер находится в любой строке окна.

Нажатие клавиши ПФ4 вызывает перемещение маркера на начало ближайшего поля той строки, где находится маркер, или на границу первого поля следующей строки. Попытка проделать эту операцию в запретных строках редактором пресекается.

Программная реализация директив разметки и табуляции представляет определенный интерес. Для распознавания директив разметки в таблицу TABDIR включены все модификации ($P = A, P = \Phi, P = F, \dots$). Поэтому каждый вариант вызывает переход в разные точки блока обработки, но структура этих входов стереотипна. Например, для разметки под ассемблер вход в обрабатывающий блок выглядит следующим образом:

```
P#A  BAL    4,TAB1
      DC     FL1'71'
      DC     CL8'АСЕМБЛР'
      DC     CL9',10,16,40'
```

Приведенный фрагмент используется для подмены исходной директивы ($P = A$) ее эквивалентом ($P = T, 10, 16, 40$). Эквивалентная директива переносится на 80-байтное поле РАЗМЕТКА и по запросу пользователя ($P =$) высвечивается в наборной строке. Замена мнемонических директив разметки, удобных для пользователя, на единый формат упрощает последующую обработку директивы. Сводится она к выполнению трех следующих шагов: формированию байта для упора при сдвиге (BYTSHIFT), переносу индикатора разметки (в приведенном примере — символы АСЕМБЛР) в информационную строку и росписи 80-байтного словаря сдвига (TABSLW). Байт упора определяет номер самой правой позиции для размещения информации в соответствии с данной разметкой. Та часть информации, которая находится левее этого байта, может раздвигаться при вставке пропущенных знаков или сдвигаться при вычеркивании лишних символов. То, что находится правее байта упора, при указанных процедурах остается на своем месте. Для ассемблера упором при сдвигах является 71-я позиция, для других алгоритмических языков — 72-я, а для произвольных текстов и таблиц — 80-я.

Словарь TABSLW используется для определения ближайшей отмеченной позиции,¹ расположенной правее текущего положения маркера. В i -й байт этого словаря заносится граница ближайшего поля. Так, например, разметке $P = T, 10, 20, 32$ соответствует следующий словарь:

1	2		9	10	11		19	20	21		31	32		80
10	10	...	10	20	20	...	20	32	32	...	32	0	...	0

Нули в байтах TABSLW означают, что правее данного разряда выделенных позиций нет.

Перемещение маркера по нажатию клавиши ПФ4 производится следующим образом. По сигналу прерывания считывается адрес маркера, из которого делением на число 80 выделяются номер строки и номер позиции. По номеру позиции из словаря TABSLW выбирается указание для сдвига, формируется новый адрес маркера и копия экрана из оперативной памяти отображается на дисплей.

Еще одна модификация директивы разметки (P) обеспечивает появление на экране шкалы с нумерацией каждой пятой позиции строки. В первой позиции шкалы находится символ отмены, который предохраняет отредактированный файл от попадания в него шкалы по мере подъема информации на экране.

7.8.3. Сдвиг вправо и вставка пропущенных символов. Если пользователь обнаружит, что в строке, находящейся на экране, пропущены один или несколько символов, то дело можно поправить следующим образом. Маркер подводится в позицию, начиная с которой нужно что-то вставить. Затем в режиме «нижний регистр» нажимается клавиша ПД1. В результате текст, начиная с помеченной позиции, сдвигается вправо, а маркер остается на месте. После этого можно набрать недостающий символ. Если нужно вставить еще один символ, то снова нажимается клавиша ПД1. Освобождается следующая позиция и т. д. Следует обратить внимание на отличие программной реализации сдвига в редакторе от аппаратного режима вставки, предусмотренного в дисплейных станциях ЕС-7920. Программный сдвиг текста происходит без перехода через границу следующего размеченного поля. Символ, выдвигающийся из обрабатываемого поля, пропадает бесследно. Можно предложить два алгоритма для реализации сдвига вправо. В первом варианте сдвиг осуществляется с конца по одному байту: $A_{n-1} \rightarrow A_n$, $A_{n-2} \rightarrow A_{n-1}$ и т. д. Во втором алгоритме по команде MVC сдвигаемая часть строки переносится в рабочую область, а затем по следующей команде MVC возвращается на прежнее место со сдвигом на один байт. В любом случае после сдвига надо очистить байт, помеченный маркером. Несмотря на явные преимущества второго варианта (за 2 команды MVC сдвигаются 2K байтов по сравнению со сдвигом K байтов в цикле за K команд MVC), авторы редактора отдали предпочтение первому алгоритму из-за компактности реализации.

7.8.4. Сдвиг влево и вычеркивание лишних символов. Для сдвига текста влево с одновременным вычеркиванием помеченного символа поступают следующим образом. Маркер подводится к удаляемому символу и в положении «нижний регистр» нажимается клавиша ПД3. Помеченный символ пропадает, а все, что было правее маркера, сдвигается на одну позицию влево. Если экран был предварительно размечен, то сдвигаемая порция ограничивается

длиной текущего поля. Содержимое всех полей, расположенных правее, остается на месте. Для сдвига влево используется одна команда MVC, в которой формируются адрес начала сдвигаемого поля ($A_i \leftarrow A_{i+1}$) и количество сдвигаемых символов. Самый правый символ обрабатываемого поля замещается пробелом.

При наборе текстов программ правый и левый сдвиги не распространяются на позиции, расположенные правее байта упора. При наборе произвольного текста в операции сдвига участвуют все байты от маркера и до 80-й позиции включительно. Процедурами сдвига удобно пользоваться для выравнивания левых или правых границ ранее набранного текста.

Приведем фрагмент редактора, реализующий сдвиги влево (вход SHIFTL) и вправо (вход SHIFTR), а также сервисную программу обработки адреса маркера (GDEMARK).

SHIFTL	MVI	MASKA+1,X'F0'	<i>формирование переклю- чателя</i>
	MVC	REGIM(8),=C'ВПРАВО'	<i>индикация режима</i>
	B	SHIFT	
SHIFTR	MVI	MASKA+1,0	<i>формирование переклю- чателя</i>
	MVC	REGIM(8),=C'ВЛЕВО'	<i>индикация режима</i>
SHIFT	BAL	11,GDEMARK	<i>на обработку адре- са маркера</i>
	SR	4,4	
	IC	4,BYTSHIFT	<i>номер последнего сдвигаемого байта</i>
	ALR	7,4	
	LA	5,СТРОКА1+2(7)	<i>адрес последнего сдвигаемого байта</i>
	SR	4,6	<i>число сдвигаемых байтов</i>
MASKA	BC	0,B4PK	<i>переключатель</i>
*** ВСТАВКА			
	MVC	1(1,5),0(5)	<i>цикл сдвига хвоста вправо</i>
	BCTR	5,0	
	BCT	4,*-8	
	MVI	0(3),C' '	<i>пробел в отмечен- ной позиции</i>
	B	OUT7920	<i>на вывод экрана</i>
*** ВЫЧЕРКИВАНИЕ			
B4PK	BCTR	4,0	<i>длина сдвига.цепоч- ки—1</i>
	STC	4,*+5	<i>формирование ко- манды MVC</i>
	MVC	0(1,3),1(3)	
	B	OUT7920	

GDEMARK	L	6,MARKER	
	SRDA	6,32	
	LA	3,СТРОКА1(7)	<i>абсолютный адрес выделенной позиции</i>
	D	6,=F'80'	<i>6=N символа, 7=N строки</i>
	MH	7,=H'80'	<i>относительный ад- рес начала отме- ченной строки</i>
	LTR	7,7	<i>проба на 1-ю стро- ку</i>
	BCR	7,11	<i>выход, если—не 1-я</i>
	B	OSHI B2	<i>здесь сдвигать нельзя</i>

7.8.5. Сдвиг вверх и вставка новых строк. Если маркер подведен в любую позицию одной из строк окна экрана (строки 2—21) и нажата клавиша ПФЗ, то текст, находящийся выше маркера, сдвигается на одну строку вверх. Одновременно с этим помеченная строка очищается, за исключением первой позиции, в которую вписывается символ отмены. В эту же позицию переводится и маркер, с тем чтобы можно было начать набор вставляемой строки. Повторяя указанную процедуру многократно, можно включить произвольное число новых строк в любое место окна экрана. В некотором смысле этот режим эквивалентен перемещению наборной строки в одну из строк окна экрана и переключению функций клавиши ВВОД на клавишу ПФЗ. Программа раздвижки строк на экране имеет вид.

*** ПОДЪЕМ ЧАСТИ ЭКРАНА ОТ СТРОКИ, ПОМЕЧЕННОЙ МАРКЕРОМ

PФЗ	BAL	11,GDEMARK	<i>на обработку адреса маркера</i>
	CL	7,=F'1760'	
	BNL	OSHI B2	<i>сдвиг ниже 21-й стро- ки</i>
	MVC	REGIM(8),=C'ВВЕРХ'	<i>индикация режима</i>
	LA	3,СТРОКА0	
	LA	4,80	
	LA	5,0(3,7)	<i>адрес конца подъема</i>
SHIFTUP	MVC	0(80,3),80(3)	
	BXLE	3,4,SHIFTUP	
	MVC	0(80,3),PUSTO	<i>очистка отмеченной строки</i>
	ST	7,MARKER	<i>маркер в начале стро- ки</i>
	BAL	11,WRNEW	<i>на запись вытолкну- той строки</i>
	B	OUT7920	

7.9. Работа с внешними устройствами

Пользователи могут получать во временное распоряжение внешние устройства из числа выделенных редактору устройств ввода-вывода: устройство ввода с перфокарт, АЦПУ, магнитные ленты. После освобождения устройства одним пользователем редактор может предоставить его другому. Технология работы с внешними устройствами состоит из следующих шагов: запрос устройства и передача сообщения оператору о приведении устройства в рабочее состояние (обеспечение готовности, установка внешних носителей и т. п.), обмен информацией между ЭВМ и выделенным устройством, освобождение устройства.

Запрос и освобождение любого устройства АЦПУ, ПК, МЛ выполняется с помощью стереотипных директив вида «тип устройства = имя носителя». Имя носителя может содержать до восьми символов; оно участвует в передаваемой оператору просьбе о приведении нужного устройства в состояние готовности или в распечатке заголовка на АЦПУ. Пробелы вместо имени носителя соответствуют освобождению устройства. Освобождая магнитную ленту, редактор производит ее перемотку и разгрузку, с тем чтобы другие пользователи не смогли записать на эту кассету свою информацию.

Процедуры ввода (ЧК, ЧЛ) и вывода (ПЧ, ПЛ) рассматривались в предыдущих разделах. Среди других директив, связанных с работой внешних устройств, отметим формирование признака конца ввода (KB) и группу процедур управления магнитной лентой. Стандартным признаком конца ввода при чтении с перфокарт или МЛ редактор считает строку, содержащую символы /* в позициях 1 и 2. Однако этот признак можно заменить, набрав директиву KB, например KB = _ _ _ _ _ END. Такой строкой, как правило, завершается любая программная единица на фортране. Для просмотра действующего в данный момент признака конца ввода директива KB набирается без параметров (KB =). В отличие от поиска по образцу признак KB считается фиксированным в позициях 1 — K анализируемой строки. Здесь K — порядковый номер последнего отличного от пробела символа в признаке KB. Обмен со всеми внешними устройствами последовательного доступа в редакторе ведется построчно. Это позволило реализовать процедуры обращения к ВУ на физическом уровне с помощью простейших канальных программ, состоящих из одной-единственной команды CCW с формируемым кодом операции. Процедуры обслуживания устройств выделены во внешнюю подпрограмму с несколькими входами. Один из них (вход GETUNIT) используется для захвата устройства во временное пользование. Входную информацию для работы этого блока составляет имя «носителя», расположенное в одном из восьми-байтовых полей NAMEAZPU, NAMECARD или NAMEML, а также

условный номер заказываемого устройства (0 — ПК, 8 — АЦПУ, 16 — МЛ), указанный в одном из общих регистров. Подпрограмма GETUNIT просматривает таблицу устройств, предоставленных редактору, на предмет поиска свободного ресурса запрашиваемого типа. Если свободное устройство обнаружено, то в соответствующую строку таблицы заносится признак занятости и производится проверка готовности ресурса. Перед проверкой готовности ПК или МЛ на операторскую консоль передается сообщение с просьбой установить требуемый носитель на выделяемое устройство. После ответа оператора на устройство выдается канальная команда, наиболее соответствующая проверке готовности. Для устройства ввода с перфокарт — это команда «холостой ход» (КОП = 03), для МЛ — *перемотка в начало* (КОП = 07), для АЦПУ — *вывод одной строки* (КОП = 09) с текстом NAMEAZPU. Возврат из подпрограммы GETUNIT на команду с адресом A, следующую за оператором CALL, свидетельствует об успешном завершении операции по захвату устройства. При отсутствии свободного устройства или при невозможности привести его в состояние готовности (последнее определяется по ответу оператора) возврат осуществляется на команду с адресом A + 4.

Второй вход подпрограммы с меткой INOUT используется для реализаций процедур обмена и всех директив управления магнитной лентой. Шифр устройства (0, 8 или 16) задается в одном из общих регистров, а в качестве буфера ввода-вывода используется копия наборной строки (СТРОКА22). В состав директив управления МЛ включено пять операций: перемотка МЛ на заданное число записей — строк в прямом или обратном направлении (директивы \pm KART N), аналогичная перемотка на заданное число файлов (директива \pm N) и запись ленточных марок (директива TM N). Если значение N в этих директивах опущено, то предполагается N = 1. Фрагмент редактора, связанный с обработкой указанных директив, приведен ниже:

Обработка директив управления лентой

BSF	MVI	CCWML,47	КОП для возврата на N файлов
	B	SKOLKO	
FSF	MVI	CCWML,63	КОП для прогона на N файлов
	B	SKOLKO	
BSR	MVI	CCWML,39	КОП для возврата на N блоков
	B	SKOLKO	
FSR	MVI	CCWML,55	КОП для прогона на N блоков
	B	SKOLKO	

TM	MVI	CCWML,31	КОП для записи TM
	LA	8,16	шифр для обмена с МЛ
	B	SKOLKO+4	
SKOLKO	LA	8,24	шифр для управления МЛ
	LA	5,1	по умолчанию N=1
	BAL	11,OBPRB	обход пробелов
	B	TESTML	уход, если N не задано
	BAL	11,NOMER	на обработку N
	B	OSIB6	ошибка в задании N
	LR	5,0	рег.5=N
TESTML	CLC	NAMEML(8),PUSTO+1	заказана ли МЛ?
	BE	OSIB15	уход, если МЛ не заказана
	CALL	INOUT,(СТРОКА22,КАК)	
	CLI	КАК,0	как завершена операция?
	BE	READY	уход, если нормально
	B	OSIB7	сбой МЛ

Код возврата, характеризующий результат завершения процедуры, заносится в байт КАК.

Третий вход (FREEUNIT) используется при освобождении устройства. Исходная информация задается шифром освобождаемого устройства. Для устройства ввода с перфокарт и АЦПУ процедура освобождения сводится к удалению признака занятости из соответствующей строки таблицы устройств. Для магнитной ленты дополнительно производится разгрузка МЛ, чтобы защитить ее от других пользователей. При освобождении устройства редактор заносит пробелы в одно из полей — NAMECARD, NAMEAZPU или NAMEML.

7.10. Внутренние подпрограммы редактора

При разработке любой системы математического обеспечения очень важно продумать ее структуру, установить взаимосвязь между отдельными блоками, закрепить определенные регистры за наиболее важными указателями, выделить типовые процедуры, упрощающие последующее программирование. Большое значение имеет наглядность обозначений часто употребляемых полей, переменных и констант. Правда, последнее нередко вступает в противоречие с ограничениями ассемблера: длина меток не более восьми символов и запрет на употребление русских букв. Поэтому иногда приходится писать русские слова английскими буквами (ПУСТО → PUSTO, ПРОБА → PROBA, МЛ → ML и т. п.), а иногда прибегать к более кратким английским словам (UNIT, RUN, INOUT и т. п.).

В этом разделе мы более подробно остановимся на сервисных подпрограммах основного модуля редактора. Затрагивать аналогичные процедуры в модулях по обслуживанию библиотек, рабочих файлов и внешних устройств мы не будем, так как они в большей степени зависят от операционной системы.

Сначала о некоторых соглашениях и обозначениях, принятых в редакторе. Под копию экрана в оперативной памяти выделено поле в 2000 байт с разбиением на 80-байтные строки. Образу экрана соответствуют подполя с наименованиями СТРОКА1, СТРОКА2,, СТРОКА24. Предшествующий им 80-байтный массив СТРОКА0 используется для хранения предпоследней строки, вытолкнутой в новый файл. Функцию наборной строки выполняет массив СТРОКА22. Директива, поступающая от пользователя, копируется в 80-байтный массив DIRLINE. Это позволяет портить наборную строку в процессе обработки директивы, но в случае необходимости директива может быть восстановлена.

Обработка директивы в существенной мере связана с указателем, который последовательно сканирует символы директивы слева направо. Роль этого указателя выполняет первый регистр общего назначения. Перед входом в блок обработки директивы в нем хранится адрес первого байта строки DIRLINE, следующий за опознанным ключевым словом, т. е. в начале указатель «смотрит» либо на первый байт поля параметров директивы, либо на продолжение названия директивы, состоящее более чем из одного слова. В процессе обработки директивы адрес в указателе последовательно увеличивается.

Наверное, читатель обратил внимание, что в приводившихся фрагментах программ фигурировали цифровые номера общих регистров, а не их символьные обозначения. Это, скорее всего, недостаток программы, затрудняющий ее модификацию. Оправдать авторов можно лишь соображениями экономичности в записи команд и железной дисциплиной в использовании регистров. Помимо первого регистра постоянные функции закреплены за регистрами: 10 (база редактора), 11 (адрес возврата из внутренних подпрограмм первого уровня), 7 (адрес возврата из внутренних подпрограмм второго уровня), 12 (адрес текста сообщения), 13 (адрес поля пользователя в многотерминальном варианте). Остальные регистры используются в качестве рабочих. Для организации циклов чаще всего применяются тройки 1—2—3 или 3—4—5.

Теперь о константах. Вместо числовых литеральных констант в редакторе очень часто используются обозначения, очень сходные с записью литеральных констант. Вместо знака равенства в них применен символ #. Например, обозначение #F1 соответствует литеральной константе =F'1', метка C#ML соответствует константе DC C'ML' и т. п. Единства в расположении «решетки» (символ #)

здесь нет, но само обозначение достаточно просто расшифровывает константу. Отказ от литеральных констант обусловлен двумя причинами. Во-первых, ассемблер может расположить их за пределами досягаемости базового регистра, если не позаботиться об операторе LTORG. Во-вторых, литеральная константа не дает возможности воспользоваться ее частью. Например, по литеральным константам =F'80' и =H'80' транслятор заведет две разные константы, занимающие вместе 6 байт. А на самом деле можно было бы обойтись одной константой F # 80, так как H # 80 эквивалентно выражению F # 80 + 2. Список внутренних сервисных подпрограмм редактора с указанием их функций приведен в табл. 7.1. Тексты некоторых из них или их аналоги уже разбирались выше по ходу изложения. На других мы остановимся ниже.

Т а б л и ц а 7.1

Входная метка подпрограммы	Выполняемые функции
BBERPX	Подъем окна экрана на одну строку вверх и запись выталкиваемой строки в новый файл
RDOLD	Чтение очередной строки из старого файла
FIND	Перепись или пропуск строк исходного текста с поиском по образцу
CVDIOLD	Перевод указателя строк старого файла в символичный формат
CVDINEW	То же, для указателя строк в новом файле
OBPRB	Обход пробелов при обработке директивы
NOMER	Преобразование числа из символического представления в машинный формат
N1N2	Обработка границ диапазона
NEXTNAME	Обработка имени библиотечной программы или ее фрагмента
SETINDEX	Обработка границ фрагмента
GDEMARK	Преобразование адреса маркера
KAKGET	Анализ результата выборки из библиотеки

7.10.1. Выборка очередной строки исходного текста. Подпрограмма RDOLD осуществляет чтение очередной строки исходного текста с помощью внешней подпрограммы GETOLD, заносит ее в копию просмотровой строки и корректирует содержимое счетчика IOLD для отображения на экране. Исходную информацию для работы подпрограммы RDOLD составляет счетчик строк IOLD, в котором хранится номер текущей строки в формате целого числа, не превосходящего 9999. Если содержимое IOLD отлично от нуля, то входной файл не пуст. Попытка прочитать из него очередную строку может закончиться удачно (KEY = 0) или неудачно, так как предшествующая строка была последней и файл уже исчерпан.

В первом случае подпрограмма GETOLD увеличит на 1 содержимое счетчика IOLD. Чтобы отразить это изменение на экране, придется переводить целое двоичное число в его символьный эквивалент. Этот перевод выполняет заключительная часть программы с меткой CVDIOLD. Она может быть использована и в качестве самостоятельной подпрограммы. Полный текст подпрограммы RDOLD приведен ниже:

Чтение очередной строки из старого файла

RDOLD	MVC	СТРОКА24(80),PUSTO	<i>очистка строки</i>
	CLC	IOLD, #F0	<i>не пуст ли файл?</i>
	BCR	8, 11	<i>выход, если пуст</i>
	CALL	GETOLD, (СТРОКА24, IOLD, KEY)	<i>на n/n чтения</i>
	CLI	KEY, 0	<i>как прочитали?</i>
	BCR	7, 11	<i>выход, если файл исчерпан</i>
	L	0, IOLD	<i>счетчик строк</i>
	BCTR	0, 0	
CVDIOLD	CVD	0, IPACK	<i>перевод в P-формат</i>
	UNPK	IZON(5), IPACK+5(3)	<i>перевод в Z-формат</i>
	OI	IZON+4, X'F0'	<i>замена знака кодом зоны</i>
	MVC	OLDCOUNT(4), IZON+1	<i>запись в инф. строку</i>
	BR	11	
.....			
PUSTO	DC	X'E0', CL79'	
IPACK	DS	0D	
.....			

7.10.2. Выталкивание строки в новый файл. Подпрограмма ВВЕРХ осуществляет подъем содержимого экрана на одну строку вверх и с помощью внешней подпрограммы PUTNEW производит запись выталкиваемой строки в новый файл. Входную информацию для работы подпрограммы составляют массивы СТРОКА1, СТРОКА2, ..., СТРОКА22. Сначала организуется цикл по перемещению строк ($СТРОКА_{i+1} \rightarrow СТРОКА_i$), в котором по команде MVC за один такт смещаются сразу три строки (240 байт). Затем еще одна команда MVC используется для сдвига последней строки. Если новое содержимое первой строки (СТРОКА1) не содержит в первой позиции символа отмены, то оно выталкивается в новый файл с соответствующей коррекцией счетчика строк INEW. Выталкиваемая строка запоминается в поле СТРОКА0. Если в первую строку попадает предложение, содержащее в первой позиции символ \, то на ее место вызывается содержимое поля СТРОКА0, т. е. восстанавливается последняя строка нового файла, записанная туда на предыдущем шаге. В этом случае коррекция счетчика INEW не производится. Если новое значение счетчика INEW должно превысить максимально допустимое число строк в выходном файле, то вместо

выталкивания на экран выдается сообщение: «НОВЫЙ ФАЙЛ ПЕРЕПОЛНЕН».

В момент, предшествующий выталкиванию очередной строки в новый файл, производится проверка режима нумерации. Если имя (позиции 47—50 информационной строки) состоит из пробелов, то выталкиваемая строка переносится в новый файл без изменения. В противном случае символы $S_1S_2S_3S_4$ из указанных позиций информационной строки заносятся в позиции 73—76, а преобразованное в символьный вид значение счетчика INEW — в позиции 77—80 выталкиваемой строки.

Подпрограмма ВВЕРХ допускает обращение из внешних подпрограмм редактора, ее полный текст приводится ниже:

Подъем экрана и запись очередной строки в новый файл			
ВВЕРХ	LM	3,5,КОНСТ2	<i>подготовка цикла</i>
ВВЕРХ1	LA	2,СТРОКА0(3)	
	MVC	0(240,2),80(2)	<i>подъем трех строк</i>
	BXLE	3,4,ВВЕРХ1	
	MVC	240(80,2),320(2)	<i>подъем 22-й строки</i>
WRNEW	CLI	СТРОКА1,X'E0'	<i>надо ли выталкивать?</i>
	BNE	WRNEW1	<i>обход, если надо</i>
	MVC	СТРОКА1(80),	
		СТРОКА0	
	BR	11	
WRNEW1	CLC	INEW, IMAX	$INEW < или = IMAX?$
	BE	OSHI B3	<i>новый файл переполнен</i>
	CLC	REGIM1+9(4),	<i>нужна ли нумерация?</i>
		PUSTO+1	
	BE	WRNEW2	<i>обход, если не нужна</i>
	MVC	СТРОКА1+76(4),	<i>номер строки</i>
		NEWCOUNT	
	MVC	СТРОКА1+72(4),	<i>имя модуля</i>
		REGIM1+9	
WRNEW2	CALL	PUTNEW,	
		(СТРОКА1,	
		INEW)	
	L	0,INEW	
CVDINEW	CVD	0,IPACK	
	UNPK	IZON(5),	
		IPACK+5(3)	
	O1	IZON+4,X'F0'	
	MVC	NEWCOUNT(4),	
		IZON+1	
	BR	11	
.			
КОНСТ2	DC	A(...)	

7.10.3. Обход пробелов. Редактор предоставляет пользователю свободу в наборе директив: директиву можно набирать не обязательно с первой позиции наборной строки, параметры могут примыкать к ключевому слову вплотную или располагаться через несколько пробелов, лишние пробелы могут встречаться и между параметрами директивы. Подпрограмма обхода пробелов OBPRB:

OBPRB	LA	2,1
	LA	3,DIRLINE+79
OBPRB1	CLI	0(1),C''
	BNE	4(11)
	BXLE	1,2,OBPRB1
	BR	11

используется для упрощения анализа содержательной части директивы. Если указатель строки DIRLINE «смотрит» на пробел, то его перемещают вправо до первого символа, отличного от пробела. Если в неисследованной части строки DIRLINE такой символ еще существует, то возврат из подпрограммы OBPRB происходит по адресу 4(11). В противном случае возврат осуществляется на команду, следующую за обращением к подпрограмме, т. е. по адресу 0 (11).

7.10.4. Перевод чисел из символьного представления в машинный формат. Очень многие директивы редактора включают в качестве аргумента числовую информацию. Это и количество переписываемых или пропускаемых строк, число повторений, границы фрагментов программ и т. п. Для всех этих чисел характерно то, что они являются целыми и принадлежат диапазону [1, IMAX]. Значение IMAX определяется максимальной емкостью рабочих файлов, но для простоты редактор считает, что $IMAX \leq 9999$. Это означает, что в символьной записи числа может присутствовать не более четырех цифр. Поэтому и цикл обработки очередной цифры в подпрограмме NOMER повторяется не более четырех раз. Досрочную обработку числа завершает любой символ, не принадлежащий диапазону [0, 9]. Приведем текст подпрограммы NOMER:

Перевод числа из символьного представления в машинный формат

NOMER	LA	2,1	<i>шаг для цикла</i>
	LA	3,3(1)	<i>конечный адрес для цикла</i>
	SR	0,0	
	SR	4,4	
NOMER1	CLI	0(1),C'0'	<i>код символа=0</i>
	BM	NOMER#0	<i>уход: встретили не цифру</i>
	CLI	0(1),C'9'	<i>код символа=9</i>
	BH	NOMER#0	<i>уход: встретили не цифру</i>
	MVC	КАК(1),0(1)	<i>извлекли очередную цифру</i>

NOMER#0	NI	КАК,Х'0F'	<i>удалили код зоны</i>
	MH	0,=H'10'	<i>умножили на 10</i>
	IC	4,КАК	
	ALR	0,4	<i>прибавили очередную цифру</i>
	BXLE	1,2,NOMER1	
	LTR	0,0	<i>проверка N=0</i>
	BZ	0(11)	<i>выход с нулевым номером</i>
	C	0,IMAX	
	BL	4(11)	<i>выход с нормальным номером</i>
	L	0,IMAX	
	B	4(11)	<i>выход с максимальным номером</i>

По выходе из подпрограммы результат перевода расположен в нулевом регистре общего назначения. Если число удовлетворяет описанным выше требованиям, то возврат осуществляется по адресу 4(11), в противном случае — на команду, следующую за обращением к подпрограмме. Задание числа N, превосходящего значение IMAX, ошибкой не считается, так как в этом случае подпрограмма заменит результат перевода на IMAX. Это позволяет во многих директивах пользоваться большим числом, например 9999, не заботясь об истинных размерах рабочих файлов или длинах программ.

Приведем программу обработки границ диапазона

N1N2	MVC	N1,#F1	<i>если не задано, то N1=1</i>
	MVC	N2,IMAX	<i>если не задано, то N2=IMAX</i>
	MVC	ТЕКСТ19+	<i>заготовка текста сообщения</i>
		+14(9),0(1)	<i>на случай ошибки</i>
	BAL	11,NOMER	<i>попытка перевести N1</i>
	B	*+8	<i>обход, если N1=0 или не задано</i>
	ST	0,N1	<i>N1 отлично от 0</i>
	CLI	0(1),C','	<i>проба на запятую</i>
	BNE	0(7)	<i>выход, если N2 не задано</i>
	LA	1,1(1)	<i>обошли запятую</i>
	BAL	11,NOMER	<i>попытка перевести N2</i>
	B	*+8	<i>обход, если N2=0 или не задано</i>
	ST	0,N2	<i>N2 отлично от 0</i>
	CLC	N1,N2	<i>анализ границ диапазона</i>
	BNH	0(7)	<i>выход, если N1<или=N2</i>
	B	OSHIВ19	<i>ошибочно задан диапазон</i>

Входной информацией для подпрограммы NOMER является указатель строки с анализируемым текстом, который предположительно «смотрит» на первую цифру числа.

7.10.5. Обработка границ числового диапазона. Границы числового диапазона обычно задают минимальный (N1) и максимальный (N2) номера строк фрагментов, используемых при формировании

нии исходного или редактируемого текстов, а также при распечатке нового файла. Эти границы должны быть целочисленными, удовлетворяющими следующему неравенству:

$$1 \leq N1 \leq N2 \leq \text{IMAX}$$

Директивы, использующие граничный диапазон, позволяют не задавать любой из параметров или опускать оба одновременно. По умолчанию считается, что опущенному значению N1 соответствует 1, а опущенному значению N2 — IMAX.

Подпрограмма N1N2 использует для обработки каждой из границ вышеупомянутую подпрограмму NOMER. Поэтому адрес возврата здесь задается в седьмом регистре. Обнаружив ошибку в задании диапазона (например, $N1 > N2$), подпрограмма формирует текст сообщения об ошибке, добавляя к нему девять символов, начиная с текущей позиции обрабатываемой директивы. Результат своей работы подпрограмма заносит в ячейки с метками N1 и N2.

7.10.6. Техника вывода информационных сообщений. В редакторе запрограммирована выдача 23 различных сообщений, полный список которых приведен в табл. 7.2. Сообщение необходимо

Т а б л и ц а 7.2

Метка	Содержание сообщения
ТЕКСТ0	МОЖНО ПРОДОЛЖАТЬ РАБОТУ
ТЕКСТ1	НЕ ТА КЛАВИША. ПОВТОРИТЕ
ТЕКСТ2	ЗДЕСЬ СДВИГАТЬ НЕЛЬЗЯ
ТЕКСТ3	НОВЫЙ ФАЙЛ ПЕРЕПОЛНЕН
ТЕКСТ4	ИСХОД. ФАЙЛ ПЕРЕПОЛНЕН
ТЕКСТ5	СВОЙ ДИСПЛЕЯ. ПОВТОРИТЕ
ТЕКСТ6	ПРОВЕРЬТЕ ДИРЕКТИВУ
ТЕКСТ7	СВОЙ ВНЕШН. УСТРОЙСТВА
ТЕКСТ8	УСТР—ВО ЗАНЯТО ДРУГИМИ
ТЕКСТ9	УСТР—ВО ЗАНЯТО ВАМИ ЖЕ
ТЕКСТ10	НЕ НАШЛИ ИМЯ=
ТЕКСТ11	ТАКОЙ БИБЛИОТЕКИ НЕТ
ТЕКСТ12	ТАКОГО ДИСКА НЕТ
ТЕКСТ13	ПРЕЖНЮЮ МОЖНО СТЕРЕТЬ?
ТЕКСТ14	В БИБЛИОТЕКЕ НЕТ МЕСТА
ТЕКСТ15	У ВАС НЕТ ЭТОГО УСТР-ВА
ТЕКСТ16	НАШЛИ. ЧИСЛО ЕЕ КАРТ=
ТЕКСТ17	ОШИБОЧНО ИМЯ=
ТЕКСТ18	ОШИБ. ИНДЕКС=
ТЕКСТ19	ОШИБ. ДИАПАЗОН=
ТЕКСТ20	КАТАЛОГ НЕ ВЕСЬ. ДАЛЬШЕ?
ТЕКСТ21	КАРТЫ ИСЧЕРПАНЫ:/*, ТМ
ТЕКСТ22	НОВЫЙ ФАЙЛ ЕЩЕ ПУСТ
ТЕКСТ23	БИБ—КА ЗАНЯТА. ПОВТОРИТЕ

переслать в информационную строку (одна команда MVC) и передать управление в стандартное место для вывода на экран (метка OUT7920). Однако ситуация несколько осложняется тем, что для различных групп сообщений приходится выполнять кое-что еще. Например, чистить или не чистить наборную строку, сохранять положение маркера на экране или возвращать его в начало наборной строки. Реализация всех этих ситуаций порождала большое количество стереотипных фрагментов, различавшихся либо длиной пересылаемого текста, либо адресом перехода. Особенно угнетало обилие «длинных» команд MVC, которые надо было изменять при замене старого текста сообщения на более удачный. И тогда было найдено очень изящное решение, которое можно рекомендовать в качестве унифицированного средства при выводе сообщений в любой другой системе. Правда, сопряжено оно с некоторым увеличением числа машинных тактов. Основная идея заключается в следующем. Надо сделать все тексты одинаковой длины. Это позволит заменить многочисленные команды MVC одной-единственной командой. Но перед ее исполнением приходится формировать адрес пересылаемого текста. А чтобы понять, как это делается, достаточно взглянуть на фрагмент программы вывода сообщений:

*** ВЫВОД СООБЩЕНИЙ С ЧИСТКОЙ НАБОРНОЙ СТРОКИ * И НАЧАЛЬНОЙ УСТАНОВКОЙ МАРКЕРА

OSHIB22	LA	12,23(12)
OSHIB21	LA	12,5 * 23(12)
OSHIB16	LA	12,16 * 23(12)
	B	READY
OSHIB20	LA	12,7 * 23(12)
	MVC	СТРОКА22(80), PUSTO
OSHIB13	LA	12,13 * 23(12)
	MVC	MARKER,F# 1680
	B	READY4

*** ВЫВОД БЕЗ ЧИСТКИ С ВОЗВРАТОМ МАРКЕРА

OSHIB23	LA	12,4 * 23(12)
OSHIB19	LA	12,23(12)
OSHIB18	LA	12,23(12)
.		
OSHIB1	LA	12,23(12)
	B	READY2

*** ВЫВОД БЕЗ ЧИСТКИ И БЕЗ ПЕРЕМЕЩЕНИЯ МАРКЕРА

OSHIB2	LA	12,2 * 23(12)
	B	READY3

READY0	LA	12,PUSTO+1	для чистки поля MESSAGE
READY	EQU	*	
READY1	MVC	СТРОКА22(80), PUSTO	чистка наборной строки
READY2	MVC	MARKER,F #1680	установка маркера
READY3	MVI	DUBL,0	сброс управляющего байта
READY4	MVC	MESSAGE(23),0(12)	текст сообщения

*** ВЫВОД НА ДИСПЛЕЙ КОПИИ ЭКРАНА

OUT7920	LA	12,ТЕКСТО	восстановление базы сообщений
	CALL	WRDISP,(СТРОКА1,1,24,MARKER)	

[.]

Перед выполнением любой директивы в 12-й регистр заносится адрес наиболее распространенного сообщения **МОЖНО ПРОДОЛЖАТЬ**. Оно имеет метку **ТЕКСТО** и свидетельствует о нормальном завершении директивы. Если из блока обработки директивы управление передано на метку **READY**, то содержимое 12-го регистра не изменено, и после чистки наборной строки пользователь увидит обычное приглашение редактора. Если же управление передано, например, на метку **OSHIB6**, то цепочка команд **LA** приведет в конечном итоге к тому, что в 12-м регистре окажется адрес константы с меткой **ТЕКСТ6**. Для этого константы с текстами сообщений должны быть упорядочены по возрастанию их номеров: **ТЕКСТО**, **ТЕКСТ1**, **ТЕКСТ2**, . . .

7.11. Основные проблемы многотерминального обслуживания

При разработке многотерминальной системы возникает масса вопросов. Разрешить ли прерывать работу любой программной единицы в любом месте для переключения на обслуживание другого пользователя? Допустить ли эту процедуру только в фиксированных точках? Как избежать тиражирования стереотипных программ? Как свести к минимуму объем работы по переключению с одного терминала на другой? По какому алгоритму обслуживать пользователей, одновременно выдавших свои заявки? Как повысить реактивность системы? И т. д. и т. п. Видимо, однозначных ответов на поставленные вопросы не существует. Некоторые из перечисленных проблем являются противоречивыми, необходимо считаться со спецификой функционирования системы, с ограничениями на ресурсы и т. п. Чтобы не увести читателя в сторону общими рассуждениями, постараемся описать решения, принятые авторами при разработке многотерминального редактора **EDIT7920**. Возможно, что существуют и более эффективные реализации, но по основным характери-

стикам (объем используемой памяти, быстроедействие) редактор EDIT7920 не уступает известным нам системам.

Первое, на что необходимо обратить внимание, — это специфика работы редактора. Каждый шаг, инициируемый директивой пользователя, порождает цепочку последовательных обменов: чтение с экрана дисплея — одна или несколько процедур обращения к внешней памяти — вывод на экран дисплея. [В промежутках между обменами редактор выполняет небольшое число машинных команд — порядка нескольких десятков. Поэтому все операции обмена вынесены за пределы редактора во внешние подпрограммы. Сформировав заявку для работы внешней процедуры, редактор готов переключиться на обслуживание другого пользователя. Так естественным образом определяются точки возможных прерываний в редакторе.

Работа любой внешней подпрограммы сводится к выполнению той или иной последовательности операций обмена с различными внешними устройствами. А любой обмен в свою очередь реализуется в три этапа. На первом — производится подготовительная работа, связанная с анализом и предварительной обработкой параметров, с формированием программы канала и т. п. Второй этап связан с запуском программы канала и временным прекращением работы внешней подпрограммы. После физического завершения процедуры обмена производится анализ результатов обмена с периферийным устройством, и работа внешней подпрограммы может быть продолжена. В связи с тем что программа канала может выполняться параллельно с работой центрального процессора, возникает возможность организовать одновременное обслуживание заявок, поступивших с разных терминалов. Для этой цели как нельзя лучше подходит механизм подзадач, реализованный во всех версиях операционных систем ЕС ЭВМ. Подзадача закрепляется за каждым потенциальным пользователем редактора. Она активизируется каждый раз, когда очередная директива пользователя порождает заявку на обмен, точнее — на втором этапе выполнения процедуры обмена. Запустив соответствующую программу канала, подзадача переходит в режим ожидания. Впоследствии она будет переведена в активное состояние после завершения канальной программы. Теперь в функции подзадачи входит дособработка (анализ байтов состояния, передача необходимых сообщений оператору и т. п.). Перед возвратом управления во внешнюю подпрограмму подзадача снова переводится в режим ожидания. Переключение на обслуживание другого пользователя можно произвести именно в тот момент, когда запущенная подзадача обмена перешла в режим ожидания. Для того чтобы не создавать больших временных задержек, обмен с внешними устройствами производится небольшими порциями, а повышению оперативности обмена с рабочими файлами содействуют буферы емкостью по 1680 байт.

Очень важным моментом для организации многотерминального обслуживания является вынос всей информации о текущем состоянии редактора в специальную область пользователя — USAREA. Количество таких областей совпадает с числом потенциальных пользователей и устанавливается при запуске редактора. В область USAREA выносятся вся переменная информация, формируемая редактором и внешними подпрограммами при обработке заказа данного пользователя. Адрес области USAREA, соответствующий обслуживаемому в данный момент пользователю, хранится в 13-м регистре. Поэтому USAREA начинается с области сохранения SAVEDIT, на которой операционная система запоминает состояние редактора в момент появления сигнала прерывания. Вслед за этим полем располагается еще несколько областей сохранения, используемых внешними подпрограммами разного уровня и исполнителями обмена. В область USAREA вынесены копия экрана, буферы для обмена с новым и старым файлами, все канальные программы, рабочие поля и указатели. Общий объем области пользователя составляет 1D10₁₆ байтов.

Все модули редактора построены таким образом, что вся текущая информация расположена либо в соответствующих полях области USAREA, либо в общих регистрах. Поэтому переключение на обслуживание другого пользователя связано с очень небольшим числом тактов. Надо сохранить текущее состояние общих регистров на одной из областей сохранения в текущей USAREA, перезагрузить адрес новой области USAREA, восстановить по ней содержание регистров и продолжить работу с прерванного места. Все эти переключения сосредоточены в одном модуле — в мониторе исполнителей обмена. Остальные программные модули редактора о переключениях и не подозревают. Единственная забота внешних подпрограмм состоит в том, чтобы при входе запоминать содержимое общих регистров в той или иной области USAREA. Выделить единую область сохранения для этой цели не удалось из-за возможных цепочек обращения из одной внешней подпрограммы в другую. Взаимодействие модулей редактора и использование различных областей сохранения (SAVEDIT, SAVCALL, SAVEDA) отражены на рис. 7.13.

7.11.1. Управление подзадачами обмена. Подзадачи обмена создаются при начальном запуске редактора. Количество их совпадает с числом дисплеев, предоставляемых редактору. На каждую подзадачу заводится и настраивается своя область USAREA. Порядок создания подзадач определяется последовательностью назначений дисплеев в задании на запуск редактора. В момент создания очередной подзадачи ей предоставляются область сохранения SAV7920 и блок управления подзадачами ECBWAIT, расположенные в массиве USAREA. Сразу же после возникновения подзадачи про-

цессор переключается на ее выполнение, так как подзадача обладает более высоким приоритетом, чем породивший ее модуль.

Начальный этап работы подзадачи состоит в обращении к операционной системе с целью предупредить ее о наличии у подзадачи

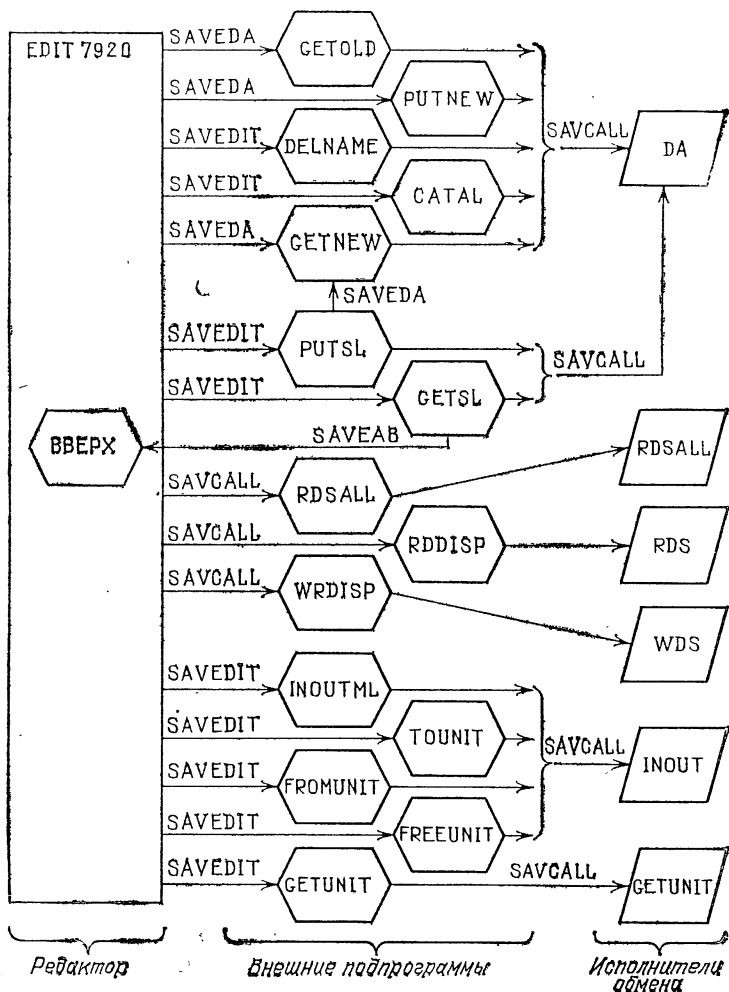


Рис. 7.3. Схема взаимодействия модулей редактора

собственной программы обработки сигнала ВНИМАНИЕ от приписанного к ней дисплея. Далее маршрут выполнения подзадачи проходит через основной блок редактора. На каких-то отрезках времени процессор будет занят выполнением активной подзадачи

т. е. будет обслуживать связанного с ней пользователя. В момент запуска канальной программы, выполняющей заказ пользователя, подзадача будет переводиться в режим ожидания. Для ее последующего пробуждения должно наступить событие, синхронизируемое программным образом через блок управления ECBPOST.

Таким образом, спустя некоторое время после порождения первой подзадачи она будет переведена в режим ожидания. А процессор возвратится в начальный блок по запуску редактора и продолжит выполнение работы по запуску следующей подзадачи. Так будет продолжаться до тех пор, пока не будут порождены подзадачи для всех потенциальных пользователей. И каждая из них будет находиться в режиме ожидания своего события.

Следует обратить внимание на специфику работы подзадач в редакторе. Дело в том, что подзадачи зачастую ассоциируются с параллельно выполняющимися программами. Это на самом деле так. Но многие программисты делают отсюда вывод, что разным подзадачам должны соответствовать разные программы, расположенные в непересекающихся участках оперативной памяти. А вот это как раз и не обязательно. Работа редактора напоминает движение по кольцевому маршруту. Начало движения совпадает с поступлением очередной директивы пользователя. Затем траектория движения определяется технологией обработки конкретной директивы. Подзадача напоминает бегуна, перемещающегося с эстафетной палочкой по этой траектории. В некоторых точках основного кольца бегун выбегает на примыкающие к нему вспомогательные кольца — внешние подпрограммы. Пробежав часть пути по маршруту подпрограммы, бегун попадает в зону отдыха — область действия канальной программы. Таких зон отдыха на вспомогательном кольце может оказаться несколько. Спустя некоторое время директива обработана, и бегун оказывается в точке своего старта. Представим себе, что два пользователя одновременно выдали одинаковые директивы. Так как процессор у ЭВМ один, то первым стартует бегун, связанный с терминалом, имеющим наименьший логический номер. Добежав до ближайшей зоны отдыха, он останавливается, и в этот момент по тому же маршруту стартует второй бегун. Выполнение канальной программы, связанной с обслуживанием первого бегуна, может затянуться. Поэтому второй бегун за это время тоже успеет попасть в зону отдыха. И чтобы они не мешали друг другу, у каждого из них в зоне отдыха должна быть своя скамеечка. Именно поэтому канальные программы, соответствующие разным подзадачам, формируются не в теле внешней подпрограммы, а на определенном поле в области USAREA. Таким образом, тела всех подзадач — это одна и та же программа с очень небольшими участками, тиражируемыми по количеству пользователей. Количество всевозможных кольцевых маршрутов в этой программе совпадает с числом различ-

ных директив, обрабатываемых редактором. А число бегунов-подзадач, находящихся в разных точках этой трассы, равно количеству обслуживаемых терминалов. В каждый момент времени может перемещаться только один бегун, все остальные — отдыхают.

7.11.2. Очередь заявок на обмены. Очередное обращение редактора к одной из внешних подпрограмм по поводу обмена с дисплеем, с библиотекой или другими периферийными устройствами в конечном итоге приводит к попаданию в монитор обмена. Монитор порождает заявку на обмен, ставит ее в очередь и пробуждает соответствующую подзадачу.

Очередь заявок строится по наиболее демократическому принципу — в порядке поступления заявок. Она размещается в таблице TABЕСВ и имеет следующую структуру:

X'00'	A(ЕСBWAIT _{i₁})
X'00'	A(ЕСBWAIT _{i₂})
.
X'00'	A(ЕСBWAIT _{i_k})
X'FF'	

Фактически эта таблица соответствует очереди заказов на обмены, представленных словами ЕСBWAIT_i. Сами заказы располагаются на полях USAREA и содержат следующую информацию. Два старших байта ЕСBWAIT хранят условный номер терминала 0, 4, 8,, 4 (K—1). Третий байт — признак состояния обмена (ПСО). Он принимает значения: X'00', если подзадача обмена запущена и редактор ждет завершения работы исполнителя обмена; X'80', если исполнитель обмена успешно завершил свою работу; X'C0', если произошло аварийное завершение работы исполнителя обмена.

7.11.3. Запуск подзадачи обмена. Как только в очереди TABЕСВ появилась новая заявка со ссылкой на управляющий блок ЕСBWAIT_i, то монитор обмена записывает номер нужного исполнителя обмена в два старших байта управляющего блока ЕСBPOST_i и по макрокоманде POST снимает ожидание с ждущей подзадачи. Так как последняя обладает более высоким приоритетом, чем породивший ее монитор, то процессор переключается на выполнение подзадачи. На этом этапе функция подзадачи заключается только в том, чтобы передать управление соответствующему

исполнителю обмена. Программа исполнителя обмена является как бы продолжением тела подзадачи. К этому моменту на поле USAREA уже сформирована соответствующая канальная программа. Так что исполнителю остается только запустить эту программу и по макрокоманде WAIT ждать физического завершения заказанной процедуры обмена. Одновременно с этим в режим ожидания переходит и подзадача, запустившая исполнитель.

Последовательность работы разных исполнителей индивидуальная. Исполнитель может включать один или несколько шагов, связанных с выполнением различных канальных программ. Он имеет право захватывать устройства во временное пользование, например, для передачи оператору сообщения о подготовке того или иного внешнего устройства, для временного блокирования записи в библиотеку и т. п.

Если исполнитель дорабатывает до конца, то возможны две ситуации: либо обмен завершен удачно и в байт ПСО операционная система занесла код X'80', либо в процессе обмена с дисплеем обнаружена неустраняемая ошибка, в результате которой оператор ЭВМ принял решение исключить данный дисплей из сферы обслуживания. Этот факт регистрируется редактором в таблице пользователей, а в байт ПСО тоже заносится код X'80'. Наконец, возможна еще одна ситуация, при которой операционная система не дает исполнителю обмена доработать до конца. Это — аварийное завершение операции обмена, при которой операционная система заносит в байт ПСО код X'C0' и уничтожает соответствующую подзадачу.

7.11.4. Исключение обработанной заявки из очереди. После нормального или аварийного завершения подзадачи обмена управление передается монитору обмена. Здесь происходит просмотр очереди TABESB в порядке поступления заявок. Как только обнаруживается «удовлетворенная» заявка (код X'80' или X'C0' в байте ПСО), то ее удаляют из очереди. В случае аварийного завершения обмена монитор восстанавливает уничтоженную подзадачу. Затем в управляющем слове ECBWAIT гасится байт ПСО и управление возвращается в ту точку редактора, из которой была порождена заявка на обмен.¹

Процессор может попасть в монитор обмена и в такой ситуации, когда ни одна из подзадач свою работу еще не завершила. Например, все канальные программы в затребованных исполнителях запущены, но ни одна операция еще не закончилась, т. е. все активизированные подзадачи находятся в режиме ожидания. В этом случае процессор вынужден переключиться на главную задачу, т. е. на монитор обмена. Просмотрев очередь заявок и убедившись, что во всех ждущих блоках ECBWAIT находятся только нулевые байты ПСО, монитор обмена выдает макрокоманду WAITM, т. е. переводит в состояние ожидания весь раздел, отведенный редактору. Из этого

состояния раздел может вывести только прерывание, связанное с нормальным или аварийным завершением работы какого-либо исполнителя.

Может оказаться и так, что во время просмотра очереди успеют выполняться еще одна или несколько заявок на обмен. Тогда монитор обработает первую из удовлетворенных заявок и возвратит управление в соответствующую точку вызова. Остальные уже удовлетворенные заявки останутся в очереди ТАВЕСВ до очередного попадания в монитор обмена.

7.11.5. Создание начальной очереди. При запуске редактора блок начальной настройки формирует в ТАВЕСВ фиктивную очередь удовлетворенных заявок. Их количество совпадает с числом терминалов, предоставленных редактору. Каждой образованной подзадаче в качестве адреса возврата назначается точка входа в редактор. Поэтому, когда монитор обмена начнет просматривать очередь, он должен будет исключить первую фиктивно удовлетворенную заявку и передать управление в начало редактора. При ближайшей попытке прочитать содержимое экрана первого дисплея мы снова попадем в монитор обмена, и вместо предыдущей заявки в хвосте очереди появится первая настоящая заявка на работу исполнителя RDSALL. Затем из очереди будет исключена начальная фиктивная заявка ко второму терминалу и т. д.

7.11.6. Защита ресурсов. Поясним, о какой защите ресурсов может идти речь в редакторе. Во-первых, это внешние устройства — АЦПУ, МЛ, устройства ввода с перфокарт, выделяемые редактору. При дефиците внешних устройств совершенно очевидно, что каждому пользователю предоставить по устройству нельзя. И они могут воспользоваться этими ресурсами по очереди. Выше уже упоминалось, что в редакторе ведется учет занятости внешних устройств, базирующийся на таблице TABUNIT. Во-вторых, приходится защищать библиотеку с исходными текстами на тот период, пока в нее происходит запись очередного модуля. Достигается это также табличными средствами. Имя библиотеки, в которую производится запись, заносится в таблицу TABBIV. Оно будет удалено из таблицы после выполнения директивы. Количество строк таблицы TABBIV совпадает с максимальным числом возможных пользователей. Если все они работают с разными библиотеками и одновременно производят запись своих модулей, то особых проблем не возникает. Запись каждого модуля производится небольшими порциями. Поэтому пользователи не очень задерживают друг друга. Чтение разных модулей из одной библиотеки, выполняемое по заказам нескольких пользователей, и даже чтение одного и того же модуля разными пользователями тоже не вызывают каких-то проблем. Может быть, в таких ситуациях имело бы смысл экономить на чтении оглавления или фрагментов модуля. Но редактор реализует

такие процедуры каждый раз заново, функционируя в составе последовательно выполняющихся независимых подзадач. Другое дело — запись в библиотеку. Во время этой процедуры изменяется оглавление библиотеки, может измениться ее длина. Поэтому совместить несколько процедур записи или разрешить чтение во время записи в библиотеку довольно сложно. И в редакторе принято самое простое решение — пока в библиотеку что-то пишут, она считается занятой. В это время любые другие процедуры, даже просмотр оглавления запрещены. Пользователь, получивший сообщение о том, что библиотека временно занята, может повторить свой заказ без перепабора директивы. Время ожидания в таких ситуациях не так уж и велико — запись 1000 строк в библиотеку занимает порядка 15—20 с.

ГЛАВА 8

ДИАЛоговые системы коллективного доступа

В связи с массовым выпуском терминального оборудования, в особенности локальных дисплейных станций, идея коллективного использования ресурсов ЭВМ (мультидоступа, коллективного доступа, разделения времени) стала особенно привлекательной прежде всего для программистов, для которых оперативное взаимодействие с машиной, утраченное в период применения чисто пакетной обработки, имеет первостепенное значение. С любым из терминов «коллективный доступ», «разделение времени» и т. п. связан один и тот же метод использования вычислительной машины, при котором каждый пользователь получает на время сеанса в монопольное использование терминал, а имеющаяся в наличии вычислительная мощность распределяется между всеми пользователями. Программные реализации этого метода получили название «систем разделения времени», а позже в связи с появлением вполне конкретной системы, имеющей название «Система разделения времени» (СРВ), — диалоговых систем коллективного доступа или диалоговых систем коллективного пользования.

Поскольку все *диалоговые системы коллективного доступа* (ДСКД), о которых пойдет речь в данной главе, построены на базе пакетной операционной системы ОС ЕС и являются системами, расширяющими ее возможности, то естественно, что они не исключают одновременного функционирования на ЭВМ обоих режимов (пакетного и разделения времени). Наоборот, так как функционирование любой ДСКД, как правило, осуществляется на фоне пакетной обработки, а сама ДСКД с точки зрения ОС ЕС — обычное пакетное задание, то при разработке таких систем важное значение придается созданию гибкого интерфейса между пакетной средой ОС ЕС и средой коллективного доступа.

Мы рассмотрим три ДСКД. Это — диалоговая система JEC (разработка ВЦ Ленинградского государственного университета им. А. А. Жданова [14]), диалоговые системы коллективного доступа PRIMUS (Московский инженерно-физический институт [9]) и БЕС7920 (Институт технической кибернетики АН БССР и Белорусский государственный институт народного хозяйства им. В. В. Куйбышева [8]). Выбор только этих трех систем связан со

следующими соображениями. Во-первых, эти системы уже эксплуатируются в десятках вычислительных центров и имеют вполне реальных разработчиков, которые их поддерживают и развивают. Во-вторых, они, имея достаточно близкие функциональные возможности, представляют определенный интерес с точки зрения специфики их реализации. И наконец, каждая из них, используя в качестве терминалов локальные дисплейные комплексы ЕС-7920, предлагает свою экранную технологию обработки данных и взаимодействия пользователя с системой.

8.1. Основные принципы построения систем

В данном разделе мы рассмотрим общие наиболее характерные проблемы, с которыми сталкиваются разработчики ДСКД, и покажем, как некоторые из них решаются в выбранных нами системах.

8.1.1. Принципы построения программного обеспечения. Одна из самых острых проблем, возникающих при реализации программного обеспечения (ПО) любой системы (не обязательно ДСКД), — это минимизация ее потребности в оперативной памяти (ОП). Поскольку еще до недавнего времени минимальный объем ОП, которой комплектовались машины Единой серии — 512К байтов, то для ДСКД критической верхней границей ее собственных затрат ОП (учитывая потребности ОС ЕС в ОП) считается величина 150 — 180 К. Потребности системы в большем объеме памяти ограничивают круг пакетных заданий, которые могут выполняться в фоновом режиме, и делают применение ДСКД малоэффективным.

Усилия разработчиков ДСКД, связанные с ограничением затрат ОП на нужды системы, идут в основном по следующим направлениям.

Прежде всего — это выбор оптимальной структуры ПО системы. В рассматриваемых системах выбор сделан в пользу параллельной динамической структуры с постоянно резидентным ядром системы и создаваемыми по мере необходимости подзадачами, обслуживающими активные процессы системы. В рамках подзадач, как правило, работают так называемые функциональные программы, динамически загружаемые в ОП для выполнения заданных функций и удаляемые из памяти, когда нужда в них отпадает. Наиболее общей является схема, по которой с подключением к системе нового пользователя для него создается отдельная подзадача, которая обслуживает его в течение всего сеанса работы, а как только система принимает от пользователя сигнал об окончании работы, подзадача уничтожается и все выделенные ей ресурсы возвращаются системе.

По такой схеме ядру системы обычно придаются функции планировщика подзадач пользователей и терминального монитора системы. Сюда же обычно включают различные общесистемные

управляющие таблицы и блоки, а также отдельные наиболее часто используемые модули. Остальное ПО системы, как правило, выносится на функциональный (прикладной) уровень. С целью минимизации размеров ядра оно обычно программируется на ассемблере.

Важное значение имеет выбор метода доступа к терминалам. Для ДСКД, ориентированных на локальные дисплеи, наиболее оптимальным (с точки зрения затрат ОП) является доступ на физическом уровне, однако при этом потребуются дополнительные усилия по программированию различных процедур, обеспечиваемых методами доступа более высокого уровня. Для обслуживания локальных дисплеев не приемлем и метод ОТМД в силу его потребностей в ОП (хотя для систем, не столь критичных к размеру ядра, некоторые из его возможностей могут оказаться весьма полезными). Из этих соображений обычно для обслуживания ЕС-7920 (локальных) берется БТМД.

В обсуждаемых ДСКД размер ядра колеблется в пределах от 40 до 60 К байт, язык программирования — ассемблер, метод доступа к ЕС-7920 — БТМД. При разработке системы основной путь для уменьшения общих затрат ОП — это разработка реентерабельных программ, благодаря чему одна копия программы в ОП используется для параллельного обслуживания всех подзадач, нуждающихся в ней одновременно.

Однако наличие большого количества мелких реентерабельных программ с недостаточно четкой структуризацией их функций может привести к обратным результатам, а, кроме того, за счет частых вызовов программы с внешних носителей время реакции системы на запросы пользователей может оказаться неприемлемым. В этом случае необходимо разумное сочетание между размером загрузочного модуля и набором функций, которые он должен выполнять. (В приложении приводится ряд практических рекомендаций по разработке реентерабельного ПО, выработанных в процессе проектирования системы БЕС7920.)

В рассматриваемых системах большая часть ПО (функциональные и обслуживающие модули) реализована в виде реентерабельных программ, а минимальные затраты ОП (в расчете на четыре активных дисплея) колеблются в пределах 100—130 К байт.

8.1.2. Распределение и защита ресурсов. Очередной проблемой, решаемой в ДСКД, является распределение ресурсов и обеспечение их сохранности (прежде всего это касается данных). Здесь возникает много различных вопросов. Как обеспечить совместное использование различными пользователями ДСКД одних и тех же наборов данных и устройств и как при этом исключить возможность испортить какой-либо из наборов данных? Как обеспечить совместное использование одних и тех же наборов данных в среде пакетной обработки и пользователями ДСКД? Все ли функции системы (ОС

ЕС и ДСКД) должны быть доступны любому пользователю или следует разграничить доступ пользователей к ним? И т. д. и т. п.

На некоторые вопросы отвечает сама ОС ЕС, предоставляя соответствующие стандартные средства; для решения других необходимо встраивать в ДСКД соответствующие программные средства.

В обсуждаемых ДСКД механизм распределения и защиты ресурсов (кроме ОП) реализован на двух уровнях — внешнем и внутреннем. Внутренний уровень — это встроенные в систему средства защиты от несанкционированного доступа пользователя (PRIMUS, БЕС7920), регулирующие распределение ресурсов между пользователями внутри ДСКД. При взаимодействии с ОС ЕС (внешний уровень) используется механизм захвата и освобождения ресурсов, обеспечиваемый средствами супервизора ОС ЕС (макрокоманды ENQ и DEQ). Что же касается распределения ОП, то оно реализуется с помощью алгоритма, по которому необходимая область ОП выделяется из общего резерва динамической памяти ДСКД при поступлении запроса на память (макрокоманда GETMAIN) и возвращается в резерв, когда программа в ней больше не нуждается (макрокоманда FREEMAIN), т. е. опять же реализуется средствами супервизора ОС ЕС.

Следует отметить одну особенность, касающуюся совместного использования общих (постоянных) наборов данных. В отличие от среды пакетной обработки, где модифицируемые в одном задании наборы данных недоступны другим заданиям до тех пор, пока полностью не завершится первое задание, в ДСКД это решается иначе. На время сеанса каждому пользователю выделяется в монопольное использование рабочая внешняя память (обычно отдельный временный набор данных). Информация из постоянного набора данных, необходимая пользователю для выполнения определенной работы (например, исходный текст для редактирования), переписывается в рабочий набор данных; после завершения работы с данными в рабочем буфере, если пользователю необходимо переписать их в постоянный набор данных, выполняется обратная операция (например, это может быть операция замены старой копии исходной программы новой, только что скорректированной). Теперь благодаря наличию буферной памяти постоянные наборы данных запрашиваются у системы в качестве ресурсов только на те короткие промежутки времени, в течение которых осуществляется перепись информации в рабочий набор или из него в постоянный набор данных.

8.1.3. Языковые средства систем. Для общения пользователя с системой во всех ДСКД разработан язык директивного типа (язык команд). Команда является основной единицей работы каждой из систем. Команда состоит из кода операции и поля параметров (операндов) команды. Способ задания параметров — позиционно-ключевой (JEC, PRIMUS) и ключевой (БЕС7920).

Большинство параметров имеет подразумеваемые значения, принимаемые системой по умолчанию. Ряд параметров можно зафиксировать, и они будут действовать в течение всего сеанса работы пользователя.

Синтаксис и лексика языков в каждой из систем различна, однако во всех системах они предельно просты. Например, команда просмотра оглавления тома в различных системах выглядит так:

#VTOC	SYSRES	(JEC)
VTOC	SYSRES	(PRIMUS)
VTOC	TOM=SYSRES	(БЕС7920)

Кроме того, для ряда операций, выполняемых в пределах одной команды, используется язык приказов (подкоманд), а иногда для запроса некоторых параметров используются диалоговые процедуры типа «вопрос — ответ».

8.1.4. Использование комплексов ЕС-7920. В рассматриваемых системах полнота использования аппаратурных возможностей комплексов ЕС-7920 различна. Так, в системах JEC и PRIMUS, первые редакции которых разрабатывались с ориентацией на работу с функционально простыми дисплеями ЕС-7906, используются в основном базовые возможности комплексов ЕС-7920, а характер использования специфических аппаратурных средств ЕС-7920 носит скорее эпизодический, нежели концептуальный характер. Кроме того, так как современные редакции этих систем поддерживают работу с дисплеями различных типов (ЕС-7906, ЕС-7920, ЕС-8564, ЕС-8561 и др.), отказ от специфических аппаратурных возможностей ЕС-7920 представляется вполне обоснованным.

В иной ситуации «оказалась» система БЕС7920 — еще в начале проектирования системы была выбрана ориентация на однородную терминальную среду, состоящую из локальных дисплеев одного типа (ЕС-7920-01), благодаря чему были сняты какие-либо ограничения на использование тех или иных возможностей дисплеев. Естественно, что в этом случае значительная часть программного обеспечения системы учитывает особенности работы с дисплеями ЕС-7920 и не может быть использована для работы с терминалами других типов. Особенности использования комплексов ЕС-7920 в этой системе будут рассмотрены в гл. 9.

8.2. Основные функциональные возможности систем

Функциональные возможности любой ДСКД, использующей для общения с пользователем язык команд, определяются совокупностью всех команд, которые система способна выполнить, а так как каждая команда в системе имеет соответствующую программную поддержку (процессор команды, или, иначе, функциональную

программу), то можно сказать другими словами, что возможности ДСКД определяются заложенным в нее функциональным программным обеспечением.

В рассматриваемых системах набор стандартных команд и алгоритмы их выполнения различны. Так, некоторые функции, в одной системе выполняемые целой последовательностью команд, в другой реализуются единственной командой. Однако во всех системах набор стандартных функций формировался на основании объективных потребностей пользователей-программистов в этих средствах, и поэтому, если в стандартном наборе каждой системы выделить определенные группы команд (в основном по объектам, обрабатываемым ими), то классификация всех команд по следующим функциональным группам оказывается общей для всех трех систем:

- редактирование текстов;
- обработка произвольных библиотечных или последовательных наборов данных;
- просмотр информации различного характера;
- модификация оглавления тома МД;
- работа с каталогом;
- организация пакетной обработки из среды ДСКД;
- связь с оператором ЭВМ;
- выполнение прикладных программ пользователя в среде ДСКД и организация в них диалоговых процедур;
- управление сеансом работы пользователя и системы в целом.

Приведенная классификация не учитывает лишь незначительную часть команд, специфичных для каждой конкретной системы. Ниже приводится характеристика каждой из перечисленных функциональных групп (за исключением двух последних, о которых мы поговорим в 8.3 — 8.4 и в гл. 9).

8.2.1. Редактирование текстов. Проблемы и цели редакторских функций подробно обсуждались в гл. 7, — и здесь они рассматриваться не будут. Рассматриваемые системы имеют развитые функции редактирования, включающие самые разнообразные сервисные процедуры, такие, как вставка и удаление отдельных строк, групп строк или фрагментов текста, различные поисковые процедуры, размножение отдельных участков текста, нумерация операторов и т. п. Хотя в каждой системе предлагаются свои алгоритмы реализации этих процедур, их назначение во всех системах одно — максимально ускорить процесс редактирования текстовой информации и получение выходной (новой или скорректированной) копии текста.

Во всех системах используется постраничное редактирование с логическим выделением из страницы 80-символьных строк (операторов). Исходные тексты для редактирования могут быть пред-

ставлены в виде последовательных наборов данных или разделов библиотек, содержащих 80-байтовые логические записи.

8.2.2. Обработка произвольных библиотечных или последовательных наборов данных. В эту группу включены команды модификации справочников произвольных библиотек и операции копирования произвольных библиотечных и последовательных наборов данных.

Команды модификации справочника обеспечивают переименование и вычеркивание разделов библиотек, а также присвоение разделам дополнительных имен.

Команды копирования (PRIMUS, BEC7920) обеспечивают копирование отдельных разделов библиотек и последовательных наборов данных, имеющих произвольный формат содержащихся в них данных; можно копировать группы разделов, а также целые библиотеки. В некоторых случаях допускается несовпадение характеристик «длина записи» и «длина блока» во входном и выходном наборах данных.

8.2.3. Просмотр информации различного характера. Команды просмотра образуют наиболее многочисленную группу команд, предназначенных для просмотра информации самого различного характера: исходных текстов, оглавления библиотеки, метки набора данных, оглавления тома, личного или системного каталога, результатов выполнения пакетного задания, различной информации состояния как самой ДСКД, так и ОС ЕС (информации об устройствах ввода-вывода системы, информации об активных заданиях системы, содержимого системных очередей, информации об активных пользователях ДСКД и свободных дисплеях и т. п.). Кроме того, в некоторых операциях просмотра доступны некоторые функции, отнесенные в другие группы (PRIMUS, BEC7920). Например, в процессе просмотра оглавления тома можно вычеркнуть отдельные или все наборы данных с тома МД, т. е. модифицировать оглавление тома.

8.2.4. Модификация оглавления тома МД. В эту группу включены команды переименования и вычеркивания произвольных наборов данных, содержащихся на томе МД, а также создание «пустого» библиотечного или последовательного набора данных. Эти операции имеют дело с блоками управления наборами данных различного формата, содержащимися в оглавлении тома.

8.2.5. Работа с каталогом. Операции с каталогом системы включают кроме его просмотра, отнесенного к группе команд просмотра, каталогизацию произвольного набора данных и вычеркивание ссылки на набор данных из каталога (PRIMUS, BEC7920). Кроме того, в BEC7920 допускается работа как с системным, так и с личным каталогом системы, а также имеются операции для повторной каталогизации набора данных и вычеркивания ссылки на набор данных из каталога с одновременным вычеркиванием самого набора данных.

8.2.6. Организация пакетной обработки из среды ДСКД. В рассматриваемых системах схема организации пакетной обработки заданий пользователей (за исключением некоторых нюансов) одинакова. С помощью обычных команд редактирования текста или специально предназначенных для этих целей команд пользователь подготавливает или корректирует ранее подготовленный текст задания и помещает его в некоторую очередь (обычно на дисках). Очередь может быть монополюс выделена каждому пользователю, или это может быть общая для всех пользователей очередь.

Для сформированной очереди специальной командой динамически запускается программа системного ввода ОС ЕС, использующая эту очередь в качестве входного потока. Задания пользователей переписываются этой программой в системную входную очередь, и дальше задания обрабатываются по обычному алгоритму пакетной обработки, принятому в ОС ЕС.

Для обработки результатов выполнения пакетного задания пользователями ДСКД обычно используются три команды: просмотр результатов на экране, перевод результатов в класс активного системного вывода (распечатка результатов) и уничтожение результатов.

Оперативный контроль за прохождением задания в среде пакетной обработки во всех системах осуществляется на уровне команд просмотра информации о состоянии системных очередей и активных заданий, периодически выдаваемых пользователями ДСКД.

8.2.7. Связь с оператором ЭВМ. Организация эффективной связи с оператором ЭВМ особенно актуальна в тех случаях, когда дисплеи удалены от ЭВМ и вынесены в специальный дисплейный класс или на рабочие места пользователей. Во всех системах имеются специальные команды, обеспечивающие обмен сообщениями между пользователями ДСКД и оператором ЭВМ. Связь с оператором организована на уровне макрокоманд супервизора ОС ЕС WTO и WTOR, хотя в системе JES, кроме того, используется аппарат команд оператора ОС ЕС (см. 8.3).

8.3. Специальные возможности системы JES

Отличительной особенностью системы JES является наличие в ней средств ведения архива на магнитных лентах. Для этой цели в систему включены команды выборки библиотечного раздела из архива и записи раздела в него, а также вспомогательная команда закрытия архивной ленты.

Представляет определенный интерес организация диалоговых процедур между прикладной программой пользователя, выполняемой как отдельное пакетное задание системы, и самим пользователем ДСКД. Для этого в прикладную программу включаются специ-

альные макрокоманды обращения к среде ДСКД, а пользователю доступны специальные команды, с помощью которых он иницирует диалог со своей прикладной программой.

В системе JEC реализован еще ряд интересных проектных решений, описанных в [14]. Назовем лишь одно из них — аппарат управления системой с консоли оператора ЭВМ (в частности, возможность динамического отключения терминалов от системы). Он реализован на базе стандартных команд оператора START и MODIFY путем ввода в их лексику дополнительных операндов, используемых только в системе JEC. Рискованно рекомендовать такой путь организации управления системой, связанный с модификацией стандартных функций компонент ОС ЕС, но отметить его необходимо как наиболее эффективный и красивый способ для организации операторного управления системой.

8.4. Дополнительные функции в системе PRIMUS

К дополнительным функциональным возможностям системы PRIMUS необходимо прежде всего отнести команду синтаксического анализа текста программ, написанных на языках фортран или ПЛ/1, и средства для динамической отладки программ, написанных на ассемблере. Поскольку алгоритмы, используемые для реализации этих возможностей в системе PRIMUS, идентичны соответствующим алгоритмам, используемым в CPB (подкоманда SCAN команды EDIT и команда TEST), мы отсылаем читателя к публикациям [6, 7].

ГЛАВА 9

КОНЦЕПЦИИ, АРХИТЕКТУРА И ВОЗМОЖНОСТИ СИСТЕМЫ БЕС7920

9.1. Назначение и возможности системы

БЕС7920 по аналогии с другими системами данного класса построен на базе пакетной операционной системы ОС ЕС и вносит в нее новое качество — режим разделения времени. Как и большинство подобных систем, БЕС7920 реализован в виде расширения пакетной операционной системы ОС ЕС и, таким образом обеспечивает одновременное функционирование обоих режимов: пакетного и разделения времени.

БЕС7920 имеет блочно-модульную структуры с иерархической зависимостью блоков снизу вверх. Благодаря такой древовидной структуре систему можно использовать на различных уровнях иерархии в соответствии с требованиями конкретного применения. Так, используя только ядро системы, можно использовать ее в качестве многотерминального монитора при проектировании другой интерактивной системы; добавив к ядру языковые средства БЕС7920, можно разработать свой собственный набор команд, выполняющих некоторые прикладные задачи; наконец, используя стандартный набор команд системы, можно обеспечить программиста эффективным инструментом для интерактивной разработки, отладки и выполнения своих программ.

Ориентация системы на однородную терминальную среду, состоящую из функционально однотипных дисплеев, позволила создать достаточно экономичную (с точки зрения использования ресурсов) и надежную систему, предлагающую эффективную технологию использования комплексов ЕС-7920.

БЕС7920 работает под управлением ОС ЕС в режиме MVT. В качестве технических средств обеспечения диалога используются локальные дисплейные комплексы ЕС-7920-01 или их зарубежные аналоги (например, IBM 3272 модели 2), подключенные через каналы ввода-вывода к ЭВМ ЕС любой модели, а в качестве системного программного обеспечения использован базисный телекоммуникационный метод доступа БТМД ОС ЕС.

Минимальный объем оперативной памяти, необходимый для функционирования БЕС7920, составляет $(50 + 16N)$ Кбайт, где N — количество активных дисплеев. Для размещения компонент системы

на дисках необходимо около 30 цилиндров внешней памяти типа ЕС-5061. Дополнительный расход внешней памяти зависит от количества активизируемых дисплеев.

9.2. Особенности использования дисплейных станций

Авторы книги сознательно отказались от создания универсальной системы, остановив свой выбор на функционально однородной терминальной среде. Это позволило использовать все аппаратные возможности терминалов для обеспечения большей производительности и создания повышенного комфорта пользователей.

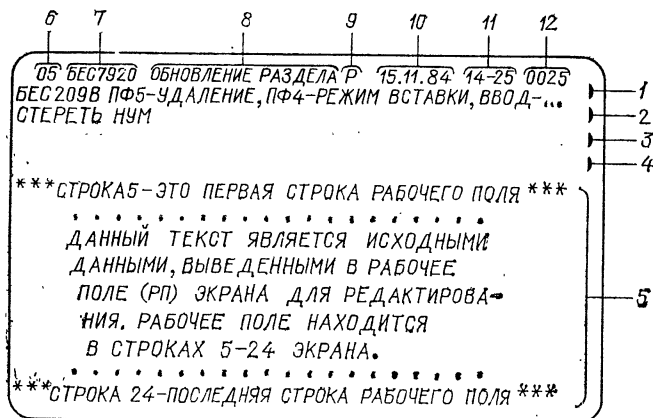


Рис. 9.1. Формат экрана в режиме редактирования текста: 1 — управляющее поле системы; 2 — поле сообщений системы; 3 — поле ввода команды (приказа); 4 — строка вставки; 5 — рабочее поле экрана; 6 — относительный номер дисплея; 7 — титул системы; 8 — наименование выполняемой команды; 9 — код формы доступа пользователя; 10 — текущая дата; 11 — текущее время дня; 12 — продолжительность сеанса работы данного пользователя

Итак, как же используются дисплеи в БЕС7920? Прежде всего — о форматах экрана. В БЕС7920 используются несколько различных форматов экрана в зависимости от режима, установленного на дисплее (режим инициализации или просмотра и т. п.), а в режиме выполнения команды — в зависимости от выполняемой операции. Поскольку наибольшее значение для пользователя имеет формат экрана, используемый при редактировании текстов, рассмотрим именно этот формат. На рис. 9.1 показан экранный кадр, типичный при работе с командой ОБНОВИТЬ (корректировка исходного текста).

Рассмотрим назначение и структуру полей, используемых при редактировании данных. Первая строка является управляющим полем системы (УПС), содержащим защищенную от ввода информацию системы. Вторая строка — это поле сообщений системы (ПСС), в которое выводятся сообщения об ошибках, информационные и предупреждающие сообщения. Оно также защищено от ввода. Третья строка используется в качестве поля ввода команды (ПВК), предназначенного для ввода управляющей информации пользователем БЕС7920. Это могут быть тексты команд, приказов или другие управляющие указания. Непосредственно при вводе команды ее текст может занимать две дополнительные строки (строки 4 и 5). При этом все три строки рассматриваются как единое поле, т. е. информация, содержащаяся в этом поле, считывается потоком. Строки 5 — 24 рассматриваются как одно поле, в котором пользователь просматривает или редактирует данные. Эта часть экрана называется рабочим полем (РП) экрана. Хотя физически эти строки представляют собой одно поле, логически оно обрабатывается построчно (в операциях редактирования или просмотра исходного текста). В некоторых операциях просмотра (например, при работе с оглавлением библиотеки или оглавлением тома) РП разбивается на дополнительные поля, каждое из которых содержит соответствующий элемент просматриваемой информации (например, для оглавления библиотеки каждая строка РП разбивается на восемь 10-байтовых полей, каждое из которых содержит имя раздела). Это позволяет использовать клавиши управления курсором, ориентированные на поля (\rightarrow), (\leftarrow и \downarrow), для быстрого подвода курсора к элементу, интересующему пользователя.

Теперь — о курсоре. Как уже говорилось выше, программы имеют возможность (с помощью определенных указаний в потоке записываемых на экран данных) установить курсор в любую позицию экрана. Используя это свойство, программы БЕС7920 стараются установить курсор таким образом, чтобы пользователь был избавлен от необходимости выполнять манипуляции по установке курсора в нужное положение. Так, в режиме ввода команды курсор устанавливается в первую позицию ПВК; если во введенной команде обнаружена синтаксическая ошибка, то курсор укажет на первый знак операнда или его значения в зависимости от того, где обнаружена ошибка.

Особо важное значение придается свойству курсора идентифицировать свое местоположение на экране при считывании информации с дисплея. Это свойство курсора, который выступает в данном случае в роли селектора, широко используется в системе. Так, различные операции над элементами информации в РП выполняются следующим образом: курсор устанавливается пользователем так, чтобы он идентифицировал выбранный на экране элемент, и нажи-

мается одна из клавиш программного функционирования, соответствующая выбранной операции. Примерами могут служить следующие операции:

1. Вычеркивание строки из РП. Курсор устанавливается в любой позиции строки, подлежащей вычеркиванию, после чего нажимается клавиша ПФ5. В результате строки, находящиеся в РП ниже курсора, сдвигаются на одну строку вверх, затирая вычеркиваемую строку, а последняя строка РП очищается двоичными нулями.

2. Программируемый режим вставки. Курсор устанавливается в строке, перед которой необходимо вставить одну или несколько строк, и нажимается клавиша ПФ4. Вся информация в РП, находящаяся ниже указанной строки (включая ее саму), убирается с экрана и сохраняется в памяти ЭВМ, освобождая место для вставки.

3. Вычеркивание разделов из библиотеки или наборов данных с тома НМД. Пользователь, используя клавиши →|, |← или _|, вызывает курсором имя удаляемого раздела или набора данных, и нажимает клавишу ПФ3.

В приведенных примерах кроме иллюстрации роли курсора в системе показано использование функциональной клавиатуры дисплея. Под функциональной клавиатурой будем понимать совокупность всех функциональных клавиш, нажатие любой из которых вызывает запрос на программное обслуживание в результате прерывания по сигналу ВНИМАНИЕ. Напомним, какие же это клавиши. Это — клавиши программного функционирования ПФ1 — ПФ12, программного доступа ПД1 — ПД3, ВВОД, СТРН ЭКР и ВЫЗОВ ТЕСТА.

В БЕС7920 за некоторыми из перечисленных функциональных клавиш закреплены стандартные (постоянные) функции. Например, ввод команды (в большинстве случаев — и ввод данных) осуществляется по клавише ВВОД; явный запрос HELP-функции — по клавише ПД1; приказы (подкоманды) всегда вводятся по клавише ПФ1; клавиша СТРН ЭКР обычно используется для восстановления последнего кадра, хотя иногда ее назначение иное. Однако большинство функциональных клавиш используется по-разному в зависимости от режима, установленного на дисплее, и характера выполняемой операции.

Таким образом, функциональная клавиатура может служить хорошим средством для сокращения числа ручных операций, выполняемых пользователем на клавиатуре дисплея. Однако перед разработчиками программного обеспечения, ориентированного на применение функциональной клавиатуры, встает много проблем. Например, как сделать программы частично или полностью независимыми от конкретных клавиш, каким образом быстро настроить программу на определенную функциональную клавишу, как в процессе

выполнения программы динамически блокировать использование той или иной клавиши либо, наоборот, разрешить ее использование и др. На эти и некоторые другие вопросы в определенной степени отвечает реализованный в системе программный аппарат обработки сигнала ВНИМАНИЕ, который будет подробно рассматриваться чуть ниже.

Несколько клавиш способны влиять на содержимое буферной памяти дисплея, не требуя при этом программного обслуживания, т. е. не вызывая прерывания по сигналу ВНИМАНИЕ. Это — клавиши СТРН ВВ (СТиРаНИЕ ВВода), СТРН КП (СТиРаНИЕ Конца Поля), ВСТВ (ВСТаВка), ВЧРК (ВыЧеРКивание), ДУБЛ (ДУБЛирование), МТКП (МеТКа Поля). Совокупность этих клавиш условно назовем *клавиатурой автономного редактирования*. Для того чтобы пользователь мог ими пользоваться при редактировании текста, необходима определенная программная дисциплина формирования выходного потока записываемых на дисплей данных. Например, чтобы воспользоваться режимом вставки, в который переводится дисплей при нажатии клавиши ВСТВ, необходимо наличие нулей (X'00') в позициях, которые следуют в поле за местом вставки. Если заполнять поле текстом, сохраняя пробелы (X'40') между текстовыми конструкциями, то воспользоваться режимом вставки в таком поле невозможно. Поэтому в БЕС7920 данные, выводимые на экран для редактирования, предварительно преобразовываются следующим образом. Все непрерывные последовательности пробелов, встречающиеся в тексте, заменяются двоичными нулями, кроме первого пробела последовательности. Выведенные таким образом данные допускают использование режима вставки, блокируя, однако, слияние соседних участков текста в строке. При считывании данных с экрана обычно выполняется обратное преобразование. Заметим, что в языке команд БЕС7920 имеется возможность запретить выполнение этого преобразования перед выводом данных на экран и соответственно после считывания их с экрана.

Рассмотрим еще несколько особенностей использования дисплейных станций. Как известно, форматизуя экран при записи на дисплей, полям можно присвоить определенные характеристики (атрибуты): поле можно сделать защищенным от ввода либо разрешить ввод в него; данные в поле можно отображать с обычной или повышенной яркостью либо сделать их вообще неотображаемыми; чисто цифровые поля могут быть защищены от ввода алфавитной информации и специальных знаков; наконец, можно сделать поле определяемым или неопределяемым фотоселектором; кроме того, если поле определяется фотоселектором, его можно сделать либо полем выборки, либо полем внимания. В БЕС7920 эти возможности используются следующим образом. Наиболее важная информация, записываемая системой на дисплей, отображается с повышенной

яркостью (например, сообщения в ПСС). Второстепенная управляющая информация, а также текст, выводимый для редактирования, отображаются с обычной яркостью. Все операции ввода (кроме ввода пароля) осуществляются на отображаемых (с обычной яркостью) полях; пароль вводится в неотображаемое поле. Управляющие поля УПС и ПСС (строки 1 и 2) и рабочая информация, выводимая исключительно для просмотра ее пользователем, защищены от ввода. Использование полей, определяемых фотоселектором, будет рассмотрено ниже.

Описанная технология использования дисплейных станций не является единственно возможной, однако, поскольку в ней учитывается большинство аппаратных возможностей комплексов ЕС-7920, ее программная поддержка может представлять определенный интерес для читателя. Рассматривая программную реализацию системы, нам придется постоянно возвращаться к тем или иным особенностям использования дисплеев. Но сначала рассмотрим языковые средства системы.

9.3. Языковые средства системы

Как и в большинстве ДСКД, для взаимодействия пользователя с системой в ней реализован язык директивного типа (язык команд). Формализованное описание действий системы для выполнения некоторой работы, запрошенной пользователем, будем называть *командой*. Таким образом, команда является единицей работы системы. До окончания выполнения одной команды нельзя инициировать выполнение другой. *Режим ввода команды* — это состояние, в котором система готова принять от пользователя очередную команду. В БЕС7920 режим ввода команды устанавливается сразу после успешной инициализации сеанса работы пользователя и восстанавливается после выполнения очередной команды (кроме команды ОКОНЧИТЬ СЕАНС).

Совокупность всех команд и формальных правил их описания будем называть *языком команд*. Формальные правила (синтаксис и лексика) языка устанавливаются на этапе технического проектирования системы и, как правило, не подлежат существенным изменениям (в БЕС7920 это прежде всего касается синтаксиса языка). Набор команд и различные грамматические (а отчасти и лексические) конструкции языка могут расширяться, изменяться или полностью заменяться.

Работа, выполняемая некоторыми командами, может быть разделена на отдельные операции, выполнение которых пользователь запрашивает с помощью *приказов (подкоманд)*. *Приказ*, следовательно, представляет собой запрос на выполнение определенной операции, являющейся частью всей работы, выполняемой командой. На-

личие приказов обеспечивает гибкость в использовании команд, выполняющих большой объем работы, и в ряде случаев позволяет сократить общее время выполнения таких команд.

Совокупность всех приказов и правил их использования составляет *язык приказов*. Язык приказов, являясь диалектом языка команд, в отличие от него имеет упрощенную лексику.

9.3.1. Классификация команд стандартного набора. Около 40 команд вместе с программной поддержкой их выполнения (процессорами команд) составляют стандартный набор команд БЕС7920. На алгоритм выполнения конкретной команды существенно влияют такие факторы, как определение объектов обработки команды, возможность вмешательства пользователя в процесс выполнения команды, а также статус команды, присвоенный ей при генерации таблицы команд системы. В связи с этим целесообразно классифицировать команды стандартного набора с учетом перечисленных факторов.

В зависимости от объекта обработки, вида и специфики выполняемой операции все команды можно разбить на следующие функциональные группы:

- редактирование исходных текстов, представленных в виде 80-символьных логических записей разделов библиотек или последовательных наборов данных;

- операции над разделами библиотек или последовательными наборами данных, необязательно содержащими текстовую информацию;

- операции над произвольными наборами данных;

- операции с томами НМД;

- организация фоновой пакетной обработки и обработка результатов выполнения пакетных заданий;

- управление сеансом работы и просмотр информации состояния системы.

С точки зрения взаимодействия системы с пользователем в процессе выполнения инициированной им команды все команды подразделяются на одношаговые и многошаговые. Под *одношаговыми* будут пониматься команды, в процессе выполнения которых система не устанавливает связь с пользователем, поскольку в них полностью детерминированы все действия, которые необходимо выполнить системе. Одношаговые команды, как правило, выполняются моментально, и пользователь лишь оповещается об успешном или аварийном окончании команды. Если время выполнения одношаговой команды значительно (свыше 8—10 с), пользователь оповещается о задержке. В отличие от одношаговых *многошаговые* команды предполагают активное участие пользователя в процессе их выполнения. После приема любой из таких команд дисплей переводится либо в режим редактирования, либо в режим просмотра ин-

формации в зависимости от характера выполняемой операции. В обоих режимах пользователь сам выбирает дальнейшее продолжение выполнения команды.

Наконец, все команды подразделяются на две группы в зависимости от статуса, присваиваемого каждой команде при генерации таблицы команд БЕС7920. Статус команды определяется двумя характеристиками: взаимодействует ли команда с наборами данных пользователя и относится ли команда к числу так называемых команд администратора (команд ограниченного использования).

Выполнение той или иной команды санкционируется пользователю в соответствии со статусом команды и формой доступа пользователя, присвоенной ему при регистрации его в качестве абонента системы. В табл. 9.1 приведен перечень команд стандартного набора и указана принадлежность каждой команды к перечисленным выше группам. В таблице использованы следующие обозначения:

РБ — раздел библиотечного набора данных;

ПН — последовательный набор данных;

С — символьная информация, представленная 80-байтовыми логическими записями фиксированной длины;

НД — набор данных с произвольной, допустимой в ОС ЕС организацией;

БН — библиотечный набор данных, содержащий произвольную информацию с любым форматом записей;

Т — том на НМД;

З — формирование и запуск пакетной обработки либо обработка результатов выполнения пакетного задания;

У — управление сеансом работы;

Ч — команда читает наборы данных абонента;

М — команда модифицирует наборы данных абонента;

А — команда администратора.

Если в таблице, например, указано «РБ,ПН,С» (в графе «объект обработки»), то это означает, что объектом обработки является текстовая информация, представленная 80-байтовыми записями фиксированной длины и находящаяся в разделе библиотеки или в последовательном наборе данных; конструкция «Ч,М,А» в графе «статус» означает, что это команда ограниченного использования (А), а кроме того, она читает наборы данных и может их модифицировать (Ч и М соответственно). Для того чтобы такая команда могла быть выполнена системой, необходимы соответствующие отметки в форме доступа пользователя. В противном случае команда будет отвергнута системой, о чем пользователь будет оповещен.

Заметим, что атрибут «команда администратора» в различных ВЦ присваивается различным группам команд (корректировкой и повторной генерацией таблицы команд).

Стандартный набор команд БЕС7920

Команда	Объект обработки	Одношаговая	Статус
Добавить раздел	РБ, ПН, С	нет	М
Заменить раздел	РБ, ПН, С	нет	М
Обновить раздел	РБ, ПН, С	нет	Ч, М
Удалить раздел	РБ	да	М
Переименовать раздел	РБ	да	М
Сформировать раздел	РБ, ПН, С	да	Ч
Копировать раздел	РБ, ПН	да	Ч, М
Перенумеровать раздел	РБ, ПН, С	да	Ч, М
Читать раздел	РБ, ПН, С	нет	Ч
Присвоить алиас	РБ	да	М
Перфорировать раздел	РБ, ПН, С	нет	Ч
Печатать раздел	РБ, ПН, С	да	Ч
Вычеркнуть набор данных	НД	да	А
Просмотреть каталог	НД	нет	Ч
Каталогизировать набор данных	НД	да	М, А
Повторно каталогизировать набор данных	НД	да	М, А
Вычеркнуть и раскаталогизировать набор данных	НД	да	М, А
Распределить память под набор данных	БН, ПН	да	А
Распечатать библиотеку	БН, С	да	Ч
Переименовать набор данных	НД	да	Ч, М
Просмотреть оглавление библиотеки	БН	нет	Ч
Отобразить метку набора данных	НД	да	Ч
Просмотреть оглавление тома	Т	нет	Ч, М, А
Разъединить управляющие тома	Т	да	М, А
Поставить задание в очередь	З	нет	
Запустить пакетную обработку	З	да	
Просмотреть результаты выполнения пакетного задания	З	нет	
Вывести результаты выполнения пакетного задания	З	да	
Уничтожить результаты выполнения пакетного задания	З	да	
Назначить наборы данных	У	да	
Обменять рабочие наборы данных	У	да	
Отобразить информацию состояния	У	нет	
Послать сообщение оператору ЭВМ	У	нет	
Установить режим динамического окончания БЕС7920	У	да	А
Отменить режим динамического окончания БЕС7920	У	да	А
Установить режим слежения за активностью пользователей	У	да	А
Запустить команду оператора ЭВМ	У	да	А

же значения операндов, кодируемые списком, разделяются между собой запятой или знаком «меньше» (эти символы набираются с помощью одной и той же клавиши, только на разных регистрах клавиатуры), например

ОБНОВИТЬ РАЗДЕЛ = ТЕХТ,ПЕЧ,ВХОД = (LIBRARY
ТОМ222)

После набора текста команды она вводится нажатием клавиши ВВОД. Если считанная системой команда не содержит синтаксических ошибок и все спецификации, сделанные ее операндами, корректны, то она принимается планировщиком команды для исполнения. В противном случае в ПСС экрана выводится соответствующее сообщение об ошибке и на дисплее восстанавливается режим ввода команды. Обычно курсор при этом указывает на тот элемент в тексте команды, в котором обнаружена ошибка.

В режиме ввода команды пользователю доступны некоторые сервисные функции, обеспечиваемые функциональной клавиатурой. В табл. 9.2 перечислены функциональные клавиши, которые могут быть активизированы в этом режиме, и функции, инициализируемые ими. Рассмотрим более подробно правила использования функциональной клавиатуры в режиме ввода команды.

Клавиша ВВОД, как уже отмечалось, может быть использована только для ввода команды. Для восстановления кадра, характерного для режима ввода команды (в частности, для очистки ПВК экрана), может быть нажата клавиша СТН ЭКР.

Клавиша ПД1, как и в любом другом установленном на дисплее режиме, используется только для активизации HELP-функции. Однако в режиме ввода команды пользователь, нажав эту клавишу, получит наиболее актуальную справочную информацию — список команд. К такому же результату приведет нажатие клавиши ВВОД при «пустом» ПВК экрана (неявная активизация HELP-функции).

Нажатие любой из клавиш ПД2 или ПД3 вызывает немедленное выполнение команды ОКОНЧИТЬ СЕАНС, а нажатие какой-либо из клавиш ПФ2, ПФ6 — ПФ12 — к выполнению одной из наиболее употребимых команд.

Процедуру подготовки и ввода команды можно значительно сократить по времени, если использовать аппарат запоминания и воспроизведения текста (или некоторого приближения к нему), обеспечиваемый функциональными клавишами ПФ1, ПФ3 — ПФ5. Логика использования этого аппарата показана в примере, приведенном на рис. 9.2. В этом примере предполагается, что пользователю необходимо вывести на печать несколько каталогизированных процедур, имеющих корень ASMC (например, ASMC, ASMCL, ASMCLG и др.). Прокомментируем этот рисунок.

ПСС: БЕС008В ВВЕДИТЕ КОМАНДУ

ПВК: ☐

ПВК: ПЕЧАТАТЬ ВХОД=SYS1.PROCLIB,
РАЗДЕЛ=ASMC_



ПСС: БЕС444В НАБЕРИТЕ В ПВК ИМЯ КОМАНДЫ
И НАЖМИТЕ ПФ4

ПВК: ПЕЧАТАТЬ ВХОД=SYS1.PROCLIB,

☐
РАЗДЕЛ=ASMC_

ПВК: ПЕЧПРОЦ_ ВХОД=SYS1.PROCLIB,РАЗДЕЛ=ASMC_



ПСС: БЕС445И КОМАНДА С ИМЕНЕМ ПЕЧПРОЦ
ДОБАВЛЕНА В . . .

ПВК: ПЕЧАТАТЬ ВХОД=SYS1.PROCLIB,
РАЗДЕЛ=ASMC_ ☐

ПВК: ПЕЧАТАТЬ ВХОД=SYS1.PROCLIB,
РАЗДЕЛ=ASMC_ ☐



.
ПСС: БЕС008В ВВЕДИТЕ КОМАНДУ

ПВК: ☐

ПВК: ПЕЧПРОЦ_



ПСС: БЕС448В В ПВК—КОМАНДА С ИМЕНЕМ
/ПЕЧПРОЦ/

ПВК: ПЕЧАТАТЬ ВХОД=SYS1.PROCLIB,
РАЗДЕЛ=ASMC_ ☐

ПВК: ПЕЧАТАТЬ ВХОД=SYS1.PROCLIB,



.....

Обозначения:

- информация, набираемая пользователем
- курсор
- — символ «подчеркивание»

Рис. 9.2. Пример использования логики запоминания/воспроизведения текста команды

1. Для того чтобы запомнить в режиме ввода команды текст часто используемой команды, необходимо набрать его в ПВК экрана и нажать клавишу ПФЗ (шаг 1). По запросу от этой клавиши текст команды считывается и без какого-либо контроля запоминается.

2. В ПСС выводится сообщение с требованием ввести имя, под которым команда будет размещена в качестве раздела в библиотеке команд. Пользователь набирает имя в ПВК (в нашем случае — ПЕЧПРОЦ), дополненное справа одним пробелом, и нажимает клавишу ПФ4. Если бы такое имя уже имелось в библиотеке команд, т. е. пользователь хотел бы просто заменить ее текст, то следовало бы нажать клавишу ПФ5.

3. После выполнения операций по запоминанию команды система восстанавливает в ПВК экрана первоначальный текст команды, измененный при наборе имени команды. Если текст команды не требует никаких корректировок, достаточно нажать клавишу ВВОД для ее инициализации (шаг 3). Команда считывается, интерпретируется и, если все в порядке, выполняется.

4. После выполнения команды на дисплее устанавливается режим ввода команды, в котором пользователь вместо набора всей команды может теперь набрать лишь имя команды и нажать клавишу ПФ1 (шаг 4).

5. Команда считывается из библиотеки команд и отображается в ПВК экрана. В нашем примере в ней достаточно добавить один символ и ввести ее, нажав клавишу ВВОД.

Нами не прокомментирована одна деталь на рисунке — назначение символа «подчеркивание», набранного в тексте команды последним (шаг 1). С помощью этого символа можно указать позицию

**Использование функциональной клавиатуры
в режиме ввода команды**

Клавиша	Функция
ВВОД	<ol style="list-style-type: none"> 1. Запрос на считывание команды, набранной в ПВК экрана 2. Невнятный запрос на активизацию HELP-функции (ПВК экрана пусто) 3. Запрос макета команды (в ПВК экрана набран только код команды)
СТРН ЭКР	Восстановление начального кадра на экране или очистка ПВК экрана
ПД1	Активизация HELP-функции (явный запрос)
ПД2	Окончание сеанса пользователем (запрос на выполнение команды ОКОНЧИТЬ СЕАНС)
ПД3	Аналогично клавише ПД2
ПФ1	Запрос на отображение текста команды, предварительно подготовленного в библиотеке команд
ПФ2	Запрос на немедленное выполнение команды ОБМЕНИТЬ РАБОЧИЕ НАБОРЫ
ПФ3	Запрос на запоминание текста команды, набранного в ПВК экрана
ПФ4	Добавление новой команды в библиотеку команд (добавление нового имени раздела в справочник библиотеки команд)
ПФ5	Замена текста команды в уже существующем разделе библиотеки команд (корректировка существующего раздела в библиотеке команд)
ПФ6	Запрос на инициализацию команды ПОСТАВИТЬ ЗАДАНИЕ В ОЧЕРЕДЬ
ПФ7	Запрос на немедленное выполнение команды ЗАПУСТИТЬ ПАКЕТНУЮ ОБРАБОТКУ
ПФ8	Запрос на немедленное выполнение команды ОТОБРАЗИТЬ ИНФОРМАЦИЮ СОСТОЯНИЯ об активных заданиях системы
ПФ9	Запрос на немедленное выполнение команды ОТОБРАЗИТЬ ИНФОРМАЦИЮ СОСТОЯНИЯ об элементах всех системных очередей
ПФ10	Запрос на немедленное выполнение команды ОТОБРАЗИТЬ ИНФОРМАЦИЮ СОСТОЯНИЯ обо всех устройствах ввода-вывода системы
ПФ11	Запрос на немедленное выполнение команды ОТОБРАЗИТЬ ИНФОРМАЦИЮ СОСТОЯНИЯ обо всех дисплеях и пользователях, подключенных к БЕС7920
ПФ12	Запрос на немедленное выполнение команды ПРОСМОТРЕТЬ ОГЛАВЛЕНИЕ БИБЛИОТЕКИ, постоянно закрепленной за дисплеем

курсора, в которую он должен быть установлен при воспроизведении команды. В воспроизведенном тексте команды (шаги 3 и 5) этот символ будет отсутствовать, а курсор будет установлен на его месте. (Курсор на рисунке обозначен знаком □.)

В заключение рассмотрим еще одну возможность, позволяющую сократить время подготовки и ввода команды, это — вызов так называемого макета команды. Суть этой операции заключается в следующем:

- в информационной базе HELP-функции для каждой команды стандартного набора заготовлен ее макет — текст команды, содержащий все используемые в ней операнды. При этом, если для операнда предусмотрено подразумеваемое значение, оно будет присутствовать в тексте команды;

- для вызова макета и отображения его в ПК экрана необходимо набрать только код команды и нажать клавишу ВВОД;

- макет команды отображается в ПК экрана. Пользователь, используя средства автономного редактирования (клавиши ВЧРК, ВСТВ, клавиши управления курсором и алфавитно-цифровую клавиатуру), должен отредактировать отображенный текст и, нажав клавишу ВВОД, ввести подготовленную таким образом команду для выполнения.

9.3.3. Лексика, синтаксис и правила ввода приказов. Как уже отмечалось выше, использование приказов допускается лишь в многошаговых командах, что содействует выполнению всей работы, проделываемой командой. Например, в процессе редактирования текста может потребоваться просто скопировать часть информации из входного буфера в выходной. Это можно было бы сделать простым «перелистыванием» кадров, однако при значительной длине копируемого фрагмента это утомительно. Проще воспользоваться приказом КОПИРОВАТЬ, указав идентификационный номер первого оператора текста, с которого необходимо продолжить редактирование, или содержащийся в нем контекст. Другим примером использования приказов может служить изменение текущих параметров работы команды.

Синтаксис приказов не отличается от синтаксиса команд. Так же, как команда, приказ состоит из mnemonic-кода и операндов, детализирующих параметры выполняемой операции. В качестве кода приказа можно использовать до четырех ключевых слов. В основном все приказы используют позиционные операнды (в отличие от операндов команд), т. е. порядок следования операндов в приказе строго фиксирован. Исключение составляет приказ ИЗМЕНИТЬ, операнды которого кодируются в ключевом формате. Правила кодирования операндов и использования разделителей точно такие же, как и для команд. Список приказов приведен в табл. 9.3.

Список приказов БЕС7920

Приказ	Выполняемая операция
ВЫЧЕРКНУТЬ	Вычеркивание одной строки или группы строк из РП. Аргументы поиска вычеркиваемых строк: идентификационные номера операторов или номера строк РП (5—24)
ПРОПУСТИТЬ	Пропуск (вычеркивание) некоторого количества операторов во входном буфере без вывода их на экран дисплея. Аргумент поиска очередного отображаемого оператора: идентификационный номер оператора или контекст, содержащийся в нем
КОПИРОВАТЬ	Копирование некоторого количества операторов из входного буфера в выходной без отображения их на экране. Аргументы поиска очередного отображаемого оператора: идентификационный номер оператора или контекст, содержащийся в нем
УСТАНОВИТЬ	Указание новой точки продолжения просмотра — возврат к информации, которая уже просматривалась, или, наоборот, пропуск очередной части информации во входном буфере
ОБМЕНЯТЬ	Взаимная замена информации в двух строках РП по их номерам (5—24)
СТЕРЕТЬ	Очистка нулями (X'00') поля идентификации каждого оператора, находящегося в РП
ВВЕРХ	«Выталкивание» некоторого количества операторов (1—20) из верхних строк РП в выходной буфер и добавление такого же количества операторов в нижние строки РП из входного буфера
ИЗМЕНИТЬ	Изменение текущих параметров обработки, используемых в выполняемой команде
ПОМОЧЬ	Активизация HELP-функции в процессе выполнения команды с указанием конкретной темы, интересующей пользователя
ОКОНЧИТЬ	Запрос на аннулирование результатов выполнения команды и немедленное завершение ее самой

Правила ввода приказа практически те же, что и для ввода команды. Исключение составляет конечная фаза ввода: для ввода приказа необходимо нажать клавишу ПФ1, а не ВВОД.

Порядок выполнения приказа идентичен схеме выполнения одношаговой команды. После считывания приказа интерпретатор производит его синтаксический и логический контроль. Если в приказе обнаружена ошибка, то в ПСС экрана выводится соответствующее

сообщение и приказ игнорируется. Если ошибок не обнаружено, система выполняет запрошенную операцию. О результатах выполнения приказа пользователь оповещается сообщением в ПСС экрана.

Нетрудно заметить, что использование некоторой функциональной клавиши в процессе выполнения многошаговой команды — это фактически ввод определенного приказа, в котором роль единственного операнда (если он необходим) выполняет курсор, идентифицирующий объект обработки. Например, при редактировании текста можно вычеркнуть одну строку из РП, указав ее курсором и нажав клавишу ПФ5. Этот «клавишный» приказ (назовем его так) можно было бы сформулировать как «вычеркнуть строку, указанную курсором». Сравните его с «наборным» приказом:

ВЫЧ[ЕРКНУТЬ] 2,5—12,22

Преимущество клавишного приказа очевидно: пользователь использует только клавиши установки курсора и клавишу ПФ5. Однако если необходимо вычеркнуть указанные десять строк, то вероятно, что ввод директивы будет выгоднее, чем девятикратное повторение клавишного приказа.

9.4. Структура программного обеспечения

В гл. 7 описан редактор текстов, программное обеспечение (ПО) которого структурно реализовано в виде программы-монолита (программы простой структуры, наиболее эффективной по быстродействию). В процессе функционирования программы-монолита практически все ее программные компоненты постоянно присутствуют в оперативной памяти (ОП). Благодаря этому обеспечивается наиболее высокая скорость ответа на запросы пользователя, но это обычно влечет за собой большие затраты ОП. Благодаря же оригинальным проектным решениям в совокупности с выбором доступа к данным на физическом уровне авторам системы удалось избежать этого недостатка.

В данном разделе мы рассмотрим несколько иные решения, связанные с организацией ПО большой интерактивной системы. Разработчики системы не ставили перед собой столь жестких ограничений на минимальный объем ОП используемой системой: требования к предельному объему ОП допускали использование около 50—60 К байтов для ядра системы и около 16—20 К байтов на каждого активного пользователя системы, т. е. на каждый обслуживаемый системой дисплей. Что касается времени ответа системы на различные запросы пользователей, разработчики системы руководствовались теми известными исследованиями, которые свидетельствуют, что при «перелистывании» кадров время ответа (вывод очередного кадра) не должно превышать 1 с, а ответ на инициирую-

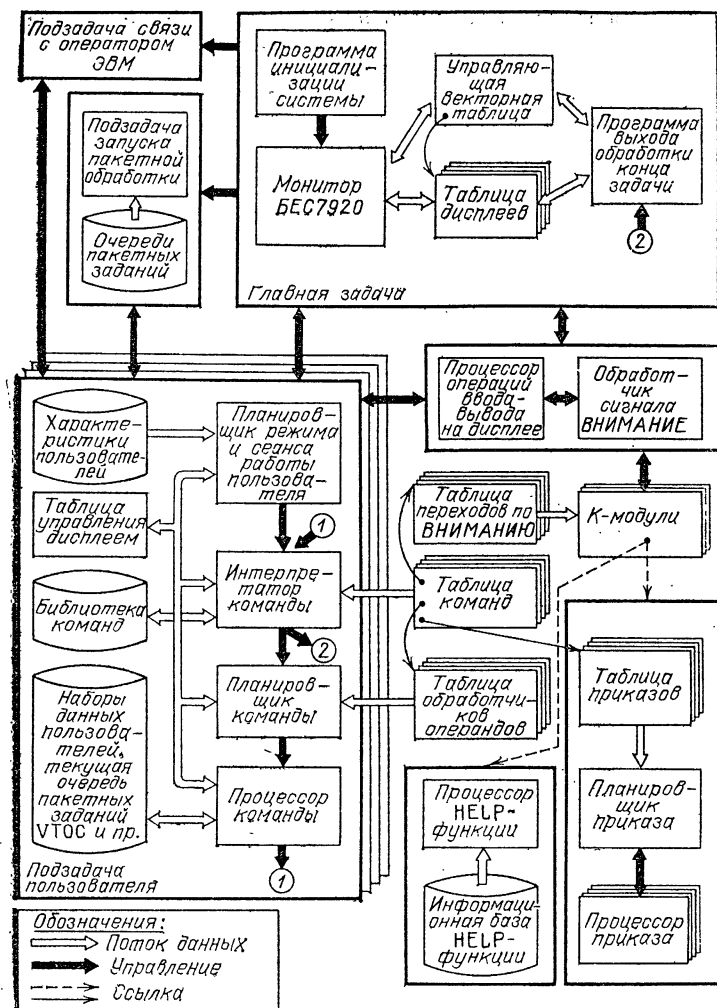


Рис. 9.3. Структурная схема программного обеспечения БЕС7920

щие действия пользователя (ввод команды, приказа и т. п.) должен поступить не позднее чем через 8—10 с.

В соответствии с этими соображениями реализация ПО системы в виде параллельной динамической структуры оказалась вполне приемлемой и эффективной. На рис. 9.3 показана упрощенная структурная схема ПО БЕС7920. Она иллюстрирует взаимосвязи между

основными программными компонентами и задачами системы го управлению и по данным. Рассматривая эту схему, необходимо учесть следующие замечания.

1. «Процессор операций ввода-вывода на дисплее» выполняется в рамках как главной задачи, так и любой порожденной ею подзадачи (хотя он и вынесен за пределы какой-либо задачи).

2. «Обработчик сигнала ВНИМАНИЕ» и все К-модули (модули поддержки функциональных клавиш) всегда выполняются в рамках х подзадач пользователей, а показанные на схеме «процессор HELP-функции» и «блок обработки приказов» являются примерами К-модулей.

3. Все таблицы системы (кроме таблицы дисплеев) являются загрузочными модулями, вызываемыми в ОП по макрокоманде LOAD и удаляемыми из памяти по макрокоманде DELETE.

4. Все таблицы (кроме таблицы управления дисплеем), показаны ли они в рамках отдельных подзадач или вынесены за их пределы, доступны всем задачам системы. Таблица управления дисплеем является «собственностью» подзадачи пользователя (у каждой подзадачи своя таблица), хотя через нее же осуществляется при необходимости связь главной задачи или системных подзадач с подзадачей пользователя. Эта таблица загружается в ОП и инициализируется первой же программой, выполняемой в рамках только что порожденной подзадачи пользователя, и сохраняется в ОП до конца сеанса пользователя.

5. На схеме не показана большая группа программ, работающих в рамках любой задачи. Они могут быть динамически выполнены вызовом по макрокоманде LINK или XCTL из любой программы, показанной на схеме. Это — обслуживающие программы, функции которых самые различные: открытие и закрытие наборов данных пользователей, запрос и освобождение различных ресурсов системы, запись и чтение рабочих наборов данных, обработка буферной памяти дисплеев и т. п.!

Ниже мы рассмотрим назначение и функции отдельных программных компонентов, показанных на рис. 9.3.

9.4.1. Программа инициализации системы и монитор БЕС7920. Программа инициализации системы — это первый загрузочный модуль, который выполняется в рамках задачи, образованной управляющей программой ОС ЕС при инициализации пункта задания (будем называть ее главной задачей БЕС7920). В ее функции входят: загрузка и инициализация управляющей векторной таблицы системы; загрузка таблицы команд системы; подсчет количества дисплеев, задействованных в задании на запуск системы, и создание на основании полученного результата таблицы дисплеев; загрузка резидентных модулей; открытие наборов данных БЕС7920; создание двух системных подзадач; наконец, открытие

блока управления данными дисплеев и запись на каждый из них начального сообщения с приглашением пользователям вступить в контакт с системой. После успешного выполнения всех перечисленных операций управление по макрокоманде XCTL передается монитору системы, входным параметром для которого является адрес управляющей векторной таблицы.

Первым действием программы является загрузка управляющей векторной таблицы — основной таблицы BEC7920, в которой собраны различные индикаторы, адресные указатели, информация о параметрах работы различных компонентов системы, блоки управления данными для различных системных наборов данных и т. п. Управляющая векторная таблица представляет собой загрузочный модуль IBZCVT, который помещается в ОП по макрокоманде LOAD, а адрес его точки входа заносится в поле TCBUSER блока управления главной задачей.

Второй таблицей, которая необходима для функционирования системы, является таблица команд BEC7920. Она представлена загрузочным модулем с именем IBZCMDTB, который заносится в ОП по той же макрокоманде LOAD. Адрес точки входа в модуль (адрес таблицы) помещается в поле CVTCMDTB управляющей векторной таблицы. Таблица команд впоследствии используется интерпретатором/планировщиком команды в подзадаче пользователя. Таблица команд состоит из отдельных элементов переменной длины, каждый из которых описывает одну команду. Для создания одного элемента таблицы используется макрокоманда COMMAND — специально разработанная для этой цели макрокоманда BEC7920, кодирование которой имеет смысл только в этом модуле. Каждый элемент таблицы содержит: информацию о допустимых кодах команды; имя процессора команды — первого модуля, которому будет передано управление после успешной инициализации команды; имя таблицы имен программ-обработчиков операндов; имя таблицы переходов по сигналу ВНИМАНИЕ; имя таблицы имен процессоров приказов; статус команды.

Наконец, на этапе инициализации системы динамически формируется еще одна таблица — таблица дисплеев DST. Таблица DST создается в области ОП, запрашиваемой динамически у управляющей программы ОС ЕС. Так как длина таблицы DST зависит от количества дисплеев, описанных в задании на запуск системы, то, прежде чем запросить память под нее, определяется количество дисплеев, равное числу сцепленных DD-предложений с именем EC7920. Таблица DST состоит из 16-байтовых элементов, каждый из которых представляет в системе один дисплей. Количество дисплеев определяется из таблицы TIOT (таблицы ввода-вывода задачи). Например, если в задании на запуск BEC7920 описаны четыре дисплея, то в таблице TIOT они будут представлены четырьмя элементами и со-

ответственно таблица DST будет также содержать четыре элемента. При этом порядок следования DD-предложений определяет так называемые относительные номера дисплеев, используемые для их идентификации в системе, и местоположение соответствующих им элементов в таблице DST. На рис. 9.4 приведен пример схемы расположения элементов в таблице DST.

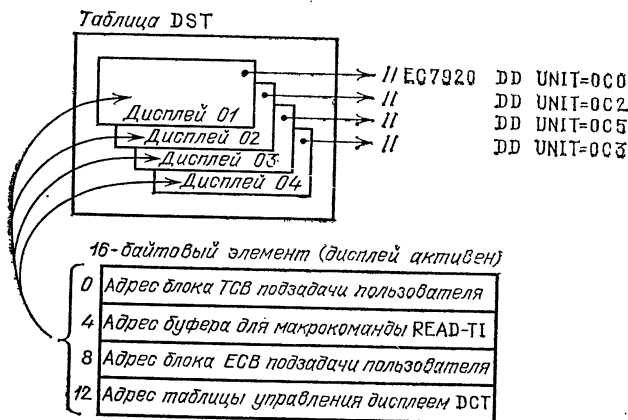


Рис. 9.4. Схема расположения элементов таблицы DST

Каждый элемент таблицы DST первоначально является пустым — содержит 16 байтов двоичных нулей. Это является признаком пассивности дисплея и соответственно отсутствия в системе подзадачи, обслуживающей данный дисплей.

После инициализации таблицы ее адрес помещается в поле CVTDSTAD управляющей векторной таблицы. Ниже приведен фрагмент программы инициализации системы, который подсчитывает количество дисплеев и создает таблицу DST:

```

. . . . .
EXTRACT TIOTADDR,FIELDS=TIOT
      L      4,TIOTADDR      адрес TIOT
      LA     4,24(4)         адрес 1-го элемента
                              «TIOT»
      SR     3,3
TIOTELM IC     3,0(4)        длина элемента
                              «TIOT»
      OC     0(4,4),0(4)     Q—конец TIOT?
      BE     NODD            да—идти на ABEND
      CLC    4(6,4),=C'EC7920' Q—DDNAME=
                              =«EC7920»?
      BE     ISDD            да—идти на подсчет

```

	LA	4,0(3,4)	адрес следующего элемента
	B	TIOTELM	
	SPACE		
ISDD	EQU	*	найдено DD EC7920*
	LA	8,1	количество DD в EC7920
	LA	4,0(3,4)	следующий элемент «TIOT»
NXTEXPCT	IC	3,0(4)	длина элемента «TIOT»
	CLC	4(6,4),=8C' '	Q-DD связано с предыдущим DD?
	BNE	SCANEND	нет
	OC	0(4,4),0(4)	Q—конец «TIOT»?
	BE	SCANEND	да
	LA	8,1(8)	+1--> в счетчик дисплеев
	LA	4,0(3,4)	следующий элемент «TIOT»
	B	NXTEXPCT	проверить следующий элемент «TIOT»
NODD	EQU	*	не определено DD с именем «EC-7920»
	WTO	'БЕСА01А В ЗАДАНИИ НЕ ОПРЕДЕЛЕНО DD С ИМЕНЕМ EC7920'	
	ABEND	16,DUMP	
SCANEND	SR	4,4	
	LA	5,16	длина элемента «DST»
	MR	4,8	длина таблицы «DST»
	GETMAIN	R,LV=(5)	получить память под «DST»
	ST	1,CVTDSTAD	запомнить адрес «DST» в «CVT»
	LR	6,1	сохранить адрес «DST»
	LXC	(6),(6),(5)	очистить нулями «DST»
.			
TIOTADDR	DS	F	

Очередной функцией программы инициализации является загрузка реентерабельных резидентных модулей системы. Список резидентных модулей оформлен сам в виде загрузочного модуля, стандартное имя которого IBZ7920L. Повторной генерацией модуля этот список можно увеличить, уменьшить или вообще заменить другим. Приведем исходный текст стандартного модуля IBZ7920L и блок загрузки резидентных модулей из программы инициализации.

Список резидентных модулей БЕС7920

*** СПИСОК РЕЗИДЕНТНЫХ МОДУЛЕЙ БЕС7920

*			
IBZ7920L	CSECT		
BLDLIST	DS	0H	<i>список—BLDL— наиболее популярных модулей</i>
MODN	DC	AL2((ENTRE-ENTRF)/76)	
LSTL	DC	AL2(76)	
ENTRF	DS	0H	
	DC	CL8'IBZATTEN', XL68'0'	<i>обработка ВНИМАНИЯ</i>
	DC	CL8'IBZLNGSS', XL68'0'	<i>«длинные» команды пересылки</i>
	DC	CL8'IBZLOGCL', XL68'0'	<i>«длинные» логические команды</i>
	DC	CL8'IBZRDWRT', XL68'0'	<i>процессор в-в на дисплее</i>
	DC	CL8'IBZSTIME', XL68'0'	<i>часы</i>
	DC	CL8'IBZWRTA', XL68'0'	<i>предпроцессор в-в (RESETPL)</i>
ENTRE	EQU	*	

Загрузка резидентных модулей БЕС7920

*** ЗАГРУЗКА РЕЗИДЕНТНЫХ МОДУЛЕЙ БЕС7920

	LOAD	EP=IBZ7920L	<i>загрузить список модулей</i>
	LR	7,0	<i>сохранить адрес начала списка</i>
	BLDL	0,(7)	<i>заполнить список—BLDL—</i>
	LTR	15,15	<i>Q—все модули найдены?</i>
	BNZ	ABEND21	<i>нет—идти на «ABEND 21»</i>
	LH	8,0(7)	<i>количество элементов в списке</i>
	LA	6,4(7)	<i>адрес первого элемента списка</i>
LOADMODA	EQU	*	
	LOAD	DE=(6)	<i>загрузить модуль</i>
	LA	6,76(6)	<i>выбрать адрес следующего элемента</i>
	BCT	8,LOADMODA	<i>Q—все модули загружены?</i>
	DELETE	EP=IBZ7920L	<i>да—удалить список из ОП</i>

.....
ABEND21 ABEND 21, DUMP

Следующим действием программы инициализации является подготовка (открытие) всех системных наборов данных БЕС7920 к работе. К ним относятся библиотека характеристик пользователей (IBZ,PASSWORD), библиотека команд (IBZ.CMDLIB), библио-

тека текстов (информационная база) HELP-функции (IBZ.HELP), наборы данных печати и перфорации. Блоки управления данными (блоки DCB) для этих наборов находятся в управляющей векторной таблице системы. Здесь же находятся и блоки DCB для двух очередей пакетных заданий, однако их открытие — это функция подзадачи запуска пакетной обработки.

Очередной шаг программы инициализации — создание двух системных подзадач — «связи с оператором ЭВМ» и «запуска пакетной обработки».

В функции первой подзадачи входит выдача макрокоманды WTOR на консоль оператора ЭВМ, благодаря которой оператор

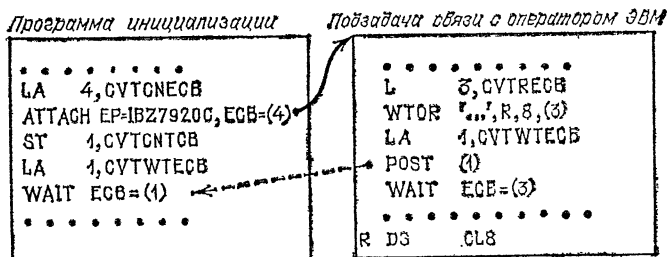


Рис. 9.5. Создание и активизация подзадачи связи с оператором ЭВМ

ЭВМ в любой момент может связаться с БЕС7920 для запроса от нее определенных действий (посылки сообщения одному пользователю или всем сразу, выдачи информации о состоянии БЕС7920, установки определенных режимов работы системы и т. п.). Схема взаимодействия программы инициализации с подзадачей показана на рис. 9.5.

Как видно из приведенной схемы, главная задача приостанавливает свою работу до тех пор, пока подзадача связи с оператором ЭВМ не выдаст макрокоманду WTOR и не отметит это событие макрокомандой POST в блоке ECB программы инициализации.

Вторая подзадача используется для временного отсоединения текущей очереди, в которую пользователи помещали свои пакетные задания, и передачи ее программе системного ввода, запускаемой динамически по команде оператора START. Более подробно об аппарате запуска пакетной обработки из среды БЕС7920 будет сказано ниже. Заметим только, что логика создания подзадачи запуска текущей очереди заданий такая же, как для подзадачи связи с оператором ЭВМ.

Наконец, после выполнения всех указанных действий программа инициализации открывает блок DCB для группы дисплеев, находящийся также в управляющей векторной таблице БЕС7920, и пишет на каждый дисплей начальное сообщение. При этом допус-

кается, что некоторые дисплеи, определенные в задании на запуск БЕС7920, могут быть выключены и запись на них начального сообщения закончится аварийно. Но эта ситуация обрабатывается программой, и впоследствии эти дисплеи могут быть подключены к системе переводом их в рабочее состояние.

Если описанные операции выполнены успешно, то программа инициализации завершает свою работу передачей управления по макрокоманде XCTL монитору БЕС7920 — загрузочному модулю с именем IBZ7920M.

Для того чтобы объяснить логику работы монитора и его функции, целесообразно напомнить о специфике БТМД — особенностях работы макрокоманды READ-TI для локальных дисплеев. Напомним их.

Во-первых, в отличие от любых других макрокоманд чтения макрокоманда READ-TI взаимодействует не с отдельным периферийным устройством, а сразу с группой устройств (в данном случае — с группой дисплеев ЕС-7927-01). Если выдана макрокоманда READ-TI, а прерывания по сигналу ВНИМАНИЕ, являющемуся запросом на считывание, ни от одного из устройств группы не последовало, то для данной группы фиксируется запрос на чтение. Если затем возникает прерывание по сигналу ВНИМАНИЕ от одного из дисплеев группы, то для этого дисплея запускается операция чтения, а запрос на чтение для группы снимается.

Во-вторых, после того как была выдана макрокоманда READ-TI, не может быть выдана другая макрокоманда READ или WRITE, адресованная одному из дисплеев той же группы, до тех пор, пока либо не завершится операция чтения, либо запрос на чтение не будет снят макрокомандой RESETPL. Может случиться так, что RESETPL была выдана в тот момент, когда уже началось считывание данных по макрокоманде READ-TI. В этом случае RESETPL не прерывает ввода данных и позволяет нормально завершиться операции считывания.

Таким образом, для того чтобы не задерживать обслуживание некоторого дисплея и быть всегда готовым к приему сообщения, постоянным состоянием системы должно быть наличие запроса на чтение по READ-TI, снимаемого лишь на те короткие промежутки, когда на некоторый дисплей нужно что-то записать или выполнить дополнительные операции считывания (отличные от READ-TI, например прочесть полный буфер дисплея).

В основу логики монитора БЕС7920 заложена именно такая технология взаимодействия с дисплеями пользователей. При этом обязательным условием является то, что только монитор имеет право выдавать макрокоманду READ-TI. Тогда работу монитора можно сравнить с бесконечным циклом (петлей), изображенным на рис. 9.6.

После выполнения всех подготовительных операций (получения буфера из буферного пула, его, очистки, формирования блока DECB и т. п.) монитор входит в блок-петлю (точка 1) — выдает макрокоманду READ-TI для группы дисплеев, определенных в задании на запуск системы. Собственно говоря, макрокоманда READ-TI выдается не в мониторе, а в «процессоре операций ввода-вывода на дисплеи» (в дальнейшем будем называть его просто *процессором в-в*), к которому тот обращается. Сам факт передачи управления в другой загрузочный модуль здесь не существен. Следует отметить следующее. Процессор в-в является единственной компонентой системы,

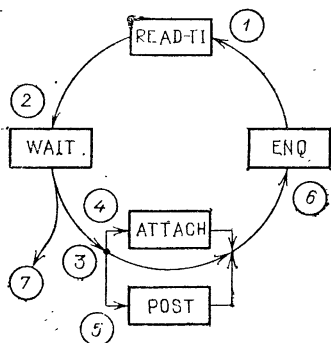


Рис. 9.6. Схема работы монитора BEC7920

осуществляющей какие-либо операции ввода-вывода на дисплеях. Поэтому в случае необходимости выполнить операцию записи или считывания все компоненты BEC7920 (монитор и программы, выполняемые в рамках подзадач пользователей и системных подзадач) сначала запрашивают у системы процессор в-в в качестве монопольно используемого ресурса системы. После того как затребованная у процессора в-в операция завершена, он освобождается. Захват

и освобождение процессора в-в осуществляются с помощью стандартных средств супервизора ОС ЕС — макрокоманд ENQ и DEQ. Поэтому, прежде чем выдать макрокоманду READ-TI, процессор в-в был получен у системы как монопольно используемый ресурс (макрокоманда ENQ), а перед переходом в точку 2 освобожден (макрокоманда DEQ). Об этом факте необходимо помнить при обсуждении логики дальнейшей работы монитора.

Итак, монитор переходит в состояние ожидания (точка 2), из которого он может выйти (попасть в точку 3) в двух случаях: пользователь одного из дисплеев нажал функциональную клавишу (выбрал фотоселектором поле внимания — в дальнейшем под активизацией функциональной клавиши будем подразумевать и этот случай) и инициировал считывание данных со своего дисплея либо одной из подзадач понадобилось выполнить операцию, отличную от READ-TI, для чего одна из ее программ выдала макрокоманду RESETPL. Могло случиться и так, что пересеклись две операции разных подзадач: в одной из них уже началось считывание данных, в то время как другой понадобилось выполнить, скажем, операцию записи на свой дисплей и она выдала макрокоманду RESETPL.

Если макрокоманда READ-TI что-то прочитала (независимо от того, пересеклась со считыванием RESETPL или нет), то монитор определяет относительный номер дисплея, выдавшего сигнал ВНИМАНИЕ, по нему отыскивает соответствующий этому дисплею элемент таблицы DST и проверяет, был ли до этого сигнала активным данный дисплей (ненулевое содержимое элемента DST). Если это начальный сигнал с данного дисплея (нулевое содержимое элемента DST), то осуществляется переход в точку 4; в противном случае — в точку 5. Если следствием окончания READ-TI была макрокоманда RESETPL и при этом не было чтения данных, то ни блок ATTACH, ни блок POST не активизируются и сразу осуществляется переход из точки 3 в точку 6.

Если от пользователя поступил начальный запрос на инициализацию сеанса работы (начальный сигнал ВНИМАНИЕ), то активизируется блок ATTACH, который при помощи одноименной макрокоманды создает стандартную подзадачу для дисплея, выдавшего сигнал ВНИМАНИЕ, запоминает адрес блока TCB порожденной подзадачи в первом слове элемента DST, соответствующего этому дисплею, и переходит в точку 6.

Если сигнал ВНИМАНИЕ поступил с активного дисплея, уже имеющего свою подзадачу, то активизируется блок POST, функции которого тривиальны — передать этой подзадаче буфер со считанными по READ-TI данными и оповестить (макрокоманда POST) ее о событии, которого она ждет (по макрокоманде WAIT). Этот механизм имеет связующее звено — элемент таблицы DST, с которым взаимодействуют и монитор, и подзадача пользователя: адрес входного буфера помещается во второе слово элемента DST, а адрес блока ECB, занесенный подзадачей пользователя в третье слово элемента DST, используется монитором и подзадачей пользователя для синхронизации совместных действий.

В точке 7, для того чтобы опять выдать макрокоманду READ-TI, монитор пытается получить процессор в-в в качестве ресурса, выдавая для этой цели макрокоманду ENQ. Если за время путешествия монитора из точки 2 в точку 6 ни одной из подзадач (системных или пользовательских) не было выдано ни одной RESETPL, то никто не мешает монитору получить этот ресурс, в результате чего произойдет переход в точку 1. Если все же в этот промежуток времени была выдана RESETPL, то это означает, что перед ее выдачей соперничающая подзадача получила в монопольное использование процессор в-в и будет его удерживать до тех пор, пока не выполнит свою операцию в-в на дисплее. До этого момента монитор будет находиться в состоянии ожидания (в точке 6) благодаря логике безусловного запроса ресурса.

Выход из этой петли (точка 7) возможен только при наличии следующих двух условий. Во-первых, в системе установлен (опера-

тором ЭВМ с консоли или пользователем с дисплеем) режим динамического окончания БЕС7920 — особое состояние системы, в котором нельзя активизировать пассивный дисплей, а, когда последний пользователь завершает свой сеанс работы, БЕС7920 автоматически завершает свою работу. Во-вторых, именно этот последний активный пользователь системы нажал в последний раз функциональную клавишу, в результате чего вывел монитор из состояния ожидания в точке 2.

Осталась одна немаловажная функция монитора, не уложившаяся в приведенную на рис. 9.6 схему, — вывод подзадачи пользователя из системы, когда сеанс работы пользователем завершен и дисплей должен быть переведен в пассивное состояние. Эту функцию в мониторе выполняет специальная (реинтерабельная) подпрограмма, которую в ОС ЕС называют программой выхода обработки конца задачи (программа ETXR). Адрес точки входа в эту программу указывается в макрокоманде ATTACH операндом ETXR при создании подзадачи пользователя. Эта программа получает управление всякий раз, когда одна из подзадач пользователей завершает свою работу в результате приема от пользователя команды ОКОНЧИТЬ СЕАНС или из-за аварийного завершения подзадачи. Программа ETXR выдает макрокоманду DETACH для блока TCB завершившейся подзадачи, удаляя ее тем самым из системы. Кроме того, элемент таблицы DST, соответствующий удаляемой подзадаче, очищается двоичными нулями, а на дисплей выводится начальное сообщение системы.

Таким образом, монитор работает только в те короткие промежутки времени, когда наблюдается активность пользователей. Остальное время он бездействует. Поэтому естественным является стремление [сделать минимальной его] постоянно резидентную часть — в БЕС7920 она (модуль IBZ7920M) занимает 1К байт ОП.

9.4.2. Подзадача связи с оператором ЭВМ. В любой ДСКД обязательной функцией системы является обеспечение возможности связаться ей самой и ее пользователям с оператором ЭВМ. Как и для любого инициированного в ОС ЕС задания роль оператора в таком взаимодействии традиционна — он должен смонтировать какие-то тома, необходимые пользователям, какие-то тома, наоборот, вывести из системы, следить за прохождением пакетных заданий пользователей ДСКД, выбирать те или иные режимы работы системы и т. п.

В БЕС7920 выбран самый простой способ связи с оператором ЭВМ — через макрокоманду супервизора WTOR. Порожденная главной задачей на этапе инициализации системы «Подзадача связи с оператором ЭВМ» выводит на главную консоль системы сообщение, позволяющее оператору при необходимости ввести в качестве ответа на него одну из команд связи с БЕС7920 (назовем это сообщение

сообщением связи). Подзадача связи с оператором ЭВМ находится в системе в течение всего времени функционирования БЕС7920, постоянно ожидая от оператора ЭВМ ввода команды связи. После того как она ее получает, она интерпретирует введенную команду связи; при необходимости эта подзадача запрашивает у оператора дополнительную информацию (например, текст сообщения, которое оператор хочет послать пользователю на его дисплей) и выполняет затребованную операцию. После этого подзадача связи с оператором ЭВМ повторно выводит на консоль сообщение связи и снова переходит в состояние ожидания.

Набор стандартных команд связи позволяет оператору выполнять различные операции, связанные с управлением системы. Так, например, можно послать сообщение конкретному пользователю или всем активным пользователям сразу, можно устанавливать и отменять различные режимы функционирования БЕС7920, получать информацию состояния БЕС7920 и т. п. Так как список команд связи оформлен в виде загрузочного модуля, список команд связи может быть изменен или расширен корректировкой исходного текста этого модуля и повторной генерацией загрузочного модуля.

Стандартным набором команд БЕС7920, выполняемых пользователем, предусмотрены средства, позволяющие пользователю послать со своего дисплея сообщение оператору ЭВМ и при необходимости получить от него ответ. Процессоры команд, обеспечивающие эти возможности, непосредственно взаимодействуют с подзадачей связи с оператором ЭВМ.

9.4.3. Формирование пакетных заданий и подзадача запуска пакетной обработки. Напомним, что для того, чтобы выполнить любое пакетное задание в ОС ЕС, его прежде всего необходимо поместить во входную системную очередь (набор данных SYS1.SYSJOBQE). Для этой операции обычно используется программа системного ввода IEFIRC, активизируемая запуском одной из каталогизированных процедур — RDR, RDR400, RDR3200 или другой.

В БЕС7920, прежде чем задания пользователей будут размещены во входной очереди ОС ЕС, они накапливаются в промежуточной очереди, обеспечиваемой БЕС7920 (в дальнейшем — очередь БЕС7920). Очередь БЕС7920 представляет собой последовательный набор данных, в котором задания располагаются последовательно в порядке поступления заявок на запись в него. Перед записью в очередь информация блокируется в буфере в ОП по двадцать 80-байтовых логических записей в блоке. Первоначально очередь пуста, а в процессе функционирования БЕС7920 сюда последовательно записываются все задания, направляемые пользователями на пакетную обработку.

Для формирования задания и постановки его в очередь БЕС7920 можно использовать одну из четырех команд: ПОСТАВИТЬ ЗАДА-

НИЕ В ОЧЕРЕДЬ, ОБНОВИТЬ РАЗДЕЛ, ДОБАВИТЬ РАЗДЕЛ и КОПИРОВАТЬ РАЗДЕЛ. В последних трех командах для этой цели указывается специальная форма операнда ВЫХОД, который обычно идентифицирует набор данных, куда записываются отредактированные данные, а в этом случае кодируется как $\text{ВЫХОД} = Q$.

Команды ПОСТАВИТЬ ЗАДАНИЕ В ОЧЕРЕДЬ и ДОБАВИТЬ РАЗДЕЛ с операндом $\text{ВЫХОД} = Q$ по алгоритму работы в точности совпадают (они имеют различные наборы сервисных функций). Сначала процессор любой из этих двух команд в цикле, кадр за кадром, считывает по запросу пользователя (клавиша ВВОД) подготовленные им в РП экрана данные, редактирует и записывает их в *выходной рабочий набор данных* (ВЫРНД) для промежуточного хранения. Так продолжается до тех пор, пока не будет получен сигнал КОНЕЦ ВВОДА (клавиша ПД2). По этому сигналу подзадача пользователя запрашивает очередь у системы в качестве монопольно используемого ресурса и, получив его, переписывает в конец очереди содержимое ВЫРНД. После переписи ресурс освобождается, и очередь становится доступной другим подзадачам; выполнение команды на этом завершается.

Описанный алгоритм является также заключительной фазой выполнения команды ОБНОВИТЬ РАЗДЕЛ с операндом $\text{ВЫХОД} = Q$. Однако эта команда дает возможность пользователю сформировать задание из текста, уже имеющегося в одном из его наборов данных. Это может быть раздел одной из его библиотек, последовательный набор данных или *входной рабочий набор данных* (ВРНД) пользователя, закрепленный за его дисплеем в качестве буфера для промежуточного хранения входных данных. В начальной фазе выполнения процессор команды копирует (при необходимости) данные из набора данных пользователя в ВРНД, а затем выводит их на экран для редактирования. Если в качестве входного набора указан ВРНД (операнд $\text{ВХОД} = *$), то операция копирования не выполняется.

Наконец, если пользователь хочет поставить в очередь задание, находящееся в его наборе данных или в ВЫРНД, без промежуточного редактирования, то скорее всего он воспользуется командой КОПИРОВАТЬ РАЗДЕЛ с операндом $\text{ВЫХОД} = Q$. Это наиболее быстрый способ доставки задания в очередь, но он пригоден только для некорректируемых текстов заданий.

Схемы движения задания к очереди БЕС7920 в различных модификациях команд показаны на рис. 9.7.

Процесс накопления заданий в очереди БЕС7920 продолжается до тех пор, пока с одного из дисплеев или с консоли оператора ЭВМ не поступит запрос на запуск пакетной обработки. Это может быть команда ЗАПУСТИТЬ ПАКЕТНУЮ ОБРАБОТКУ, введенная пользователем с дисплея, или команда связи RDR, введенная

оператором в качестве ответа на сообщение связи. В обоих случаях активизируется (выводится из состояния ожидания) подзадача запуска пакетной обработки.

На этапе инициализации системы функции подзадачи запуска пакетной обработки просты — она открывает набор данных, отведенный под очередь БЕС7920, подготавливая его для приема пакетных заданий пользователей, и, отметив это событие в главной задаче, выдает макрокоманду WAIT.

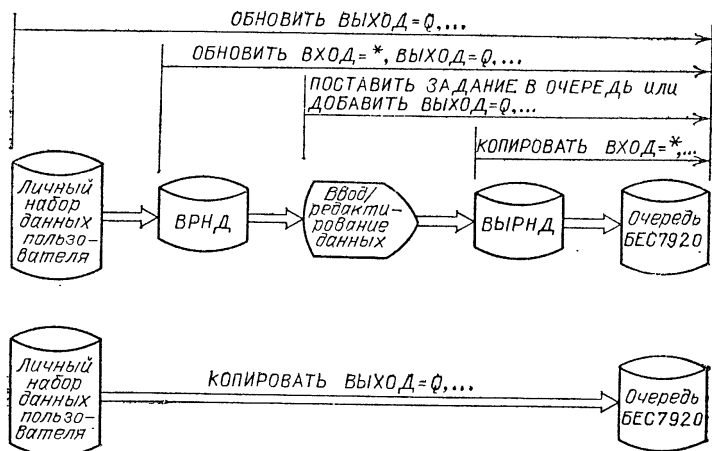


Рис. 9.7. Схемы движения задания при выполнении различных команд БЕС7920

Выведенная из состояния ожидания командой пользователя или командой оператора ЭВМ, она выполняет уже другие действия.

Прежде всего она пытается монопольно захватить очередь БЕС7920 как ресурс системы. Если к этому моменту у системы уже были запросы на этот ресурс (т. е. имеются подзадачи пользователей, желающие дописать свои задания в очередь), то подзадача будет ждать до тех пор, пока все запросы не будут обслужены и очередь не будет предоставлена ей в монопольное использование. Получив ресурс, она прежде всего проверяет, есть ли в очереди хоть одно задание. Если очередь пуста, то она отмечает это событие соответствующим кодом в блоке ЕСВ подзадачи, запросившей запуск пакетной обработки, и на этом прекращает выполнение каких-либо дальнейших операций, переходя в состояние ожидания. В противном случае она закрывает очередь БЕС7920 и приступает к выполнению очередной операции — динамическому запуску программы системного ввода ОС ЕС (программа IEFIRC).

Запуск программы IEFIRC производится имитацией соответствующих действий оператора ЭВМ, когда он вводит команду START. Например, для считывания входного потока заданий, находящегося в наборе данных JCL на томе IBZVOL (на устройстве прямого доступа), он мог бы ввести команду:

S RDR.R,SYSDA,IBZVOL,DISP=(OLD,DELETE),
DSN=JCL

Нечто подобное выполняет и подзадача запуска пакетной обработки. Она динамически формирует текст команды START для запуска каталогизированной процедуры RDR1600, подставляя в него регистрационный номер тома, ту или иную диспозицию очереди и имя набора данных, используемого в качестве очереди БЕС7920. Затем сформированная в ОП команда START запускается с помощью команды SVC 34 при нулевом ключе защиты программы, и функции подзадачи по запуску пакетной обработки на этом завершаются. Программа, выполняющая указанные действия:

Динамический запуск процедуры системного ввода

JFCB 9 *фиктивная секция для блока 'JFCB'*

*** СЧИТЫВАНИЕ БЛОКА 'JFCB' ОЧЕРЕДИ

RDJFCB (QDCB)		<i>читать 'JFCB'</i>
LA 9,AREA176		<i>R9—база 'JFCB'</i>
MVI TEMPAR,C'		<i>очистить пробелами перем. часть текста</i>
MVC TEMPAR+1(65),TEMPAR		
LA 6,20		<i>иниц-ть счетчик длины текста</i>
LA 7,TEMP		<i>адрес начала формируемой части</i>
MVC 0(6,7),JFCBVOLS		<i>рег. ном. тома --> в текст команды</i>
IBZDSNL (1),JFCBVOLS,MAXL=6		<i>подсчитать длину имени</i>
AR 6,1		<i>увеличить сч. на длину имени</i>
AR 7,1		<i>продвинуть адрес</i>
MVC 0(11,7),=C',DISP=(OLD,'		
LA 6,11(6)		
LA 7,11(7)		
MVC 0(10,7),=C'KEEP),DSN='		
LA 1,10		
TM JFCBIND2,1		<i>Q—временный набор?</i>
BZ *+14		<i>нет</i>
MVC 0(12,7),=C'DELETE),DSN='		
LA 1,12		

AR	8,1	
AR	7,1	
MVC	0(44,7),	<i>переслать имя очереди</i>
	JFCBDSNM	
IBZDSNL	(1), JFCBDSNM,	<i>подсчитать дл. имени</i>
	MAXL=44	
LA	6,4(6,4)	
STH	6,STRTPARM	

*** ДИНАМИЧЕСКИЙ ЗАПУСК КОМАНДЫ 'START'

MODESET	KEY=ZERO,	<i>уст. 0-й ключ защиты</i>
	MODE=SUP	
LA	1,STRTPARM	<i>R1=параметры ко-</i>
		<i>манды 'START'</i>
SVC	34	
MODESET	KEY=NZERO,	<i>восст. ключ защиты</i>
	MODE=PROB	

.....

STRTPARM	DC	H'88',H'0',CL20'S RDR1600,IBZ,
		SYSDA'
TEMPAR	DC	CL68' '
QDCB	DCB	DSORG=PS,DDNAME=QUE,EXLST=
		QEXLST
QEXLST	DC	X'87',AL3(AREA176)
AREA176	DC	XL176'0'

Здесь используются две макрокоманды BEC7920 — JFCB и IBZDSNL. По первой из них создается фиктивная секция JFCB (базируемая регистром 9), обеспечивающая символическую адресацию полей блока JFCB. Вторая макрокоманда подсчитывает длину любого имени, указанного вторым операндом, и помещает ее в регистр, определенный первым операндом. Ключевой операнд MAXL определяет максимально допустимую длину имени. Подсчет длины осуществляется до первого пробела в пределах поля, ограниченного значением MAXL.

«Старая» очередь BEC7920 после того, как задания из нее будут переписаны во входную системную очередь, не возвращается в среду BEC7920 — в зависимости от указанной в команде START диспозиции она будет либо уничтожена, либо сохранена. Поэтому очередной функцией подзадачи является обеспечение системы «новой» очередью.

Новая очередь создается динамическим вызовом программы распределения памяти на устройствах прямого доступа ОС ЕС (программа DASDM). Входными параметрами для этой программы являются адрес блока управления устройством (UCB), на котором расположен том магнитных дисков (регистр 1), и адрес 176-байтовой области ОП, содержащей блок управления файлом задания JFCB для распределяемого набора данных (регистр 0). Таким образом,

подзадаче необходимо отыскать том магнитных дисков, на котором можно было бы разместить новую очередь, и сформировать для нее блок JESB.

Что касается выбора тома, то сначала проверяется том, на котором была расположена старая очередь БЕС7920. Если на нем есть свободный участок подходящих размеров, то новая очередь создается на этом томе. В противном случае просматриваются другие тома памяти, находящиеся в системе в оперативном состоянии. Здесь опускается рассмотрение варианта, когда нет ни одного тома, подходящего для размещения очереди,— этот случай обрабатывается системой.

Для формирования второго входного параметра для программы DASDM используется считанный ранее блок JFCBDSNM (имя набора данных) и поле JFCBVOLS (регистрационный номер тома). В поле JFCBDSNM заносится имя, формируемое в следующем виде.

IBZ. ГГДДД. ЧЧМСС. QUE#НН

где ГГДДД — текущая дата, получаемая у супервизора ОС ЕС; ЧЧМСС — текущее время суток, НН — порядковый номер поколения очереди. Загрузив регистры 0 и 1 соответствующими параметрами, управление (по команде SVC 32) передается программе DASDM. Результат выполнения операции определяется по коду завершения программы DASDM, возвращаемый ею в регистре 15. В табл. 9.4 перечислены все коды завершения программы DASDM,

Т а б л и ц а 9.4

Основные коды завершения программы DASDM

Код	Значение
X'00'	Нормальное завершение; внешняя память получена
X'04'	Набор данных с указанным именем уже есть на томе
X'08'	Нет места во VTOC тома
X'0C'	Неисправимая ошибка ввода-вывода
X'10'	Абсолютная дорожка не доступна
X'14'	На томе нет свободного участка запрошенных размеров
X'18'	Некорректный атрибут «длина записи»
X'1C'	Некорректный атрибут DSORG или DISP
X'30'	Неправильное количество дорожек в запросе разделения цилиндров
X'34'	Неправильный указатель блока JFCB
X'38'	Участок, запрошенный под оглавление, больше, чем размер свободного участка на этом томе
X'4C'	Не определены параметры распределения для нового набора данных

имеющие смысл при распределении памяти для библиотечного или последовательного набора данных, а ниже приведен фрагмент программы, запрашивающей внешнюю память под очередь БЕС7920.

Создание новой очереди БЕС7920

JFCB 3, *генерация фикт. секции для 'JFCB'*

LA 3, AREA176 *R3=база фикт. секции*

.....

*** ПОИСК ПОДХОДЯЩЕГО ТОМА НМД

.....

LINK EP=IBZSETVL *поиск тома*
LR 9,1 *адрес блока 'UCB'*

*** СФОРМИРОВАТЬ ИМЯ НАБОРА В ВИДЕ 'IBZ.ГГДДД.
ЧЧММСС.QUE#XX'

MVI JFCBDSNM,C' '
MVC JFCBDSNM+1(43), JFCBDSNM
MVC JFCBDSNM(4),=C'IBZ.'
TIME
ST 1,WORD
UNPK JFCBDSNM+4(5),WORD
OI JFCBDSNM+8,X'F0'
MVI JFCBDSNM+9,C'.'
ST 0,WORD
UNPK JFCBDSNM+10(7),WORD
MVI JFCBDSNM+16,C'.'
MVC JFCBDSNM+17(4),=C'QUE#'
AP COUNT,=P'1' *+1 --> в номер поколения очереди*
UNPK JFCBDSNM+21(2),COUNT
OI JFCBDSNM+22,X'F0'
MODESET KEY=ZERO,MODE=SUP
LR 0,3 *R0=адрес блока 'JFCB'*
LR 1,9 *R1=адрес блока 'UCB'*
SVC 32 *выполнить программу 'DASDM' ОС ЕС*
LR 8,15 *сохранить код завершения 'DASDM'*
MODESET KEY=NZERO,MODE=PROB
B *+4(8) *перейти на соотв. блоки*
B OK *нормальное завершение*
B DUPLICAT *имя уже есть на томе*

.....

B NOSPACE *не определены параметры распр-я*

.....

WORD	DS	F	<i>рабочее слово</i>
AREA176	DC	XL176'0'	<i>область, содержащая 'JFCB' старой очереди</i>
COUNT	DC	P'1'	<i>счетчик номера поколения очереди</i>

Создав новую очередь БЕС7920, подзадача запуска пакетной обработки настраивает соответствующие поля в управляющей векторной таблице на новую очередь и переходит в состояние ожидания новой команды запуска пакетной обработки.

Программа системного ввода IEFIRC, инициированная запуском каталогизированной процедуры RDR1600, выполняется асинхронно по отношению к подзадаче запуска пакетной обработки. Поэтому в ОС может одновременно присутствовать несколько активных программ системного ввода, использующих очереди БЕС7920. Каждая из программ IEFIRC прекращает свое существование и выводится из системы сразу же, как только ею будет прочитана запись «конец файла» из входного потока (очереди БЕС7920). К сожалению, обратной связи с программой IEFIRC нет, поэтому узнать программным способом о результатах ее выполнения невозможно.

Несколько слов о процедуре RDR1600. Это — каталогизированная процедура БЕС7920, отличающаяся от стандартной процедуры системного ввода ОС ЕС размером блока ее наборов данных: для набора данных входного потока (IEFRDER) и для набора данных, используемого для промежуточного хранения данных входного потока (IEFDATA), размер блока установлен 1600 байт.

9.4.4. Планировщик сеанса работы пользователя. После запуска БЕС7920 на экране каждого дисплея, определенного в задании, должен быть отображен начальный кадр системы, свидетельствующий о том, что данный дисплей поддерживается системой. Этот же кадр выводится на экран и после окончания пользователем своего сеанса работы или в результате аварийного завершения подзадачи пользователя. Вывод этого кадра, как уже было сказано выше, является функцией главной задачи.

Если с какого-либо дисплея, находящегося в этом (пассивном) состоянии, принят сигнал ВНИМАНИЕ (т. е. на дисплее нажата некоторая функциональная клавиша), то монитор создает подзадачу пользователя для данного дисплея. Первой программой, которая получает управление, когда подзадача становится активной, является *планировщик сеанса работы пользователя* (в дальнейшем просто — планировщик сеанса). Прежде всего эта программа выводит на «свой» экран сообщение, предлагающее альтернативу — либо перевести дисплей в режим обучения (нажатием клавиши ПД1), либо приступить к инициализации сеанса (нажатием клавиши СТРН ЭКР).

Если пользователь выбирает режим обучения, он получает доступ только к текстовой информации HELP-функции системы, при просмотре которой его активность ограничена функциональной клавиатурой. В этом режиме ни одна команда БЕС7920 и ни один ресурс системы ему не доступны и не могут быть им модифицированы. Выйти из режима обучения можно в любой момент, нажав клавишу СТРН ЭКР.

Однако активная работа пользователя может быть начата им только после инициализации сеанса работы. Для этого пользователь, прежде чем он сможет ввести первую команду, должен *идентифицировать себя* (сначала вводом своего псевдонима, а затем пароля). *Псевдоним* и *пароль* (один или несколько) пользователя — это произвольные последовательности знаков, гравировка которых имеется на алфавитно-цифровой клавиатуре дисплея. Длина и псевдонима, и пароля не должна превышать восьми знаков. Каждому псевдониму можно поставить в соответствие до семи паролей.

Псевдоним и пароль вводятся последовательно по запросу системы, причем ввод пароля осуществляется в неотображаемое поле. В случае ошибочного ввода планировщик сеанса предоставляет возможность повторить его. После трехкратного неправильного ввода (либо псевдонима, либо пароля) планировщик сеанса «отправляет» пользователя к системному программисту за консультациями, а сам иницирует псевдоокончание сеанса работы. Подзадача пользователя выводится главной задачей из системы, а дисплей переводится в пассивное состояние.

На основании чего же система идентифицирует пользователя? По этому поводу в БЕС7920 приняты следующие соглашения. Каждый, кто пожелает вступить в контакт с системой, должен быть предварительно зарегистрирован в ней в качестве пользователя БЕС7920. Информация, характеризующая отдельного пользователя, с помощью специальных обслуживающих компонент системы заносится в *библиотеку характеристик пользователей* (набор данных IBZ, PASSWORD). Каждому пользователю в этой библиотеке соответствует отдельный раздел, имя которого является псевдонимом абонента и хранится в *справочнике библиотеки*. Кроме псевдонима в элементе справочника содержатся все пароли и *код формы доступа* (КФД), присвоенные пользователю.

В качестве данных раздел содержит 80-символьные несблокированные записи, каждая из которых идентифицирует один так называемый *корень имен наборов данных*, с которыми пользователь может работать, и место (*регистрационный номер тома*), где эти наборы данных размещены. (Ниже мы поясним, что понимается под «корнем имен наборов данных».) Кроме того, в каждой записи присутствует *индикатор вида обработки* наборов данных. Последней записью раздела является запись «*конец данных*», содержащая в

первых восьми байтах все двоичные единицы. Если пользователю разрешены в КФД и чтение, и модификация любых наборов данных, то раздел может содержать единственную запись «конец данных». На рис. 9.8 показана структура элемента справочника набора данных IBZ.PASSWORD и форматы записей разделов.

Форма доступа пользователя регулирует доступ пользователя к данным, а также санкционирует использование команд администратора (команд ограниченного применения). Форма доступа кодируется одной из следующих букв: В, С, F, G, K, L, O, P, а интерпретируется системой на основании двоичной структуры этих символов (в коде ДКОИ). Символ КФД отображается в управляющем

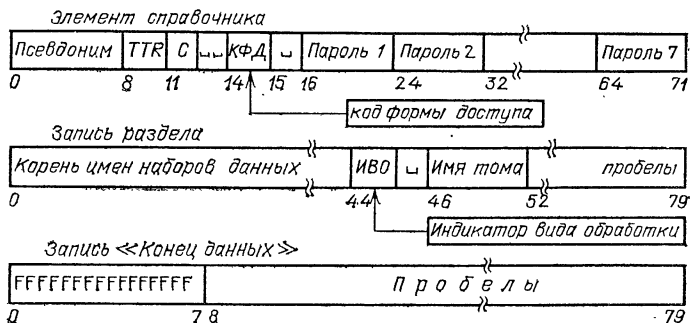


Рис. 9.8. Структура библиотеки характеристик пользователей

поле системы (см. рис. 9.1) после завершения процедуры инициализации сеанса. На рис. 9.9 показана структура КФД и перечислены допустимые в БЕС7920 коды (в символьном и двоичном изображении).

Рассмотрим пример. Пусть пользователю с псевдонимом USER0001 и двумя паролями ИВАНОВ и ПРОФЭСОР разрешено читать любые наборы данных, а модифицировать только те, имена которых начинаются символьной последовательностью (иначе говоря, имеющими корень) LAB22.GROUP1. и которые расположены на томах с регистрационными номерами LAB22 и LAB23. Кроме того, пользователь может произвольным образом использовать набор данных LAB22.COMMON, находящийся на томе LAB22. Наконец, пользователю запрещено использовать команды администратора. Тогда элемент оглавления набора данных IBZ.PASSWORD, соответствующий данному пользователю, и записи раздела имели бы структуру, изображенную на рис. 9.10.

9.4.5. Процессор операций ввода-вывода на дисплее, обработчик сигнала ВНИМАНИЕ и К-модули. В условиях многотерминального обслуживания для организации последовательного доступа из функциональных программ системы к «своим» дисплеям необходим некий программный интерфейс между этими программа-

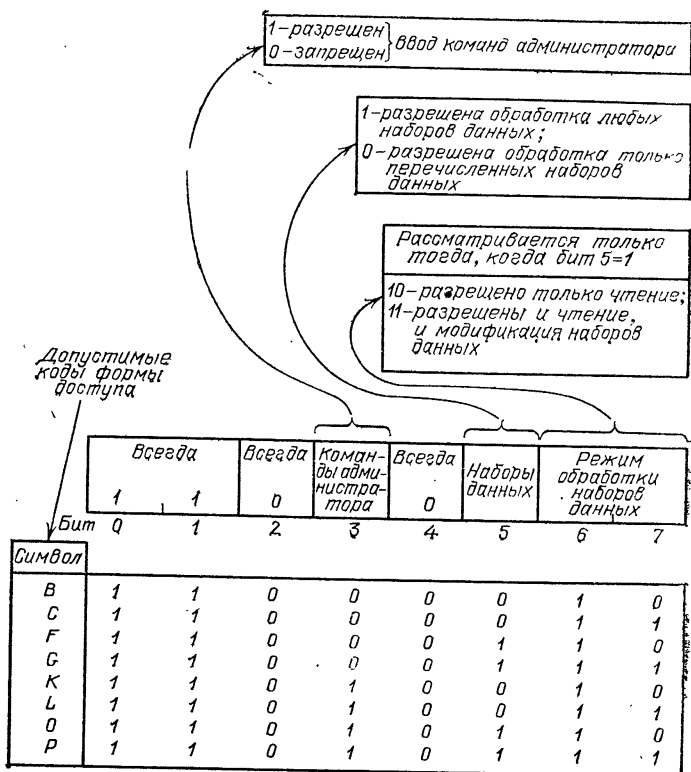


Рис. 9.9. Структура кода формы доступа пользователя

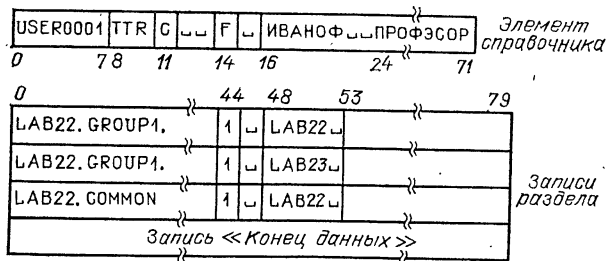


Рис. 9.10. Пример организации информации, характеризующей одного пользователя БЕС792)

ми и логикой БТМД. Это в основном связано со спецификой обработки сигнала ВНИМАНИЕ в БТМД, обсуждавшейся выше, и необходимостью избавить функциональные программы от учета этой специфики. Другими словами, функциональные программы должны иметь возможность монопольно связываться со своим дисплеем, считая, что он единственный в системе.

В БЕС7920 роль такого интерфейса выполняют три программные компоненты, обеспечивающие высокий логический уровень программирования операций ввода-вывода для дисплеев. Одна из них уже рассматривалась — это монитор системы, функционирующий в рамках главной задачи. Основная его функция — идентификация источника (дисплея) сигнала ВНИМАНИЕ и активизация соответствующей подзадачи пользователя, ожидающей появления этого сигнала со своего дисплея. Кроме вывода этой подзадачи из состояния ожидания монитор передает ей буфер со считанными по READ-TI данными и настраивается на прием очередного сигнала ВНИМАНИЕ. Вторая компонента, — «процессор операций ввода-вывода на дисплее» — является единственным звеном системы, непосредственно взаимодействующим с логикой (в частности, с процессором в-в) БТМД. Основная его функция — трансляция любой заявки на ввод-вывод, поступающей из программ БЕС7920, в один из эквивалентов макрокоманд чтения-записи БТМД, инициализация и первичная обработка выполнения ее. Третья компонента — «обработчик сигнала ВНИМАНИЕ» — является средством для перевода активной подзадачи пользователя в состояние ожидания до появления сигнала ВНИМАНИЕ с того дисплея, с которым эта подзадача взаимодействует.

Так как основным состоянием тракта терминального ввода-вывода системы является запрос на чтение, выданный монитором по макрокоманде READ-TI, то для снятия этого запроса из подзадач пользователей используется вспомогательная программа, занимающая промежуточное место между функциональными программами и процессором в-в. Это — *предпроцессор в-в*, основными функциями которого являются захват в монопольное использование логической цепочки «функциональная программа — процессор в-в — процессор в-в БТМД — дисплей» и разрыв связи «процессор в-в — монитор» с помощью макрокоманды RESETPL. В дальнейшем предпроцессор в-в будем рассматривать как специальный вход процессора в-в для подзадач пользователей.

На рис. 9.11 показана взаимосвязь перечисленных компонент и выделены различные логические уровни их взаимодействия (предполагается, что в системе активизированы три дисплея, для каждого из которых уже создана своя подзадача пользователя).

Для взаимодействия подзадач пользователей с процессором в-в используются две макрокоманды БЕС7920 — IBZRD (читать) и

IBZWRT (писать). Смысл операнда «тип операции» и результаты выполнения этих макрокоманд полностью идентичны соответствующим модификациям макрокоманд чтения и записи БТМД. Использование операции ЧТЕНИЕ НАЧАЛЬНОЕ в подзадачах пользователя запрещено. Макрорасширения команд IBZRD и IBZWRT заполняют соответствующими спецификациями поля блока DECB

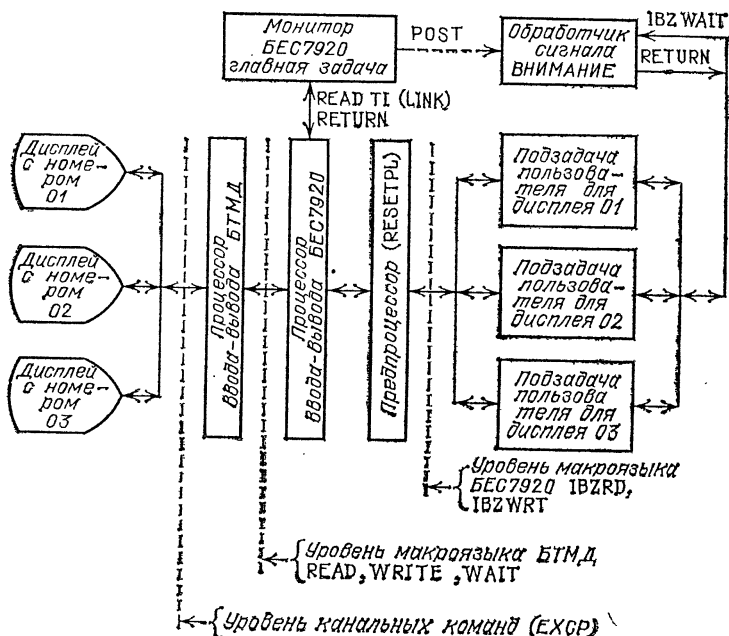


Рис. 9.11. Логические уровни организации ввода-вывода для дисплеев в BEC7920

и по макрокоманде LINK динамически связываются с процессором в-в, передавая ему в качестве параметра блок DECB. Поскольку управление передается предпроцессору в-в, в нем сбрасывается запрос на ЧТЕНИЕ НАЧАЛЬНОЕ, сделанный монитором системы, процессор в-в захватывается в качестве монополюсильно используемого ресурса и только потом осуществляется переход в процессор в-в.

Процессор в-в транслирует запрос подзадачи пользователя на язык БТМД, инициирует выполнение заказанной операции, выдавая соответствующую макрокоманду БТМД, ждет ее завершения, анализирует результаты ее выполнения, организует обработку ошибок ввода-вывода и в случае необходимости повторяет операцию, завершившуюся с ошибкой. Если заявка поступила из монитора

системы (а это может быть запрос только на ЧТЕНИЕ НАЧАЛЬНОЕ), процессор в-в, выполняющийся в данном случае как программа главной задачи, прежде чем перейти в состояние ожидания, от имени главной задачи освобождает сам себя как монопольно используемый ресурс, давая возможность захватить себя задаче, выдающей RESETPL. Выйти из состояния ожидания в рамках главной задачи он может либо в результате выдачи макрокоманды RESETPL из предпроцессора в-в, либо при появлении сигнала ВНИМАНИЕ.

Если подзадача пользователя в соответствии с ее программной логикой должна перейти в состояние ожидания и ждать запроса пользователя на обслуживание, она выдает макрокоманду IBZWAIT. По этой макрокоманде управление передается «обработчику сигнала ВНИМАНИЕ» (в дальнейшем — обработчику ВН), который, выполняясь как программа данной подзадачи, переводит ее в состояние ожидания. При появлении сигнала ВНИМАНИЕ процессор в-в возвращает управление монитору системы, а тот, определив относительный номер дисплея, пославшего ВНИМАНИЕ, а по нему и подзадачу, которую необходимо вывести из состояния ожидания, отмечает это событие в соответствующем блоке ЕСВ подзадачи пользователя, тем самым активизируя обработчик ВН, выполняющийся в рамках этой подзадачи.

Каким же образом внутри отдельной подзадачи обрабатывается сигнал ВНИМАНИЕ и как влияют различные источники этого сигнала на поведение обработчика ВН?

Здесь необходимо напомнить о том, что первым символом считанных с дисплея данных является так называемый идентификатор ВНИМАНИЯ (ИВ) — односимвольный код, определяющий источник полученного сигнала. Так как вместе с активизацией обработчика ВН монитор передает ему и буфер со считанными данными, то ИВ становится доступным обработчику ВН.

В БЕС7920 обработка сигнала ВНИМАНИЕ выполняется в соответствии со следующими соглашениями. В любой момент времени каждой активной подзадаче поставлена в соответствие одна «таблица переходов по сигналу ВНИМАНИЕ» (для краткости — таблица ВН) и шкала блокировки источников сигналов ВНИМАНИЕ (шкала ВН). Адрес таблицы ВН находится в поле ДСТАЕТВА таблицы управления дисплеем DCT, а шкала — это 32-разрядное слово (поле DCTSCALE), старшие 20 бит которого используются в качестве ключей блокировки источников сигнала ВНИМАНИЕ. Каждый бит шкалы ВН (0—19) соответствует строго определенному элементу таблицы ВН. Таблица ВН состоит из 20 8-символьных элементов, каждый из которых соответствует определенной функциональной клавише и может содержать либо пробелы, либо звездочки, либо некоторое имя.

Приведем исходный текст таблицы ВН, используемый при выполнении команды ПРОСМОТРЕТЬ ОГЛАВЛЕНИЕ ТОМА (см. стр. 185).

Пробелы в элементе этой таблицы являются признаком того, что использование соответствующей функциональной клавиши запрещено; звездочки — это указание вернуть управление программе;

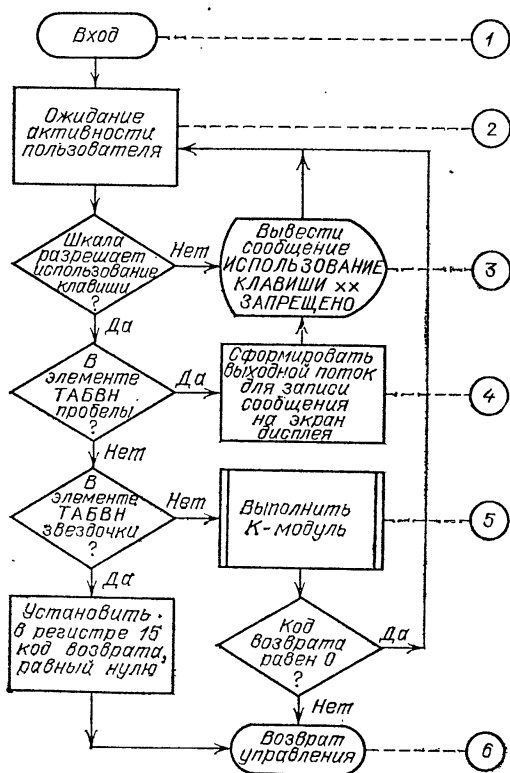


Рис. 9.12. Блок-схема обработки сигнала ВНИМАНИЕ. 1 — вход по макрокоманде IBZWAIT из любой программы, выполняющейся в рамках подзадачи пользователя; 2 — после вывода программы из состояния ожидания во входном буфере подзадачи первый байт — это идентификатор сигнала ВНИМАНИЕ; 3 — по макрокоманде IBZWRT управление передается процессору операций в-в на дисплее; 4 — поток выходных данных изменит на экране только ПСС (строка 2); 5 — К-модуль может сам писать на дисплей и читать с него, может обратиться к программе обработки сигнала ВНИМАНИЕ; может быть временно изменена шкала сигналов ВН; 6 — код возврата возвращается в регистре 15. Ненулевой код возврата может быть следствием выполнения только К-модуля

	DCT	2	
		
WAIT	LA	5,DCTDECB	адрес 'ЕСВ' из 'DCT'.
	WAIT	ЕСВ=(5)	ожидание
* В R7 НАХОДЯТСЯ СЧИТ. ДАННЫЕ (1-Й БАЙТ—ИДЕНТ. 'ВНИМАНИЯ')			
		
	SR	8,8	получить поряд. номер
	IC	8,0(7)	элемента таблицы переходов,
	N	8,=A(127)	соответствующего полученному
	SH	8,=H'107'	идентификатору 'ВН'
	L	6,DCTSCALE	шкала блокировки источника 'ВН'
	LR	14,8	
	LTR	14,14	Q—сдвигать шкалу нужно?
	BZ	*+12	нет—обойти блок сдвига шкалы
	SLD	6,1	да—сдвинуть шкалу так, чтобы
	BCT	14,*—4	искомый бит оказался нулевым
	SLL	8,3	
	CL	6,MAXMASK	Q—бит 0=1?
	BL	ILLEGAT	нет
	L	15,DCTAETBA	адрес таблицы —ВН—
	AR	8,15	адрес соотв. элем. таблицы
	CLI	0(8),C' '	Q—использование кл. запр.?
	BE	ILLEGAT	да
	XR	15,15	
	CLI	0(8),C'*'	Q—возврат в вызов. пр-му?
	BE	EXIT	да
	LINK	EPLOC=(8)	вызвать К-модуль
	LTR	15,15	Q—код возврата=0?
	BZ	WAIT	да—на 'WAIT'
EXIT	L	13,4(13)	возврат в вызывающую
	RETURN	(14,12),T,RC=(15)	программу
ILLEGAT	EQU	*	выдать сообщ. 'использование клавиши запрещ.'.
		
	DS	OF	
MAXMASK	DC	XL4'80000000'	константа для опр. знач. бита 0

выдавшей макрокоманду IBZWAIT, т. е. обратившейся к обработчику ВН. Если элемент содержит значение, отличное от пробелов и звездочек, то его содержимое рассматривается как имя модуля, которому необходимо передать управление с возвратом. При этом принято, что если модуль возвращает управление с ненулевым кодом завершения, то обработчик с этим же кодом завершения возвращает управление программе, обратившейся к нему. В противном случае обработчик ВН опять переводит подзадачу в состояние ожидания. В БЕС7920 модули, имена которых указаны в таблице ВН и работают по указанным соглашениям, называются модулями поддержки функциональных клавиш или просто К-модулями.

Поясним взаимодействие шкалы и таблицы ВН. Прежде всего в процессе функционирования системы таблицы ВН обычно не модифицируются, благодаря чему являются реентерабельными. Однако часто необходимо временно запретить использование той или иной клавиши. Для этих целей используется модифицируемая в процессе функционирования подзадачи шкала ВН. Если в шкале бит, соответствующий определенной клавише, равен нулю, то при обработке ИВ соответствующий элемент таблицы не рассматривается. Описанный алгоритм обработки сигнала ВН показан на рис. 9.12. Исходный текст обработчика сигнала ВНИМАНИЕ приводится на стр. 187.

Объясним некоторые детали этой программы. Входным параметром для обработчика ВН является адрес таблицы управления дисплеем DCT, из которой используются следующие поля: адрес блока управления событием ECB (поле DCTECB), в котором монитор по макрокоманде POST отметит событие, связанное со считыванием данных с дисплея; шкала ВН (поле DCTSCALE), перенесенная в таблицу DCT из таблицы команд (при инициализации выполнения команды) либо сформированная динамически в подзадаче; адрес таблицы ВН (поле DCTAETBA). Для символической адресации перечисленных полей используется макрокоманда DCT, единственный операнд которой указывает номер регистра, используемого для базирования имен из фиктивной секции, образуемой макрокомандой. При определении места элемента в таблице ВН, соответствующего принятому ИВ, используется тот факт, что элементы в таблице упорядочены в порядке возрастания младших семи бит кодов ИВ (X'6B', X'6C', ..., X'70', ..., X'7E').

9.5. HELP-функция как средство обучения и информационного обслуживания пользователей

Трудности, испытываемые пользователями, которые слабо знакомы с заданной интерактивной системой, наиболее часто характеризуются вопросами: «Где он (пользователь) находится?», «Что он может здесь сделать?», «Как он попал сюда?» и «Куда он может пе-

рейти и как он туда попадет?». Хорошо спроектированная система позволяет пользователю в любое время получить ясный ответ на перечисленные вопросы.

Роль подсказчика в большинстве интерактивных систем выполняют специальные компоненты, в качестве обобщенного названия которых часто используется термин «HELP-функция». В различных системах возможности, степень развития, программная реализация и информационное наполнение HELP-функции различны, однако в большинстве систем ее назначение, как правило, одинаково — предоставить пользователю некую справочную информацию в процессе его взаимодействия с системой.

В системе БЕС7920 кроме отдельной программной компоненты, специально предназначенной для оказания информационной помощи пользователю, в самом алгоритме диалогового взаимодействия с ним учтена возможность появления асинхронных запросов на информационную помощь. Кроме того, в системе принят ряд соглашений по организации информационной базы HELP-функции, которым должны следовать как программные компоненты БЕС7920 и диалоговые прикладные программы пользователя, так и сам пользователь, включающий свою текстовую информацию в информационную базу.

В БЕС7920 HELP-функция может быть активизирована как по запросу пользователя (явный запрос), так и самой системой (неявный запрос). Явный запрос ассоциируется в системе со следующими событиями: нажата клавиша ПД1 (в БЕС7920, кстати, используется только для этих целей), введена «пустая» команда, введена любая команда без операндов, введен приказ HELP. Неявные запросы связаны, как правило, с ошибками пользователя. Реакцией системы на ошибки пользователя обычно является вывод сообщения в ПСС экрана (строка 2). Здесь в основном будут рассмотрены алгоритмы работы HELP-функции, активизируемой в результате явных запросов.

Характер информации, выдаваемой пользователю на экран его дисплея, зависит от того, каким способом и в какой момент активизирования HELP-функции.

Клавиша ПД1. Если клавиша ПД1 нажата в тот момент, когда сеанс работы на дисплее еще не начат, т. е. дисплей находился в пассивном состоянии, то для данного дисплея устанавливается режим обучения. Этот режим характеризуется тем, что пользователь сам планирует процесс обучения: при желании он может просмотреть все, что имеется в информационной базе HELP-функции (хотя сделать это достаточно сложно — в настоящее время это около 2000 кадров), или (что более реально) может выбрать интересующее его направление обучения, избрав один из предложенных ему вариантов в кадрах типа меню. Выбор варианта по дисциплине меню осуществ-

ляется при помощи клавиш. программного функционирования ПФ1—ПФ12.

Если клавиша ПД1 нажата в процедуре ввода команды, пользователь получит на экране своего дисплея список всех команд редактора с указанием их мнемонических кодов. Он может просмотреть информацию об интересующей его команде, выбрав ее по указанной клавише ПФ, или «пролистать» информацию обо всех имеющихся в списке командах, последовательно нажимая клавишу ПД1.

Клавиша ПД1, активизированная уже в процессе выполнения команды, приводит к следующим последствиям. Последний рабочий кадр, отображаемый на экране дисплея, запоминается, и на экран выводится информация о выполняемой команде. Информационное содержание помощи в этот момент может быть различным, однако, как правило, первым кадром является опять же меню, предлагающее на выбор информацию на темы: «каковы функциональные возможности выполняемой команды», «какие приказы можно использовать», «какие функциональные клавиши и какие соответствующие им программные функции можно использовать», «какие особые ситуации могут встретиться при выполнении данной команды».

Информация по каждой теме также оформлена в виде списка или перечня, т. е. в виде меню, поэтому пользователю не трудно быстро «добраться» до той специфической информации, которая ему нужна. Здесь важно то, что пользователь не пользуется ни одной алфавитно-цифровой клавишей (а только функциональной клавиатурой), что сводит к минимуму вероятность внесения ошибки или неправильного выбора.

Ввод «пустой» команды. Если в процедуре ввода команды пользователь нажал клавишу ВВОД, не набрав никакой информации в поле ввода команды, эта ситуация расценивается системой как запрос на информационную помощь, и по аналогии с нажатием клавиши ПД1 в этом режиме на экран выводится список команд системы.

Ввод команды без операндов. Если пользователь ввел команду, не задав ни одного операнда, то на дисплее сохраняется режим ввода команды, т. е. режим обучения не устанавливается, и на экран выводится так называемый макет команды. Кроме самого макета, содержащего код команды, все предусмотренные в ней операнды и некоторые значения, принимаемые в них по умолчанию, в кадре отображаются содержательное название команды, краткие сведения обо всех операндах и правилах их кодирования. Используя клавиатуру, пользователь может отредактировать текст команды, изменив значения некоторых операндов и определив недостающие значения в операндах, не имеющих подразумеваемых значений. Затем, нажав клавишу ВВОД, он может сразу же инициировать выполнение данной команды.

Приказ HELP. В тех командах, в процессе выполнения которых допускается обработка] приказов (подкоманд), HELP-функция может быть активизирована вводом приказа HELP. С помощью этого приказа можно указать конкретную тему, интересующую пользователя. Он может узнать алгоритм работы выполняемой команды, порядок использования функциональной клавиатуры, перечень приказов, применимых для данной команды. Кроме того, можно запросить подробную расшифровку сообщения, отображенного в ПСС экрана, а также любого сообщения, указав его идентификационный номер.

После просмотра справочной информации с помощью HELP-функции пользователь должен деактивизировать ее. Это выполняется нажатием клавиши СТН ЭКР. В результате на дисплее восстанавливается та операционная обстановка, которая предшествовала активизации HELP-функции. Например, если до активизации HELP-функции в РП экрана находилась некоторая рабочая информация, она будет восстановлена.

Процедура деактивизации HELP-функции бывает необходима и в тех случаях, когда пользователь, просмотрев некоторую цепочку кадров, хочет возвратиться в начало и начать просмотр информации на другую тему. Хотя средства организации информационной базы допускают рекурсию кадров, иногда такой способ «достижения вершины дерева» бывает более предпочтителен. Однако для пояснения последнего предложения необходимо рассмотреть организацию информационной базы HELP-функции и способов ее наполнения.

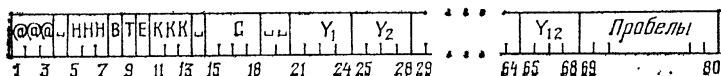
Информационная база HELP-функции физически представлена в системе в виде библиотеки IBZ.HELP. Библиотека состоит из разделов, имена которых — HELPxxxx, где HELP — стандартный префикс, xxxx — четырехсимвольный код раздела HELP-функции. Разделы, содержащие макеты команд, должны иметь имена HELPMxxx, где xxx — последние три символа имени процессора соответствующей команды. Все записи в библиотеке должны быть фиксированной, равной 80 байт длины.

Логическая организация информационной базы такова. Текст, предназначенный для отображения на экране в рамках HELP-функции и представленный в библиотеке отдельным разделом, может быть предварительно разбит на кадры или оставаться сплошным. Будем говорить, что раздел форматизован, если хранимый в нем текст разбит на кадры. Каждый кадр не должен содержать более 20 строк текста (исключение составляют макеты команд).

Форматизованные текстовые кадры могут быть двух типов — кадр типа меню (М-кадр) и кинокадр (К-кадр). Один раздел может содержать несколько кадров этих двух типов (вообще говоря, сколько угодно много кадров). Раздел, содержащий макет команды, должен быть представлен единственным текстовым кадром (не более 23 строк

текста), оформленным специальным образом. Кроме того, в библиотеке допускается наличие служебных, не отображаемых на экране разделов, содержащих цепочку ссылок к другим разделам (такой раздел назовем С-кадром). Каждый С-кадр представлен отдельным разделом и содержит одну-единственную запись.

Логическая взаимосвязь разделов в информационной базе и графицы текстовых кадров в разделе указываются с помощью оператора форматизации раздела — специальной управляющей записи следующего вида:



Символы @@@ в первых трех позициях записи являются признаком оператора форматизации. Колонки 5—7 содержат номер кадра внутри данного раздела. Назначение этого поля носит чисто иллюстративный характер. Колонки 8 и 10 (поля В и Е соответственно) могут содержать символы определения границ (начала и конца соответственно) текста, который необходимо отображать на экране с повышенной яркостью. Естественно, что используемые для этих целей символы не должны использоваться в тексте данного кадра в качестве информационных. Колонка 9 определяет тип кадра (С — С-кадр, М — М-кадр, — К-кадр, S — S-кадр). Поле КKK (колонки 11—13) может указывать количество кадров в данном разделе. Так же, как поле ННН, это поле служит для иллюстративности. Поле С (колонки 15—18) представляет собой иерархическую ссылку к разделу. Смысл этого поля будет объяснен ниже. Заметим лишь, что если код раздела в этом поле опущен или в нем указаны четыре символа «звездочка», то это интерпретируется как «тупик». Смысл четырехсимвольных кодов разделов Y_i зависит от типа кадра. В С-кадрах они определяют последовательность выборки разделов при каждом очередном нажатии клавиши ПД1. В М-кадрах каждый Y_i (i = 1, ..., 12) соответствует определенной клавише ПФ_i, т. е. при нажатии, скажем, клавиши ПФ2 будет выбран раздел с кодом, находящимся в колонках 25—28 оператора форматизации. В К-кадрах Y_i игнорируются.

Для каждого дисплея, на котором активизирована HELP-функция, программная логика сохраняет лишь по одному выбранному последним оператору каждого типа — К-типа, М-типа и С-типа, т. е. каждый очередной кадр, выбранный в результате ссылки, заменяет кадр того же типа, считанный ранее.

За кадрами различных типов закреплена следующая иерархия: К-кадр → С-кадр → М-кадр. Это означает следующее. Если последний выбранный кадр является К-кадром, а пользователь продолжает запрашивать очередные кадры информации (клавиша

ПД1), то очередной раздел будет выбран в соответствии с полем иерархической ссылки в операторе форматизации данного кадра. Если при этом в поле С указан «тупик» (пробелы или звездочки), то выбирается ссылка из поля С ранее считанного С-кадра (при условии, что все ссылки Y_i обработаны). Если и здесь указан «тупик», то выбирается ссылка из М-кадра, а если и она отсутствует, то на экран выводится текст стандартного К-кадра (раздела с именем HELPFFFF), информирующего пользователя о достижении конца цепочки кадров. Если и в этой точке будет нажата клавиша ПД1, то будет выбран стандартный начальный раздел с именем HELP0000. В этой точке пользователь может деактивизировать HELP-функцию или начать новый цикл просмотра.

Таким образом, в целом об информационной базе HELP-функции можно говорить как о совокупности древовидных и сетевых

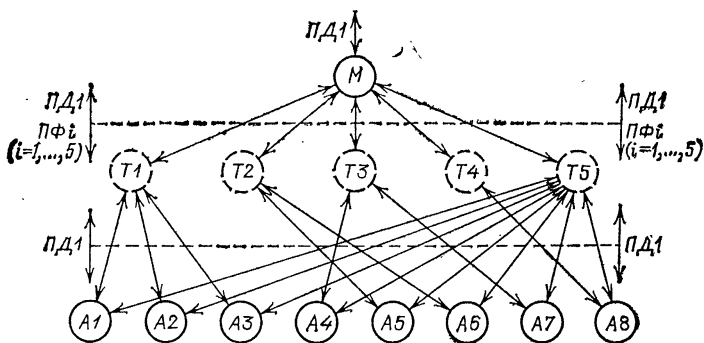


Рис. 9.13. Сетевая структура для литературных источников

структур, в которых отношения между данными описываются с помощью операторов форматизации различных типов.

Рассмотрим пример построения сетевой структуры, используя операторы форматизации. Предположим, что у нас есть восемь литературных источников, тексты которых разбиты на К-кадры и помещены в информационную базу HELP-функции. Во всех К-кадрах указаны «тупиковые» ссылки. Каждый литературный источник обозначим кодом A_i ($i = 1, \dots, 8$). Предположим далее, что в разделах A_1 , A_2 и A_3 есть информация на тему T_1 , в A_5 и A_6 — на тему T_2 , в A_4 и A_7 — на тему T_3 , в A_8 — на тему T_4 . Необходимо сформировать сеть, которая позволила бы пользователю на выбор получать информацию на каждую тему, а при желании последовательно просматривать все литературные источники. Указанную сетевую структуру можно описать схемой, представленной на рис. 9.13.

Верхний узел является меню (М-кадром), второй уровень содержит С-кадры, не отображаемые на экране. Роль их вспомогательна: они регулируют отношения в сети между М-кадрами и К-кадрами. Движение в указанных стрелками направлениях вызывают функциональные клавиши ПФ1—ПФ5 или ПД1. Очевидно, что все

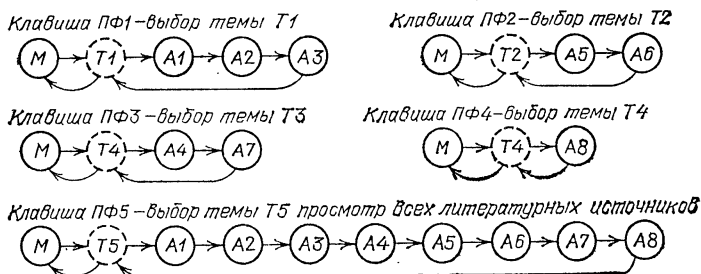


Рис. 9.14. Пути просмотра К-кадров в зависимости от выбора, сделанного в меню

пути, на которых встречаются отображаемые на экране кадры, могут быть представлены в виде простых цепочек, изображенных на рис. 9.14.

В соответствии с выбранной структурой операторы форматизации для разделов с кодами M, Ai и Tj могут быть представлены в следующем виде:

	Тип кадра	Иерархическая ссылка	Указатели разделов Y _i
A _i :	@@@	□	****
T1:	@@@	□	A1 □ A2 □ A3 □
T2:	@@@	□	A5 □ A6 □
T3:	@@@	□	A4 □ A7 □
T4:	@@@	□	A8 □
T5:	@@@	□	A1 □ A2 □ A3 □ A4 □ A5 □ A6...
M:	@@@	□	T1 □ T2 □ T3 □ T4 □ T5 □
Колонки:	1 3 9	15 18	21 25 29 33 37 41

Ссылка к разделу, не показанному в данной структуре

9.6. Перспективы развития системы БЕС7920

Вместе с включением некоторых дополнительных возможностей в набор стандартных функций системы (связанных в основном с динамической отладкой программ) и заменой некоторых «старых» функций новыми (более технологичными) авторы БЕС7920 связывают перспективы развития системы в большей степени с реализацией метода модульно-композиционного программирования.

Прежде всего, что понимается под методом модульно-композиционного программирования?

В основе наиболее распространенного в настоящее время метода модульного программирования лежит декомпозиция программ на элементарные, логически завершенные программные единицы — модули и объединение их в программы-монолиты с жестко детерминированными связями между модулями, включенными в программу (в особенности это характерно для программ, разрабатываемых на фортране). Однако практика разработки сложных программных изделий этим методом показала, что зачастую синтез эффективной модульной структуры оказывается затруднительным, поскольку первоначальное закрепление функций за отдельными модулями сделано без учета особенностей их последующего объединения.

Метод модульно-композиционного программирования, обладая всеми возможностями модульного программирования, обеспечивает более гибкую технологию синтеза новых программ на основании имеющихся программных фондов. Программный модуль рассматривается как элемент структуры в базе операционной обстановки для программирования, включающей в свою очередь базу знаний о программах в виде совокупности функциональных свойств, внутренних и внешних характеристик модулей. В общем случае синтез новой программы сводится к построению схемы композиции программных модулей на основании спецификаций и знаний, накопленных о них, а композиционная схема фактически представляет собой формализованное описание программы динамической структуры.

Процесс создания композиционной схемы обычно является итерационным, и наиболее эффективным для него представляется интерактивный режим взаимодействия пользователя с системой (хотя эксплуатация синтезированных в системе программ может осуществляться как в диалоговом, так и в пакетном режиме).

Программная реализация данного метода получила название «диалоговой системы модульно-композиционного программирования», для которой в качестве «сырья» используются фонды программных модулей, написанных в основном на фортране и ассемблере. Ядром системы является БЕС7920, обеспечивающая систему языком команд и интерпретатором с него, редактором текстов, механизмом распределения ресурсов машины между пользователями, службой поддержки базы знаний. В число дополнительных служб системы должны быть включены служба обслуживания композиционных схем, средства организации диалога в фортран-программах, средства диагностики и отладки синтезированных программ. Предполагается включить в систему язык программирования, объединяющий возможности некоторого диалекта языка фортран и языка команд БЕС7920, и интерпретатор с этого языка.

Процесс создания композиционной схемы можно упрощенно описать следующим алгоритмом. Для решения той или иной задачи программист выбирает необходимые загрузочные модули из совокупности модулей, имеющихся в программном фонде системы, с которой устраивающей его точностью определяет схему их выполнения и основные связи между ними (как по управлению, так и по данным) и инициирует их выполнение в диалоговом режиме. (Выбранная схема выполнения модулей является первым приближением к композиционной схеме.) При наличии несогласованности между модулями, а также в точках, определенных программистом, организуется диалог между системой и пользователем, в процессе которого можно изменить дальнейшую последовательность выполнения загрузочных модулей, скорректировать выходные данные выполненного модуля и входные данные очередного модуля, зафиксировать трассу выполнения модулей или ее часть в качестве новой программной единицы (некоторого очередного приближения композиционной схемы), выполнить ее с другими исходными или промежуточными данными и т. п. Поскольку связи между модулями внутри синтезируемой программы могут быть не полностью определены, это учитывается в построенной композиционной схеме — при ее выполнении в точках с недетерминированными связями между модулями либо будут использоваться предварительно подготовленные спецификации, либо будет автоматически инициироваться диалог с пользователем. Композиционные схемы могут ссылаться на другие композиционные схемы, причем допускаются рекурсии.

Средства организации диалога в фортран-программах предназначены для разработки диалоговых программ, выполняющихся как в среде БЕС7920, так и в среде прикладных систем, совместимых с БЕС7920. Диалог организован на базе предварительно описанных кадров, подготовленных на специальном языке высокого уровня, с последующим управлением сценарием диалога из фортран-программы. Имеется развитый аппарат работы с матрицами, доступный пользователю в режиме диалога и не требующий дополнительного программирования.

Средства организации диалога ориентированы на работу с локальными дисплеями ЕС-7920 и допускают использование практически всех возможностей аппаратуры и сервиса БЕС7920: форматизацию экрана, использование функциональной клавиатуры, использование селекторного пера, использование всех типов полей на экране и т. п. Из фортран-программы можно управлять информационным обслуживанием пользователя, активизируя (когда это необходимо) HELP-функцию.

ДИАЛОГ С ПРИКЛАДНОЙ ПРОГРАММОЙ НА БАЗЕ ППП ДИАФОР

Пакет программ ДИАФОР (ДИАлог на ФОРтрane) [2] был задуман как набор инструментальных средств, облегчающих создание диалоговых прикладных программ пользователям ЕС ЭВМ, работающим на фортране. Рассмотрим основные соображения, которыми руководствовались авторы пакета ДИАФОР.

Стандартные версии трансляторов с фортрана, эксплуатируемые на ЕС ЭВМ, — обычный транслятор уровня G и оптимизирующий транслятор уровня H — практически не располагают диалоговыми возможностями. В фортране ЕС нет операторов, обеспечивающих двусторонний обмен информацией с активным терминалом, кроме разве что обмена с пультом оператора. Средства преобразования данных, к которым мы неявно прибегаем, используя оператор FOR-MAT, скрыты от программиста. И хотя в операционных системах ОС ЕС такая возможность пользователю предоставляется*), но она требует для своего функционирования специальным образом сгенерированной версии ОС. Это означает, что диалоговая программа, работающая на одной ЭВМ, в условиях другой операционной обстановки работать не будет.

Фортран ЕС не приспособлен для работы с текстовыми данными. Привлечение для этого однобайтовых логических переменных связано с различного рода ухищрениями — присвоением значений однобайтовых логических переменных четырехбайтовым, отождествлением (EQUIVALENCE) переменных логического и арифметического типов и т. п. Все это приводит к достаточно замысловатым программам, обладающим невысокой эффективностью. Такие объекты, как экран, строки экрана, отдельные поля строк, плохо согласуются с фортрановскими представлениями о массивах. Попытка задать значения текстовых констант без учета длины соответствующих переменных приводит к многочисленным сообщениям транслятора о нарушении размерностей. И хотя эти синтаксические ошибки не носят криминального характера (уровень ошибок, равный 4, не приводит к снятию задания), видеть такие сообщения в своей программе неприятно. Дополнительное неудобство в формировании длинных строк вызывает и жесткий формат фортрановского бланка,

*) Имеется в виду служебная программа CONV, описание которой приведено в руководстве по ОС ЕС.

требующий размещения текста оператора только в позициях 7 — 72. В таких условиях очень трудно добиться соответствия между расположением текстовых величин на бланке и на экране дисплея.

Еще одна трудность, возникающая при реализации диалоговых систем, связана с установлением соответствия между объектами исходной программы (переменными, массивами, точками входа) и адресами рабочей программы, выполняющейся в ЭВМ. Проблема заключается в том, что объектный модуль, получающийся в результате трансляции фортрановской программы, не содержит никакой информации о распределении памяти под внутренние величины исходного модуля. Поэтому не так-то просто узнать, в какой ячейке оперативной памяти расположена интересующая нас величина, изменить или отобразить это значение по запросу пользователя во время счета. И даже установив такое соответствие, не хотелось бы заставлять пользователя помнить программные обозначения тех или иных величин, думать о лишних точках при задании целых значений вещественных переменных, придерживаться жесткого расположения данных в строке и заставлять отсчитывать позиции... одним словом, предоставить больше естественных возможностей по сравнению с ограничениями распространенных языков программирования.

Напомним и о некоторых технических проблемах. Фортран ЕС, например, не располагает какими-либо средствами синхронизации процесса вывода информации с физическими возможностями пользователя, просматривающего результаты счета на экране. Скорость вывода информации из ЭВМ и ограниченность объема экрана могут привести к тому, что интересующие нас данные промелькнут как кадры ускоренной съемки. В диалоговом общении большую роль играют средства, упрощающие процедуры выбора из числа тех возможностей, которые доступны пользователю в данной ситуации. Это и применение функциональной клавиатуры, и возможность выбора среди представленного на экране меню с помощью курсора или светового пера и т.п.

Вот далеко не полный перечень вопросов, которые приходилось решать в процессе создания и развития ППП ДИАФОР.

Аналогичные проблемы пришлось решать и авторам других инструментальных систем подобного рода [12]. Ниже мы рассмотрим основные программные решения, заложенные в базовых процедурах пакета, и наметим некоторые пути их совершенствования.

10.1. Организация обмена с дисплеем

В ранних версиях пакета построчный обмен производился с консолью оператора, которая на большинстве моделей ЭВМ Единой Серии представлена электрифицированной пишущей машинкой

(ЭПМ). Это наложило свой отпечаток на выбор базовых процедур обмена — подпрограмм INPULT и OUTEXT, выполняющих соответственно ввод и вывод символьной строки. Реализации этих процедур использовали либо стандартные средства фортрана, либо макрокоманды операционной системы (INLOG и OUTLOG в ДОС ЕС, WTO и WTOR в ОС ЕС). Текст программы INPULT приведен в [2].

Переход на работу с дисплейными станциями обусловил появление модулей обмена типа RWO7920, реализация которых для разных операционных систем подробно рассматривалась в гл. 7. Эти модули, собственно, и являются нижним уровнем, на котором базируются все остальные процедуры обмена.

10.1.1. Имитация пишущей машинки. В простейших диалоговых системах дисплей может имитировать режим работы с пишущей машинкой. Обмен информацией между ЭВМ с дисплеем происходит по одной строке. Отображение ее на экране можно производить либо по принципу «снизу—вверх», подобно тому как движется бумага на ЭПМ, либо по принципу «сверху — вниз». В первом случае, который представляется нам более естественным, 24-я строка экрана (самая нижняя строка) играет роль наборной при вводе и информационной при выводе. В этой строке пользователь набирает все сообщения, адресованные прикладной программе. Набор очередного сообщения заканчивается нажатием любой функциональной клавиши, например кнопки ВВОД. Это служит сигналом программе обмена для приема директивы пользователя и передачи ее вызывающей программе на предмет анализа и исполнения поступившего указания. Содержимое экрана в этот момент смещается на одну строку вверх, имитируя продергивание бумаги на ЭПМ после нажатия клавиши «возврат каретки». Для удобства пользователя в момент предоставления ему инициативы наборную строку желательно чистить, а курсор переводить в первую позицию наборной строки. Чтобы привлечь внимание пользователя, можно либо прибегнуть к звуковому сигналу, либо к выдаче в первой позиции наборной строки специального символа (вопросительного знака, звездочки и т.п.), либо к комбинации обоих этих средств. При выводе сообщений, адресованных пользователю, содержимое экрана предварительно «поднимается» на одну строку вверх.

Работа по принципу «снизу — вверх» требует хранения копии экрана в оперативной памяти и при каждом выводе связана с выдачей полного экрана. В этом смысле более экономным является работа по принципу «сверху—вниз», когда заполнение экрана ведется подобно тому, как мы пишем на листе бумаги. Самое первое сообщение ЭВМ появляется в самой верхней части экрана. Под ним набирается первое сообщение пользователя, ниже — ответы ЭВМ и т. д. Используя 24-ю строку, программа обмена должна

стереть содержимое экрана и начать заполнять его заново. Если экран не гасить, то новые сообщения начнут напозать на содержимое предыдущей страницы, и будет трудно отделить текущую информацию от старых данных. Преимущества работы по принципу «сверху — вниз» состоят в том, что каждая процедура ввода-вывода связана с обменом только одной строки и нет необходимости в хранении копии экрана. Однако, по мнению большинства пользователей, первый вариант считается более удобным для восприятия информации. Кроме того, в любой момент времени пользователь видит на экране максимальный фрагмент предыстории своего общения с ЭВМ.

Реализация описанных алгоритмов на базе модуля RWO7920 осуществляется программами, приведенными ниже:

Построчный обмен по принципу «снизу — вверх»

SUBROUTINE RDLINE (СТРОКА)

REAL *8 ЕКРАН(240),СТРОКА(10),PUSTO(8*','_')/

C ПОД'ЕМ ЭКРАНА НА ОДНУ СТРОКУ В ОПЕРАТИВНОЙ ПАМЯТИ

DO 1 K=1,230

1 ЕКРАН(K)=ЕКРАН(K+10)

C ОЧИСТКА НАБОРНОЙ СТРОКИ

DO 2 K=231,240

2 ЕКРАН(K)=PUSTO

C ВЫВОД МОДИФИЦИРОВАННОГО ЭКРАНА

CALL WRDISP(ЕКРАН,1,24,1840)

C ЧТЕНИЕ НАБОРНОЙ СТРОКИ

CALL RDDISP(ЕКРАН(231),24,1,K,M)

C ПЕРЕСЫЛКА ВВЕДЕННОЙ СТРОКИ ПО АДРЕСУ

DO 3 K=1,10

3 СТРОКА(K)=ЕКРАН(K+230)

GO TO 6

C ДОПОЛНИТЕЛЬНЫЙ ВХОД ДЛЯ ВЫВОДА СТРОКИ

ENTRY WRLINE(СТРОКА)

DO 4 K=1,230

4 ЕКРАН(K)=ЕКРАН(K+10)

DO 5 K=1,10

5 ЕКРАН(K+230)=СТРОКА(K)

CALL WRDISP(ЕКРАН,1,24,1840)

6 RETURN

END

Программа построчного обмена по принципу «сверху — вниз»

SUBROUTINE RDLINE (СТРОКА)

LOGICAL *1 СТРОКА(80),PUSTO(80)/80*','_')/

INTEGER НОМЕР/25/

```

INPUT=1
GO TO 1
ENTRY WRLINE (СТРОКА)
INPUT=0
1 IF (HOMEP.LE.24)GO TO 3
  DO 2 K=1,24
2 CALL WRDISP(PUSTO,K,1,0)
  HOMEP=1
3 IF(INPUT.EQ.1)CALL RDDISP(СТРОКА,HOMEP,1,K,M)
  IF(INPUT.EQ.0)CALL WRDISP(СТРОКА,HOMEP,1,
* HOMEP*80)
  HOMEP=HOMEP+1
RETURN
END

```

В первой программе использованы восьмибайтовые переменные для ускорения процедур присваивания. Однако это налагает некоторые ограничения на расположение выдаваемого массива: он должен начинаться с границы двойного слова. Одномерный массив ЕКРАН также сокращает время выполнения программы за счет отсутствия двойных циклов и уменьшения числа тактов, связанных с индексацией элементов массива. Объединение программ ввода и вывода в одну подпрограмму удобно тем, что не приходится выносить массив ЕКРАН в общую память. Во второй программе массив СТРОКА, используемый для обмена, представлен] однобайтовым массивом логического типа. Это снимает какие-либо ограничения на его' расположение в памяти и позволяет вести посимвольную обработку, однако увеличивает число машинных тактов. Внутренняя переменная HOMEP служит указателем текущей строки. Ее начальное значение (HOMEP = 25) выбрано таким образом, чтобы первое же обращение прикладной программы к дисплею произвело чистку экрана.

Конечно, фортран не самое лучшее средство для построения эффективных машинных программ. Так, например, подъем содержимого экрана в оперативной памяти на фортране требует выполнения 230 операторов присваивания плюс накладные расходы на организацию цикла. На ассемблере эта же операция требует выполнения всего восьми команд MVC. Фрагмент программы на ассемблере, реализующей чтение строки в режиме «снизу — вверх», приведен ниже:

Программа ввода строки в режиме «снизу—вверх»

```

RDLINE  CSECT
        PROLOG
        MVC   АСТР,0(1)
        LM    3,5,КОНСТ
*** ЦИКЛ ПОД'ЕМА СТРОК 2—22

```

```

ВВЕРХ    MVC    0(240,3),80(3)
          BXLE   3,4,ВВЕРХ
*** ПОД'ЕМ СТРОК 23,24
          MVC    0(160,3),80(3)
*** ОЧИСТКА НАБОРНОЙ СТРОКИ
          MVC    СТРОКА24(80),PUSTO
*** ВЫВОД МОДИФИЦИРОВАННОГО ЭКРАНА
          CALL   WRDISP,(ЕКРАН,#F1,#F24,#F1840)
*** ЧТЕНИЕ НАБОРНОЙ СТРОКИ
          CALL   RDDISP,(СТРОКА24,#F24,#F1,К,К+4)
*** ПЕРЕСЫЛКА ВВЕДЕННОЙ СТРОКИ ПО АДРЕСУ
          L      1,АСТР
          MVC    0(80,1),СТРОКА24
          EPILOG
#F1      DC      F'1'
#F24     DC      F'24'
#F1840   DC      F'1840'
АСТР     DS      F
К        DS      2F
КОНСТ    DC      A(ЕКРАН,240,ЕКРАН+22*80)
PUSTO    DC      CL80'—'
ЕКРАН    DC      CL1920'—'
СТРОКА24 EQU  ЕКРАН+23*80
          END

```

Эта программа занимает меньше оперативной памяти, чем аналогичная программа на фортране, и примерно раз в 100 быстрее работает. Однако мы довольно часто будем прибегать к фортрановским программам из-за ограниченности объема пособия и сложности типографского набора.

10.1.2. Вывод с накоплением строк. С целью сокращения числа обращений к внешнему устройству полезно накапливать выводимые строки в оперативной памяти и по мере заполнения «экрана» осуществлять выдачу накопленной информации за один прием. После вывода необходимо предоставить пользователю возможность ознакомиться с очередной порцией результатов прежде, чем начать формирование и отображение следующей страницы. Это можно сделать либо по регулируемой временной задержке с помощью макрокоманд операционной системы (SETIME в ДОС ЕС или STIMER в ОС ЕС), либо по сигналу прерывания, который возникнет в момент нажатия пользователем той или иной функциональной клавиши.

Существует много алгоритмов накопления информации в буфере-образе экрана и его отображения на дисплее. Один из них реализован следующей программой:

Программа вывода с накоплением строк

```

SUBROUTINE WRPAGE(СТРОКА,J)
REAL *8 ЕКРАН(240),СТРОКА(10),PUSTO/8*'—',/
INTEGER НОМЕР/1/
C ПРИЕМ ОЧЕРЕДНОЙ СТРОКИ
I=10*(НОМЕР-1)
DO 1 K=1,10
1 ЕКРАН(I+K)=СТРОКА(K)
НОМЕР=НОМЕР+1
C ПРОВЕРКА УСЛОВИЙ ВЫВОДА
IF (НОМЕР.LE.24.AND.J.EQ.0)RETURN
C ВЫВОД НАКОПЛЕННОЙ ИНФОРМАЦИИ
CALL WRDISP(ЕКРАН,1,24,1840)
C ОЖИДАНИЕ СИГНАЛА ОТ ПОЛЬЗОВАТЕЛЯ
CALL RDDISP(ЕКРАН,1,0,K,I)
C ПРИВЕДЕНИЕ ОБРАЗА ЭКРАНА В НАЧАЛЬНОЕ
СОСТОЯНИЕ
НОМЕР=1
DO 2 K=1,240
2 ЕКРАН(K)=PUSTO
RETURN
END

```

Входные параметры СТРОКА и J соответствуют очередной выводимой строке и указанию по режиму отображения. При $J = 0$ накопление строк ведется до заполнения экрана, после чего происходит отображение очередной страницы и ожидание сигнала прерывания. При $J = 1$ после переноса очередной строки на буфер производится отображение тех строк, которые накопились к данному обращению. Возврат в вызывающую программу в любом случае произойдет только после разрешения со стороны пользователя, т.е. после нажатия любой функциональной клавиши. Для этой цели в программе используется оператор фиктивного чтения — обращение к RDDISP с нулевым числом вводимых строк. Заполнение буфера в оперативной памяти ведется сверху вниз. После выдачи полного или частично заполненного буфера на экран дисплея производится очистка буфера.

Функции приведенной программы можно существенно расширить, если добавить анализ кода функциональной клавиши, нажатой пользователем. Можно выделить: код, по которому содержимое экрана будет отправляться на печать (твердая копия); код, разрешающий продолжение вывода; код, по которому управление должно быть передано в заданную точку вызывающей программы, и т. п.

10.2. Использование функциональной клавиатуры для работы в режиме «меню»

Диалог в режиме «меню» предполагает выдачу на экран некоторого сообщения, занимающего одну или несколько строк, с указанием на возможные варианты продолжения работ. Меню может быть организовано по-разному — одна строка соответствует одному «блюду» или в пределах одной строки располагается несколько предлагаемых на выбор процедур. В большинстве ранних диалоговых систем работа в режиме меню строилась следующим образом. Каждому ответу в меню присваивался порядковый номер, и ответ пользователю заключался в наборе выбранного числа.

Наличие функциональной клавиатуры сокращает процедуру набора до минимума. Если число предлагаемых вариантов невелико ($n \leq 18$), то с каждым вариантом продолжения из данной точки сценария диалога можно связать определенную функциональную клавишу. Тогда вместо набора номера выбранного блюда, что связано с нажатием двух—трех клавиш, пользователю остается нажать всего одну кнопку. Если число вариантов достаточно велико, то вместо функциональных клавиш можно воспользоваться маркером. Его надо подвести к соответствующей позиции меню и нажать любую функциональную клавишу. Поступающий в ЭВМ номер позиции курсора — целое число из диапазона от 0 до 1919 — в сочетании с 18 кодами функциональной клавиатуры дают достаточно много комбинаций для того, чтобы запрограммировать любые алгоритмические развилки.

Для реализации диалога в режиме меню в составе ППП ДИАФОР опробовались различные программы:

```
MENU1 (MENU,LIN1,LN,KODFK)
MENU2 (MENU,LIN1,LN,KYPCOP)
MENU3 (MENU,LIN1,LN,&M1,&M2,...,&M18)
MENU4 (MENU,LIN1,LN,NCTP,NPOS)
```

Несмотря на разнообразие параметров, функции всех этих программ одинаковы. Сначала в копию экрана, хранимую в оперативной памяти, вписывается предлагаемый текст — массив MENU. Он содержит LN полных 80-байтовых строк и должен быть отображен на экране дисплея, начиная со строки с номером LIN1. Затем буфер отображается на экран и по команде фиктивного чтения определяются адрес позиции курсора и номер функциональной клавиши, нажатой пользователем после изучения меню. А дальнейшие действия программ могут быть разными. Например, можно передать вызывающей программе код нажатой клавиши (KODFK) или координаты курсора (KYPCOP), можно произвести возврат на разные метки вызывающей программы (&Mi) в зависимости от сложившейся

ситуации, и т. п. Переход от адреса позиции курсора (КУРСОР) к его координатам в терминах «номер строки» (NСТР) и «номер позиции» в строке (NPOS) на фортране производится довольно просто:

$$NСТР = КУРСОР / 80 + 1$$

$$NPOS = КУРСОР - 80 * NСТР + 79$$

10.3. Обработка символьных и числовых данных

Программы обмена с дисплеем обеспечивают передачу информации без какой-либо ее переработки. Таким образом можно передавать только символьные данные. Поэтому при работе с числовой информацией необходимо позаботиться о соответствующем формате преобразования. Получив одну или несколько символьных строк в результате чтения с экрана, прикладная программа должна разобраться с поступившей информацией — выделить ключевые слова, определить расположение числовых данных, позаботиться об их преобразовании в требуемый машинный формат и присвоить полученные значения соответствующим переменным.

Введение жестких форматов заставляет пользователя отсчитывать номера позиций на экране, думать о количестве набираемых цифр, не забывая о простановке точек после целых значений вещественного типа, задавать обязательно все величины, запрограммированные в списке процедур ввода. Все это очень затрудняет процесс общения с ЭВМ для человека. Предоставление пользователю большей свободы на вводе, конечно, усложняет анализ поступающих данных. Однако дополнительные затраты не так уж и велики. Ниже рассматривается ряд возможностей, предоставляемых пакетом ДИАФОР для преобразования числовой и текстовой информации, и обсуждаются предложения по их совершенствованию.

10.3.1. Анализ символьной информации. При обработке введенного текста довольно часто приходится выполнять такие процедуры, как распознавание очередного символа или группы символов, обход незначащих пробелов, определение координат вхождения того или иного ключевого слова, пропуск ошибочного фрагмента текста до ближайшего разделителя и т.п. Проще всего для реализации такого рода процедур воспользоваться ассемблером, так как он позволяет выполнить большинство из перечисленных выше операций за одну-две машинные команды.

Для обработки текстовых данных очень удобна команда проверки и перекодировки (TRT) в сочетании с удачно подобранным словарем. Предположим, что введенный текст, закодированный символами ДКОИ, находится в 80-байтовом массиве СТРОКА, а в первом регистре общего назначения хранится абсолютный адрес

текущего символа. Тогда анализ одиночного символа можно выполнить с помощью команды CLI. Например,

```
CLI 0(1),C'='   проба на знак «равно»
CLI 0(1),C',,   проба на запятую
CLI 0(1),C'('   проба на открывающую скобку
```

Для сравнения с заданным ключевым словом можно воспользоваться командой CLC. Например,

```
CLC 0(2,1),=C'ДА'
CLC 0(8,1),=C'ПЕРЕПИШИ'
```

Более широкие возможности предоставляет макрокоманда IF, текст которой приведен в [2]. С помощью макрокоманды IF можно проанализировать текущий или следующий символ строки, а также цепочку символов, начиная с текущего или следующего символа. Например,

```
IF THIS=БУКВА,THEN=M1,ELSE=M2
IF NEXT=ЗАПЯТАЯ,THEN=M3
IF THIS=(,ELSE=M4
```

Здесь метки M1, M2, . . . определяют адрес перехода в случае выполнения (THEN) или невыполнения (ELSE) проверяемого условия. Когда с помощью макрокоманды IF анализируется ключевое слово, то значением параметра THIS или NEXT является метка ключевого слова:

```
IF THIS=DA,THEN=...
. . . . .
DA DC C'ДА'
```

При анализе символа строки на цифру макрокоманда IF совмещает проверку логического условия с удалением кода зоны, т. е. с выделением двоичного эквивалента цифры и записью его в младший байт второго регистра.

Машинная команда TRT может быть использована для обхода пробелов или пропуска фрагментов текста до ближайшего разделителя. Для этого достаточно построить словарь, в котором пропускаемым символам поставлены в соответствие нулевые коды. Например,

```
TRT 0(L,1),SLOWAR
. . . . .
SLOWAR DC 63X'01',X'00',192X'01'
```

В этом примере производится обход пробелов, начиная с текущего символа, адрес которого равен 0(1). Значение L определяет количество символов в правой части массива СТРОКА. Так как среди отображаемых символов ДКОИ пробел имеет самый наименьший

код ($X'40' = 64_{10}$), то словарь можно сократить на 63 байта:

TRT 0(L,1),SLOWAR1 — 64

SLOWAR1 $\cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot$
DC X'00'492X'01'

Можно сэкономить объем оперативной памяти и за счет полного отказа от словаря. Но тогда придется организовать циклический перебор пропускаемых символов. Например, внутренняя подпрограмма обхода пробелов может быть построена следующим образом:

```
OBPRB  LM      2,3,КОНСТ
OBPRB1 CLI      0(1),C'—'
        BNE      4(11)
        BXLE     1,2,OBPRB1
        BR       11
```

КОНСТ $\cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot$
DC A(1,СТРОКА + 79)

Обращение к такой подпрограмме имеет вид:

BAL 11,OBPRB

Возврат из подпрограммы OBPRB по адресу 0(11) свидетельствует о том, что правее текущего символа находятся одни пробелы. При возврате по адресу 4(11) в первом регистре находится адрес ближайшего «непробела».

Анализ вхождения ключевого слова в исследуемую строку может быть выполнен с помощью подпрограммы-функции JKEY, обращение к которой на фортране имеет вид:

J1=JKEY(СТРОКА,J,LKEY,KEY)

Поиск ключевого слова, заданного параметром KEY и имеющего длину LKEY символов, производится начиная с байта СТРОКА(J). Для этого организуется циклическое сравнение по команде CLC:

CLC 0(LKEY,R),KEY

При каждом повторении цикла адрес в регистре R увеличивается на 1. Так повторяется до тех пор, пока либо не будет обнаружено заданное ключевое слово, либо не будет исчерпан массив СТРОКА. В первом случае значению функции JKEY присваивается номер байта, расположенного вслед за ключевым словом. Во втором случае функции JKEY присваивается нулевое значение. Текст подпрограммы JKEY приведен в [2].

10.3.2. Перевод чисел из символьного представления в машинные форматы. Задача преобразования числа из символьного представления в нужный машинный формат особой трудности не составляет. Ее можно реализовать либо на ассемблере, либо на фортране

с небольшой потерей эффективности. Надо только договориться о допустимых вариантах представления числовых данных в анализируемой строке и выбрать символ-разделитель, которым числовое значение должно заканчиваться. ДИАФОР допускает преобразование целых чисел и вещественных чисел, записанных в форме с фиксированной точкой. Пробелы в записи чисел при переводе игнорируются, а в качестве терминального символа используется запятая. Выполняется процедура преобразования с помощью программной секции INUMB, имеющей дополнительный вход — FNUMB. Оба эти входа имитируют работу фортрановских функций со следующими обращениями:

$I = \text{INUMB}(\text{СТРОКА}, J, \text{IRES})$

$R = \text{FNUMB}(\text{СТРОКА}, J, \text{IRES})$

Переменная J определяет номер байта в массиве СТРОКА, с которого должно начинаться число. Перевод целых и вещественных чисел выполняется по единой схеме:

$$S = S \cdot 10 + \alpha_i.$$

Здесь α_i — очередная цифра, выделенная в записи числа. Накапливаемое значение суммы S формируется в виде целого числа. После появления десятичной точки, допустимой только в записи вещественного числа, начинается подсчет корректирующего множителя 10^{-m} (m — общее количество цифр в дробной части числа). Если переводится целое число, то результат (S) находится в нулевом регистре общего назначения. Для вещественного числа значение рассматривается как мантисса, к которой присоединяется порядок X'46'. Полученное таким образом ненормализованное число с плавающей запятой домножается на корректирующий множитель 10^{-m} и результат помещается в нулевой арифметический регистр. Перед выходом из подпрограммы к содержимому обоих нулевых регистров приформировывается знак числа.

Значение указателя J продвигается (увеличивается на 1) одновременно с обработкой очередного символа числа. Поэтому при возврате в вызывающую программу переменная J определяет байт в массиве СТРОКА, прервавший процесс перевода. Выходная переменная IRES принимает следующие значения:

$$\text{IRES} = \begin{cases} +1, & \text{если символьная запись числа соответствовала предполагаемому формату и правее него в массиве СТРОКА ничего не было,} \\ 0, & \text{если ошибок в записи числа не обнаружено и число завершилось запятой, т. е. признаком продолжения списка,} \\ -1, & \text{если до разделителя обнаружен недопустимый символ.} \end{cases}$$

Текст подпрограмм — функций INUMB и FNUMB, составленных на ассемблере, — приведен в [2].

Единая схема преобразования целых и вещественных чисел, реализованная в этих подпрограммах, имеет определенные преимущества. К ним следует отнести простоту алгоритма, возможность не задавать точку в записи целых чисел вещественного типа, возможность получить сразу два машинных формата для целых чисел в обоих нулевых регистрах. Однако преобразование чисел целого типа выполняется существенно медленнее, так как алгоритм не использует специализированные команды типа PASC и CVB.

Включение в состав допустимых форматов числовых данных экспоненциального представления не слишком сильно усложнит программу перевода. После обнаружения символа E надо преобразовать порядок p и домножить результат перевода на $10^{\pm p}$. Эту операцию можно выполнить либо в цикле, либо воспользоваться разумно подобранным массивом констант. Подобного рода операция будет продемонстрирована в п. 10.3.3.

10.3.3. Формирование выводной строки. Назовем выводной строкой 80-байтовый массив оперативной памяти, снабженный указателем первой свободной позиции. Пусть для определенности идентификатор СТРОКА задает начало выводного поля, а указатель J определяет байт СТРОКА(J), начиная с которого размещается преобразованная для вывода информация. Прикладная программа может иметь не одну, а несколько выводных строк, каждая из которых снабжена своим указателем. Пользователь сначала формирует отдельные поля выводной строки, а после заполнения всей строки выдает ее на дисплей.

Очередная порция выводимых данных представляет собой числовую или текстовую информацию, которую необходимо разместить в выводной строке, начиная со свободного места. Количество позиций w , выделяемое для размещения очередной порции, задается пользователем. Текстовые данные переносятся в выводную строку без какого-либо преобразования, при этом первый символ выводимого текста записывается в байт СТРОКА(J), второй — в байт СТРОКА(J + 1) и т.д. Числовые данные перед записью в выводную строку должны быть преобразованы из соответствующего машинного формата в символьное представление. После этого число заносится в выводную строку, прижимаясь к правой границе выделенного поля, т.е. младшая цифра числа заносится в байт СТРОКА(J + w - 1), следующая — в байт СТРОКА(J + w - 2) и т.д. Байты левой части выделенного поля, оставшиеся свободными, заполняются пробелами. Такой способ формирования числовых данных позволяет расположить на экране дисплея числовые таблицы с наиболее естественным расположением цифр в колонках.

После переноса очередной порции данных в выводную строку продвигается связанный с ней указатель ($J = J + w$), с тем чтобы подготовиться к приему следующей порции.

Заполнение выводной строки с сопутствующим преобразованием числовых данных выполняется с помощью программы FORM:

CALL FORM (DATA, FORMAT, СТРОКА, J, IER)

За одно обращение на свободное место выводной строки может быть перенесена либо цепочка произвольных символов в кодировке ДКОИ, либо цепочка пробелов, либо одно числовое значение целого или вещественного типа. Выбором перечисленных возможностей управляет параметр FORMAT, в качестве которого может выступать один из редактирующих указателей вида: Aw, Xw, Iw, Fw.d, Ew.d. Параметр DATA определяет значение данного, передаваемого в выводную строку.

Пример 1.

CALL FORM (IK, 'I4—'CTPK22, J1, IOSH)

или

CALL FORM (IK, 3H I4—, CTPK22, J1, IOSH)

Преобразуется переменная целого типа KI по явно заданному формату I4. Пробел после цифры в формате I4 играет роль разделителя. Он необходим, так как ширина поля в формате Iw может содержать две цифры, например I12.

Пример 2.

DATA IFORM/4HI4— —/

.....

CALL FORM (K + 2, IFORM, CTPK24, J2, IRES)

Преобразуется выражение целого типа по формату I4, заданному как значение переменной IFORM. Тип переменной IFORM роли не играет.

Пример 3.

CALL FORM (—52.4, 'E12.4', CTPK1, K, IER)

Вещественное число —52.4 заносится в выводную строку с преобразованием по формату E, заданному в обращении явно.

Пример 4.

CALL FORM ('ОБРАЗЕЦ ВЫВОДА ТЕКСТА', 'A21',
СТРОКА, I, J)

Текст, заданный явно в обращении, переносится в выводную строку.

Пример 5. Очистка выводной строки:

CALL FORM (J, 'X80', СТРОКА, I, J)

Здесь первый и последний параметры указаны фиктивно для того, чтобы соблюсти порядок и количество аргументов в обращении.

Пример 6.

INTEGER ТЕКСТ(6)/'ОБРАЗЕЦ ВЫВОДА ТЕКСТОВ'/
CALL FORM(ТЕКСТ,ЗНА22,СТРОКА,I,J)

Вариант с неявным заданием текста, построенный с учетом того, что транслятор с фортрана в ЕС ЭВМ не считает тяжелой ошибкой отсутствие разбиения символьных данных на элементы по четыре буквы:

ТЕКСТ(6)/4НОВА, 4НЗЕЦ_,.../

Поясним смысл некоторых ограничений на числовые характеристики редактирующих указателей. Для текстовых данных и цепочек пробелов длина передаваемой информации не может превышать общую длину выводной строки ($w \leq 80$). Выбор $w=15$ в качестве максимальной длины числового поля объясняется довольно естественными соображениями. Целое число максимальной длины, представимое в ЕС ЭВМ, требует для своей записи не более 11 позиций ($-2^{+32} = -2147483648$). Вещественные числа нормальной длины в ЕС ЭВМ представляют семь-восемь значащих десятичных знаков. Поэтому самое сложное представление вещественного числа в экспоненциальном формате занимает не более 15 позиций:

$$\pm 0.a_1a_2a_3a_4a_5a_6a_7 = E \pm p_1p_2$$

По той же причине количество дробных разрядов в формате F или число значащих цифр в формате E не должно превышать семи ($d \leq 7$). Остальные ограничения ($d \leq w - 2$ для формата F и $d \leq w - 8$ для формата E) связаны с необходимостью предусмотреть позиции для размещения знака числа, десятичной точки, десятичного порядка.

Параметр IER является выходным. Его значение определяет характер ошибки, допущенной пользователем при обращении к подпрограмме FORM:

$$IER = \begin{cases} -1, & \text{если длина свободной части выводной строки меньше, чем длина } w \text{ передаваемой порции,} \\ 0, & \text{если с параметрами все в порядке,} \\ 1, & \text{если длина выделяемого поля } w \text{ недостаточна для приема числового значения по форматам I или F,} \\ 2, & \text{если неправильно задан редактирующий указатель или нарушены ограничения на значения } w \text{ и } d. \end{cases}$$

В случае $IER = 1$ текущее поле выводной строки в соответствии со стандартами, фортрана расписывается звездочками. При

$IER=2$ и $IER=-1$ фактической записи в выводную строку не происходит.

Особенности преобразования числовых данных. При составлении программы FORM преследовались две цели. Во-первых, хотелось максимально совместить процедуры преобразования целых и вещественных чисел. Во-вторых, требовалось достичь высокого быстродействия. Совместить обработку целых чисел по формату Iw и вещественных чисел с фиксированной запятой по формату $Fw.d$ довольно просто: достаточно умножить преобразуемое значение на 10^d , после чего процедура обработки становится абсолютно идентичной как для целых чисел, так и для чисел с фиксированной запятой. Единственная «лишняя» операция для формата Iw заключается в умножении числа на 10^0 . Для того чтобы направить по этому же руслу перевод числа по формату $Ew.d$, необходимо сначала определить десятичный порядок числа q , а затем разделить исходное число на 10^q . Полученная после этого мантисса числа может преобразовываться так же, как и число с фиксированной запятой.

Большинство известных алгоритмов перевода двоичных чисел с плавающей запятой в десятичную систему использует для определения десятичного порядка циклы с умножением или делением исходного числа на 0.1 до тех пор, пока результат не окажется приведенным к диапазону (0.1, 1). На вычислительных машинах типа М-20 или БЭСМ-6, где диапазон допустимых чисел находится в интервале от 10^{-19} до 10^{19} , количество повторений такого цикла может достигать 19, хотя на практике редко превышает 5—6. В ЭВМ Единой Серии диапазон допустимых чисел существенно шире — от 10^{-78} до 10^{+75} . Поэтому определять десятичный порядок в цикле, который может повторяться до 78 раз, крайне невыгодно.

В связи с этим специально для программы FORM был разработан быстрый алгоритм нахождения десятичного порядка, не использующий циклических процедур. Обозначим через X исходное число. Если не прибегать к специальным ухищрениям, то число X нормализовано, и его можно представить в виде

$$X = x \cdot 16^p = y \cdot 10^q \quad (1)$$

Здесь x — двоичная мантисса ($0.0625 \leq x \leq 1 - 2^{-24}$), p — машинный (шестнадцатеричный) порядок ($-64 \leq p \leq 63$), y — нормализованная десятичная мантисса ($0.1 \leq y < 1$) и q — искомый десятичный порядок ($-76 \leq q \leq 75$). Логарифмируя левую и правую части равенства (1), мы получим

$$\lg x + p \lg 16 = \lg y + q.$$

Учитывая, что мантиссы x и y достаточно близки, можно принять $q \approx [p \lg 16] \pm 1 = [1.2041p] \pm 1 = [1.2p] \pm 1$. Таким образом, в качестве достаточно хорошего приближения к десятичному по-

рядку можно взять величину $q^* = \left[\frac{p \cdot 12}{10} \right]$. Выбор q^* в указанном виде удобен тем, что он позволяет всего за две операции над целыми числами определить старшую (q_1) и младшую (q_2) цифры десятичного порядка. Тем самым основное преобразование двоичной мантиссы x может быть выполнено всего за два умножения (деления) — на 10^{q_1} и 10^{q_2} . Конечно, абсолютной гарантии попадания получающейся таким образом мантиссы y в интервал $(0,1)$ описанное преобразование не дает, так как порядок q^* может отличаться от истинного на 1. Однако после выполнения основного преобразования можно проанализировать величину мантиссы и в случае необходимости произвести еще одно умножение (деление) на 10, скорректировав соответствующим образом q^* .

Программа формирования десятичного порядка

.....			
	LPE	0,A(DATA)	
	STE	0,ZONA1	
	SR	11,11	
	IC	11,ZONA1	
	LA	1,0	
	MVC	Q10,=X'000C'	$q_{10}=00 \ 0 \rightarrow$
	SH	11,=H'64'	
	BZ	E2	
	BP	E1	
	LPR	11,11	
	LA	1,4	
	MVI	Q10+1,X'0D'	$q_{10}=00 \ 0 \rightarrow$
E1	M	10,=F'12'	$12 * p_{16}$
	D	10,=F'100'	
	STC	11,Q10	$q_{10}=0q_1 0 \text{ знак}$
	SRDL	10,32	$11_{\text{per}}=10 * q_2$
	D	10,=F'10'	$11_{\text{per}}=q_2$
	IC	10,Q10+1	$10_{\text{per}}=q_1$
	SLDL	10,2	$10_{\text{per}}=4, \ 11_{\text{per}}=4q_2$
	EX	0,DMQ1(1)	
	EX	0,DMQ2(1)	
	SLL	11,2	$q_2 0000$
	EX	11,Q2	$q_{10}=0q_1 q_2 \text{ знак}$
E2	LTER	0,0	
	BZ	E5	
E3	CE	0,=E'1'	
	BL	E4	
	AP	Q10(2),=P'1'	$q_{10}=q_{10}+1$
	DE	0,=E'10'	

	B	E5	
E4	CE	0,=E'0,1'	
	BNL	E5	
	SP	Q10(2),=P'1'	$q\bar{q}_n = q_{i_c} - 1$
	ME	0,=E'10'	
E5	.	.	.
DMQ1	DE	0,E0(11)	
	ME	0,E0(11)	
DMQ2	DE	0,E10(10)	
	ME	0,E10(10)	
Q2	OI	Q10+1,0	
E0	DC	E'1,1E1,1E2,1E3,1E4,1E5,1E6,1E7,1E8,1E9'	
E10	DC	E'1,1E10,1E20,1E30,1E40,1E50,1E60,1E70'	
ZONA1	DS	F	
Q10	DS	H	

Прокомментируем наиболее интересные места фрагмента приведенной программы, выполняющей предварительную обработку числа по формату Ew.d. Двухбайтовое поле Q10 предназначено для размещения десятичного порядка q в упакованном формате. Сначала на это поле пересылается число +00. Если шестнадцатеричный порядок отрицателен, то начальное значение q заменяется на -00. Выделение цифр десятичного порядка начинается командой с меткой E1. После умножения на 12 и деления на 100 в регистрах 11 и 10 соответственно получаются частное, равное старшей цифре q_1 , и остаток, который приходится еще раз разделить на 10 для выделения младшей цифры q_2 . Эти цифры приформировываются к соответствующим позициям Q10. С их же помощью осуществляется и деление (умножение) исходного числа на 10^q .

Так как значения q_1 и q_2 находятся в двух смежных регистрах общего назначения, то с помощью одной команды длинного сдвига SLDL формируются значения $4*q_1$ и $4*q_2$, которые используются для выбора соответствующих констант 10^{q_1} и 10^{q_2} . Использование команд EX упрощает процедуру выбора нужной операции и умножения (содержимое R1 = 0) или деления (содержимое R1 = 4) на соответствующую константу.

Дальнейшая обработка цифровой части числа ведется почти одинаково для любого форматного указателя. Единственная особенность для формата Fw.d состоит в том, что перед умножением на 10^d надо предупредить возможность переполнения. Это достигается сравнением исходного числа со специально подобранной константой EMAX:

```

CD  0,EMAX
BM  PEREBOR

```

```

EMAX DC X'4E0000007FFFFFFFFF'

```

Если число слишком велико, то оно заведомо не может быть размещено по формату F на поле, содержащем не более 15 позиций ($w \leq 15$). В этом случае управление передается команде с меткой PEREBOR и на текущее поле выводной строки заносится w звездочек. После умножения на 10^d мы фактически имеем дело только с целой частью числа. Сначала она округляется (к положительному числу прибавляется 0.5, а из отрицательного вычитается 0.5), а потом денормализуется путем сложения с ненормализованным нулем. Затем денормализованная мантисса пересылается в регистр общего назначения и по команде CVD преобразуется в упакованный формат десятичного числа. Его редактирование производится по команде EDMK, использующей один из заранее заготовленных 20-байтовых шаблонов:

Iw:	40	40	20	20	20	21	21	
Fw.d Ew.d }:	40	21	20	21	4B	20	20
									d байт

К шаблону формата Ew.d справа примыкают еще 5 байт, в которых формируется десятичный порядок числа. Команда EDMK позволяет не только сформировать символьное представление числа, но и определить адрес старшей значащей цифры, перед которой должен быть помещен знак числа. Если этот адрес принадлежит правым w байтам шаблона, то преобразование завершилось удачно, и содержимое шаблона переносится в выводную строку с соответствующей коррекцией указателя J. В противном случае на текущее поле выводной строки заносится w звездочек.

Полный текст программы FORM заинтересованный читатель может найти в работе [11].

10.4. Изменение параметров прикладной программы в режиме диалога

На базе программ ввода текстовых строк (INPULT, RDLINE, RDDISP) и преобразования числовой информации (INUMB, FNUMB) в прикладной программе можно построить любые процедуры обращения к пользователю за сменой тех или иных параметров. И в составе пакета ДИАФОР имеются некоторые средства, облегчающие реализацию процедур подобного рода. Одно из них — ввод списка значений, количеством которых пользователь может

управлять. Выполняется такая процедура с помощью подпрограммы — функции GETNUM:

$Y = \text{GETNUM}(J, \text{IRES})$

При первом обращении к функции GETNUM указатель J должен иметь нулевое значение. В этом случае подпрограмма предоставляет пользователю возможность ввести очередную строку, содержащую список чисел, количество которых регулирует пользователь. Если список не пуст, то из него выбирается первое значение, переводится в машинный формат вещественного числа и присваивается функции GETNUM, т. е. записывается в нулевой регистр

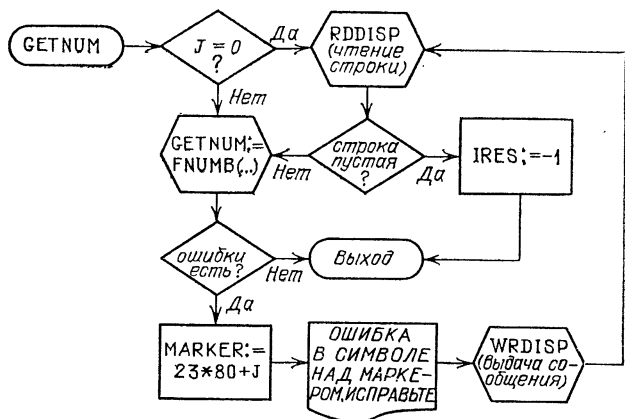


Рис. 10.1. Блок-схема ввода списка числовых значений

с плавающей запятой. По мере преобразования числа указатель J перемещается таким образом, что «смотрит» на очередной обрабатываемый байт введенной строки. Поэтому после вычисления значения функции указатель J «смотрит» на символ, прервавший процедуру перевода. Если во введенной строке было набрано несколько чисел, то повторное обращение к функции GETNUM позволяет выбрать следующее число. В этом случае между двумя обращениями к GETNUM нельзя менять значение указателя J.

Код возврата, который засылается в переменную IRES после вычисления значения функции, определяет состояние списка. При $\text{IRES} = -1$ список пуст с самого начала, т. е. пользователь в ответ на приглашение не набрал ни одного числа. Нулевое значение IRES означает, что очередное обработанное число кончилось запятой, т. е. список еще не исчерпан. Значение $\text{IRES} = 1$ свидетельствует о том, что обработано последнее число списка. Текст программы GETNUM, приведенный в [2], для работы с дисплеем

не годится, так как для выдачи сообщения об ошибке там использовались стандартные средства фортрана — вывод на устройство с номером 15. Однако аналогичную программу несложно написать на ассемблере или фортране по блок-схеме, приведенной на рис. 10.1. В ней предполагается, что числовая информация набирается пользователем в 24-й строке экрана.

Так как программа GETNUM осуществляет преобразование числовых данных с помощью функции FNUMB, то в записи целых вещественных значений точку набирать не обязательно. Кроме того, с помощью функции GETNUM можно изменять значения переменных целого типа.

Функцией GETNUM можно воспользоваться для построения подпрограммы GETMAS,¹ реализующей запланированный ввод значений элементов вещественного массива. Так как вариант программы GETMAS, приведенный в [2], не обеспечивает работу с дисплеем в ошибочных ситуациях, нами представлена более подходящая модификация этой программы.

Программа ввода значений элементов одномерного массива

```
SUBROUTINE GETMAS(A,N)
  DIMENSION A(N)
  J=0
  DO 4 K=1,N
1  A(K)=GETNUM(J,IRES)
  IF (IRES) 2,4,3
2  CALL WRDISP('ОТКАЗ ОТ ВВОДА НЕДОПУСТИМ',
  *1,1,MARKER)
  J=0
  GO TO 1
3  J=0
4  CONTINUE
  RETURN
END
```

Тривиальное обобщение за счет вложения еще одного цикла позволяет построить аналогичную программу для ввода значений двумерного массива.

Однако эти средства воздействия пользователя на свою программу в ходе вычислений предполагают жесткую схему диалога, в которой все заранее запланировано. Гораздо большая гибкость может быть достигнута за счет предоставления инициативы пользователю. Если число варьируемых параметров равно всего пяти, то количество возможных вариантов их изменения превышает 300. Поэтому запрограммировать все эти ситуации в реальной задаче не представляется возможным.

Наиболее разумно либо пойти по линии позиционного формирования значений варьируемых переменных, либо выбрать ключевой способ изменения аргументов. В первом случае фиксируется порядок варьируемых переменных. Числовые значения, которые пользователь будет вводить по запросу программы, должны следовать в таком же порядке. Например,

A1, BETA, I, D4, GAMMA

—12.5, 8.9, 15, 0.005, 18

Для того чтобы не заставлять пользователя вводить все значения, соответствующие полному списку переменных, можно договориться следующим образом. Если в числовых данных подряд следуют две или большее количество запятых, то значения соответствующих переменных должны быть сохранены. Например,

—12.5,,,0.005

Такая запись может означать, что меняться должны значения переменных A1 и D4. Реализовать такую процедуру довольно просто: ее можно встроить в прикладную программу, используя операторы присваивания конкретным переменным и запрограммировав условия их обхода в случае отсутствия соответствующих данных, а лучше выделить в подпрограмму, к которой можно будет обращаться из разных точек с одинаковыми списками варьируемых переменных. Удобнее всего было бы объединить варьируемые переменные в массив и построить программу, подобную GETMAS. Блок-схема одного из возможных вариантов такой процедуры приведена на рис. 10.2.

В предлагаемом варианте не решена проблема смешанного управления параметрами целого и вещественного типа. Ведь для этого нужно знать тип изменяемого параметра и в зависимости от типа пользоваться то функцией INUMB, то функцией FNUMB. Конечно, можно договориться, что варьируемые параметры сгруппированы в массив таким образом,¹ что первые p_1 параметров целого типа, а следующие p_2 — вещественного.

Несмотря на простоту реализации процедуры ввода, применение позиционного списка усложняет работу пользователя в тех случаях, когда число варьируемых параметров достаточно велико. Гораздо удобнее задавать значения изменяемых переменных с помощью списка параметров:

A1 = —12.5, D4 = 0.005

Здесь слева от знака равенства указывается имя ключевого параметра, которое либо в точности повторяет название соответствующей переменной программы, либо между идентификаторами ключей и переменных каким-то способом установлено взаимно однозначное соответствие. Аналогичный аппарат используется в фортране при вво-

де данных с использованием оператора **NAMELIST**. Дополнительное удобство ключевого списка заключается в том, что он позволяет задавать значения переменных в произвольном порядке.

Принем небольшого числа ключевых параметров в конкретной задаче несложно запрограммировать. После ввода одной или нескольких строк с данными, поступившими от пользователя в виде

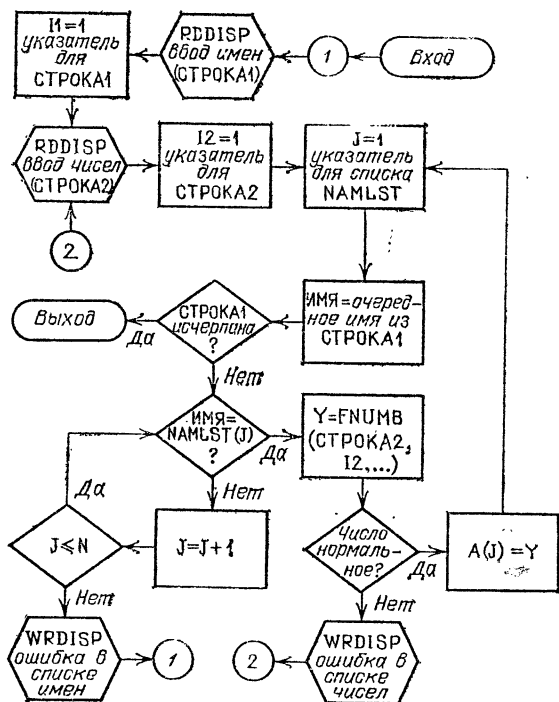


Рис. 10.2. Блок-схема обработки связанных списков имен и значений параметров

ключевого списка, надо организовать циклический просмотр с помощью подпрограммы **JKEY**. Определив вхождение имени того или иного параметра, надо преобразовать числовое значение с помощью функции **INUMB** или **FNUMB** и присвоить его соответствующей переменной. Приведем фрагмент программы обработки ключевого параметра.

```

. . . . .
CALL RDISP(СТРОКА,24,1,KEY,MARKER)
J=0
J1=JKEY(СТРОКА,J,3,'A1=')

```

```

IF (J1)20,20,10
10 A1=FNUMB(СТРОКА,J1,M)
IF (M.LT.0) GO TO 30
20 J1=JKEY(СТРОКА,J,3,'D4=')

```

.....

Когда число варьируемых параметров превышает три — пять, невыгодно включать такие однообразные фрагменты в прикладную программу. Их разумно вынести в подпрограмму. Некоторую сложность при этом составляет обилие передаваемых параметров. На каждую переменную такой подпрограмме приходится сообщать три параметра: имя переменной в форме текстовой константы для ее опознания в списке, программное обозначение этой переменной (т. е. ее адрес) для присвоения значения и тип переменной для выбора формата машинного представления при переводе числового значения. В конкретной ситуации число параметров можно сократить за счет включения текстовых констант и соответствующих им значений типов переменных в тело подпрограммы. Однако в этом случае подпрограмма лишается общности.

В пакете ДИАФОР принят несколько иной подход к изменению параметров прикладной программы в режиме диалога. Вместо того чтобы передавать модифицируемые переменные через механизм параметров, было решено воспользоваться аппаратом общей памяти. С этой целью прикладная программа размещает все варьируемые параметры в блоке общей памяти с именем INPUT:

```
COMMON /INPUT/A1,BETA,I,D4,GAMMA
```

Тем самым порядок программных переменных, допускающих изменение со стороны пользователя, жестко зафиксирован, и подпрограммы пакета ДИАФОР могут работать с ними как с элементами одномерного массива. С целью сокращения количества параметров, передаваемых прикладной программой, информация об именах и типах варьируемых переменных объединяется в два одномерных массива:

```

INTEGER NAMLST(5)/'A1___','BETA ','I___','D4___',
'GAMM'/
INTEGER NAMTYP(5)/1,1,0,1,1/

```

Порядок текстовых констант в массиве NAMLST совпадает с порядком соответствующих переменных в блоке INPUT. Значения элементов массива NAMTYP определяют тип варьируемых параметров. Единицы соответствуют переменным вещественного типа, нули — переменным целого типа. Процедура изменения параметров в порядке, определяемом пользователем, обеспечивается в пакете ДИАФОР двумя подпрограммами — INLIST и PUTLST:

```

CALL INLIST (NAMLST,NAMTYP,N,NC)
CALL PUTLST (NAMLST,NAMTYP,N,NC)

```

Параметры в обоих обращениях имеют одинаковый смысл. Значение N соответствует длине массивов NAMLST и NAMTYP. Параметр NC является выходным. В эту переменную заносится количество символов, содержащихся в сообщении пользователя. Нулевое значение NC соответствует пустому сообщению и свидетельствует об отказе пользователя внести какое-либо изменение.

В процедуре INLIST реализован ввод варьируемых параметров в виде ключевого списка. Программа INLIST предоставляет инициативу пользователю без выдачи каких-либо сообщений. Если список значений не помещается в одной строке, то пользователь должен завершить последнее число запятой. По этому признаку ему будет предоставлено право ввести следующую строку. Процедура INLIST просматривает введенное сообщение слева направо и, выделив имя очередного ключа, организует его поиск в массиве NAMLST. Порядковый номер, зафиксированный при совпадении ключа с i-м элементом массива имен, определяет местоположение соответствующей переменной в блоке общей памяти и ее тип по значению NAMTYP(i).

Процедура PUTLST исторически появилась в составе пакета раньше и была ориентирована на пользователя невысокой квалификации. Она изменяет параметры прикладной программы в два приема. Сначала у пользователя запрашивается список имен переменных, которые он в данный момент хотел бы изменить. Затем ему предлагают ввести числовые значения указанных переменных. Диалог проистекает по следующему сценарию:

ЭВМ: ЗНАЧЕНИЯ КАКИХ ПАРАМЕТРОВ ВЫ ХОТИТЕ
 ИЗМЕНИТЬ?

ЧЕЛОВЕК: D4,I,GAMMA

ЭВМ: ВВЕДИТЕ ИХ ЗНАЧЕНИЯ В ТАКОМ ЖЕ
 ПОРЯДКЕ

ЧЕЛОВЕК: 0.005, 15, 18

Этот вариант ближе к позиционному способу, однако пользователю предоставлено право каждый раз изменять последовательность переменных и их количество в списке.

Варианты программ PUTLST и INLIST, ориентированные на работу в режиме консоли оператора, приведены в [2]. Их легко приспособить для работы с дисплеем, изменив процедуру вывода сообщений об ошибках. Обратим внимание на некоторые недостатки указанных программ. Во-первых, они ориентированы на изменение простых переменных целого и вещественного типа стандартной длины. Было бы удобно иметь возможность модифицировать отдельные элементы массивов, разрешать задание числовых данных в экспоненциальном формате. Во-вторых, анализ имен построен в расчете на сравнение первых четырех символов. Это ограничение можно было бы легко обойти. Например, задавать имена переменной длины

в логическом массиве NAMLST с однобайтовыми элементами. А длины имен совместить с массивом NAMTYP, задавая для переменных целого типа, например, отрицательные длины, а для переменных вещественного типа положительные. В-третьих, следовало бы перестроить и систему индикации ошибок — зафиксировать для сообщений определенную строку экрана, использовать положение курсора для указания ошибочного элемента. Остановимся несколько подробнее на основных моментах реализации процедур, обеспечивающих выборочное изменение элементов массивов.

Если сценарий диалога таков, что в каждой конкретной точке пользователю предоставляется возможность изменить любые элементы одного массива, то такую процедуру можно реализовать следующим образом. Включим в состав параметров, передаваемых подпрограмме, текстовую константу с именем массива, идентификатор массива (т. е. адрес его первого элемента) и граничные значения. Конечно, такого рода подпрограмму с переменным числом параметров проще написать на ассемблере. Но можно и на фортране, если разнести входы с одномерными и двумерными массивами, используя оператор ENTRY. Разумно предположить, что сообщение, принимаемое пользователем, похоже на ключевой список:

$$A(3)=800, \quad A(8)=-12.2, \quad A(2)=1.5$$

или

$$R(1,3)=5.4, \quad R(2,7)=8, \quad R(2,2)=-1.8$$

Анализ такого сообщения можно построить по следующему алгоритму:

1. Установить указатель обрабатываемой строки в начальное положение.
2. С помощью подпрограммы JKEY установить ближайшее вхождение имени массива.
3. «Перешагнуть» открывающую скобку ($J=J+1$) и с помощью подпрограммы INUMB обработать индексное выражение.
4. «Перешагнуть» через символы “) = “ и присвоить заданное значение соответствующему элементу массива.
5. Если обработанное значение завершилось запятой и сообщение не исчерпано, перейти к п. 2. Если сообщение завершено запятой, правее которой ничего нет, предоставить пользователю возможность ввести следующую строку и начать обработку с п. 1.

Работа подпрограммы завершается после обработки числового значения, правее которого в строке ничего нет.

Для массивов небольшой размерности, все компоненты которых нужно показать на экране дисплея, можно предложить более удобную схему модификации элементов. Установив единый формат расположения значений элементов массива на экране, можно преобразовать

зовать их из машинного представления в символьное с помощью подпрограммы FORM и выдать на экран. Пользователь скорректирует любые из этих значений, после чего все элементы массива считываются с экрана, преобразуются в нужный машинный формат и записываются по своим адресам в оперативную память.

Немного сложнее реализовать подпрограмму обработки сообщений пользователя, когда в пределах одной строки изменяются элементы нескольких массивов. Если, например, все модифицируемые массивы одномерны, то можно попытаться построить следующий аналог подпрограммы INLIST:

```
SUBROUTINE CHNGAR (LISTAR,LENTYP,N)
COMMON /MASSIV/R(1)
DIMENSION I(1),LISTAR(N),LENTYP(N)
EQUIVALENCE (R(1),I(1))
. . . . .
```

Формальный массив LISTAR используется для задания первых четырех символов имен модифицируемых массивов прикладной программы. Аргумент LENTYP определяет максимальное количество элементов в каждом из этих массивов и тип входящих в них элементов (для массивов целого типа задается отрицательное значение размерности, для массивов вещественного типа — положительное). С целью сокращения числа передаваемых параметров модифицируемые массивы собраны в блоке общей памяти с именем MASSIV. Обработка очередного фрагмента введенного сообщения производится по следующему алгоритму:

1. Организуется цикл перебора имен из массива LISTAR и их сравнение с текущим именем изменяемого элемента. При каждом несовпадении к переменной IND прибавляется длина очередного «неудовлетворительного» массива.

2. После совпадения осуществляется переход к символу, следующему за открывающей скобкой, и преобразование индексного значения. Сумма последнего и значения переменной IND определяют порядковый номер k модифицируемого элемента в блоке общей памяти с именем MASSIV.

3. С помощью подпрограммы JKEY перемещаем указатель на символ, следующий за знаком «равно», и в зависимости от знака размерности выявленного массива обращаемся к подпрограмме INUMB или FNUMB. Результат преобразования числа в машинный формат присваивается переменным I(k) или R(k) соответственно. В случае, когда среди модифицируемых массивов должны встречаться и одномерные, и двумерные массивы, можно предложить следующий путь. Сконцентрируем все одномерные массивы в одном блоке общей памяти, а все двумерные в другом. Тогда процедура обработки сообщения пользователя немного усложнится, так как каждый

фрагмент придется тестировать дважды: сначала вести поиск среди имен одномерных массивов, а в случае неудачи — и среди имен двумерных массивов. А в остальном, кроме увеличения числа передаваемых параметров, работа подпрограммы CHNGAR принципиально не изменится.

10.5. Отображение выходных параметров по запросу пользователя

При программировании сценария диалога между пользователем и прикладной программой полезно предусмотреть точки, в которых программист имеет возможность просмотреть на экране дисплея значения интересующих его переменных. Если число таких переменных невелико, то проще всего заранее запрограммировать вывод всех этих значений на экран дисплея в виде удобочитаемой таблицы, а при появлении соответствующего запроса пользователя выдавать на экран эту таблицу целиком. Однако фрагмент программы, приведенной ниже и реализующей процедуру подобного рода, показывает, что эти однообразные операции удобнее переложить на подпрограмму.

Фрагменты программы подготовки и вывода результатов:

```
J=1
CALL FORM('ALPHA=','A6_',CTPK23,J,IRES)
CALL FORM(ALPHA,'F8.4',CTPK23,J,IRES)
CALL FORM(J,'X2_',CTPK23,J,IRES)
CALL FORM('D4=','A3_',CTPK23,J,IRES)
CALL FORM(D4,'E12.4',CTPK23,J,IRES)
. . . . .
CALL WRDISP(CTPK23,23,1,MARKER)
```

Приведем один из простейших вариантов подпрограммы форматного преобразования и вывода результатов:

```
SUBROUTINE OUTVAR (NAMLST,FORMAT,N)
LOGICAL *1 FORMAT(1),CTPOKA(80)
REAL *8 NAMLST(1)
COMMON /OUT/R(1)
J=1
DO 2 K=1,N
CALL FORM(NAMLST(K),'A8_',CTPOKA,J,IRES)
2 CALL FORM(R(K),FORMAT(1+(K-1)*5),CTPOKA,
* J,IRES)
CALL WRDISP(CTPOKA,23,1,23*80)
RETURN
END
```

Массив NAMLST состоит из восьмибайтовых элементов, в которых подготовлены текстовые константы вида ' $S_1S_2S_3S_4S_5S_6 =$ ', соответствующие именам отображаемых переменных. Логический массив FORMAT состоит из однобайтовых переменных, в которых порциями по 5 байтов заданы форматы, используемые для преобразования машинных значений в символьный вид. Программа OUTPUT переносит в выводную строку имя переменной, выбирает ее значение из очередного элемента блока общей памяти и преобразует его в соответствии с очередным редактирующим указателем. После обработки всех переменных массив СТРОКА выводится на экран дисплея.

По такому же принципу можно построить программу, выдающую на экран информацию, состоящую из нескольких строк. Несложно сконструировать и программу, отображающую на экран элементы двумерного массива по единому формату.

Несколько сложнее устроены программы вывода, в которых заранее не известны порядок и количество переменных, затребованных пользователем. К их числу относится программа GETLST, полный текст которой приведен в [2]. Программа GETLST «знает» список всех переменных прикладной программы, значения которых пользователь может запросить. Для уменьшения количества параметров в обращении эти переменные упорядочены в блоке общей памяти с именем OUTPUT. Подобно программам PUTLST и INLIST, подпрограмме GETLST передаются два массива — NAMLST и NAMTYP, содержащие список имен выводных переменных и характеристику их типов. Диалог с пользователем выполняется в два этапа.

ЭВМ: ЗНАЧЕНИЯ КАКИХ ПАРАМЕТРОВ ВЫ ХОТИТЕ
УЗНАТЬ?

ЧЕЛОВЕК: C4, GAMMA, I, A5

ЭВМ: $-0.180000E+02$ 145 18 $0.958667E-03$

Список переменных, которые набираются на запрос программы, должен содержать только те имена, которые включены в состав массива NAMLST. Однако их количество и порядок могут быть произвольными. Программа GETLST сначала устанавливает соответствие между заданными именами и местоположением затребованных значений в блоке общей памяти, а затем преобразует их в соответствии со стандартными форматами вывода вещественных или целых чисел.

ЗАКЛЮЧЕНИЕ

Опыт разработки описанных в книге систем приводит к мысли, что можно построить универсальный диалоговый монитор, обеспечивающий общение пользователя с прикладной программой на уровне директив, близких по формату к предложениям естественного языка. В такой монитор могут быть перенесены некоторые сервисные блоки редактора — прием и анализ директив, выдача сообщений о нормальном или аварийном завершении процедур, программы обработки числовых параметров и т. п.

Естественно, что исполнителями директив будут сменные программы, зависящие от специфики решаемой задачи. Однако многие функции могут быть общими для всех прикладных программ. К их числу можно отнести директивы без параметров, управляющие сменой содержимого экрана, получением твердой копии экрана, выдачей справочной информации и т. п. Ряд директив с параметрами также можно отнести к разряду универсальных. Например, ввод заданного числа перфокарт с исходными данными, построчный обмен с магнитной лентой, просмотр диалоговых кадров с информацией от предыдущих шагов, возврат к указанному шагу сценария диалога и т. п. По аналогии с редактором можно было бы заводить архивы с исходными данными и результатами расчетов, формировать из них нужные фрагменты для новых исследований или для оформления отчетных документов.

Мы далеки от мысли, что можно написать программу такого монитора, которая была бы одинаково хороша на любой случай жизни. Однако набор сервисных и функциональных программ общего назначения, использование единой технологии обработки сообщений пользователя, набор эффективных приемов программирования стандартных ситуаций в значительной мере облегчают создание конкретных диалоговых систем. При их построении очень важно разработать формат экрана, на котором помимо данных, адресованных участникам диалога, должна отражаться служебная информация, связанная с функционированием систем. Большое значение имеют структура входного языка и информационная точность сообщений системы, удобная мнемоника и выразительность средств общения, возможность сокращать служебные слова.

Если все переменные прикладной программы вынести за пределы программы в область, аналогичную USAREA, то появляется возможность организовать многотерминальную работу. В этом случае следует обратить внимание на достижение максимального быстродействия исполнителей, чтобы не заставлять пользователей скучать у застывших дисплеев.

Все эти благие пожелания адресуются читателям, к которым нам хотелось бы обратиться с лозунгом одной из телевизионных программ — «Делай с нами, делай, как мы, делай лучше нас!».

П Р И Л О Ж Е Н И Е

РЕЕНТЕРАБЕЛЬНЫЕ ПРОГРАММЫ И РЕКОМЕНДАЦИИ ПО ИХ РАЗРАБОТКЕ

В главе 8 упоминалось о том, что одним из путей уменьшения общих затрат ОП при реализации ПО параллельной динамической структуры является разработка реентерабельных программ. Поскольку в документации по ОС ЕС вопросам разработки реентерабельных программ уделено крайне незначительное внимание, мы сочли возможным поделиться нашим скромным опытом разработки таких программ.

Прежде всего напомним, что понимают под термином «реентерабельность». *Реентерабельность* определяют как возможность повторного входа в программу до того, как она полностью завершила работу в результате предыдущего входа в нее. Иначе говоря, реентерабельность — это свойство программы, позволяющее двум или более задачам одного или нескольких заданий использовать ее одновременно.

Формально характеристика реентерабельности может быть присвоена только в целом всему загрузочному модулю (независимо от того, обладает ли он в действительности свойством реентерабельности и состоит ли он из одной программной секции или нескольких). Характеристика реентерабельности присваивается загрузочному модулю программистом на этапе редактирования связей в программе. Для этого при вызове РЕДАКТОРА СВЯЗЕЙ указывается характеристика RENT в поле PARM:

```
// LKED EXEC PGM=IEWL,PARM='RENT,...'
```

РЕДАКТОР СВЯЗЕЙ, помещая загрузочный модуль в библиотеку, одновременно сделает в элементе оглавления соответствующую отметку о том, что модуль является реентерабельным (устанавливает 0 в 1-ый бит в поле PDS2ATR элемента оглавления).

Формального присвоения атрибута реентерабельности некоторому загрузочному модулю достаточно для того, чтобы программы супервизора ОС ЕС, выполняющие функции макрокоманд LOAD, LINK, XCTL и ATTACH, рассматривали его таковым и организовали использование единственной его копии в ОП различными задачами, однако ясно, что одного этого условия недостаточно для корректного использования данного модуля различными

параллельными процессами. Прежде всего необходимо, чтобы все программные секции, входящие в состав загрузочного модуля, в действительности обладали свойством реентерабельности, т. е. чтобы каждая из них не модифицировалась во время своего выполнения и, кроме того, не могла быть модифицирована другими загрузочными модулями.

Как же достичь этого и какие адекватные приемы программирования при этом использовать? Ниже приводится ряд практических рекомендаций по разработке реентерабельного ПО, выработанных в процессе проектирования системы БЕС7920.

Оформление входа в программную секцию и выхода из нее. Обычно первое, что должна сделать программа, претендующая на реентерабельность, — это обеспечить реентерабельную область сохранения. Для этих целей можно использовать макрокоманду FROMAS, предназначенную для оформления программных секций в системе БЕС7920 (соответствующее макроопределение приводится ниже).

```

MACRO
&C      FROMAS &BASE,&RENT=YES,&L=88,*
              &SBT=YES, *&SVAR=YES
&C      AIF    ('&BASE' EQ ''),NOCS
CSECT
USING   &C,&BASE уст. базу прог. секции
SAVE    (14,12),* сохранить регистры
LR      &BASE(1),15 загрузить регистр
              базы
      AGO      .NOBASE
.NOCS    ANOP
&C      SAVE   (14,12)
.NOBASE  AIF    ('&SVAR' EQ 'YES').NOBASEA
      MEXIT
.NOBASEA AIF    ('&RENT' NE 'YES').OLD
***ПОЛУЧИТЬ РЕЕНТЕРАБЕЛЬНУЮ ОБЛАСТЬ СОХРАНЕНИЯ
      LA      0,&L.(0,0) длина реент. области сохр.
      AIF    ('&SBT' NE 'YES').NOSBT
      L      1,16 адрес "CVT" ОС ЕС
      L      1,0(1) адрес двойного слова
      L      1,4(1) адрес "TCB" акт. задачи
      L      1,X'A8'(1) адрес "DCT" из "TCBUSER"
      IC     1,X'1CF'(1) относ. номер дисплея
      SLL    1,24
      OR     1,0
.NOGBT   BAL   1,*+4
      SVC    10
      AGO    .NEW
  
```

.OLD	ANOP		
	CNOP	0,4	
	BAL	1,BR&SYSNDX	
	DC	22F'0'	стат. обл. сохранения
.NEW	ANOP		
BR&SYSNDX	ST	13,4(1)	
	ST	1,8(13)	
	LM	14,1,12(13)	восст. регистры 14,15,0,1,
	L	13,8(13)	загр. R13
	MEND		

Макроопределение FROMAS

Как видно из текста макроопределения, с помощью макрокоманды FROMAS можно получить как динамическую (реентерабельную), так и статическую область сохранения. Кроме обеспечения области сохранения, по макрокоманде FROMAS можно создать стандартный пролог программной секции (присвоение имени программной секции, сохранение содержимого регистров вызывающей программы, указание регистра базы и загрузка его адресом точки входа в программную секцию).

Поясним назначение операнда SBT макрокоманды FROMAS. Дело в том, что программы, пролог которых генерируется с помощью данной макрокоманды, работают в рамках подзадач пользователей, количество которых соответствует числу активных дисплеев. Каждый активный дисплей имеет однозначный относительный номер, и этот факт использован при запросах динамической памяти программами подзадачи пользователя: она запрашивается из подпула с номером, равным относительному номеру дисплея. (Относительный номер дисплея содержится в поле DCTRLN таблицы DCT, а указатель таблицы DCT — в поле TCBUSER блока TCB подзадачи пользователя.)

С помощью операнда L макрокоманды FROMAS можно получить динамическую память большего размера, чем та, которая требуется для области сохранения регистров. Назначение этой возможности макрокоманды будет рассмотрено чуть ниже.

Для возврата управления в вызывающую программу предусмотрена макрокоманда BACKAS (текст соответствующего макроопределения приведен ниже).

	MACRO	
&C	BACKAS	&SVAR=YES,&RENT=YES,&L=88,* &SBT=YES,&RC=0
&C	DS	0H выход из программы
	AIF	('&SVAR' NE 'YES').NORENT
	L	13,4(13) адр.обл.сохр.выз-щей программы

AIF	('&RENT' NE 'YES').NORENT	
LA	0,&L.(0,0)	<i>длина освобождаемой области</i>
AIF	('&SBT' NE 'YES').NOSBT	
L	1,16	<i>адрес 'CVT' ОС ЕС</i>
L	1,0(1)	<i>адрес двойного слова</i>
L	1,4(1)	<i>адрес 'TCB'</i>
L	1,X'A8'(1)	<i>адрес 'DCT' из поля 'TCBUSER'</i>
IC	1,X'1CF'(1)	<i>поле 'DCTRLN'</i>
SLLE	1,24	<i>номер подпула → в старш.байт</i>
OR	0,1	

.NOSBT ANOP

***ОСВОБОДИТЬ СВОЮ (РЕЕНТЕРАБЕЛЬНУЮ) ОБЛАСТЬ

*

L	1,8(13)	<i>адрес обл.сохр.</i>
LA	1,0(1)	
SVC	10	

.NORENT ANOP

SPACE

L	14,12(13)	<i>восстановить адрес возврата</i>
MVI	12(13),0	<i>отметить обл.сохр. как 'НЕИСП.'</i>
LM	0,12,20(13)	<i>восстановить регистры 0—12</i>
AIF	('&RC' EQ '(15)').EXIT	
LA	15,&RC.(0,0)	<i>установить код возврата</i>

.EXIT ANOP

BR	14 +++	<i>возврат управления</i>
MEND		

Макроопределение BACKAS

С помощью ее можно оформить стандартный эпилог реентерабельной программы, в который, кроме функции восстановления регистров вызывающей программы, включена операция по освобождению динамической области сохранения, полученной по макрокоманде FROMAS. Смысл операнда RC — тот же, что и в макрокоманде RETURN супервизора ОС ЕС.

Использование в программе системных макрокоманд. Как известно, многие макрокоманды ОС ЕС (макрокоманды супервизора, управления данными и других компонент) используют списки параметров и рабочие области для обмена информацией между программой пользователя и соответствующими компонентами ОС ЕС. Макрокоманды, закодированные в *стандартной форме* (операнды MF и SF закодированы в виде MF=I и SF=I или опущены), вызывают генерацию этих списков и областей внутри макрорасширения вместе с некоторым исполнительным кодом, содержащим вызов соответствующей компоненты ОС ЕС. Иллюстрацией использования стандартной формы макрокоманд указанного типа может служить следующий фрагмент программы:


```

      * * * * *
      PRINT GEN
      OPEN (IN,(INPUT))
+     CNOP 0,4
+     BAL 1,*+8
+     DC AL1(128)
+     DC AL3(IN)
+     SVC 19

      * * * * *
      READ INDECB,SF,IN,INAREA,400
+     CNOP 0,4
+     BAL 1,*+24
+INDECB DC F'0'
+     DC X'00'
+     DC X'80'
+     DC AL2(400)
+     DC A(IN)
+     DC A(INAREA)
+     DC A(0)
+     L 15,8(1,0)
+     L 15,48(0,15)
+     BALR 14,15
      CHECK INDECB
+     LA 1,INDECB
+     L 14,8(0,1)
+     L 15,52(0,14)
+     BALR 14,15

      * * * * *
      PRINT NOGEN
IN     DCB DSORG=PS,MACRF=(R),DDNAME=IND
INAREA DS CL400
      . . . . .

```

Очевидно, что использование стандартной формы приведенных макрокоманд неприемлемо при разработке реентерабельных программ. Для организации реентерабельной связи между программой пользователя и соответствующими программами ОС ЕС предусмотрены две другие формы макрокоманд: описательная и исполнительная. *Описательная форма* макрокоманды используется для построения списка параметров, который должен быть передан программе, вызываемой с помощью исполнительной формы той же макрокоманды. *Исполнительная форма* макрокоманды используется совместно с одним или двумя списками параметров, полученными с помощью описательной формы. Расширение исполнительной формы представляет собой выполняемые команды, не-

обходимые для настройки списка параметров и передачи управления вызываемой программе. Таким образом, аналогичный фрагмент реентерабельной программы должен содержать ряд дополнительных операций, связанных с выносом из программы в некоторую динамическую область памяти всех изменяемых ее частей:

* * * * *

```
GETMAIN R,LV=DYNARL
LR 9,1
USING DYNAR, 9
MVC DYNAR(DYNARL1),RDYNAR
OPEN (IN,(INPUT)),MF=(E,OPNPARM)
```

* * * * *

```
READ INDECB,SF,IN,INAREA,400,MF=E
CHECK INDECB
```

* * * * *

```
CLOSE IN,MF=(E,OPNPARM)
FREEMAIN R,LV=DYNARL,A=(9)
```

* * * * *

```
RDYNAR DS 0D списки параметров,DECB,DCB
ROPNP OPEN (INR,(INPUT)),MF=L
READ RINECB,SF,MF=L
RDCB DCB DSORG=PS,MACRF=(R),DDNAME=INDD
DYNARL1 EQU *—RDYNAR
RINREA DS CL400
DYNARL EQU *—RDYNAR
LTORG
```

* * * * *

```
DYNAR DSECT
OPNPARM OPEN (IN,(INPUT)),MF=L
READ INDECB,SF,MF=L
IN DCB DSORG=PS,MACRF=(R),DDNAME=INDD
INAREA DS CL400
```

* * * * *

Использование рабочих областей. В приведенном выше примере в неявном виде было уже показано, что рабочие области, необходимые для работы программы (область INAREA), должны быть вынесены за пределы программы, если необходимо обеспечить ее реентерабельность. Как обычно, динамическая область необходимой длины запрашивается с помощью регистровой (реентерабельной) формы макрокоманды GETMAIN и освобождается после ее использования с помощью аналогичной формы макрокоманды FREEMAIN. Однако частые запросы на получение и освобождение памяти по мере того, как появляется необходимость в ней, могут привести к снижению быстродействия программы и к излишней

фрагментации памяти раздела, отведенного заданию. В связи с этим более предпочтительным может оказаться объединение всех запросов на динамическую память в один запрос. Выделение динамической памяти программе в виде одного непрерывного участка дает еще одно преимущество: для базирования полученных областей и сохранения адресных указателей не требуется большого количества регистров. В связи с этим можно рекомендовать запрашивать динамическую память вместе с реентерабельной областью сохранения в прологе программы, а освобождать ее в эпилоге. Вероятно, сейчас становится понятным назначение операнда L в макрокомандах FROMAS и BACKAS. Используя эти макрокоманды, приведенный выше фрагмент программы можно было бы изменить следующим образом:

```

READPGM FROMAS 12,RENT=YES,L=SVL
      USING SVAR,13
SVAR      DSECT
SAVE      DS 18F реент., область сохранения
           DS 4F  спец. область 'БЕС7920'
DYNAR     EQU * доп.дин.память программы
OPNPARM   OPEN (IN,(INPUT)),MF=L список парам-ров 'OPEN'
           READ INDECB,SF,IN,MF=L блок DECB 'INDECB'
IN         DCB DSORG=PS, MACRF=(R),DDNAME=INDD
INAREA    DS CL400 буфер ввода
SVL       EQU *—SVAR
READPGM   CSECT

      * * * * *
      MVC DYNAR(DYNARL1), RDYNAR
      OPEN (IN,(INPUT)),MF=(E,OPNPARM)

      * * * * *
      READ INDECB,SF,IN,INAREA,400,MF=E
      CHECK INDECB

      * * * * *
      CLOSE IN,MF=(E,OPNPARM)

      * * * * *
      BACKAS RC=00,L=SVL

      * * * * *
RDYNAR    DS 0D
ROPNP     OPEN (INR,(INPUT)),MF=L
           READ RINDECB,SF,MF=L
RDCB      DCB DSORG=PS,MACRF=(R),DDNAME=INDD
DYNARL1   EQU *—RDYNAR

```

Таким образом, основные усилия разработчика реентерабельной программы обычно направлены на обеспечение постоянства машинного кода программы в процессе ее выполнения путем пере-

носа всех модифицируемых элементов программы в некоторую динамическую память, приобретаемую из фонда свободной памяти раздела.

В заключение отметим, что при отладке логики реентерабельной программы обычно используются те же методы и средства, которые применяются при отладке обычных (не реентерабельных) программ. Однако в условиях мультизадачной обработки поиск ошибок, связанных с нарушением реентерабельности программы, весьма затруднителен в связи с асинхронностью параллельных процессов и отсутствием в ОС ЕС каких-либо адекватных средств динамической отладки мультизадачной логики. Единственным (скорее «утешительным», нежели эффективным) средством диагностики реентерабельности программы является режим RENT, задаваемый при трансляции исходного текста с языка ассемблера:

|| EXEC ASMFC,PARM.ASM=(RENT, . . .)

В соответствии с этим режимом транслируемая программа проверяется на возможные нарушения, допущенные программистом в отношении реентерабельности программы: транслятор включает в диагностическую часть листинга программы сообщение с кодом IFO229 (код серьезности 4), если есть подозрение, что некоторая машинная команда пытается модифицировать что-либо в теле программы.

СПИСОК ЛИТЕРАТУРЫ

Основной

1. Данилович В. П., Одинцов Б. В., Пеледов Г. В. Справочник системного программиста по операционной системе ОС ЕС/Под ред. Л. Д. Райкова.— М.: Финансы и статистика, 1982.— 288 с.
2. Кетков Ю. Л., Максимов В. С., Рябов А. Н. Введение в системное программирование на языке ассемблера ЕС ЭВМ.— М.: Наука, 1982.— 264 с.
3. Операционная система ОС ЕС: Справочное пособие/В. П. Данилович, В. В. Митрофанов, Б. В. Одинцов, Г. В. Пеледов; Под ред. Л. Д. Райкова.— М.: Финансы и статистика, 1980.— 480 с.
4. Хромов В. И., Ульянов С. А. Введение в программирование для систем телеобработки данных.— М.: Финансы и статистика, 1982.— 176 с.

Дополнительный

5. Блажнов В. Ю., Молчанов В. А. RLTCAM — пакет для создания программ телеобработки на языке ПЛ-1.— В кн.: Прикладная информатика. М.: Финансы и статистика, 1982, вып. 2, с. 69—78.
6. Борисевич В. Ф. и др. Система разделения времени ЕС ЭВМ/Под ред. Э. В. Ковалевича.— М.: Финансы и статистика, 1982.— 239 с.
7. Брич З. С., Капплевич Д. В., Терехова О. Г. Программирование на фортране ЕС ЭВМ в режиме разделения времени.— М.: Финансы и статистика, 1982.— 192 с.
8. Диалоговый редактор исходных текстов. Язык команд/В. И. Зверев, Г. В. Андреев, Н. С. Буракова, Л. П. Кабанова, Ю. Н. Сотсков.— Ин-т техн. киберн. АН БССР, Минск, 1983.— 150 с.
9. Диалоговая система коллективного доступа PRIMUS/В. В. Васильков, В. М. Иванов, В. В. Еломапова, П. В. Чеукасов.— Моск. инж.-физ. ин-т, 1980.— 25 с. (Инф. бюлл. ВНИИЦ «Алгоритмы и программы», 1982, вып. 1, рег. № ГФАП — П005027).
10. Кетков Ю. Л., Максимов В. С. Редактирование символьной информации в режиме диалога под управлением ДОС ЕС.— Тезисы докл. IV Всесоюз. конф. «Системное и теоретическое программирование», Кишинев, 1983, с. 202—203.
11. Кетков Ю. Л., Максимов В. С., Тафорица Н. М. Форматные преобразования числовой информации, предназначенной для вывода на ЕС ЭВМ.— НИИ прикл. мат. и киберн. при Горьковском гос. ун-те, 1978.— 29 с. (Инф. бюлл. ВНИИЦ

«Алгоритмы и программы», 1980, вып. 4, рег. № ГФАП — П004372).

12. Кузмин А. Я., Кирдяпкин А. В., Соболев А. Н. О применении программных средств пакета КОДИАЛ для решения конструкторских задач в диалоговом режиме.— УСиМ, 1984, № 2, с. 47—49.
13. Любимский Э. З., Малинкин А. В. Современные диалоговые редакторы.— Препринт Ин-та прикл. матем. АН СССР, М., 1979.— 51 с.
14. Новые средства программирования для ЕС ЭВМ: Транслятор с языка Алгол-68 и диалоговая система ЈЕС/Г. Ф. Дейкало, Б. А. Новиков, А. П. Рухлин, А. Н. Терехов.— М.: Финансы и статистика, 1984.— 207 с.
15. Программное обеспечение комплексов ЕС-7920.— В кн.: Математическое обеспечение ЕС ЭВМ, вып. 30. Ин-т мат. АН БССР, НИИ ЭВМ, Минск, 1981.— 212 с.
16. Руско-Усков Ю. А. Программирование ввода-вывода и управление экранами для локального комплекса ЕС-7920 на языке ПЛ-1.— Вопросы радиоэлектроники, Серия АСУПР, 1984, вып. 3, с. 68—70,

*Владимир Иванович Зверев
Юлий Лазаревич Нетков
Владимир Сергеевич Максимов*

**АЛФАВИТНО-ЦИФРОВЫЕ ДИСПЛЕИ ЕС-7920
В ДИАЛОГОВЫХ СИСТЕМАХ**

Серия «Библиотека программиста», вып. 48

*Редактор О. И. Сухова
Художественный редактор Т. Н. Кольченко
Технический редактор Е. В. Морозова
Корректор И. Я. Кристаль*

ИБ № 12822

Сдано в набор 23.12.85. Подписано к печати 06.08.86. Т-16798.
Формат 84×108/32. Бумага книжно-журнальная.
Гарнитура обыкновенная. Печать высокая. Усл. печ. л. 12,6.
Усл. кр.-отт. 12,81. Уч.-изд. л. 15,94, Тираж 41 500 экз, Заказ 2385.
Цена 95 коп.

Ордена Трудового Красного Знамени издательство «Наука»
Главная редакция физико-математической литературы
417071 Москва В-71, Ленинский проспект, 15

2-я типография издательства «Наука»
421099 Москва Г-99, Шубинский пер., 6

СЕРИЯ «БИБЛИОТЕЧКА ПРОГРАММИСТА»

1. А. Л. Б р у д н о. Алгол.— 1968.
2. Ж. Б е р т е н, М. Р и т у, Ж. Р у ж и ё. Работа ЭВМ с разделением времени / Пер. с франц.; Под ред. С. С. Лаврова.— 1970.
3. С. С. Л а в р о в, Л. И. Г о н ч а р о в а. Автоматическая обработка данных (хранение информации в памяти ЭВМ).— 1974.
4. А. Л. Б р у д н о. Алгол.— Изд. 2-е.— 1971.
5. В. А. В а с и л ь е в. Язык алгол-68, Основные понятия / Под ред. С. С. Лаврова.— 1972.
6. Ю. А. П е р в и н. Основы фортрана.— 1972.
7. Ж. Б е р т е н, М. Р и т у, Ж. Р у ж и ё. Работа ЭВМ с разделением времени.— Изд. 2-е.— 1972.
8. И. Л. Б р а т ч и к о в. Синтаксис языков программирования / Под ред. С. С. Лаврова.— 1975.
9. Л. Дж. К о э н. Анализ и разработка операционных систем / Пер. с англ.; Под ред. В. Ф. Тюринина.— 1975.
10. Д. У о л ш. Руководство по созданию документации для математического обеспечения / Пер. с англ.; Под ред. Н. И. Козлова.— 1975.
11. Б. М. П а в л о в, И. Н. П о с о х о в. Математическое обеспечение ЭВМ типа М-20.— 1975.
12. В. Я. К а р п о в. Алгоритмический язык фортран. Фортран-Дубна / Под ред. Н. Н. Говоруна.— 1976.
13. А. И. С а л т ы к о в, Г. И. М а к а р е н к о. Программирование на языке фортран / Под ред. Н. Н. Говоруна.— 1976.
14. В. Н. Б у с л е н к о. Автоматизация имитационного моделирования сложных систем / Под ред. и с послесловием Н. П. Бусленко.— 1977.
15. В. Ф. Д ё м ц и н, Л. В. Д о б р о л ю б о в, В. А. С т е п а н о в. Системы программирования на алголе / Под ред. Е. А. Гребеникова.— 1977.
16. Г. М. А д е л ь с о н - В е л ь с к и й, В. Л. А р л а з а р о в, М. В. Д о н с к о й. Программирование игр.— 1978.
17. В. В. К о б е л е в. Машинная графика для системы БЭСМ-алгол / Под ред. Г. Г. Рябова.— 1978.
18. С. С. Л а в р о в, Г. С. С и л а г а д з е. Автоматическая обработка данных. Язык лисп и его реализация.— 1978.
19. Г. Л. М а з н ы й. Программирование на БЭСМ-6 в системе «Дубна» / Под ред. Н. Н. Говоруна.— 1978.
20. Г. Г. Б е л о н о г о в, А. П. Н о в о с е л о в. Автоматизация процессов накопления, поиска и обобщения информации.— 1979.
21. Ф. П. Б р у к с м л. Как проектируются и создаются программные комплексы, Мифический человек-месяц. Очерк по систем-

- ному программированию / Пер. с англ.; Под ред. А. П. Ершова.— 1979.
22. Н. П. Брусеинов. Миникомпьютеры / Под ред. Л. Н. Королева.— 1979.
 23. В. Ф. Жиров. Математическое обеспечение и проектирование структур ЭВМ / Под ред. Л. Н. Королева.— 1979.
 24. Транслятор альфа-6 в системе «Дубна» / Под ред. А. П. Ершова.— 1979.
 25. С. А. Усов. Диалоговый монитор Димон / Под ред. В. М. Брябрина.— 1970.
 26. Ю. М. Безбородов. Сравнительный курс языка PL/1 (на основе алгола-60).— 1980.
 27. В. Р. Хисамутдинов, В. С. Авраменко, В. И. Легоньков. Автоматизированная система информационного обеспечения разработок.— 1980.
 28. Н. И. Козлов. Организация вычислительных работ.— 1981.
 29. М. Крайн, О. Лемуан. Введение в регенеративный метод анализа моделей / Пер. с англ.; Под ред. В. В. Калашникова.— 1982.
 30. Ю. М. Безбородов. Индивидуальная отладка программ.— 1982.
 31. Л. А. Осипов. Язык аналитик и его сравнение с языками алгол и фортран.— 1982.
 32. В. М. Пентковский. Автокод эльбрус / Под ред. А. П. Ершова.— 1982.
 33. С. Н. Бушев, М. С. Бесфамильный. Программно-аппаратные методы управления данными / Под ред. С. В. Емельянова.— 1982.
 34. К. Жаблон, Ж. К. Симон. Применение ЭВМ для численного моделирования в физике. / Пер. с франц.; Под ред. В. В. Александрова и Ю. С. Вишнякова.— 1983.
 35. З. Д. Усманов, Т. И. Хаитов. Программирование состояний коллекций.— 1983.
 36. Г. Г. Белоногов, Б. А. Кузнецов. Языковые средства автоматизированных информационных систем.— 1983.
 37. В. Н. Пильщиков. Язык плэпер.— 1983.
 38. Ю. М. Безбородов. От фортрана к PL/1.— 1984.
 39. А. И. Салтыков, Г. И. Макаренко. Программирование на языке фортран. Изд. 2-е.— 1984.
 40. А. В. Гуляев, Н. В. Макаров-Землянский, И. В. Машечкин. Диалоговый комплекс программы Краб / Под ред. Л. Н. Королева.— 1985.
 41. В. А. Евстигнеев. Применение теории графов в программировании / Под ред. А. П. Ершова.— 1985.
 42. В. Ф. Тюрин. Операционная система Диспак.— 1985.
 43. А. Н. Андрияпов, С. П. Бычков, А. И. Хорошилов. Программирование на языке симула-67.— 1985.
 44. Ю. М. Баяковский, В. А. Галактионов, Т. Н. Михайлова. Графор. Графическое расширение фортрана.— 1985.
 45. С. А. Абрамов. Элементы анализа программ.— 1986.
 46. В. И. Зуев, В. М. Крюков, В. И. Легоньков. Управление данными в вычислительном эксперименте.— 1986.
 47. Е. Г. Ойхман, Ю. В. Зюзин, Ю. В. Новоженков. Графические системы для СМ ЭВМ.— 1986.

95 коп.

5B166

3-433

