

Е.П. БАЛАШОВ • В.Л. ГРИГОРЬЕВ •
МИКРО- Г.А. ПЕТРОВ
И МИНИ-ЭВМ

Допущено Министерством высшего и среднего
специального образования СССР
в качестве учебного пособия
для студентов вузов,
обучающихся по специальности
"Электронные вычислительные машины"

Ленинград
ЭНЕРГОАТОМИЗДАТ
Ленинградское отделение
1984

ББК 32.973

Б 20

УДК 681.31—181.48(075.8)

Рецензенты: *Б. М. Казан и В. В. Сташин (кафедра ЭВМ Московского института инженеров железнодорожного транспорта); В. Я. Петров (кафедра ЭВМ Московского высшего технического училища имени Н. Э. Баумана)*

Балашов Е. П. и др.

Б 20 Микро- и мини-ЭВМ/Е. П. Балашов, В. Л. Григорьев, Г. А. Петров: Учебное пособие для вузов. — Л.: Энергоатомиздат. Ленингр. отд-ние, 1984.—376 с., ил.

В пер.: 1 р. 10 к.

В книге систематизированно, с единых позиций изложены вопросы организации и проектирования аппаратных и программных средств микро- и мини-ЭВМ. Большое внимание уделено архитектуре малых ЭВМ, программному обеспечению и периферийному оборудованию. В качестве примеров использованы лучшие образцы серийных отечественных и зарубежных машин.

Для студентов вузов, обучающихся специальностям «Электронные вычислительные машины», «Прикладная математика», «Робототехнические системы», а также для аспирантов и научно-технических работников.

Б $\frac{24050000-141}{051(01)-84}$ 279—84

ББК 32.973

6Ф7

ЕВГЕНИЙ ПАВЛОВИЧ БАЛАШОВ
ВЯЧЕСЛАВ ЛЕОНИДОВИЧ ГРИГОРЬЕВ
ГЕННАДИЙ АЛЕКСЕЕВИЧ ПЕТРОВ

МИКРО- И МИНИ-ЭВМ

Редактор *Ю. В. Долгополова*
Художественный редактор *Д. Р. Стеванович*
Технический редактор *А. Г. Рябкина*
Корректор *А. Н. Акимов*

ИБ №

Сдано в набор 26.03.84. Подписано в печать 31.10.84. М-21216. Формат 84×108¹/₃₂. Бумага типографская № 2. Гарнитура литературная. Высокая печать. Усл. печ. л. 19,74. Усл. кр.-отт. 19,74. Уч.-изд. л. 23,26. Тираж 73 000 экз. Заказ 808. Цена 1 р. 10 к.

Ленинградское отделение Энергоатомиздата. 191041, Ленинград, Марсово поле, 1.

Владимирская типография Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 600000, г. Владимир, Октябрьский пр., 7.

© Энергоатомиздат, 1984

ПРЕДИСЛОВИЕ

В программе экономического и социального развития нашей страны на 1981—1985 гг. и на период до 1990 г. особое место уделено развитию средств вычислительной техники, которые широко используются практически во всех областях народного хозяйства.

ЦК КПСС и Совет Министров СССР приняли постановление «О развитии работ по автоматизации машин, оборудования и приборов с применением микропроцессорных средств и созданий на этой базе автоматизированных предприятий и технологических комплексов».

В соответствии с инструктивным письмом Минвуза СССР № 93-081-789/10 от 14 сентября 1982 г. «Об изучении в высших учебных заведениях вопросов применения микропроцессорных средств и микро-ЭВМ» рекомендовано включить в учебный процесс высших учебных заведений изучение вопросов проектирования микропроцессорных систем для специальностей, связанных с проектированием микропроцессорных наборов, конструированием и производством электронно-вычислительной аппаратуры на их основе (специальности 0608, 0611, 0629, 0648 и др.) и разработкой программного обеспечения микро-ЭВМ и автоматизированных систем различного назначения (специальности 0640, 0646, 0647, 0738 и др.), а также специальностей по разработке средств и систем автоматизации и механизации технологических процессов в различных отраслях народного хозяйства (специальности 0606, 0628, 0634, 0635, 0636, 0637, 0639, 0642, 0654 и др.).

Все вышеизложенное ставит вопрос о необходимости подготовки специалистов в области разработки микро- и мини-ЭВМ на основе микропроцессоров и комплектов типовых БИС, а также применения указанных средств в системах различного назначения. Однако в настоящее время одной из основных трудностей в решении данной задачи является отсутствие систематизированной учебно-методической литературы по указанному кругу вопросов.

Предлагаемое учебное пособие является одной из первых попыток систематизировать вопросы организации и проектирования микро- и мини-ЭВМ на основе мирового и отечественного опыта подобных разработок, а также опыта, приобретенного авторами при выполнении ряда НИР и при подготовке соответствующих учебных курсов.

Отличительной чертой настоящего пособия является единый подход к рассмотрению организации и проектиро-

вания аппаратных и программных средств для микро- и мини-ЭВМ, которые в настоящее время различаются, по существу, только стоимостью и комплектом программ, составляющих систему программного обеспечения.

При этом рассмотрение проводится как с позиций проектировщика микро- и мини-ЭВМ, так и с позиций пользователя. В качестве примеров конкретных машин, иллюстрирующих основные вопросы, анализируемые в пособии, используются лучшие образцы отечественных и зарубежных микро- и мини-ЭВМ, получившие большое распространение.

В тексте книги термином «процессоры СМ» определяются процессоры микро-ЭВМ «Электроника-60», мини-ЭВМ СМ-3 (4), имеющие единую архитектуру и программную совместимость.

Важное место в пособии занимают вопросы, посвященные программному обеспечению малых ЭВМ, и методы его разработки на кросс-средствах и резидентных средствах.

Рассматриваются вопросы организации мультипрограммной работы микро- и мини-ЭВМ, что связано с их широким использованием в системах реального времени. Обсуждаются вопросы организации распределенных систем, создание которых становится экономически оправданным с появлением микропроцессоров, и анализируются основные области применения микро- и мини-ЭВМ.

В основу настоящего пособия положены курсы лекций, прочитанных авторами с 1975 г. в Ленинградском электротехническом институте имени В. И. Ульянова (Ленина), Ленинградском институте авиационного приборостроения и Рязанском радиотехническом институте, а также на ряде факультетов повышения квалификации преподавателей и инженеров.

Изложение материала иллюстрируется практическими примерами, связанными с реальными НИР, выполненными под руководством и при непосредственном участии авторов. В конце каждой главы приводятся контрольные вопросы для проверки усвоения материала.

Учебное пособие предназначено для студентов высших учебных заведений по специальностям 0608, 0647 и 0654 и может быть рекомендовано студентам смежных специальностей 0646, 0648, 0627 и др., а также аспирантам и инженерно-техническим работникам.

Отзывы о книге, замечания и пожелания просьба присылать по адресу: 191041, Ленинград, Марсово поле, д. 1, Ленинградское отделение Энергоатомиздата.

Авторы

МНОГОФУНКЦИОНАЛЬНОСТЬ И СПЕЦИАЛИЗАЦИЯ В ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКЕ

1-1. ПРИНЦИП МНОГОФУНКЦИОНАЛЬНОГО ИСПОЛЬЗОВАНИЯ АППАРАТНЫХ И ПРОГРАММНЫХ СРЕДСТВ

Анализ эволюции систем обработки информации показывает, что по мере развития систем, усложнения и расширения реализуемых ими функций наиболее эффективными и жизнеспособными являются системы, в которых расширение функциональных возможностей элементов, находящихся на различных уровнях иерархии системы, опережает рост их сложности.

Диалектические противоположности «многофункциональность и специализация», «интеграция и дифференциация функций» являются источником развития систем.

Развитие систем идет по диалектической спирали. Расширение витков спирали во времени соответствует расширению спектра функций (числа задач), реализуемых системой. Каждая точка на витке спирали соответствует определенному соотношению между многофункциональными и специальными элементами (модулями) системы. Переход на новый виток соответствует скачку в «технологии» создания системы.

На каждом этапе своего развития системы представляют собой совокупность многофункциональных и специализированных элементов, соотношение между которыми изменяется от преобладания многофункциональных элементов в начале этапа до преобладания разнородных специализированных объектов в конце его. Каждому этапу развития соответствует определенный виток диалектической спирали.

Диалектические противоположности многофункциональность и специализация постоянно сосуществуют на всех уровнях развития систем. Для систем обработки информации многофункциональность и специализация сосуществуют на всех уровнях аппаратных и программных средств.

Анализ развития вычислительной техники убедительно иллюстрирует периодическое изменение соотношения между многофункциональными и специализированными средствами на различных этапах развития.

Определим многофункциональный элемент (МФЭ) как элемент, реализующий неединичный набор функций. МФЭ может реализовать определенную совокупность функций из конкретного, свойственного МФЭ набора.

Наряду с МФЭ следует различать многофункциональное использование элементов (устройств). Будем считать, что элемент (устройство) используется многофункционально, если участвует в реализации более одной функции устройства (системы), в состав которого он входит. Многофункционально могут использоваться как многофункциональные, так и специализированные элементы. Так, например, микропрограмма суммирования используется как «подпрограмма» при выполнении различных арифметических операций в микро-ЭВМ.

Следует различать понятия «универсальность» и «многофункциональность», хотя четкие грани между ними практически невозможно провести. Достаточно задуматься над таким термином, как «универсальная ЭВМ». По-видимому, универсальность может рассматриваться как предельный случай многофункциональности. Нередко определение системы как многофункциональной более полно и точно отражает характеристики и возможности системы.

Если первые поколения микропроцессорных комплектов на основе больших интегральных схем (МПК БИС) представляли собой набор модулей с «жесткой» логикой, ориентированных на применение в конкретных системах большого тиража, то последующие поколения МПК БИС благодаря использованию принципов микропрограммирования нашли разнообразные области применения вследствие возможности функциональной ориентации. Микропроцессор является примером многофункционального модуля с программной реализацией основных функций.

Агрегатный принцип построения современных вычислительных и управляющих комплексов на базе микро- и мини-ЭВМ, многоуровневая модульная организация программного обеспечения позволяют создать достаточно экономичные структуры на основе многофункциональных и специализированных аппаратных и программных средств, обеспечивающих решение заданного комплекса задач.

1-2. ЭВОЛЮЦИЯ СРЕДСТВ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Эволюция функций и эволюция технологий. В системах сосуществуют два параллельно идущих эволюционных процесса: эволюция функций и эволюция технологий. Анализ

развития систем показывает, что эволюция функций опережает эволюцию технологий. Всегда потребности общества в реализации новых функций опережают возможности их эффективной реализации.

Все функции систем можно разделить на основные и дополнительные. Основные функции определяют существование систем определенного класса. Дополнительные функции относятся к сервисным и улучшают эксплуатационные характеристики, повышают эффективность использования систем.

Основными функциями вычислительной системы (ВС) являются: обработка и хранение информации, обмен информацией и управление процессами в системе. Дополнительные функции ВС обеспечивают эффективную загрузку системы, диалог с пользователем и контролируют работоспособность ВС. Указанные функции реализуются в ВС совокупностью аппаратных и программных средств. Возможности эффективной реализации определенного набора функций существенно зависят от уровня развития соответствующей технологии.

Использование микропроцессорной техники позволяет существенно расширить спектр основных и дополнительных функций систем. В настоящее время реализуются большие интегральные схемы и сверх-БИС со степенью интеграции 10^4 — 10^5 элементов на кристалл. Существенное уменьшение размеров схемных элементов и улучшение их характеристик достигается за счет усовершенствования процессов диффузии, травления, легирования и применения прецизионных методов литографии, в частности электронно-лучевой и рентгенолучевой литографии. Разрабатываются БИС с функциональной плотностью 10^5 — 10^6 элементов на кристалл. Разработка таких схем должна стать одним из решающих факторов создания сверхвысокопроизводительных систем обработки информации четвертого поколения.

В процессе развития систем определенного класса сохраняется совокупность их базовых функций. Наиболее интенсивным изменениям подвергаются дополнительные (сервисные) функции. Причем эти изменения направлены на увеличение производительности систем и совершенствование взаимодействия пользователя с системой. Основные и дополнительные функции многоуровневой системы можно представить как дерево функций, реализуемых отдельными подсистемами и элементами, находящимися на различных уровнях системной иерархии. Реальное число уровней декомпозиции функций системы до уровня функций, реализу-

мых функциональными и конструктивными модулями системы, составляет не более 6—8. На последних уровнях декомпозиции реализуются элементарные функции (операторы) систем соответствующего класса.

Расширение функциональных возможностей системы приводит к развитию дерева функций. Можно утверждать, что существует взаимное вложение деревьев функций развивающихся систем определенного класса. В дереве функций развивающейся системы идет наиболее интенсивный рост ветвей дополнительных функций.

Введение новых базовых функций в систему порождает новые классы (подклассы) систем. Так, например, введение в состав базовых функций вычислительных машин новой базовой функции сопряжения с управляемой системой привело в 1960-х годах к созданию управляющих цифровых вычислительных машин (УЦВМ), которые в настоящее время представлены семействами (рядами) управляющих вычислительных комплексов (УВК), широко используемых для создания автоматизированных систем управления.

В рамках стран социалистического содружества разрабатывается семейство мини-машин.

Относительное и временное разрешение противоречий в системах. Противоречия, возникающие в системах в процессе их развития, разрешаются на определенных этапах развития систем конкретного класса и проявляются в дальнейшем в трансформированном виде на новом качественном уровне развития систем.

По аналогии с деревом функций систем можно ввести понятие «дерево противоречий системы». Противоречия систем по своей сути отражают противоречия между функциями, которые должны быть реализованы на различных уровнях или различных этапах проектирования системы, и соответствующей структурой, реализующей определенную совокупность функций. Противоречия между функциями и структурой являются, по существу, противоречиями между содержанием и формой.

Разрешение противоречий между функциями и структурой приводит к созданию более совершенных систем. Разрешение противоречий на верхних уровнях системной организации приводит к необходимости разрешения цепочки противоречий, возникающих при создании системы. Проектирование систем есть процесс последовательного разрешения противоречий на отдельных этапах создания систем и противоречий, возникающих на различных уровнях систем-

ной организации при создании модулей (элементов) соответствующего уровня.

Диалектические противоречия в системе представляют собой совокупность противоречий в отдельных подсистемах — противоречий между функциями, реализуемыми подсистемой, и разработкой подсистемы соответствующей структуры.

Устойчивость базовых функций, реализуемых системами определенного класса, определяет относительную устойчивость их функционально-структурной организации и обуславливает относительное постоянство дерева противоречий систем определенного класса.

На различных этапах развития ВС специалистам в области «вычислительной технологии» предстоит разрешать вечные противоречия между функциональными возможностями и сложностью технической реализации модулей ВС, между объемом хранимой информации и быстродействием запоминающих устройств, проблеме организации эффективного диалога пользователя с ВС при минимальных затратах аппаратных и программных средств.

Так как системы находятся в непрерывном развитии, то спектр задач и перечень противоречий, подлежащих разрешению, неисчерпаемы.

Преимущество функционально-структурной организации многоуровневых систем. В процессе развития старая система входит в новую в качестве подсистемы. Исчерпав возможности развития, система включается в подсистему, является составной частью новой сложной системы, и в дальнейшем ее развитие идет на уровне подсистемы. Закономерность отражения базовой совокупности функций систем-прототипов в функционально-структурной организации многоуровневых систем непосредственно вытекает из закономерности вложения функций в сложных развивающихся системах.

Вычислительные системы в своих различных конфигурациях повторяют, и неоднократно, структуру машины Дж. Неймана и включают в себя все функциональные (не конструктивные!) узлы механических арифмометров. Аналогично микропроцессор повторяет структуру как классических ЭВМ предшествующих поколений, так и современных мини-ЭВМ и содержит следующие типовые функциональные устройства: арифметико-логическое, регистры общего назначения — сверхоперативную память, устройство управления и синхронизации, устройства ввода и вывода информации (входные и выходные мультиплексоры).

На каждом иерархическом уровне систем обработки информации реализуется совокупность функций элементарной машины, выражаемая операторами: P — обработки информации, C — управления процессом, M — хранения информации, T — обмена информацией.

Функциональный модуль i -го уровня (ΦM_i) системы обработки информации должен реализовать следующую совокупность функциональных операторов: обработки информации P_i , хранения M_i , управления C_i , обмена T_i с предыдущими и последующими уровнями системы.

Реализация оператора P_i модулем ΦM_i определяется функциональным назначением модуля определенного уровня. Для организации обработки информации в ΦM_i необходимо хранение операндов и управляющих слов, что обуславливает необходимость реализации оператора M_i . Операторы T_{i-1} и T_{i+1} необходимы для обмена операндами и управляющими словами модуля ΦM_i с модулями $(i-1)$ -го и $(i+1)$ -го уровней системы. Необходимость реализации оператора C_i определяется тем, что модули $(i-1)$ -го уровня являются управляющими по отношению к ΦM_i , а модуль ΦM_i управляет функционированием модулей $(i+1)$ -го уровня.

Операторы обработки последующих уровней формируются из операторов предыдущих уровней путем их последовательной декомпозиции.

Сформулированная закономерность преемственности функционально-структурной организации развивающихся систем и требование функциональной полноты набора P -, C -, M -, T -операторов для многоуровневой системы могут служить основой для формализованного структурного синтеза функционально-ориентированных систем обработки информации (см. главу 8). До настоящего времени эта задача решалась интуитивно.

Сжатие этапов развития систем. Общеизвестны закономерность ускорения во времени темпов научно-технического процесса, сокращение сроков от формирования определенной научной идеи до ее практической реализации. По-видимому, постепенное сжатие диалектической спирали развития по временной оси является общей закономерностью эволюции систем.

Следует отметить, что времена разработки и освоения систем в производстве становятся соизмеримыми с жизненным циклом систем. Скорость работы ЭВМ за два последних десятилетия возросла на 6—7 порядков, объем оперативной памяти увеличился на 5—6 порядков. За 25

лет сменилось четыре поколения ЭВМ. Развитие микроэлектронных средств обработки информации идет таким образом, что каждые 2—3 года появляется новое поколение интегральных микросхем и микропроцессорных комплектов БИС.

Непрерывно расширяются области применения микропроцессорных функционально-ориентированных систем — вычислительных и управляющих систем различного назначения, реализуемых на основе МПК БИС.

Учитывая соизмеримость сроков разработки и внедрения систем со сроками их морального старения, можно выделить следующие альтернативы: сокращение сроков проектирования за счет использования типовых проектных решений и автоматизации процессов проектирования; разработка систем с «экстраполяцией» их развития.

Первый аспект достаточно хорошо осознан и все шире используется создателями систем различного назначения; второй, несмотря на большой потенциальный выигрыш его применения, внедряется значительно медленнее. В данном случае речь идет о системах, способных к реконфигурации, к развитию. Подобный подход используется, например, при создании рядов программно-совместимых моделей вычислительных машин: Единой системы ЭВМ, Системы малых машин типа СМ; при создании ряда агрегатных средств электроизмерительной техники и при проектировании многоуровневых автоматизированных систем управления производством (АСУП) и технологическими процессами (АСУТП).

1.3. ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ И УПРАВЛЯЮЩИЕ СИСТЕМЫ

Функциональная и структурная организация систем. Функциональная организация отражает проявление свойств системы во внешней среде. Структурная организация отражает состав и взаимосвязи элементов в системе.

Функционально-структурная организация отражает как особенности строения (структуры) и поведения системы во внешней среде, так и внутреннее взаимодействие элементов в процессе функционирования системы.

Функциональная организация системы — это модель системы, построенная на основе функциональных элементов, отражающая основные функциональные связи между ними.

Структурная организация системы — это модель системы на основе структурных (конструктивных) элементов, отражающая их взаимосвязи в процессе функционирования.

Для описания функциональной организации системы используются: алгоритмические языки, аналитическое, графическое и табличное описание, временные диаграммы и словесное описание.

В связи с интенсивным развитием методов автоматизированного проектирования систем разработаны и широко применяются новые методы принятия решений: ситуационного управления, нежесткой логики и другие методы, базирующиеся на концепциях искусственного интеллекта.

Для описания структурной организации системы используются обычно наглядное графическое описание и табличное представление связей между элементами системы. В автоматизированных системах для представления структуры системы используется аппарат теории графов.

В классической теории систем принято выделять следующие виды представлений иерархических систем: статифицированное, отражающее иерархию структурных решений; многослойное, отражающее иерархию целей системы; многоэшелонное, описывающее совместно функциональную и структурную иерархии.

Описание, отражающее функционально-структурную организацию системы, должно соответствовать концептуальной модели проектируемой системы. Таким образом, описание должно быть открытым и допускать возможность расширения (сужения) спектра функций, реализуемых системой. Описание должно предусматривать возможность перехода от одного уровня рассмотрения системы к другому, т. е. обеспечивать построение виртуальных систем любого уровня.

В настоящей работе наряду с традиционными для вычислительной техники средствами описания функционально-структурной организации систем (структурные схемы, граф-схемы алгоритмов, программы и микропрограммы) дополнительно используются (см. § 8-4): многоуровневое представление функции системы — дерево функций; обобщенные функционально-информационные карты (ОФИК); диаграммы временной активности элементов в процессе функционирования системы; операторные модели функционирования системы.

Основу описания функционально-структурной организации составляет дерево функций системы, адекватное по

своей структуре принципам иерархической организации систем.

Адекватность функционально-структурной организации систем. Анализ эволюции систем убедительно доказывает эффективность и жизнеспособность систем, структура которых максимально соответствует реализуемым функциям.

В процессе эволюции систем расширение и усложнение функций, реализуемых системами, и непрерывное совершенствование и создание новых технологий, пригодных для практического воплощения систем, приводят к непрерывному совершенствованию функционально-структурной организации систем, их прогрессивному развитию.

В области систем обработки информации известно правило: «В идеальном случае каждому реализуемому алгоритму соответствует определенная структура системы (устройства)». Однако понятно, что бесконечное разнообразие алгоритмов не может быть отображено на соответствующее разнообразие материальных структур.

Одни и те же функции могут быть реализованы многофункциональными и специализированными средствами. Таким образом, при формировании структуры системы определенного функционального назначения необходимо разрешать диалектическое противоречие «многофункциональность — специализация» на всех уровнях организации системы. Стратегия эволюционного синтеза систем предусматривает максимальное использование многофункциональных модулей системы. Использование их на всех уровнях системной организации позволяет создавать системы с минимальным числом типов элементов, реализующие заданную совокупность функций. Такие системы обладают наиболее высокими характеристиками информационной и конструктивной надежности и эффективности.

Изменение условий применения системы, новые требования, предъявляемые к системе, приводят к количественным и качественным изменениям в спектре функций, реализуемых системой. Это в свою очередь приводит к необходимости изменения принципов функционально-структурной организации системы с целью более полной адаптации к изменяющимся условиям эксплуатации. Изменение принципов функционально-структурной организации имеет место и при совершенствовании технологий создания системы.

В процессе развития систем формируются различные классы и виды систем, функционально-структурная организация которых наиболее адекватна определенным условиям применения.

При создании новых систем имеет место целенаправленный процесс воспроизведения заданной совокупности функций создаваемой системы. Определяющим, первичным при создании системы является ее функционально-проблемная ориентация. Каждая система должна эффективно и экономично реализовать заданную совокупность основных и дополнительных функций, т. е. должна удовлетворять определенным функциональным требованиям. При невыполнении этих условий система либо перестает существовать, либо требует создания дополнительных подсистем для удовлетворения заданного набора функций, либо теряет смысл.

Если в процессе проектирования не придается должного внимания определенной, казалось бы, второстепенной, функции, то система становится неэффективной в процессе эксплуатации. Первичность функционального назначения создаваемой системы и вторичность формирования ее структуры определяет необходимость в функционально-структурном подходе (ФСП) к синтезу систем. Таким образом, ФСП является естественным и единственно возможным при создании развивающихся систем. Использование ФСП предполагает проведение глубокого предварительного анализа основных и вспомогательных функций системы и связей между ними.

Функциональное назначение системы (совокупность реализуемых функций) и условия существования (функционалирования) определяют структурную организацию системы. Система может удовлетворять функциональному назначению, имея различную структурную организацию.

Заданная совокупность функций системы может быть реализована как многофункциональными, так и специализированными компонентами. Количественное соотношение между специализированными и многофункциональными компонентами зависит от возможностей технологии создания системы и, в свою очередь, влияет на качество создаваемой системы.

Как отмечалось выше, в рационально созданных системах преобладают многофункциональные компоненты. При равных функциональных возможностях наиболее эффективна и экономична система с минимальной структурой. Такая система при заданной совокупности реализуемых функций и определенной производительности содержит наименьшее число компонентов. Естественно, что сравнение ведется для компонентов одного уровня сложности систем с эквивалентными конструктивными, надежностными и экономическими показателями.

Специфической чертой проблемно-ориентированных ЭВМ является ограничение машинных ресурсов (аппаратных и программных средств) применительно к определенному кругу основных областей использования. В классе проблемно-ориентированных систем выделяют подклассы: мини-ЭВМ, микро-ЭВМ и электронные клавишные вычислительные машины (ЭКВМ).

Выделяются также специализированные ЭВМ, предназначенные, как правило, для реализации одного алгоритма (вычисления характеристик случайных процессов, реализации преобразования Фурье, управления системой наведения и т. д.). Узкая ориентация специализированных машин позволяет обеспечить высокие характеристики их производительности при сравнительно простой архитектуре и ограниченных возможностях элементной базы за счет наиболее полного соответствия структуры реализуемому алгоритму. Современные специализированные вычислительные машины реализуются, как правило, на основе микропроцессорных комплектов БИС и представляют собой функционально-ориентированные информационно-измерительные или информационно-управляющие системы.

Микропроцессорные системы, микро- и мини-ЭВМ представляют собой результат сложной и долгой работы специалистов в области электроники и вычислительной техники на пути создания эффективных средств обработки информации и управления. В микропроцессорных информационно-управляющих системах осуществляется целесообразное сочетание многофункциональных (БИС микропроцессора, БИС оперативной памяти) и специализированных модулей (интерфейсных БИС, БИС постоянной памяти и специальных схем).

Развитие вычислительных систем идет непрерывно. Пользователей не удовлетворяют возможности универсальных, проблемно-ориентированных и специализированных машин. С целью повышения производительности универсальных и проблемно-ориентированных средств вычислительной техники создаются специализированные функциональные расширители, предназначенные для работы в составе вычислительного или управляющего комплекса. Функциональные расширители разгружают основной процессор от выполнения сложных, часто встречающихся операций. Создаются функциональные расширители для воспроизведения различных функций (тригонометрических, показательных, специальных), аппаратной реализации функций программного обеспечения, сопряжения с пери-

ферийным оборудованием и управляемой системой.

Анализ развития систем с позиций функционально-структурного подхода приводит к следующим выводам:

1. Структура системы определяется совокупностью реализуемых функций.

2. Между реализуемыми функциями и структурой системы не существует взаимно-однозначного соответствия. Один и тот же набор функций может быть реализован различными совокупностями многофункциональных и специализированных элементов, объединенных в систему.

3. При изменении условий функционирования происходит изменение спектра реализуемых системой функций, приводящее к соответствующим изменениям в структуре системы. Таким образом осуществляется адаптация функционально-структурной организации системы к изменяющимся условиям существования.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Приведите примеры многофункционального использования устройств в ЭВМ.

2. Какие основные и дополнительные функции реализуются в вычислительных системах?

3. Какие средства описания функциональной организации вычислительных систем можно использовать?

4. В чем заключается функциональная и структурная организация систем?

ГЛАВА ВТОРАЯ

ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ И СИСТЕМА КОМАНД

2-1. ФОРМАТЫ ДАННЫХ

Основными типами данных, которые встречаются при обработке информации в мини- и микро-ЭВМ, являются числа, представленные в двоичной системе счисления, а также алфавитно-цифровые символы. Для кодирования символов используются специальные коды, наиболее распространенными из которых являются 7- и 8-битные двоичные коды. Некоторые малые ЭВМ также осуществляют обработку чисел, представленных в двоично-десятичной системе счисления. Для записи и индикации адресов и данных применяются удобные и компактные восьмеричная и шестнадцатеричная системы счисления.

Беззнаковое целое n -битное число X может принимать значения в диапазоне $0 \leq X \leq 2^n - 1$, где n для мини- и микро-ЭВМ равняется 8, 12, 16 битам. Для представления чисел с многократной точностью используется соответствующее число ячеек памяти, в каждой из которых хранится одно слово. Для выборки таких чисел из памяти необходимо задать адрес младшего слова, а остальные слова выбираются путем увеличения адреса текущей ячейки памяти на 1.

Дробное n -битное число X принимает значения в диапазоне $0 \leq X \leq 1 - 2^{-n}$.

Для представления отрицательных чисел наибольшее распространение получил немодифицированный дополнительный код вследствие таких его достоинств, как однозначное представление отрицательного и положительного нуля, отсутствие циклического переноса, использование одного разряда для представления знака числа. При этом признаком переполнения, например, при сложении двух чисел в дополнительном коде является наличие переноса только в знаковый разряд из старшего разряда кода или только из знакового разряда кода.

Представление числа X с плавающей запятой в ЭВМ для достижения требуемой точности при выполнении арифметических операций осуществляется следующим образом: $X = M \cdot 2^E$, где M — мантисса, представленная в нормализованной форме, $0,5 \leq |M| < 1$; E — порядок, представляющий собой целое со знаком.

В процессорах СМ с целью увеличения числа значащих цифр в мантиссе используется тот факт, что для нормализованной формы представления чисел в двоичной системе счисления старший бит мантиссы всегда равен 1, и это позволяет не представлять его в разрядной сетке мантиссы. Такой старший разряд мантиссы называют обычно скрытым разрядом, что обеспечивает разрядность мантиссы 24 бита (рис. 2-1). В некоторых малых ЭВМ используется короткий формат машин ЕС ЭВМ для расширения диапазона представления чисел с плавающей запятой: $X = M \cdot 16^E$

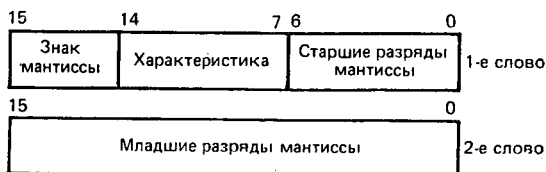


Рис. 2-1. Формат данных процессора СМ с плавающей запятой

где M — нормализованная мантисса, наименьшим значением которой является код 0001, а E — двоичное целое (характеристика).

Рассмотрим пример представления числа $1/32$ в процессорах СМ в форме с плавающей запятой. Двоичный код числа $1/32$ равен 0,00001; мантисса после нормализации путем сдвига кода влево на 4 разряда будет соответствовать коду 0,10000, а порядок равен —4. Характеристика с избытком 128 для данного порядка равна 0111100. Таким образом, число $1/32$ в форме с плавающей запятой при использовании скрытого бита имеет вид:

0	0111100	00 000 000 000 000 000 000
Знак мантиссы	Характеристика	Мантисса

Некоторые мини- и микро-ЭВМ могут выполнять арифметическую обработку чисел, представленных в десятичной системе счисления. Это позволяет избежать потерь времени на прямое и обратное преобразование чисел из одной системы счисления в другую и потерь в точности представления исходных чисел, которые могут иметь место при переходе от одной системы счисления к другой. Например, десятичное число 0,3 нельзя представить абсолютно точно в двоичной системе счисления. Поэтому для представления десятичных чисел используют специальные двоично-десятичные коды — двоично-кодированную форму. Например, в 8-битной ячейке памяти можно хранить двухразрядное десятичное число, каждая цифра которого кодируется четырьмя битами. Одним из наиболее распространенных кодов для кодирования цифр десятичного числа является код с весами 8, 4, 2, 1. Для хранения многоразрядных десятичных чисел используется последовательное запоминание в ячейках памяти цифр десятичного числа. Для сложения и вычитания чисел в десятичной системе счисления в микро-ЭВМ используется не одна команда, как это имеет место в больших ЭВМ, а две: команда двоичного сложения и команда десятичной коррекции. Подробнее процедура сложения десятичных чисел будет рассмотрена ниже.

Отрицательные десятичные числа представляются также с использованием десятичного дополнительного кода. Для кодирования знакового разряда S используются, например, такие комбинации:

$$S = \begin{cases} 0000 & \text{— для положительных чисел,} \\ 1001 & \text{— для отрицательных чисел.} \end{cases}$$

Рассмотрим пример сложения двух десятичных чисел (+52) и (—43) с использованием дополнительных кодов:

$$\begin{array}{r}
 + \begin{array}{l} 0000, 0101\ 0010 \\ 1001, 0101\ 0111 \end{array} \quad (+52) \\
 \hline
 + \begin{array}{l} 1001, 1010\ 1001 \\ 0110\ 0110 \end{array} \quad \begin{array}{l} (-43 \text{ в дополнительном коде}) \\ \text{Двоичное сложение} \\ \text{Десятичная коррекция} \end{array} \\
 \hline
 0000, 0000\ 1001 \quad (+9)
 \end{array}$$

Для представления алфавитно-цифровой информации, включающей в себя буквы алфавита, цифры, математические, пунктуационные и другие специальные символы, в ЭВМ получили распространение 7- и 8-битные символьные коды.

2.2. СТРУКТУРА И ТИПЫ КОМАНД

Под командой понимают совокупность сведений, необходимых процессору для выполнения определенного действия при реализации программы. Множество команд, реализуемых в ЭВМ, образует ее систему команд, выбор которой является сложнейшей и важнейшей задачей проектирования ЭВМ, так как система команд определяет область и эффективность ее применения. Несмотря на то, что подавляющее большинство алгоритмов может быть реализовано посредством достаточно ограниченного и простого набора команд, большинство мини- и микро-ЭВМ имеет 60—120 базовых команд. Под базовой понимают команду, которая определяет выполняемую операцию без учета модификаций данной команды за счет использования различных режимов адресации, регистров общего назначения, являющихся источниками и приемниками результатов, и т. п. Например, МП КР580 имеет 78 базовых команд, а процессоры СМ — около 60, однако с учетом модификаций число команд соответственно равняется 244 и примерно 400. Это позволяет в ряде случаев существенно сокращать длину программ, а следовательно, уменьшать время решения задачи и емкость памяти. Таким образом, система команд определяет возможности машины.

Теоретически ограничения на число команд ЭВМ нет; например, при введении команд из нескольких слов можно выделить больше бит под код операции. Каждый дополнительный бит в коде операции удваивает число команд. Однако, как показывает практика программирования, при реализации обширной системы команд программисты

обычно начинают использовать только ее некоторое подмножество. Кроме того, чем сложнее команда, тем более вероятно, что ее действия можно интерпретировать группой команд, а это уменьшает эффективность всей системы команд. Но, с другой стороны, чем сложнее команда, тем быстрее выполняется программа из-за сокращения числа обращений к памяти.

Проектирование системы команд оказывает влияние на структуру ЭВМ. Оптимальную систему команд иногда определяют как совокупность команд, которая удовлетворяет требованиям проблемно-ориентированных применений таким образом, что избыточность аппаратных и аппаратно-программных средств на реализацию редко используемых команд оказывается минимальной. В различных программах ЭВМ частота появления команд различна; например, по данным фирмы DEC в программах для ЭВМ семейства PDP-11 наиболее часто встречается команда передачи MOV(B), на ее долю приходится приблизительно 32 % всех команд в типичных программах. Систему команд следует выбирать таким образом, чтобы затраты на редко используемые команды были минимальными.

При наличии статистических данных можно разработать (выбрать) ЭВМ с эффективной системой команд. Одним из подходов к достижению данной цели является разработка команд длиной в одно слово и кодирование их таким образом, чтобы разряды таких коротких команд использовать оптимально, что позволит сократить время реализации программы и ее длину.

Другим подходом к оптимизации системы команд является использование микроинструкций. В этом случае отдельные биты или группы бит команды используются для кодирования нескольких элементарных операций, которые выполняются в одном командном цикле. Эти элементарные операции не требуют обращения к памяти, а последовательность их реализации определяется аппаратной логикой.

Сокращение времени выполнения программ и емкости памяти достигается за счет увеличения сложности логики управления. Мини-ЭВМ «Электроника-100» является примером использования такого подхода.

Важной характеристикой команды является ее формат, определяющий структурные элементы команды, каждый из которых интерпретируется определенным образом при ее выполнении. Среди таких элементов (полей) команды выделяют следующие: код операции, определяющий выпол-



Рис. 2-2. Классификация команд

няемое действие; адрес ячейки памяти, регистра процессора, внешнего устройства; режим адресации; операнд при использовании непосредственной адресации; код анализируемых признаков для команд условного перехода.

Классификация команд по основным признакам представлена на рис. 2-2. Важнейшим структурным элементом формата любой команды является код операции (КОП), определяющий действие команды. Большое число КОП в процессоре очень важно, так как аппаратная реализация команд экономит память и время. Но при выборе ЭВМ необходимо концентрировать внимание на полноте операций с конкретными типами данных, а не только на числе команд, на доступных режимах адресации. Число бит, отводимое под КОП, является функцией полного набора реализуемых команд. При использовании фиксированного числа бит под КОП для кодирования всех m команд необходимо в поле КОП выделить $\log_2 m$ двоичных разрядов. Однако, учитывая ограниченную длину слова мини- и микро-ЭВМ, различное функциональное назначение команд, источники и приемники результатов операций, а также то, что не все команды содержат адресную часть для обращения к памяти и периферийным устройствам, в малых ЭВМ для кодирования команд широко используется принцип кодирования с переменным числом бит под поле КОП для различ-

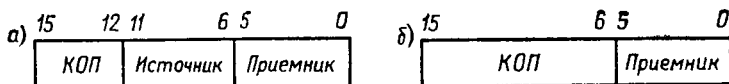


Рис. 2-3. Форматы команд процессоров СМ: а — двухадресная команда; б — одноадресная

Таблица 2-1

Примеры кодирования двухадресных команд в процессорах СМ

КОП	Мнемоника команды	Комментарий
0001	MOV	Передача данных Сравнение Сложение Вычитание
0010	CMP	
0110	ADD	
1110	SUB	
0000	—	Кодирование групп одноадресных команд
1000	—	

ных групп команд. На рис. 2-3, а приведен формат двухадресной (двухоперандной) команды процессоров СМ. Четырехбитный КОП (биты 15—12) кодирует ряд двухоперандных операций, приведенных в табл. 2-1. Биты (11—6) и (5—0) для команд данного типа определяют адреса источника и приемника данных. Как видно из табл. 2-1, комбинации 0000 и 1000

поля КОП определяют группы одноадресных команд (рис. 2-3, б), примеры которых иллюстрирует табл. 2-2. КОП 1 (биты 15—12), соответствующий кодам 0000 и 1000, определяет группу одноадресных команд, а КОП 2 (биты 11—6) кодирует конкретную операцию команд данной группы. Таким образом, команды, использующие один операнд, кодируются 10-битным КОП (биты 15—6).

По своей структуре команды в зависимости от того, используются они для обращения к памяти, периферийным устройствам или регистрам процессора, подразделяются на адресные и безадресные. Большинство адресных команд малых ЭВМ является одноадресными. При этом адрес указывается в команде явно или косвенно путем адресации

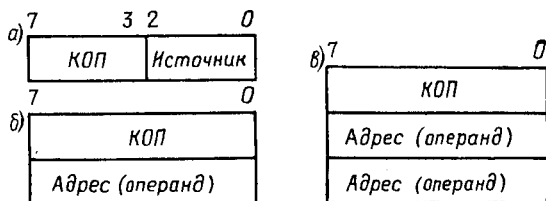


Рис. 2-4. Форматы команд МП КР580: а — однобайтная команда; б — двухбайтная; в — трехбайтная

регистра процессора, в котором хранится исполнительный адрес памяти. Для указания правила использования адресной части в командах большинства ЭВМ используются 2—3 бита, определяющие режим адресации. Однако некоторые ЭВМ, как, например, микро-ЭВМ СМ 1800, не имеют таких полей, и каждая команда использует только один режим адресации.

На рис. 2-4 приведены примеры команд МП КР580 длиной в один, два и три байта.

Таблица 2-2

Примеры кодирования одноадресных команд в процессорах СМ

КОП		Мнемоника команды	Комментарий
Постоянная часть	Переменная часть		
0000	000001	JMP	Безусловный переход
0000	101000	CLR	Очистка приемника
0000	101001	COM	Инвертирование
0000	101010	INC	Инкремент
0000	101110	DEC	Декремент
0000	101100	NEG	Отрицание
0000	0××××	—	Кодирование команды передачи управления
1000	0××××	—	

Примечание: знак × соответствует нулю или единице.

2-3. КОМАНДЫ ПЕРЕДАЧИ ДАННЫХ

Данная группа команд включает в себя подгруппы команд передачи кодов внутри процессора, из регистров процессора в память, из памяти в регистры процессора и команды передачи кодов между процессором и периферийными устройствами. В процессорах, допускающих различную длину данных, например байты, слова, двойные слова, для каждой длины имеются отдельные команды передачи данных.

Команды передачи кодов между регистрами процессоров. Типичный формат команд данной группы, являющихся двухадресными, приведен на рис. 2-3, а. Поле источника определяет адрес регистра, из которого передается данное, а поле приемника — адрес регистра, принимающего данное.

В процессорах СМ для обработки данных, представленных байтами, имеется команда перестановки байтов SWAB. Например, команда SWAB R1 меняет местами

старший байт с младшим в регистре R1. Некоторые процессоры имеют команды обмена (exchange), которые обменивают содержимое двух регистров, ячеек памяти или регистра и памяти. В МП КР580 команда обмена XCHG обменивает содержимое регистровых пар H, L и D, E.

Команды пересылки кодов между процессором и памятью. По командам этой группы выполняют запоминание данных из регистров процессора в памяти — store (ST) и загрузку регистров процессора данными из памяти — load (LD), однако имеются разновидности мнемоники команд. В процессорах СМ используется мнемоника MOV(B) для передачи в обоих направлениях, так как MOV(B) является двухадресной командой, в которой поля S и D в зависимости от режима адресации кодируют регистр или ячейку памяти, что также обеспечивает передачи типа память — память. Некоторые команды содержат вместо адреса памяти и регистра данное, передаваемое в процессор или память при использовании непосредственного режима адресации. Иногда адрес ячейки памяти указывается неявным образом путем использования косвенной регистравой адресации. В МП КР580 команда MOV r, M (рис. 2-5, а) передает данное из памяти в регистр r, указанный в поле D. Адрес ячейки памяти хранится в регистрах процессора H, L, которые предварительно необходимо загрузить требуемым адресом. Большинство современных мик-

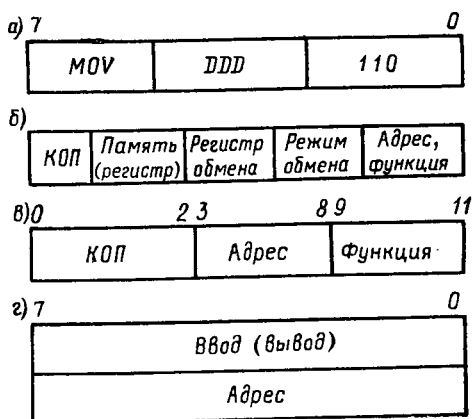


Рис. 2-5. Форматы команд передачи данных: а — между регистром и памятью в МП КР580; б — обобщенный формат команды ввода-вывода; в — команды ввода-вывода мини-ЭВМ «Электроника-100»; г — МП в микро-ЭВМ СМ-1800

ро- и мини-ЭВМ имеет команды включения данных в стек и извлечения из стека. Например, в МП КР580 команды PUSH гр и POP гр обеспечивают включение в стек и извлечение из стека данных в регистровые пары В и С, D и Е, H и L. В качестве указателя стека используется специальный регистр — указатель стека. Для включения и извлечения содержимого аккумулятора и флажков используются отдельные команды PUSH PSW и POP PSW.

Команды передачи кодов между процессором и периферийными устройствами. В различных моделях малых ЭВМ процедура обмена кодов реализуется посредством команд ввода и вывода, содержащих в общем случае поля, приведенные на рис. 2-5, б. Иногда отдельные команды не используются, а буферные регистры периферийных устройств рассматриваются как часть адресного пространства памяти, что применяется в процессорах СМ. Поля команд ввода-вывода определяют следующие действия (рис. 2-5, б):

1. КОП определяет вид операции обмена с периферийным устройством.

2. Поле памяти (регистра) определяет использование регистра в качестве источника (приемника) информации или в качестве адреса ячеек памяти.

3. Поле регистра обмена кодирует номер регистра, выполняющего одну из вышеназванных функций.

4. Поле режима обмена определяет, что необходимо выполнять: ввод, вывод, управление или проверку состояния периферийных устройств.

5. Поле «адрес периферийного устройства и функция» определяет, какое устройство должно быть выбрано и какие действия необходимо выполнить. Примерами функций, которые необходимо выполнять в режимах управления и проверки периферийных устройств, являются следующие: установить или сбросить «флажки» готовности, проанализировать значение флажка, инициировать операцию, выполнить перемотку, разрешить или запретить прерывание и др. В некоторых ЭВМ поля «адрес» и «функция» не выделяются отдельно, а код данных полей определяет устройства и выполняемую функцию. Данный подход более универсален по сравнению с разделением полей, но требует более сложных схем декодирования в интерфейсных модулях периферийных устройств. В мини- и микро-ЭВМ имеется множество разновидностей обобщенного формата команды обмена. Команды обмена процессора с периферийными устройствами для микро-ЭВМ СМ-1800 и мини-ЭВМ «Электроника-100» приведены на рис. 2-5, в, г.

В СМ-1800 адрес команд ввода и вывода (PORT) определяет адрес периферийного устройства и выполняемую функцию (см. пример интерфейса СМ-1800 с АЦП в § 4-2), обмен всегда выполняется только с аккумулятором. В ЭВМ «Электроника-100» разряды 0—2 идентифицируют КОП обмена 110, разряды 3—8 определяют адрес устройства, а разряды 9—11 указывают, какую функцию необходимо выполнять в адресуемом устройстве.

2-4. КОМАНДЫ ОБРАБОТКИ ДАННЫХ

Данную группу команд с точки зрения выполняемых над данными операций можно подразделить на арифметические, логические команды и команды сдвига, а по числу используемых операндов их можно классифицировать на одно- и двухоперандные операции. Операнды двухоперандных команд хранятся в регистрах процессора или в регистре и памяти, а при использовании непосредственной адресации один из операндов указывается в самой команде. Результат формируется в регистре-приемнике или в аккумуляторе. Команды данной группы формируют признаки результатов: перенос из старшего разряда, переполнение результата, нулевой результат и др.

Базовой арифметической командой в любой ЭВМ является команда двоичного сложения, имеющая следующий формат: ADD dst, src, где src — источник операнда, а dst — приемник результата. Большинство ЭВМ имеет команду вычитания SUB dst, src, которая обеспечивает вычитание операнда-источника из операнда-приемника с запоминанием результата в приемнике. В командах сложения (вычитания) с непосредственным операндом к содержимому регистра или ячейки памяти прибавляется (вычитается) константа, определенная в специальном поле команды. Например, в процессорах СМ 16-битный операнд является вторым словом команды, а в 8-битном МП КР580 — вторым байтом.

При выполнении арифметических операций сложения (вычитания) с многократной точностью перенос (заем) должен распространяться от младших разрядов к старшим. Эти операции упрощаются при использовании команд сложения с переносом ADC dst, src и вычитания с займом SBB dst, src. Например, команда ADC dst, src складывает два операнда и добавляет единицу, если C=1 до выполнения команды. В некоторых ЭВМ источником одного из опе-

рандов и приемником результата всегда является аккумулятор, который явно к команде не адресуется. Например, в МП КР580 команда сложения содержимого аккумулятора А и регистра r ADDr обеспечивает следующее: $(A) + (r) \rightarrow (A)$.

При обработке знаковых целых неодинаковой длины разрядность короткого формата необходимо увеличить до разрядности длинного. Длину числа, представленного в дополнительном коде, можно увеличить до любого числа бит путем копирования его знакового разряда слева. Например,

$$+5_{10} = 0101_2 = 0000\ 0101_2 = 0000\ 0000\ 0000\ 0101_2$$

или

$$-5_{10} = 1011_2 = 1111\ 1011_2 = 1111\ 1111\ 1111\ 1011_2.$$

Во многих процессорах имеется команда расширения знака SXT (sign extend).

Сравнение кодов выполняется с помощью команды вычитания или чаще специальной команды сравнения CMP dst, src. При этом содержимое источника и приемника не изменяется, а для анализа результата сравнения операндов (больше, меньше, равно) команда CMP формирует признаки результата операции. Если использовать маскирование, то можно сравнивать отдельные биты двух операндов.

Арифметические однооперандные операции увеличения на 1 (инкремент) и уменьшение на 1 (декремент) являются распространенными операциями для организации счетчиков при просмотре таблиц. Эти операции обеспечивают командой INC dst, под действием которой $(dst) := (dst) + 1$, и командой DEC dst, под действием которой $(dst) := (dst) - 1$. Операндами в этих командах может быть содержимое регистров или ячеек памяти в некоторых процессорах. Следует заметить, что обычно команда INC dst не эквивалентна команде сложения с непосредственным операндом ADD dst, #1 с точки зрения формирования флажков: ADD влияет на признак переноса C, а INC не изменяет его. Это обеспечивает правильное выполнение условных переходов:

REP:

DEC CNT ; (CNT) := (CNT) - 1;

BNE REP ; перейти к REP, если (CNT) $\neq$ 0.

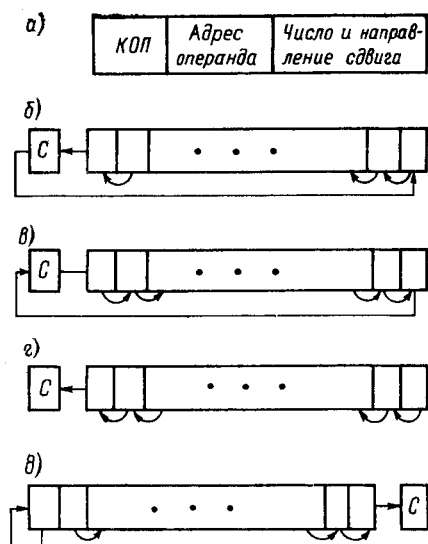


Рис. 2-6. Форматы команд сдвига: а — обобщенный формат команды сдвига; б — логический сдвиг влево; в — логический сдвиг вправо; г — арифметический сдвиг влево; д — арифметический сдвиг вправо

Эта проблема возникает при выполнении арифметических операций сложения (вычитания) с повышенной точностью, умножения, деления. В таких циклах необходимо сохранить значение признака С, поэтому команды INC и DEC не изменяют С. В процессорах СМ введена команда «вычитание единицы

и ветвление» SOB, которая объединяет команды DEC и BNE (условный переход по признаку «не равно нулю») и не изменяет значения признаков.

Во многих малых ЭВМ нет команд аппаратного умножения и деления, при необходимости эти операции реализуются посредством обращения к соответствующим подпрограммам или используются специальные расширители.

В подгруппу логических двухоперандных команд входят команды, реализующие операции логического умножения AND dst, src, логического сложения OR dst, src, исключающего ИЛИ (неравнозначности) XOR dst, src. В процессорах СМ нет команды AND, а имеется команда проверки разрядов BIT sec, dst (bit test), которая выполняет логическую операцию $(src) \wedge (dst)$, формируя признаки N, Z в соответствии с данной операцией. При этом (src) и (dst) не изменяются. Такая команда может быть использована для проверки любого из разрядов dst. Например, команда BIT#30, R3 обеспечивает проверку 3-го и 4-го разрядов R3.

Команда BIC src, dst (bit clear) сбрасывает разряды dst, соответствующие 1 в src, т.е. она эквивалентна логической функции запрета $(src) \wedge (dst)$. Логические операции позволяют эффективно обрабатывать отдельные биты, когда в слове данных упаковано много булевых переменных (флажков).

Группа команд обработки данных содержит также команды логического и арифметического сдвига кодов на один или несколько разрядов, хранящихся в регистрах или аккумуляторе процессора, а для некоторых ЭВМ и в ячейках памяти. Логический сдвиг кода может выполняться вправо или влево, быть простым (с потерей сдвигаемых разрядов) или циклическим через перенос. Обобщенный формат команды сдвига представлен на рис. 2-6, а. КОП определяют тип сдвига: логический или арифметический, простой или циклический, направление сдвига. Код числа сдвигов может храниться в поле команды или в регистре, в соответствии с этим команды многократного сдвига иногда называют статическими или динамическими.

Однако обычно в 8- и 16-битных ЭВМ из-за ограничений разрядности слова в команде поле, определяющее число сдвигов, отсутствует и сдвиг выполняется на один разряд вправо или влево в соответствии с КОП. Для выполнения сдвига на несколько разрядов команда в программе повторяется требуемое число раз или организуется цикл. Поле источника операнда определяет, в каком регистре или аккумуляторе необходимо сдвинуть код. Пример реализации команд левого ROL R и правого ROR R циклического сдвигов содержимого регистра R приведен на рис. 2-6, б, в. Если сдвигается только содержимое аккумулятора, как это имеет место в МП К580, то в команде поле R отсутствует.

Кроме команд логического сдвига некоторые ЭВМ имеют команды арифметического сдвига. При арифметическом сдвиге кода вправо знаковый разряд не сдвигается. На рис. 2-6, г, д представлены примеры выполнения команд арифметического левого и правого сдвигов в процессорах СМ. При выполнении левого арифметического сдвига знаковый разряд сдвигается в триггер переноса, а остальные разряды сдвигаются на один разряд влево, что при повторном сдвиге приводит к потере знакового разряда.

2-5. КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ И ДОПОЛНИТЕЛЬНЫЕ КОМАНДЫ

Команды передачи управления имеют важное значение, так как используются для изменения естественного порядка следования команд и организации циклических участков в программах.

Простейшей командой управления программой является команда безусловного перехода (JMP), которая загружа-

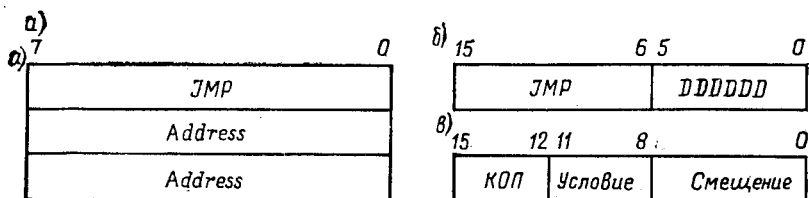


Рис. 2-7. Форматы команд передачи управления: а — команда безусловного перехода МП КР580; б — команда безусловного перехода процессоров СМ; в — команда разветвления процессоров СМ

ет адрес перехода, указанный в команде, в программный счетчик. На рис. 2-7, а, б приведены форматы команд безусловного перехода для МП КР580 и процессоров СМ. В МП КР580 адрес перехода передается из второго и третьего байтов команды JMP, а в процессорах СМ в поле D указывается адрес перехода с возможностью использования любого режима адресации кроме регистровой.

Команды условного перехода проверяют указанное в команде условие и загружают в программный счетчик адрес перехода, если условие истинно. Переходы, которые формируют целевой адрес с использованием относительной адресации к программному счетчику, называются командами разветвлений (branch). Команды разветвлений процессоров СМ содержат 8-битное смещение для образования целевого адреса, что уменьшает длину команды разветвления и обеспечивает позиционную независимость программы (рис. 2-7, в). Смещение представляет собой число в дополнительном коде и при выполнении разветвлений добавляется в программный счетчик. Если поле смещения m -битное, то переход возможен в диапазоне адресов: от $2^{m-1}-1$ до -2^{m-1} .

Другой разновидностью реализации передачи управления в программе является использование команды пропуска следующей команды по условию. Следующая команда пропускается, если проверяемое условие истинно, за счет добавления в программный счетчик величины, соответствующей длине следующей команды.

Для вызова подпрограмм используется команда CALL dst, которая запоминает адрес возврата в стеке до перехода к подпрограмме. После завершения подпрограммы управление возвращается к вызывающей программе командой возврата RET, которая восстанавливает программный счетчик из вершины стека. В ЭВМ без стека команда CALL

dst загружает адрес возврата в регистр или ячейку памяти, а возврат из подпрограммы выполняется посредством команды JMP с косвенной адресацией.

Помимо безусловных команд вызова подпрограмм и возврата из подпрограммы, в микро-ЭВМ СМ-1800 есть команды обращения CALL CCC Addr и возврата RET CCC по различным признакам (наличие или отсутствие переноса, положительный или отрицательный результат и др.), где CCC — трехразрядное поле кода признаков (подробнее о подпрограммах см. в § 7-5). Кроме того, в СМ-1800 безусловную передачу управления можно обеспечить командой PCNL, которая загружает программный счетчик адресом из регистровой пары H, L.

В ЭВМ помимо основных рассмотренных команд имеются дополнительные (специальные) команды. Почти во всех ЭВМ предусмотрена команда NOOP (нет операции), которая является удобной при отладке программ. Например, команду, которую необходимо исключить из программы, можно заменить на NOOP, не изменяя остальной части программного модуля. Иногда команда NOOP используется также при построении точных программных задержек. В некоторых ЭВМ кроме обычной аппаратной начальной установки системы (начального сброса) после включения питания имеется команда сброса RESET, реализующая данную функцию программным путем.

Для останова программы используется команда останова HALT, при этом программный счетчик содержит адрес следующей выполняемой команды и состояние машины не изменяется. Для выхода из состояния останова необходимо, например, с пульта управления нажать клавишу продолжения программы. В процессорах СМ имеется команда ожидания WAIT, отключающая процессор от общей шины до появления сигнала внешнего прерывания. Это позволяет производить обмен информацией с высокой скоростью. При прерывании в стеке запоминается адрес команды, следующей за WAIT, и после выхода из подпрограммы обработки прерывания программный счетчик автоматически восстанавливается, и продолжается выполнение прерванной программы. Во многих ЭВМ обычно имеется команда установки и сброса отдельных признаков.

2-6. РЕЖИМЫ АДРЕСАЦИИ ИНФОРМАЦИИ

При выполнении программ многие команды требуют обращения к памяти для выборки данных, записи промежуточных и окончательных результатов вычислений. Поэтому

механизм адресации в значительной степени определяет способность ЭВМ эффективно осуществлять обработку информации. Для малых ЭВМ это является особенно важным из-за ограниченной длины их команд, что часто является причиной использования команд в два или три слова. Поэтому в ЭВМ для преодоления данного ограничения имеется множество режимов адресации, которые позволяют: а) определять полный адрес памяти меньшим числом бит, тем самым сокращая длину команды; б) обращаться к ячейкам памяти, адреса которых вычисляются во время обработки, что обеспечивает удобный доступ к данным различной структуры; в) вычислять адреса данных относительно позиции команды таким образом, что программу можно загружать в любую область памяти без всяких изменений адресов в программе.

Все режимы адресации можно разделить на две группы. К первой относятся режимы, в которых исполнительный адрес определяется одним значением кода в команде. Режимы адресации второй группы используют содержимое адресной части команды и одного или нескольких регистров для формирования исполнительного адреса. К первой группе основных режимов адресации, используемых в малых ЭВМ, можно отнести прямую, регистровую, регистровую косвенную, непосредственную, автоинкрементную, автодекрементную, а ко второй — страничную, индексную, относительную.

Прямая адресация. В данном режиме адресации код адреса в команде является исполнительным адресом обращения к памяти. Однако указание полного прямого адреса требует много бит в команде ЭВМ с большим адресным пространством, и для уменьшения его некоторые ЭВМ используют короткую прямую адресацию, обеспечивающую доступ к ограниченной части адресного пространства. Из-за своей простоты и гибкости режим прямой адресации имеется во многих малых ЭВМ.

Регистровая адресация. В данном режиме адресации операнд содержится в одном из регистров процессора.

Непосредственная адресация позволяет в команде задавать операнд, который передается в процессор из памяти вслед за кодом операции для двух- или трехсловных команд.

Регистровая косвенная адресация получила широкое распространение в малых ЭВМ и по быстродействию приближается к прямой адресации, так как косвенный адрес извлекается из внутреннего регистра процессора и не тре-

буется дополнительного цикла памяти. В регистровой косвенной адресации регистр или регистровая пара содержит исполнительный адрес операнда. Загрузка регистров осуществляется с использованием команд с непосредственной адресацией. Использование режима регистровой косвенной адресации позволяет вычислять адреса памяти во время выполнения программы, что требуется в процедурах передачи данных, при просмотре элементов массива и т. п.

Автоинкрементная и автодекрементная адресация. Автоинкрементная адресация вычисляет исполнительный адрес практически так же, как и при регистровой косвенной адресации, а затем осуществляется увеличение содержимого регистра. В ЭВМ с байтовой адресацией содержимое регистра необходимо увеличить на 1 для указания следующего байта, на 2 — для указания слова, при этом размер операнда определяется кодом операции.

В автодекрементном режиме адрес операнда формируется вычитанием 1 или 2 из регистра адреса. Отличие от автоинкрементной адресации состоит в том, что вычитание производится до использования содержимого регистра как исполнительного адреса. Сочетание автоинкрементного и автодекрементного режимов адресации обеспечивает эффективное использование любого регистра в качестве указателя стека. В процессорах СМ использование автоинкрементного косвенного режима по отношению к программному счетчику, являющемуся одним из регистров общего назначения, позволяет получить абсолютный режим адресации.

Страничная адресация. При использовании страничного режима адресации память разбивается на ряд страниц одинаковой длины. Адресация страниц осуществляется или с программного счетчика, или с отдельного регистра страниц, а адресация ячеек памяти внутри страницы — адресом, содержащимся в команде. При этом обычно адрес (номер) страницы формируется одним из следующих путей: а) номер страницы располагается в базовой (нулевой) странице; б) номер страницы данных формируется путем пристыковки (конкатенации) старших разрядов программного счетчика, что позволяет получать адрес той же страницы, в которой находится команда; в) номер страницы берется из регистра страниц, в который программа ранее загрузила номер нужной страницы.

Например, в мини-ЭВМ «Электроника-100» память разделена на 32 страницы по 128 ячеек каждая. Исполнительный адрес формируется путем пристыковки пяти старших

бит (0—4) программного счетчика и семи младших бит (5—11) регистра команд, что позволяет обращаться к любой ячейке памяти в текущей странице.

Биты 3, 4 определяют признак адресации (ПА): 00 — прямая адресация к нулевой странице; 01 — прямая адресация к текущей странице; 10 — косвенная адресация к нулевой странице; 11 — косвенная адресация к текущей странице. При использовании только прямой адресации для формирования исполнительного адреса время выборки операнда не увеличивается и требуется один цикл обращения к памяти.

При использовании одноуровневой косвенной адресации для выборки операнда из памяти «Электроники-100» требуется два цикла памяти. Развитием одноуровневой косвенной адресации в некоторых ЭВМ является косвенная многоуровневая адресация, которая оказывается удобной для пользователя при организации распределения памяти, дает гибкость при программировании. Для указания числа уровней адресации при использовании многоуровневой адресации обычно старший бит выбранного кода определяет, требуется ли обращение за следующим адресом (1) или за операндом (0). Каждый уровень косвенной адресации требует дополнительного обращения к памяти и увеличивает время выполнения команды.

Индексная адресация удобна для обращения к массивам и таблицам. Для образования исполнительного адреса к адресной части команды прибавляется смещение из индексного регистра, называемое индексом. Индексный регистр является программно-доступным, и его содержимое может изменяться, что позволяет изменять исполнительные адреса без модификации адресной части команды. В качестве индексных регистров обычно используется один или несколько специальных регистров, но в некоторых ЭВМ их функцию выполняют регистры общего назначения или ячейки памяти, что приводит к увеличению времени выполнения программ. Когда индексный режим используется для доступа к массиву, адрес в команде соответствует базовому адресу массива, а значение индексного регистра — индексу компонента массива.

Относительная адресация. При использовании относительной адресации исполнительный адрес формируется путем сложения базового адреса с адресным полем команды. В качестве базового адреса используется содержимое программного счетчика. Использование относительной адресации позволяет строить программы, свободно перемещаемые

в памяти, за счет того, что в команде всегда указано смещение по отношению к содержимому программного счетчика. Смещение интерпретируется как знаковое целое, представленное в дополнительном коде, что обеспечивает переход в любом направлении. Например, для 8-битного кода смещения может выполняться максимальный переход на число адресов в диапазоне от $+127_{10}$ до -128_{10} по отношению к программному счетчику, при этом содержимое программного счетчика увеличивается для формирования адреса следующей команды до прибавления смещения. При перемещении программ из одной области памяти в другую смещение не изменяется, так как относительные позиции команд и данных сохраняются. Следует отметить, что так как содержимое программного счетчика постоянно увеличивается, формируемые исполнительные адреса изменяются для одного и того же смещения, т. е. это динамический способ адресации. Для выполнения операции сложения значений смещения и программного счетчика в малых ЭВМ используется, как правило, арифметико-логическое устройство (АЛУ), а иногда и специальный сумматор.

Ранее рассмотренный режим страничной адресации с конкатенацией (присоединением) адресной части команды и старших разрядов программного счетчика имеет по сравнению с данным режимом существенный недостаток для пользователя, хотя он и проще из-за отсутствия операции сложения. Для примера рассмотрим ситуацию, когда программа выполняется на странице, адресное пространство которой находится в диапазоне адресов 4000_8-4177_8 . Если программный счетчик содержит адрес 4170_8 , то все адреса, превышающие его больше чем на 7, находятся на следующей странице и не могут быть достигнуты с использованием страничной адресации. Только косвенное обращение, например, через нулевую страницу может обеспечить такой переход.

В заключение необходимо отметить, что единства в вопросах о важности различных режимов адресации нет. Различия в их выборе, по-видимому, объясняется тем, что разработчики ЭВМ ориентируют их на определенные области применения.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как формируется характеристика при представлении данных с плавающей запятой?
2. Какие типы команд образуют систему команд ЭВМ?

3. С какой целью формируются признаки результатов операций? Перечислите основные признаки.

4. Объясните назначение полей команд передачи данных.

5. Каким образом формируется исполнительный адрес при выполнении команд передачи управления?

6. Какие основные режимы адресации используются в малых ЭВМ и каким образом формируются исполнительные адреса для каждого из режимов?

7. Какими достоинствами и недостатками обладают режимы адресации, используемые в ЭВМ?

ГЛАВА ТРЕТЬЯ

АРХИТЕКТУРА МИКРО- И МИНИ-ЭВМ

3-1. ОСОБЕННОСТИ СТРУКТУРНОЙ ОРГАНИЗАЦИИ СИСТЕМ ОБРАБОТКИ ИНФОРМАЦИИ НА ОСНОВЕ БИС

Изменения элементно-технологической базы привели к изменению сложившихся принципов проектирования вычислительной техники. На смену им приходят идеи проблемной ориентации систем, параллельной и конвейерной обработки информации, использования табличных методов обработки данных и принятия решений, развиваются принципы регулярности и однородности структур на различных уровнях организации, становится реальной возможностью идея создания адаптивно перестраиваемых вычислительных систем, конфигурация которых изменяется в процессе решения задачи с целью наиболее эффективной организации вычислительного процесса и обеспечения живучести системы.

Характерной чертой развития средств ВТ является аппаратная реализация функций математического обеспечения, что позволяет существенно повысить производительность вычислительных систем.

Дальнейшее развитие получает принцип 3М — модульность, магистральность, микропрограммируемость.

Модульная организация систем. Принцип модульной организации предполагает построение вычислительных и управляющих машин и систем на основе набора модулей. Под модулем в данном случае понимается конструктивно, функционально и электрически законченное вычислительное устройство, позволяющее самостоятельно или в совокупности с другими модулями решать вычислительные или управляющие задачи заданного класса. Следует различать функциональные и конструктивные модули.

Модульный подход при проектировании микро- и мини-ЭВМ позволяет создать семейство ЭВМ, отличающихся друг от друга функциональными возможностями и характеристиками, способствует стандартизации элементов все более высоких уровней и сокращению затрат на проектирование систем, а также упрощает наращивание мощности и реконфигурацию систем, отодвигает время морального старения технических средств.

При решении вопроса о функциональном составе модулей в соответствии с рассмотренными выше закономерностями развития систем сосуществуют две диалектические противоположности: многофункциональность и специализация модулей. Повышение универсальности модулей рассматривается как средство, обеспечивающее сокращение номенклатуры, снижение затрат на проектирование и изготовление, высокую серийность, а следовательно, и низкую стоимость; специализация модулей — как средство достижения высокой эффективности систем за счет оптимального согласования структуры вычислительных средств и реализуемых ими алгоритмов и функций, а также уменьшения избыточности структуры модулей.

Оба этих подхода нашли сегодня практическое применение. Особо следует отметить прогресс в области микропроцессорных средств, реализующих первый из упомянутых путей. Однако низкая эффективность использования МП в ряде применений, являющаяся следствием несоответствия их структуры характеру выполняемых задач, недоиспользование значительной доли аппаратной и программной частей комплектов, сложная программная реализация сравнительно простых функций привели к появлению огромного числа типов МПК, увеличению их состава, включению специализированных БИС, реализующих главным образом функции обмена и связи с объектами управления.

Специализация модулей низшего конструктивного уровня ведет к несоблюдению основного требования при проектировании БИС — минимизации числа их типов и, за исключением создания систем специального назначения с экстремальными параметрами, не может быть выдвинута в качестве основной концепции проектирования модулей. Практика подтвердила целесообразность решения вопроса о функциональном составе модулей соединением рассмотренных крайних направлений путем создания систем в виде совокупности многофункциональных и специализированных модулей, проблемно и функционально ориентированных в рамках определенных классов задач, алгоритмов, функций.

Магистральный способ обмена информацией. Можно выделить два принципа взаимосвязи элементов модулей и самих модулей в системе: принцип произвольных связей, реализующий правило «каждый с каждым», и принцип упорядоченных связей — магистральный, позволяющий минимизировать число связей. Магистральный принцип обеспечивает обмен информацией между функциональными и конструктивными модулями различного уровня с помощью магистралей, объединяющих входные и выходные шины. Обычно выделяются следующие магистрали: шины данных, шины адресные, шины управления. Адресные шины и шины данных обычно однобайтные или двухбайтные.

Магистральный обмен информацией является одним из аспектов обеспечения регулярности структуры МПС как на уровне БИС, так и на уровне связей между конструктивными модулями МПС.

Применение магистрального обмена позволяет минимизировать число связей между блоками, обеспечить регулярность операционного устройства управления путем сосредоточения большинства комбинационных схем в АЛУ и коммутаторах, обеспечить стандартизацию интерфейсов, сократить число выводов БИС.

Необходимо отметить взаимосвязь схемотехнических и структурных решений, которая проявляется при реализации данного принципа обмена в виде специальных двунаправленных буферных каскадов с тремя устойчивыми состояниями и при использовании временного мультиплексирования каналов обмена.

Микропрограммная организация управления. Для современных систем обработки информации характерна многоуровневая организация программного управления. Принцип микропрограммного управления обеспечивает наибольшую гибкость при организации многофункциональных микропроцессорных модулей и путем определенной комбинации микрокоманд позволяет осуществить проблемную ориентацию микро- и мини-ЭВМ. Принцип микропрограммного управления позволяет использовать макрооперации в МПС, что существенно эффективнее использования стандартных подпрограмм и соответствует адаптации МПС к реализуемому алгоритму.

Микропрограммное управление повышает гибкость устройств за счет смены микропрограмм, увеличивает регулярность устройств за счет широкого использования матричных структур типа памяти, обеспечивает параллелизм решения задач при рассредоточенном управлении и распре-

деленности памяти, повышает надежность устройств за счет применения серийно освоенных БИС памяти, упрощает контроль функционирования устройства (контроль блока микропрограммного управления сводится, по существу, к контролю содержимого ЗУ).

Передача управляющих слов в виде зашифрованных кодовых последовательностей соответствует условиям минимизации числа выводов БИС и снижению числа соединений в модулях.

Регулярность структуры. Принцип регулярности предполагает закономерную повторяемость элементов структуры и связей между ними. Регулярность системы следует рассматривать на различных уровнях ее организации. Применение данного принципа позволяет повысить эффективность интегрального исполнения устройств вследствие увеличения интегральной плотности, уменьшения длин связей на кристалле, упрощения и сокращения времени топологического и схемотехнического проектирования БИС, упрощения процедур контроля топологии, уменьшения числа пересечений, сокращения числа типов функциональных и конструктивных элементов, повышения серийности и снижения стоимости средств.

Основными путями увеличения регулярности структуры вычислительных и управляющих систем (ВУС) являются широкое использование структур и устройств типа памяти (ОЗУ, ПЗУ, ПЛМ), применение регистровых структур, отказ от закрепления определенных микроопераций за регистрами, выполнение регистров общего назначения в виде ячеек ОЗУ, использование магистрального способа обмена, стандартизация интерфейсов, использование принципа микропрограммного управления, вертикальное (поразрядное) разбиение структуры ЭВМ на БИС, развитие многопроцессорных параллельных систем.

3-2. АРХИТЕКТУРА ЭВМ

Под архитектурой ЭВМ понимается абстрактное представление машины в терминах основных функциональных модулей, языка ЭВМ, структуры данных. Архитектура не определяет особенностей реализации аппаратной части ЭВМ, времени выполнения команд, степени параллелизма при выполнении программы, ширины шин и других аналогичных характеристик. Архитектура отображает аспекты структуры ЭВМ, которые являются видимыми для пользователя: систему команд, режимы адресации, форматы и

длину данных, набор регистров ЭВМ, доступных пользователю. Одним словом, термин «архитектура» используется для описания возможностей, предоставляемых ЭВМ, а термин «организация» определяет, как эти возможности реализованы.

Все ЭВМ содержат следующие функциональные блоки, имеющие свою микроархитектуру: процессор, состоящий из арифметико-логического устройства и устройства управления, память, устройства ввода и устройства вывода информации (рис. 3-1). Объединение функциональных блоков в ЭВМ осуществляется посредством следующей системы шин: шины данных, по которой осуществляется обмен информацией между блоками ЭВМ, шины адреса, используемой для передачи адресов, по которым осуществляется обращение к различным устройствам ЭВМ, и шины управления для передачи управляющих сигналов.

Для связи пользователя с ЭВМ предусмотрен пульт управления, который позволяет выполнять такие действия, как пуск ЭВМ; останов, под действием которого прекращается поступление сигналов с генератора тактирующих сиг-

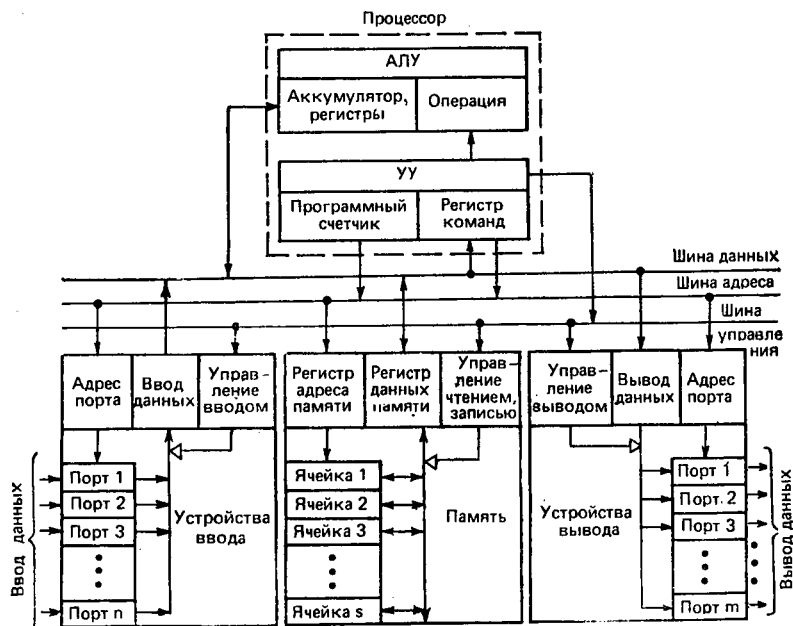


Рис. 3-1. Структурная схема ЭВМ

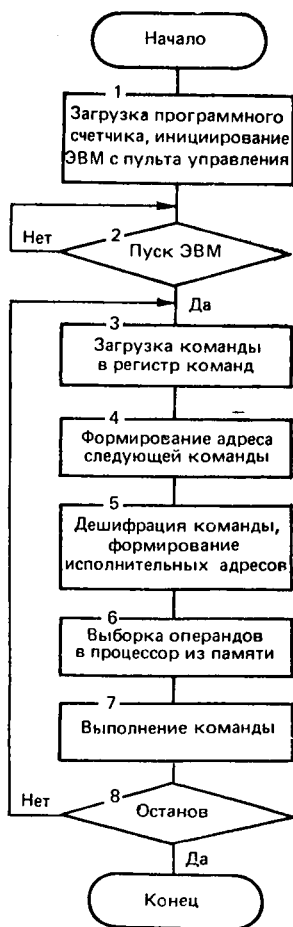
Рис. 3-2. Обобщенный алгоритм функционирования ЭВМ

налов и процессор переходит в состояние ожидания; загрузка начального адреса программы в программный счетчик; индикация содержимого ячеек памяти и регистров процессора; пошаговое выполнение команд программы при ее отладке.

Обобщенный алгоритм функционирования ЭВМ представлен на рис. 3-2. Одно из основных различий между ЭВМ — это организация системных шин, обеспечивающих связь между отдельными блоками ЭВМ; по этому признаку все структуры малых ЭВМ могут быть классифицированы следующим образом: ЭВМ с многошинной структурой; ЭВМ с общей шиной.

Типичная архитектура ЭВМ с многошинной структурой представлена на рис. 3-3. Основная особенность ее организации состоит в том, что для каждого способа обмена информацией с периферийными устройствами используется отдельная группа шин: отдельные шины для программного режима обмена информацией с прерыванием или без прерывания выполняемой программы и для ввода-вывода информации в режиме прямого доступа периферийных устройств к памяти, которые передают блоки данных с большой скоростью (накопители на магнитных дисках, лентах, быстродействующие аналого-цифровые преобразователи и другие устройства). Протоколы обмена данными, структура шин и быстродействие при обмене для каждой из групп шин могут быть оптимальным образом адаптированы к обслуживаемым периферийным устройствам в соответствии с выбранным методом.

Другой популярной структурой шин, которая использует



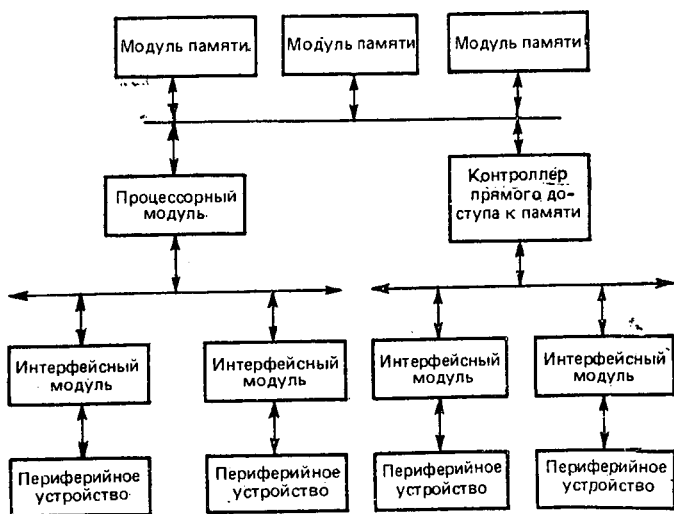


Рис. 3-3. ЭВМ с многошинной структурой

ется во многих ЭВМ, является структура с общей шиной (рис. 3-4). В этом случае все блоки ЭВМ объединяются посредством одной группы шин, в которую входят подмножества шин для передачи данных, адресов и управляющих сигналов. При такой организации системы шин обмен информацией между процессором, периферийными устройствами и памятью выполняется по единому правилу, отдельные команды ввода-вывода для обращения к периферийным устройствам в системе команд ЭВМ отсутствуют. Это позволяет повысить гибкость и эффективность ЭВМ, так как

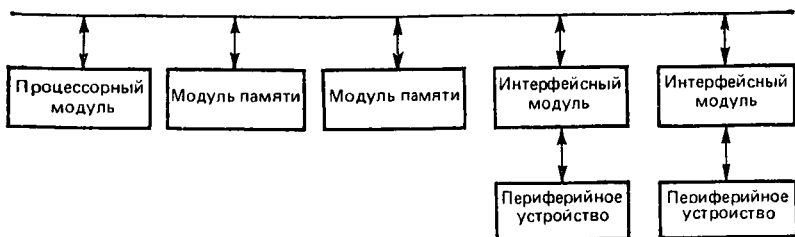


Рис. 3-4. ЭВМ с одношинной структурой

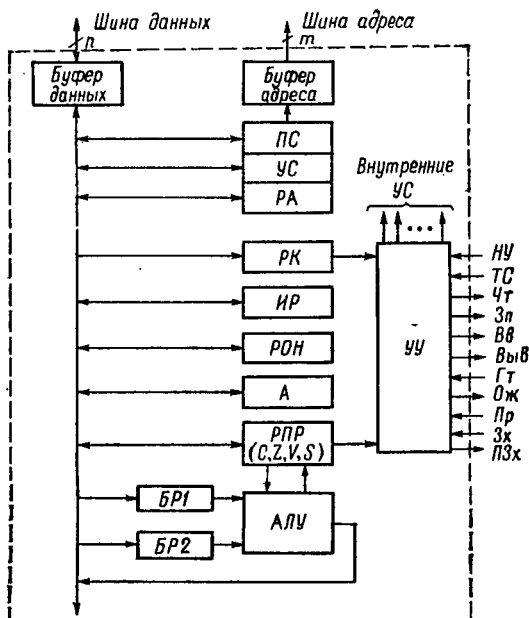
весь набор команд обращения к памяти может использоваться для передачи и обработки содержимого регистров периферийных устройств. Кроме того, другим важным достоинством является простота структуры шин и минимизация числа связей для обмена информацией между устройствами ЭВМ.

3-3. ОДНОКРИСТАЛЬНЫЕ МП

Под процессором понимают функциональный блок ЭВМ, предназначенный для логической и арифметической обработки информации на основе принципа программного управления. Основными характеристиками, определяющими различия в организации процессоров, являются система команд и режимы адресации; структура внутренних и внешних шин, используемых для сопряжения процессора с памятью и периферийными устройствами; число регистров и их функциональное назначение; организация арифметико-логического устройства и устройства управления; организация стековой памяти; обслуживание запросов прерываний; возможность наращивания разрядности процессора и модификации системы команд.

С точки зрения различий в реализации процессоров, определяемых возможностями используемой элементной базы и технологии, процессоры подразделяют на микропроцессоры (МП), реализованные на больших интегральных схемах (БИС), и процессоры, построенные на основе схем малой и средней степени интеграции. В свою очередь, МП, реализованные на одной БИС, называют однокристальными, а на нескольких БИС — многокристальными. Типичным примером отечественного однокристального МП является 8-битный МП КР580. Характерными чертами однокристальных МП являются фиксированный набор команд и разрядность без возможности ее наращивания. Многокристальные МП подразделяются на МП секционного типа, в которых обрабатывающее устройство МП строится из отдельных секций с возможностью наращивания разрядности, и МП с фиксированной разрядностью обрабатывающего устройства. Типичными представителями МП секционного типа являются МП серии К589 и К584, соответственно с 2-битной и 4-битной секцией.

Структурная схема типичного представителя класса однокристальных МП с отдельными шинами адреса и шинами данных приведена на рис. 3-5. МП выбирает коман-



ды из программной памяти микро-ЭВМ, выполняет их дешифрацию, обращение к памяти данных и устройствам ввода-вывода информации в соответствии с типом реализуемых команд. Кроме того, МП должен идентифицировать входные сигналы в МП, такие, как запросы прерываний, сигналы ожидания для синхронизации работы МП с медленнодействующими устройствами памяти и ввода-вывода информации, сигналы прямого доступа к памяти и другие, а также формировать необходимую последовательность внутренних и внешних сигналов для обращения к памяти, периферийным устройствам. Рассмотрим состав, назначение основных блоков МП и его функционирование.

Программный счетчик (ПС), или счетчик адреса команды, используется для хранения адреса команды, подлежащей выполнению. Начальный адрес ПС загружается с пульта управления микро-ЭВМ или ПС после начальной установки переходит в состояние 0, и выполнение программы начинается с нулевой ячейки программной памяти. В эту ячейку можно поместить команду безусловной пере-

дачи управления любому адресу программной памяти. После выборки команды или первого слова команды, содержащего код операции, в ПС путем увеличения его содержимого формируется адрес следующего слова выбираемой команды или следующей команды. Данная процедура характерна для линейной последовательности естественного порядка команд. При выполнении команд безусловного или условного перехода, обращения к подпрограммам адрес следующей команды в ПС формируется путем загрузки в него адресной части вышеуказанных команд. Код адреса из ПС передается в программную память через однонаправленную m -битную шину адреса.

Регистр команд (РК) принимает выбранную из памяти программ команду и хранит ее в течение цикла выполнения команды. Загрузка РК осуществляется по n -битной шине данных; если команда хранится в двух или более ячейках памяти, то для загрузки каждой части команды, например байта, требуется отдельный машинный цикл выборки команды. Ограниченная разрядность 8-битных МП приводит к тому, что для двух- и трехбайтных команд значительная доля (50—75 %) общего времени реализации команды может потребоваться для выборки команд и операций, разрядность которых может также превышать 8 бит. В 8-битных МП регистр команд обычно хранит первый байт команды, являющийся кодом операции, а второй или третий байты хранятся во вспомогательных регистрах, как правило, программно недоступных.

Указатель стека (УС) хранит адрес последней занятой ячейки в области стековой памяти. Под стековой памятью понимают оперативную память, в которую запись или выборка слова производится по принципу: последний записанный элемент выбирается из памяти первым. Адрес ячейки стековой памяти, по которому выполняется запись или чтение слова, называют вершиной стека. Начальный адрес стека устанавливается в указателе стека программным путем, и под действием команд записи или чтения информации из стека указатель стека автоматически изменяется, например, для МП КР580 увеличивается на 1, когда код извлекается из стека, и уменьшается на 1 при записи, т. е. стек «растет» снизу вверх. Некоторые МП используют внутреннюю стековую память команд для запоминания только адресов команд для возврата из подпрограмм. Однако емкость такой памяти ограничена, а кроме того, в ней нельзя запоминать содержимое регистров прерываемой программы, слово—состояние программы и другую информа-

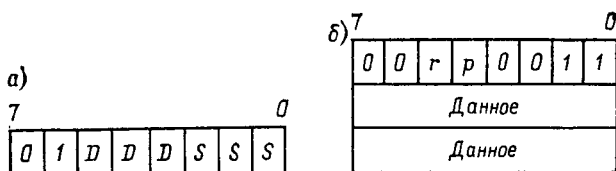


Рис. 3-6. Форматы команд передачи данных (а) и загрузки регистровой пары (б) в МП КР580

цию при переключении программ. Поэтому подавляющее большинство выпускаемых МП использует внешнюю стековую память, емкость которой для 16-битного указателя стека может составлять до 64К слов. Адресация стековой памяти также осуществляется через шину адреса МП.

В процессорах микро- и мини-ЭВМ широкое использование нашли регистры общего назначения (РОН), представляющие собой сверхоперативную память МП. РОН обычно используются для временного хранения исходных операндов при выполнении бинарных логических и арифметических операций, промежуточных результатов вычислений, иногда в РОН хранятся индексы, если в МП отсутствуют специальные индексные регистры (ИР). Это позволяет повысить быстродействие микро-ЭВМ за счет сокращения пересылок кодов между МП и памятью. Регистры общего назначения являются программно-доступными, и обращение к ним осуществляется посредством команд передачи данных. Так как число РОН обычно составляет 8—16, то для их адресации достаточно в поле адреса команд 3—4 разрядов соответственно. Например, в МП КР580, содержащем 6 РОН, формат команды для передачи данных между РОН имеет вид, представленный на рис. 3-6, а. В МП КР580 8-битные РОН В, С, D, E, H, L могут для некоторых команд рассматриваться как 16-битные регистры (В, С; D, E и H, L), что позволяет обрабатывать 16-битные данные параллельным способом. На рис. 3-6, б приведен формат команды с непосредственной адресацией «загрузить регистровую пару», код которой указывается в поле гр.

Число регистров в МП зависит от технологии и иногда представляет собой критический параметр для характеристики быстродействия микро-ЭВМ, используемой для решения задачи в режиме реального времени, которая требует интенсивной обработки запросов прерываний, что, в свою

очередь, может требовать частого запоминания и восстановления содержимого РОН при переключении МП с одной программы на другую. Естественно, время переключения увеличивается с числом РОН и поэтому в некоторых МП применяется дублирование массивов РОН, причем каждый массив используется одной программой или ограниченным числом программ, что, соответственно, исключает необходимость запоминания и восстановления содержимого РОН или значительно сокращает число этих процедур при переключении программ.

Аккумулятор А используется для хранения одного из операндов и результата операции, т. е. является источником и приемником информации. Команды ввода и вывода данных обычно осуществляют обмен кодов с периферийными устройствами также через аккумулятор. Некоторые МП содержат несколько аккумуляторов.

Арифметико-логическое устройство (АЛУ) предназначено для арифметической и логической обработки данных, выполнения операции сдвига, формирования признаков результатов операций. Если МП имеет только одну внутреннюю магистраль, как показано в МП на рис. 3-5, то на входах АЛУ необходимы буферные регистры (БР), которые выполняют функцию временного хранения операндов, участвующих в операции. Рассмотрим выполнение команды сложения содержимого аккумулятора с содержимым одного из РОН на примере аккумулятора А и регистра В (ADD В) для МП КР580. Для реализации команды потребуется три микротакта (МТ):

1-й МТ: (БР1) := (В);
 2-й МТ: (БР2) := (А);
 3-й МТ: (А) := (БР1) + (БР2).

После формирования в аккумуляторе результата выполнения арифметических и логических команд на регистре признаков (РПР) формируются признаки результатов операций Z, S, C, V и др. Эти признаки используются командами условий передачи управления при разветвлениях в программах. Команда условного перехода определяет тип

7							0	
1	1	C	C	C	0	1	0	
Адрес								
Адрес								

Рис. 3-7. Форматы команды условного перехода МП КР580

Таблица 3-1

Кодирование признаков в команде
условного перехода МП КР580

С	С	С	Значение признаков
0	0	0	NZ, Z = 0, результат $\neq 0$
0	0	1	Z, Z = 1, результат = 0
0	1	0	NC, C = 0, нет переноса
0	1	1	C, C = 1, есть перенос
1	0	0	PO, P = 0, результат чет- ный
1	0	1	PE, P = 1, результат нечет- ный
1	1	0	P, S = 0, результат поло- жительный
1	1	1	M, -S = 1, результат отри- цательный

признака, по которому требуется осуществить переход. На рис. 3-7 показан формат команды условного перехода МП КР580, а табл. 3-1 иллюстрирует кодирование признаков в 3-битном поле команды ССС. Если значение признака истинно, то в ПС загружается адресная часть команды условного перехода, иначе сохраняется естественный порядок следования команд.

В некоторых МП, например в КР580, возможно сложение десятичных чисел, представленных в двоично-десятичном коде. Однако сложение выполняется не посредством одной команды десятичного сложения, а с использованием двух команд: команды двоичного сложения ADC г и команды десятичной коррекции аккумулятора DAA. Рассмотрим пример сложения двух чисел, одно из которых (+59) в исходном состоянии хранится в аккумуляторе А, а другое (+79) — в регистре В.

Исходное положение:

(A): 0 1 0 1 1 0 0 1 (+59);
(B): 0 1 1 1 1 0 0 1 (+79)

1. Выполняется команда сложения ADC В:

+	0 1 0 1	1 0 0 1	(A)
	0 1 1 1	1 0 0 1	(B)
	1 1 0 1	0 0 1 0	(A)

При сложении младших тетрад сформировался перенос $AC=1$ из младшей тетрады результата, являющийся признаком необходимости коррекции, т. е. прибавления кода 0110. Коррекция необходима также, если при сложении тетрад формируется код в диапазоне 1010—1111, т. е. промежуточный результат больше 9, но меньше 16.

2. Выполняется команда коррекции DAA:

+	1 1 0 1	0 0 1 0	(A)
	0 1 1 0	0 1 1 0	
	1 0 0 1 1	1 0 0 0	(A)

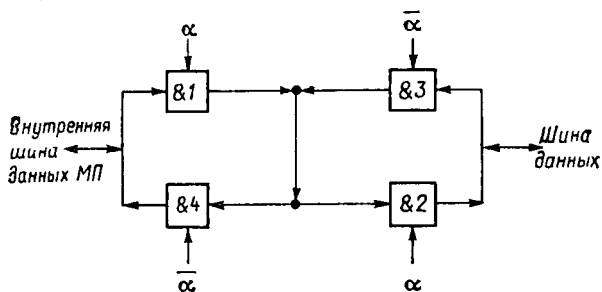


Рис. 3-8. Двухнаправленный буфер данных

Таким образом, в результате выполнения двух команд в аккумуляторе А сформировался двоично-десятичный код числа 38, а разряд сотен запоминается в регистре признаков результата (РПР) как признак переноса С из старшего разряда аккумулятора.

Обмен кодами между памятью команд, памятью данных, периферийными устройствами и микропроцессором осуществляется через двухнаправленный буфер шины данных, который изолирует внешнюю шину данных от внутренней. Принцип организации двухнаправленного буфера шины данных для передачи одного бита кода числа представлен на рис. 3-8. При передаче информации в МП с внешней шины данных на внутреннюю признак α , определяющий направление обмена, равен 0 и обмен осуществляется через логические схемы &3 и &4. При $\alpha=1$ обмен происходит в обратном направлении через &1 и &2. В МП обычно используют выходные буферы с тремя устойчивыми состояниями: 0, 1 и состоянием высокого выходного сопротивления (плавающее состояние), что позволяет упростить подключение к одной шине нескольких устройств. Доступ к шине в любой момент времени имеет только одно устройство, а остальные можно рассматривать практически не подключенными к шине, если они находятся в состоянии высокого выходного сопротивления.

Шина адреса используется для выдачи микропроцессором кода адреса программной памяти из ПС, адреса стековой памяти из УС, адреса памяти данных из регистра адреса (РА). Буфер адреса также имеет три состояния.

Формирование последовательностей внутренних и внешних управляющих сигналов, а также анализ сигналов из других устройств микро-ЭВМ выполняется устройством

управления, которое может быть реализовано на основе структуры с жесткой логикой или на основе программируемой логики, использующей память и программируемые логические матрицы. Остановимся более подробно на рассмотрении типичных входных и выходных управляющих сигналов, необходимых для функционирования микро-ЭВМ и называемых обычно системными управляющими сигналами.

Управляющие сигналы записи (Зп) и чтения (Чт) информации используются для управления обменом кодами между МП и памятью данных, программной памятью. Управляющие сигналы ввода (Вв) и вывода (Выв) разрешают обмен информацией с выбранным периферийным устройством. Пара сигналов «ожидание» (Ож) и «готовность» (Гт) выполняет функцию синхронизации МП с памятью и периферийными устройствами. Наличие сигнала Гт на входе МП указывает, что, например, данные из памяти готовы для передачи в МП. При отсутствии сигнала Гт МП переходит в состояние ожидания и формирует соответствующий выходной сигнал. На рис. 3-9, а показан фрагмент диаграммы переходов МП при реализации машинного цикла чтения информации из памяти. В состоянии T_1 с шины адреса МП в память передается код адреса. В состоянии T_2 МП анализирует сигнал Гт: если сигнал готовности из памяти отсутствует ($Гт=0$), то МП перейдет в состояние ожидания $T_{ож}$, и только при $Гт=1$ перейдет в состояние T_3 , в котором формируется управляющий сигнал Чт и выполняется прием кода по шине данных из памяти в МП. Такой протокол обмена обеспечивает сопряжение МП с памятью любого быстродействия. Формирование сигнала го-

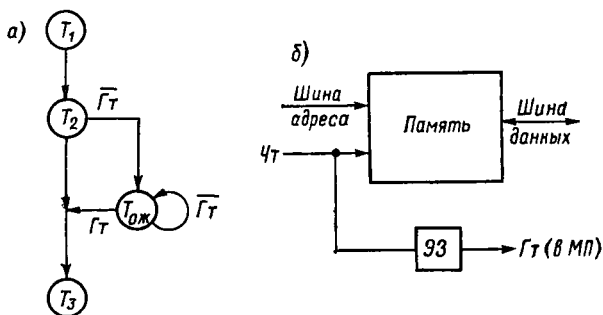


Рис. 3-9. Фрагмент диаграммы переходов МП (а) и формирование сигнала готовности (б)

товности из памяти осуществляется внешними по отношению к модулю памяти схемами, как это показано на рис. 3-9, б. Длительность задержки, реализуемой элементом ЭЗ, определяется исходя из разницы между временем обращения к памяти и длительностью состояния МП.

Сигнал прерывания (Пр) используется для организации обмена информацией с процессором или для временного переключения его на обслуживание какого-либо внешнего события, возникающего асинхронно по отношению к выполняемой программе. Большинство МП реагирует на данный сигнал только после выполнения текущей команды и при условии, что прерывание разрешено, т. е. процессор не замаскирован. Для обслуживания более одного источника запросов в режиме прерывания запросы источников объединяются по схеме ИЛИ на входе прерывания и процессор затем анализирует, какой источник запроса имеет более высокий приоритет в данный момент. Прохождением запросов прерывания в процессор управляет сигнал «разрешение прерывания», формируемый триггером разрешения прерывания, который является программно-доступным посредством команд процессора «разрешить прерывание» и «запретить прерывание». Кроме того, этот триггер после восприятия очередного прерывания автоматически переходит в состояние, соответствующее блокированию дальнейших запросов прерываний, выход из которого возможен посредством команды «разрешить прерывание». Поэтому при изучении конкретного процессора необходимо внимательно анализировать все ситуации, которые приводят к маскированию запросов прерываний.

Сигнал «захват» (Зх) обычно используется для организации обмена информацией между процессором и периферийными устройствами (быстродействующие АЦП, дисплеи, диски и др.) в режиме прямого доступа к памяти. В случае сигнала Зх процессор переходит в режим ожидания по захвату, шины адреса и данных переводятся в состояние высокого выходного сопротивления и доступ к данным шинам получает устройство, инициирующее сигнал для такого вида обмена. Отличительной особенностью режима захвата является то, что МП переходит в состояние захвата после выполнения текущего машинного цикла команды, не дожидаясь ее завершения, что позволяет сократить время реакции МП. Подтверждением перехода МП в состояние захвата является сигнал подтверждения захвата (ПЗх).

Сигнал начальной установки (НУ) обычно устанавливает ПС в 0, сбрасывает в 0 триггеры разрешения прерывания, захвата и другие схемы МП для инициирования выполнения программы. При этом содержимое РОН, аккумулятора и других устройств не определено.

Тактирующие (синхронизирующие) сигналы (ТС) используются для синхронизации работы МП и микро-ЭВМ.

Основными достоинствами однокристалльных МП являются следующие: однокристалльный МП является функционально и конструктивно завершенным модулем, реализованным на одной БИС, что требует минимального числа компонентов для построения микро-ЭВМ; надежность системы, построенной на основе однокристалльных МП, выше за счет выполнения соединений в едином технологическом процессе и значительного сокращения паяных соединений.

3.4. ОДНОКРИСТАЛЛЬНЫЙ 8-БИТНЫЙ МП

Организация МП КР580. Типичным представителем 8-битных однокристалльных МП, используемым для построения микро-ЭВМ СМ-1800, является МП КР580ИК80А (далее для краткости — КР580). Данный МП, выполненный по НМОП-технологии, содержит примерно 5000 элементов и реализован в 40-выводном корпусе. Число базовых команд МП КР580 составляет 78, время выполнения команд для тактовой частоты 2 МГц лежит в диапазоне 2—9 мкс. Команды МП могут быть одно-, двух- и трехбайтными. Двух- и трехбайтные команды хранятся в соседних ячейках памяти. МП КР580 имеет четыре режима адресации:

1. Прямая адресация. В этом режиме второй и третий байты команды содержат исполнительный адрес команды, причем во втором байте — младшие разряды, а в третьем — старшие.

2. Регистровая адресация — для обращения к внутренним регистрам МП.

3. Непосредственная адресация, при которой в команде указывается 8- или 16-битный операнд.

4. Косвенная регистровая адресация.

Структурная схема МП КР580 приведена на рис. 3-10. Функциональное назначение выводов МП следующее:

$A_{15}-A_0$ — шина адреса с тремя состояниями, обеспечивает адресацию памяти емкостью до 64К байт, адресацию 256 портов ввода и 256 портов вывода информации;

D_7-D_0 — двунаправленная шина данных с тремя

состояниями, обеспечивает обмен информацией между МП, памятью и периферийными устройствами;

DBIN — выходной сигнал приема, указывает памяти и периферийным устройствам, что шина данных находится в режиме приема информации в МП;

$\overline{WR}$ — выходной сигнал выдачи, используется для управления выдачей информации из МП в память и периферийные устройства, активным является сигнал $\overline{WR}=0$;

INT — входной сигнал прерывания, воспринимаемый МП после выполнения текущей команды или в режиме останова; прерывание не воспринимается МП, если он находится в режиме захвата или в режиме запрещенных прерываний, что обеспечивается установкой триггера разрешения прерываний в состояние 0 командой DI;

INTE — выходной сигнал разрешения прерываний, индицирует состояние триггера разрешения прерываний; данный триггер сбрасывается в 0 после команды DI, а также после приема сигнала прерывания INT или сигнала сброса RESET;

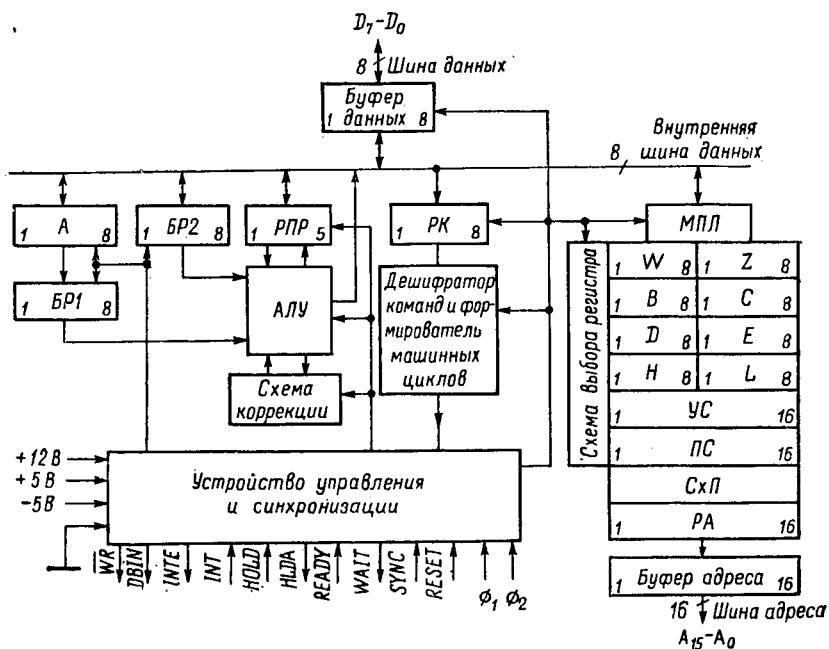


Рис. 3-10. Структурная схема МП КР580

HOLD — входной сигнал захвата, переводит МП в состояние захвата после завершения обмена данными между МП, памятью или периферийными устройствами в текущем машинном цикле; после захвата МП шины данных и адреса переходят в состояние высокого выходного сопротивления;

HLDA — выходной сигнал подтверждения состояния захвата МП;

READY — входной сигнал готовности, информирует МП, что данные из внешнего источника переданы на шину данных, синхронизирует работу МП с более медленноресурсными памятью или периферийными устройствами; при нулевом значении этого сигнала МП переходит в состояние ожидания T_w ;

WAIT — выходной сигнал ожидания, подтверждающий, что МП находится в состоянии ожидания T_w ;

RESET — входной сигнал сброса, обеспечивает установку в состояние 0 регистра команд, программного счетчика, триггеров разрешения прерывания и подтверждения захвата; при этом состояние остальных регистров не изменяется;

SYNC — выходной сигнал синхронизации, определяет начало машинного цикла;

ϕ_1, ϕ_2 — тактовые сигналы.

МП КР580 содержит шесть 8-битных РОН, которые обозначаются В, С, D, Е, Н, L и могут объединяться в 16-битные пары В и С; D и Е; Н и L. Регистры W, Z являются программно-недоступными и выполняют вспомогательные функции, например используются для временного хранения адресов. Содержимое программного счетчика (ПС) автоматически увеличивается на 1 при выборке каждого байта команды, при этом первый байт, определяющий код операции, всегда передается в регистр команд (РК). Указатель стека (УС) хранит адрес последней занятой ячейки в стековой памяти. Содержимое УС уменьшается перед записью кода в стек и увеличивается после чтения.

Обмен данными внутри МП осуществляется по внутренней 8-битной шине данных через двунаправленный мультиплексор (МПЛ). На регистр адреса (РА) коды передаются из регистровых пар, УС и ПС при обращении к памяти и периферийным устройствам. Схема приращения (СхП) обеспечивает добавление 1 к содержимому регистров или вычитание 1 из него. АЛУ используется для выполнения всех арифметических и логических операций,

сдвигов и т. п. Арифметические операции выполняются в дополнительных кодах. Аккумулятор А является источником одного из операндов при выполнении бинарных операций, и приемником результата. Буферные регистры БР1 и БР2 выполняют вспомогательные функции.

АЛУ после выполнения команды формирует признаки результатов, которые фиксируются в 5-битном регистре признаков РПР. АЛУ формирует следующие признаки: нулевого результата Z, знака S, переноса C из старшего бита, переноса AC из третьего бита аккумулятора, четности P. Признаки AC и C при выполнении операций над числами в десятичной системе счисления используются для коррекции результата.

Первый байт команды, выбранный из программной памяти, передается по внутренней шине данных на РК. Выход РК связан с дешифратором команд ДШК, который определяет тип выполняемой операции. Реализация любой команды складывается из машинных циклов, число которых может быть от одного (для самой короткой команды) до пяти (для самой длинной). Для передачи одного байта между МП и памятью или периферийным устройством требуется три состояния машинного цикла, а максимальное число состояний машинного цикла — пять. Время одного состояния определяется периодом следования тактирующих сигналов. Внутреннее преобразование данных в МП выполняется в состояниях T_4 , T_5 . Начало каждого машинного цикла идентифицируется синхросигналом SYNC. Минимальная длительность одного состояния T равняется 0,5 мкс для $F_{\max} = 2$ МГц, но имеется три состояния, продолжительность которых может быть значительно больше: это состояние ожидания, захвата и ожидания при останове.

В МП КР580 имеется десять типов машинных циклов, которые могут быть при выполнении команды: выборка байта команды, чтение из памяти, запись в память, чтение из стековой памяти, запись в стековую память, ввод, вывод, обработка прерывания, останов, обработка прерывания при останове. При этом первым машинным циклом всегда является выборка команды. Для определения типа машинного цикла в первом состоянии каждого машинного цикла на шину данных передается 8-битный код, который запоминается во внешнем регистре и используется для формирования системных управляющих сигналов для обращения к памяти, периферийным устройствам. Рассмотрим назначение каждого разряда в коде слова состояния:

INTA (D_0) — признак подтверждения прерывания, используется для разрешения передачи команды вызова подпрограммы обработки запроса прерывания;

$\overline{WO}$ (D_1) — признак записи-вывода, указывает, что в текущем машинном цикле будет выполняться запись в память или вывод кода в порт вывода (при $WO = 0$);

STACK(D_2) — признак указывает, что на шину адреса передается адрес из указателя стека, используемый для адресации области стековой памяти;

HLTA (D_3) — признак подтверждает выполнение команды останова HLT;

OUT(D_4) — признак вывода, указывает, что шина адреса содержит адрес порта вывода, а шина данных будет содержать данные при сигнале $\overline{WR} = 0$;

M1(D_5) — признак указывает, что МП находится в цикле выборки первого байта команды;

INP(D_6) — признак ввода, указывает, что шина адреса содержит адрес порта ввода и входные данные передаются по шине данных при сигнале $DBIN = 1$;

MEMR(D_7) — признак указывает, что шина данных будет использована для приема данных из памяти.

Диаграмма переходов при выполнении машинного цикла в МП КР580 представлена на рис. 3-11, на котором используются следующие обозначения T_w — состояние ожидания МП; T_{wn} — состояние ожидания МП при выполнении команды останова HLT; T_1 — состояние МП, $i = 1 \div 5$.

Каждый машинный цикл содержит 3, 4 или 5 состояний в зависимости от типа выполняемой команды. Первое состояние T_1 идентифицируется синхросигналом SYNC, который используется для загрузки слова состояния в регистр. Кроме того, в T_1 на шину адреса передается адрес устройства, к которому будет обращение.

В состоянии T_2 МП анализирует признак подтверждения останова HLTA и сигнал готовности READY и соответственно переходит в состояние T_{wn} или T_w . Если имел место сигнал захвата HOLD, то внутренний триггер захвата устанавливается в состояние 1.

В состоянии T_3 действия определяются типом машинного цикла. В цикле выборки МП интерпретирует код на шине данных как байт КОП, а в циклах чтения из памяти, записи в память, ввода и вывода информации осуществляет обмен данными.

Состояния T_4 и T_5 являются необязательными, так как используются для внутренних преобразований в МП. Та-

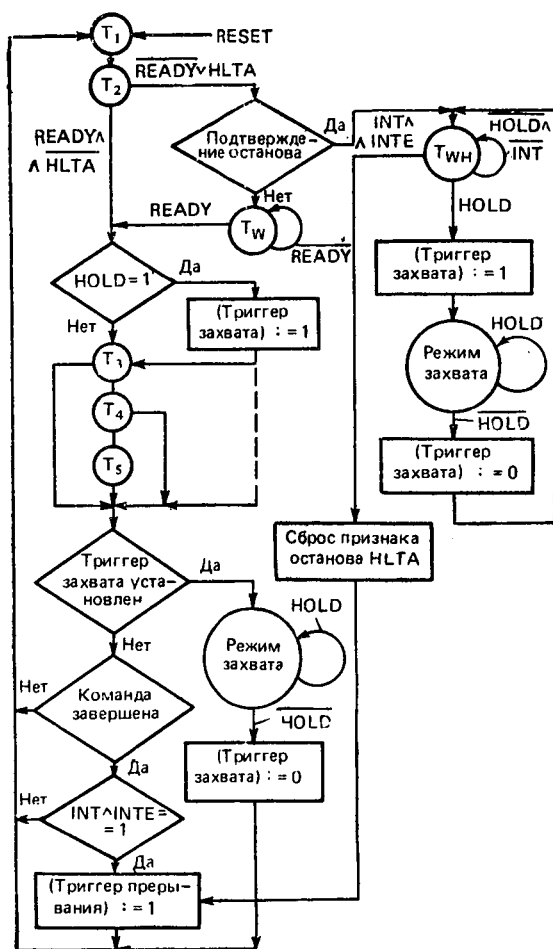


Рис. 3-11. Диаграмма переходов для машинного цикла МП КР580

ким образом, МП может переходить в следующее состояние T_1 после T_3 , T_4 или T_5 .

Формирование системных управляющих сигналов. Набор управляющих сигналов, таких как чтение из памяти ($\overline{MEMR}$), запись в память ($\overline{MEMW}$), ввод информации ($\overline{I/OR}$), вывод информации ($\overline{I/OW}$), подтверждение прерывания ($\overline{INTA}$), обеспечивает прием и передачу кодов

между МП, памятью и периферийными устройствами в определенные интервалы времени в соответствии с диаграммой переходов машинного цикла. Данные сигналы непосредственно не формируются микропроцессором КР580, для их формирования используются сигналы приема (DBIN) и записи ($\overline{WR}$) из МП и необходимые признаки из слова состояния:

$$\overline{MEMR} = \overline{DBIN \wedge MEMR};$$

$$\overline{MEMW} = \overline{\overline{WR} \wedge \overline{WO}};$$

$$\overline{I/OR} = \overline{DBIN \wedge INP};$$

$$\overline{I/OW} = \overline{\overline{WR} \wedge OUP};$$

$$\overline{INTA} = \overline{DBIN \wedge INTA}.$$

Слово состояния загружается в 8-битный регистр, например К589ИР12, посредством сигнала синхронизации SYNC и тактирующего сигнала $\emptyset_2$ уровня ТТЛ, формируемого генератором тактовых сигналов.

На рис. 3-12 показано формирование системных управляющих сигналов в соответствии с вышеприведенными выражениями. Другие признаки слова состояния могут использоваться при тестировании МП. Остановимся несколько подробнее на обработке запросов прерывания. Устройство, обслуживаемое МП в режиме прерывания, инициирует прерывание путем формирования на входе INT МП сигнала 1, который может возникнуть в любом машинном цикле команды. Однако, как видно из диаграммы переходов, текущая команда должна завершиться и только после этого МП переходит к машинному циклу подтверждения прерывания, который имеет ряд отличий от цикла выборки команды. Во-первых, в слове состояния дополнительно к признаку машинного цикла выборки команды M1 содержится признак INTA, подтверждающий, что сигнал прерывания воспринят МП. Во-вторых, ПС не увеличивается на 1 в данном цикле, так как, увеличенный после выборки предыдущей команды, он должен быть запомнен в стековой памяти без изменения. В-третьих, вместо команды из программной памяти на шину данных передается код, представляющий собой однобайтную команду RST. Команда RST обеспечивает запоминание содержимого программного счетчика в стеке, и в ПС формируется один из восьми начальных адресов, в которых хранятся первые команды подпрограмм обслуживания

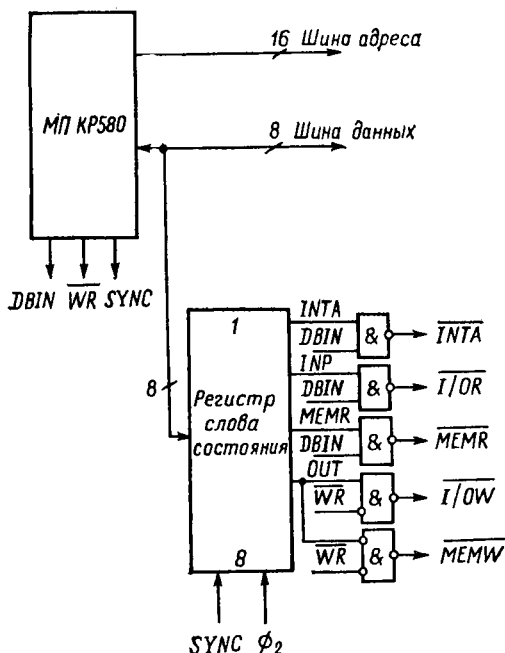


Рис. 3-12. Формирование системных управляющих сигналов

прерываний. Если область памяти в 8 байт недостаточна для размещения подпрограммы, что, как правило, имеет место, то путем использования команд перехода ее можно расширить. Таким образом, первые 64 ячейки памяти (0000H — 003FH) зарезервированы для подпрограмм обработки прерываний.

Система команд МП КР580. Все множество команд данного МП можно подразделить на следующие 5 групп:

1. Команды пересылки кодов, обеспечивающие пересылку данных между регистрами или памятью и регистрами. Команды данной группы не формируют признаков результатов операций.

2. Арифметические команды, обеспечивающие выполнение операций сложения и вычитания, изменения кодов на 1. Один операнд для бинарных операций хранится в аккумуляторе А, другой — в регистре или ячейке памяти, а результат помещается в аккумулятор. Такие операции, как умножение и деление, выполняются программным пу-

Таблица 3-2

Система команд МП КР580

Группа команд	Мнемоника команды	Описание команды	Число байт	Признаки результата операции				
				Z	S	P	C	AC
Команды пересылки кодов	MOV D, S	$(D) \leftarrow (S)$	1	—	—	—	—	—
	MVI D, data	$(D) \leftarrow \text{data}$	2	—	—	—	—	—
	LXI rp, data	$(D) \leftarrow \text{data}$	3	—	—	—	—	—
	LDA addr	$(A) \leftarrow (\text{addr})$	3	—	—	—	—	—
	STA addr	$(\text{addr}) \leftarrow (A)$	3	—	—	—	—	—
	LHLD addr	$L \leftarrow (\text{addr}), H \leftarrow (\text{addr} + 1)$	3	—	—	—	—	—
	SHLD addr	$(\text{addr}) \leftarrow L, (\text{addr} + 1) \leftarrow (H)$	3	—	—	—	—	—
	LDAX rp	$A \leftarrow (rp)$	1	—	—	—	—	—
	STAX rp	$((rp)) \leftarrow A$	1	—	—	—	—	—
	XCHG	$(H) \leftrightarrow (D), (L) \leftrightarrow (E)$	1	—	—	—	—	—
Команды арифметической обработки	ADD S	$(A) \leftarrow (A) + (S)$	1	+	+	+	+	+
	ADI data	$(A) \leftarrow (A) + \text{data}$	2	+	+	+	+	+
	ADC S	$(A) \leftarrow (A) + (S) + (C)$	1	+	+	+	+	+
	ACI data	$(A) \leftarrow (A) + \text{data} + (C)$	2	+	+	+	+	+
	DAD rp	$(H, L) \leftarrow (H, L) + (rp)$	1	—	—	—	+	—
	SUB S	$(A) \leftarrow (A) - (S)$	1	+	+	+	+	+
	SUI, data	$(A) \leftarrow (A) - \text{data}$	2	+	+	+	+	+
	SBB S	$(A) \leftarrow (A) - (S) - (C)$	1	+	+	+	+	+
	SBI data	$(A) \leftarrow (A) - \text{data} - (C)$	2	+	+	+	+	+
	INR D	$(D) \leftarrow (D) + 1$	1	+	+	+	—	+
	INX rp	$(rp) \leftarrow (rp) + 1$	1	—	—	—	—	—
	DCR D	$(D) \leftarrow (D) - 1$	1	+	+	+	—	+
	DCR rp	$(rp) \leftarrow (rp) - 1$	1	—	—	—	—	—
	DAA	Десятичная коррекция	1	+	+	+	+	+
Команды логической обработки	ANA S	$(A) \leftarrow (A) \wedge (S)$	1	+	+	+	0	+
	ANI data	$(A) \leftarrow (A) \wedge \text{data}$	2	+	+	+	0	+
	XRA S	$(A) \leftarrow (A) \oplus (S)$	1	+	+	+	0	0
	XRI data	$(A) \leftarrow (A) \oplus \text{data}$	2	+	+	+	0	0
	ORA S	$(A) \leftarrow (A) \vee (S)$	1	+	+	+	0	0
	ORI data	$(A) \leftarrow (A) \vee \text{data}$	2	+	+	+	0	0
	CMP S	$(A) - (S)$	1	+	+	+	+	+
	CPI data	$(A) - \text{data}$	2	+	+	+	+	+
	RLC	$(A_{n+1}) \leftarrow (A_n),$ $(A_0) \leftarrow (A_7), (C) \leftarrow (A_7)$	1	—	—	—	+	—
	RRC	$(A_n) \leftarrow (A_{n+1}),$ $(A_7) \leftarrow (A_0), (C) \leftarrow (A_0)$	1	—	—	—	+	—
	RAL	$(A_{n+1}) \leftarrow (A_n),$ $(C) \leftarrow (A_7), (A_0) \leftarrow (C)$	1	—	—	—	+	—
	RAR	$(A_n) \leftarrow (A_{n+1}),$ $(C) \leftarrow (A_0), (A_7) \leftarrow (C)$	1	—	—	—	+	—
	STC	$(C) \leftarrow 1$	1	—	—	—	1	—
	CMC	$(C) \leftarrow \overline{(C)}$	1	—	—	—	+	—
	CMA	$(A) \leftarrow \overline{(A)}$	1	—	—	—	—	—

Группа команд	Мнемоника команды	Описание команды	Число байт	Признаки результата операции				
				Z	S	P	C	AC
Команды передачи управления	JMP addr	$(PC) \leftarrow (addr)$	3	—	—	—	—	—
	Jcond addr	$(PC) \leftarrow (addr)$, если условие CCC в поле cond истинно	3	—	—	—	—	—
	CALL addr	$(PC) \leftarrow (addr)$	3	—	—	—	—	—
	Ccond addr	$(PC) \leftarrow (addr)$, если условие CCC в поле cond истинно	3	—	—	—	—	—
	RET	Возврат из подпрограммы	1	—	—	—	—	—
	Rcond	Возврат из подпрограммы, если условие CCC в поле cond истинно	1	—	—	—	—	—
	PCHL	$(PC) \leftarrow (H, L)$	1	—	—	—	—	—
Команды ввода и вывода, обращения к стеку и управления МП	RSTn	Команда повторного старта	1	—	—	—	—	—
	IN port	$(A) \leftarrow (port)$	2	—	—	—	—	—
	OUT port	$(port) \leftarrow (A)$	2	—	—	—	—	—
	PUSH rp	$((SP)) \leftarrow (rp)$	1	—	—	—	—	—
	PUSH PSW	$((SP)) \leftarrow (A)$ и (Z, S, P, C, AC)	1	—	—	—	—	—
	POP rp	$(rp) \leftarrow ((SP))$	1	—	—	—	—	—
	POP PSW	(Z, S, P, C, AC) и $(A) \leftarrow ((SP))$	1	+	+	+	+	+
	XTHL	$(H, L) \leftrightarrow ((SP))$	1	—	—	—	—	—
	SPHL	$(SP) \leftarrow (H, L)$	1	—	—	—	—	—
	EI	Разрешение прерывания	1	—	—	—	—	—
	DI	Запрещение прерывания	1	—	—	—	—	—
	HLT	Останов	1	—	—	—	—	—
	NOP	Пустая команда	1	—	—	—	—	—

тем с использованием подпрограмм. Отрицательные числа при выполнении арифметических операций необходимо преобразовывать в дополнительный код.

3. Логические команды реализуют операции логического сложения и умножения, исключающего ИЛИ, инвертирования, левого и правого сдвигов и некоторые другие. Исходные операнды хранятся в регистрах или ячейках памяти, а результат помещается в аккумулятор.

4. Команды передачи управления, в число которых входят команды безусловной и условной передачи управ-

ления, обращения и выхода из подпрограмм. Данные команды не формируют признаков результатов операций.

5. Команды ввода и вывода информации, обращения к стековой памяти. Кроме того, в эту группу входит ряд команд управления работой МП.

Система команд микропроцессора КР580 приведена в табл. 3-2, в которой использованы следующие условные обозначения: D, S — приемник (Destination) или источник (Source) информации, в качестве которых используются регистры МП В, С, D, Е, H, L, аккумулятор А и ячейки памяти М (Memory); data — 8- или 16-битное данное; addr — 16-битный адрес памяти; (...) — содержимое ячейки памяти или регистра МП; (SP) — содержимое указателя стековой памяти; ((SP)) — содержимое ячейки стековой памяти; port — 8-битный адрес периферийного устройства; gr — регистровая пара В, С; D, Е; H, L или указатель стека SP; CCC — 3-битный код признака, используемого в командах передачи управления (см. табл. 3-1); n — номер команды повторного старта, $n=0\div7$.

3-5. ОДНОКРИСТАЛЛЬНЫЙ 16-БИТНЫЙ МП

Наиболее известен 16-битный микропроцессор 8086¹ фирмы Intel. Он выполнен по НМОП-технологии, имеет 40-контактный корпус и одно напряжение питания +5 В. Номинальная частота однофазных сигналов синхронизации составляет 5 МГц. На кристалле размером 5,5×5,5 мм размещено 29 000 транзисторов. Для достижения номинальной производительности необходима память с циклом 500—800 нс и временем обращения при считывании 290—460 нс. Все регистры и двунаправленная мультиплексная шина AD адреса и данных 16-битные. Старшие 4 бита адреса мультиплексируются с сигналами состояния. Следовательно, длина физического адреса памяти составляет 20 бит, что обеспечивает адресное пространство 1М байт. Микропроцессор прямо адресует 256 портов ввода и 256 портов вывода и косвенно — 64К 8-битных портов ввода-вывода.

В архитектуре микропроцессора видна тенденция сохранения программной совместимости с 8-битным микропроцессором 8080. Хотя прямой совместимости вверх не достигнуто, ассемблерные программы 8080 легко трансформируются в программы 8086 (имеется специальная программа преобразования).

¹ Отечественный аналог — микропроцессор К1810ВМ86.

Функционально микропроцессор состоит из двух функционально автономных устройств: шинного интерфейса и операционного (исполнительного) устройства. Шинный интерфейс содержит секцию управления, 6-байтную очередь команд (эквивалент регистра команд), 4 сегментных регистра и программный счетчик (называемый указателем команд IP). Это устройство обеспечивает взаимодействие с остальными компонентами ЭВМ и нахождение в очереди команд достаточного числа байт. Как только в очереди фиксируются два свободных байта, инициируется обращение к программной памяти. Операционное устройство выполняет все преобразования данных, последовательно считывая и интерпретируя байты из очереди команд.

Микропроцессор допускает два режима системной конфигурации: максимальный и минимальный, которые определяются уровнем сигнала на входе $MN/\overline{MX}$. В зависимости от выбранного режима изменяется интерпретация 8 управляющих сигналов. При минимальной конфигурации управляющие сигналы для памяти и периферийных устройств генерирует сам микропроцессор, а в максимальном режиме необходим специальный контроллер шины. Для реализации мультимикропроцессорных систем предусмотрен выходной сигнал $\overline{LOCK}$ блокировки шины. При низком уровне этого сигнала блокируется (запрещается) доступ к системной шине другим компонентам системы. Формированием этого сигнала управляет пользователь, применяя в программах префикс LOCK.

Память представляет собой массив 1М байт, причем любые два смежных байта образуют слово, т. е. слово может начинаться с четного или нечетного адреса. В первом случае слово передается за один цикл шины, а во втором требуется два цикла шины, поэтому целесообразно размещать слова данных по четным адресам. Длина команд составляет 1—6 байт, и их размещение в памяти на производительность не влияет. Микропроцессор оперирует 16-битными операндами, поэтому все адресные объекты должны иметь длину 16 бит. Следовательно, для формирования 20-битных адресов необходим дополнительный механизм. Таким механизмом в микропроцессоре 8086 является сегментация памяти.

Можно считать, что память состоит из произвольного числа сегментов емкостью по 64К байт. Начальный адрес каждого сегмента (20 бит) кратен 16, т. е. содержит нули в 4 младших битах. Таким образом, сегменты могут начинаться на границах блоков по 16 байт и их начальные

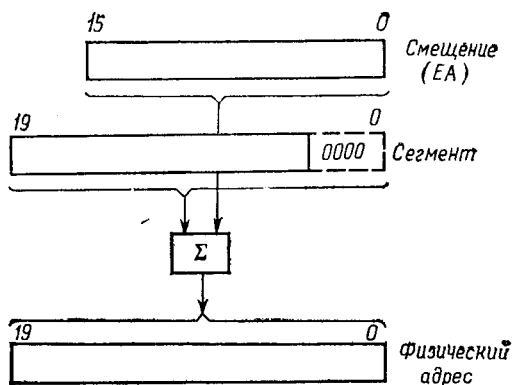


Рис. 3-13. Формирование физического адреса в МП 8086

адреса имеют вид $\times \times \times \times 0$. Микропроцессор управляет 4 сегментами (программы (кода), данных, стека и дополнительного сегмента данных), начальные адреса которых без младших нулей хранятся в 16-битных сегментных регистрах CS, DS, SS и ES. Команды с обращением к памяти формируют 16-битный исполнительный (эффективный) адрес EA, представляющий собой смещение в сегменте. Физический 20-битный адрес памяти равен сумме содержимого EA и сегментного регистра (рис. 3-13). При адресации портов ввода-вывода сегментные регистры не используются.

В программной модели микропроцессора, приведенной на рис. 3-14, заштрихованы регистры, в известной степени эквивалентные регистрам микропроцессора 8080. Для удобства рассмотрения регистры программной модели разделены на три группы (файла).

Регистры общего назначения представлены 16-битными регистрами AX, BX, CX и DX, допускающими раздельную адресацию старших (H) и младших (L) 8-битных регистров. Двойственный характер РОН обеспечивает простую обработку 8- и 16-битных данных. Остальные регистры можно адресовать только как 16-битные. В основном все РОН в арифметических и логических операциях используются одинаково. Но имеется значительное число команд, которые специализируют некоторые РОН, что отражено в их названиях. Регистр AX выполняет функции аккумулятора, он является источником и получателем в операциях

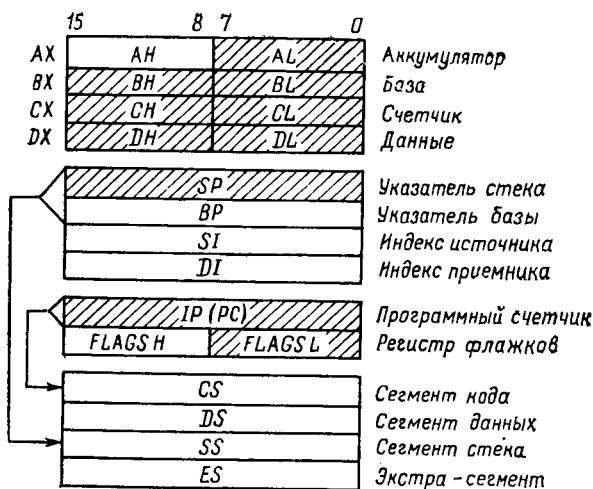


Рис. 3-14. Программная модель МП 8086

ввода-вывода, с ним связаны операции преобразования, десятичной коррекции, умножения и деления. Регистр BX в некоторых командах участвует как регистр базового адреса и более или менее соответствует регистрам H, L микропроцессора 8080. Регистр CX используется как счетчик в операциях сдвигах, командах закидывания и в операциях с цепочками байт и слов. Наконец, регистр DX неявно адресуется в операциях умножения и деления, а также содержит адрес портов ввода или вывода в режиме косвенной адресации.

Группа указательных и индексных регистров включает в себя 16-битные регистры SP, BP, SI и DI. Они обычно содержат внутрисегментные смещения и обеспечивают косвенную адресацию и динамическое вычисление исполнительных адресов. Гибкость таких вычислений достигается тем, что рассматриваемые регистры могут участвовать в арифметических и логических операциях как регистры общего назначения. Регистры SP (указатель стека) и BP (указатель базы) адресуют данные в текущем сегменте стека, а не в сегменте данных. Поэтому, если сегмент специально не определен, смещения в SP и BP относятся к текущему сегменту стека. Индексные регистры SI (источника) и DI (получателя) содержат смещения, которые обычно относятся к текущему сегменту данных.

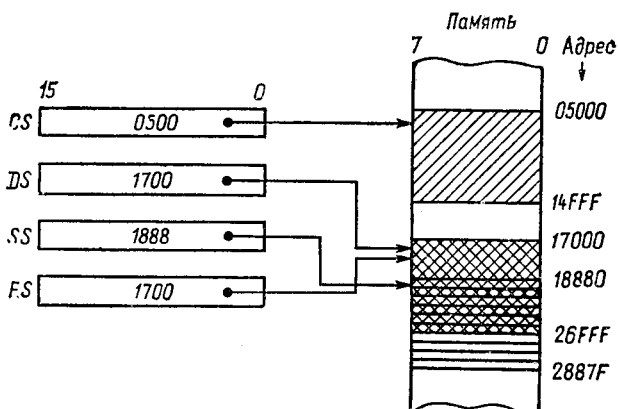


Рис. 3-15. Адресация сегментов в МП 8086

Четыре 16-битных сегментных регистра CS, DS, SS и ES используются для задания текущих сегментов по 64K байт, адресуемых при выполнении программы (рис. 3-15). Каждый регистр адресует конкретный текущий сегмент, что отражено в их названиях: код, данные, стек, дополнительные данные. Сегментные регистры автоматически действуют в качестве базовых регистров при всех обращениях к памяти, т. е. исполнительный адрес суммируется с содержимым определенного сегментного регистра. Выборка всех команд осуществляется из текущего сегмента кода, а смещением служит содержимое указателя команд IP. Все исполнительные адреса данных (кроме адресов, формируемых через BP или SP, а в операциях с цепочками — через DI) суммируются с содержимым DS. Обращения к стеку, включая неявные действия при прерываниях, вызовах подпрограмм и возвратах, а также явные стекосые включения и извлечения, осуществляются через SS. В операциях с цепочками и использованием DI физический адрес формируется через ES. Отметим, что любой команде с вычислением исполнительного адреса может предшествовать определяемый пользователем однобайтный префикс, в котором двухбитное поле явно определяет используемый сегмент.

Основное назначение сегментных регистров — динамическое перемещение программ в памяти, которое необходимо в мультипрограммной среде. По существу, для такого перемещения достаточно модифицировать содержимое

сегментного регистра CS при условии, что программа сама не изменяет его содержимого. Кроме того, модифицируя содержимое DS, прикладная программа может манипулировать данными такого объема, который превышает емкость сегмента.

16-битный регистр состояния (регистр флажков) FLAGS разделен на две половины, младшая из которых FLAGSL полностью соответствует регистру флажков микропроцессора 8080. В старшей половине введены 4 новых флажка: OF — арифметического переполнения; DF — направления (определяет направление сканирования массива в операциях с цепочками); IF — прерывания (выполняет функцию маскирования внешних прерываний); TF — прослеживания (обеспечивает выполнение программы по командам).

Система команд микропроцессора 8086 не является подмножеством системы команд микропроцессора 8080. Хотя сохранена преимущество большинства команд, но некоторые редко используемые команды (например, условные вызовы и возвраты) исключены и введено много новых команд.

Команды манипулируют одним или двумя операндами, а результат замещает один (часто любой) из операндов. Многие команды обрабатывают 8- и 16-битные операнды. Первым операндом в двухоперандной команде обычно является регистр¹ или память, а во втором — регистр или константа (непосредственный операнд). В общем, нумерация операндов чисто условная и не отражает никакой направленности в передаче данных.

Общий формат двухоперандной команды, когда вторым операндом является регистр, показан на рис. 3-16. Первый байт содержит код операции и два однобитных поля: *d* (направление) и *w* (слово). Поле *d* определяет направление передачи, относящееся ко второму операнду — регистру, идентифицируемому 3-битным полем *reg* второго байта команды. Если *d* = 1, то направление пе-

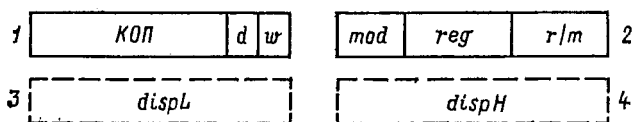


Рис. 3-16. Формат двухоперандной команды

¹ Разумеется, речь идет о содержимом регистра.

редачи — в регистр, а если $d=0$, то направление — из регистра. Поле w определяет тип операнда: слово ($w=1$) или байт ($w=0$).

Второй байт команды, называемый постбайтом, имеет три поля. Поле reg определяет второй операнд, обязательно находящийся в регистре. В этом поле кодируются следующие 8- или 16-битные регистры:

$reg (r/m)$	000	001	010	011	100	101	110	111
8 бит	AL	CL	DL	BL	AH	CH	DH	BH
16 бит	AX	CX	DX	BX	SP	BP	SI	DI

Поле режима mod определяет используемый режим адресации. В частности, оно влияет на интерпретацию содержимого поля r/m (регистр/память) при нахождении первого операнда. Поле mod имеет следующий общий смысл

$$mod = \begin{cases} 00, 01, 10 & \text{— операнд содержится в памяти;} \\ 11 & \text{— операнд содержится в регистре.} \end{cases}$$

Когда $mod=11$, поле r/m определяет 8- или 16-битный регистр в соответствии с приведенным выше кодированием.

Если $mod \neq 11$, это поле определяет, каким образом используется необязательное смещение $disp$:

$$mod = \begin{cases} 00, disp=0 & \text{(байты } disp \text{ L и } disp \text{ H отсутствуют);} \\ 01, disp=disp \text{ L} & \text{(с расширением знака до 16 бит);} \\ 10, disp=disp \text{ H, } disp \text{ L} & \text{(два байта смещения).} \end{cases}$$

В этом же случае ($mod \neq 11$) поле r/m определяет, как формируется исполнительный адрес EA команды:

r/m	EA
000	$(BX) + (SI) + disp$
001	$(BX) + (DI) + disp$
010	$(BP) + (SI) + disp$
011	$(BP) + (DI) + disp$
100	$(SI) + disp$
101	$(DI) + disp$
110	$(BP) + disp$
111	$(BX) + disp$

Приведенные правила имеют только одно исключение: если $mod=00$ и $r/m=110$, то $EA=disp \text{ H, } disp \text{ L}$. Такой способ формирования EA называется прямой адресацией, т. е. адрес содержится в команде.

Следовательно, операнд в памяти может адресоваться прямо (16-битное смещение) или косвенно (8- или 16-бит-

ное смещение). Во втором случае допускается адресация посредством содержимого базового регистра (BP или BX), индексного регистра (SI или DI), а также суммы содержимого базового и индексных регистров. Всего получается 24 режима адресации: 3 способа интерпретации поля *mod* и 8 способов интерпретации поля *г/м*.

Режимы адресации спроектированы с учетом эффективной реализации языков высокого уровня. Например, к простой переменной можно обратиться в режиме прямой адресации, а к элементу массива — в режиме косвенной адресации посредством BX, SI и смещения. Режим адресации через BP предназначен для доступа к данным из сегмента стека, а не из сегмента данных. При реализации рекурсивных процедур и компиляторов языков высокого уровня с блоковой структурой данные часто запоминаются в сегменте стека.

Система команд насчитывает 113 базовых команд, объединяемых в следующие группы:

Команды передачи данных. Команды данной группы подразделяются на четыре разновидности:

а) команды передачи данных между регистрами и памятью, включения в стек и извлечения из стека (адресуемого SP), а также команда обмена содержимым источника и приемника;

б) команды ввода, вывода, табличного преобразования;

в) команды загрузки исполнительного адреса в регистры общего назначения, а также загрузки 4-байтного адресного объекта в регистры-указатели (начальный адрес сегмента и смещение в сегменте); эти команды предоставляют пользователю возможность управлять механизмом адресации микропроцессора;

г) команды передачи содержимого регистра флажков (запоминание в памяти и загрузка из памяти регистра FLAGSL, включение в стек и извлечение из стека всего регистра FLAGS).

Арифметические команды. Микропроцессор имеет команды сложения, вычитания, умножения и деления двоичных знаковых и беззнаковых чисел. Произведение и делимое представляются числами двойной длины. Предусмотрены команды коррекции сложения и вычитания упакованных двоично-десятичных чисел, а также команды коррекции сложения, вычитания, умножения и деления неупакованных двоично-десятичных чисел. Интересно отметить, что для деления коррекция делимого осуществляется до выполнения операции.

Логические команды и команды сдвига. В эту группу входят поразрядные операции инверсии, конъюнкции, дизъюнкции и исключающего ИЛИ, а также команда TEST проверки операнда. Последняя выполняет поразрядную конъюнкцию, устанавливает по результату флажки, но нигде не загружает результат. Наряду с командами однобитных (статических) сдвигов имеются команды многобитных (динамических) сдвигов, в которых константа сдвига находится в регистре CL.

Команды передачи управления (переходы, вызовы, возвраты) имеют две разновидности: внутрисегментные («близкие»), целевой адрес которых, загружаемый в указатель команд IP, находится в текущем сегменте кода, и междусегментные («далекие»), целевой адрес которых определяет новые значения регистров CS и IP. Передачи управления могут быть прямыми (целевой адрес находится в команде) и косвенными (целевой адрес вычисляется с использованием стандартных режимов адресации). В 16 командах условных переходов проверяются отношения знаковых и беззнаковых чисел.

Имеются 4 команды управления циклами, которые рассчитаны на передачу числа повторений цикла в регистре CX (счетчик).

Команды обработки цепочек данных. Команды этой группы манипулируют цепочками данных, т. е. последовательностями байт или слов в памяти. Время выполнения таких операций, как передача цепочки, проверка и сканирование, значительно сокращается по сравнению с их программной реализацией.

Микропроцессор имеет двухуровневую векторную приоритетную систему прерываний с двумя линиями запросов прерываний. Вход NMI немаскируемого прерывания имеет больший приоритет, чем вход INTR маскируемых прерываний. Прерывания воспринимаются между командами, причем запросы INTR воспринимаются при условии, что $IF=1$. Реагируя на прерывание, микропроцессор автоматически включает в стек содержимое регистров FLAGS, CS и IP. После этого сбрасываются биты IF и TF, а в регистры CS и IP загружаются новые значения векторов прерываний, что инициирует выполнение соответствующей подпрограммы обслуживания прерывания. Таблица векторов прерываний занимает первые 1024 байта физической памяти. Каждый вектор представлен 4 байтами и содержит новые значения, загружаемые в регистры CS и IP.

В мультипроцессорных системах с разделенными ресурсами необходим механизм управляемого доступа к ресурсам. Обычно такой механизм реализуется операционной системой, но требуются и некоторые аппаратные средства. Одно из простых средств, реализованных в микропроцессоре 8086, заключается в заблокированном обмене. Специальный однобайтный префикс LOCK, который может предшествовать любой команде, вызывает формирование сигнала LOCK, блокирующего доступ к шине на время выполнения команды от других компонентов системы. Примером программной блокировки служит следующий фрагмент:

Метка	Код	Операнд	Комментарий
CHECK: LOCK	MOV	AL, 1	; Установка блокировки
	XCHG	SEMA, AL	; Обмен семафора и AL
	TEST	AL, AL	; Установка флажков по AL
	JNZ	TEST	; Зацикливание до освобождения
	***		; Критический сегмент программы

	MOV	SEMA, 0	; Освобождение ресурса

3-6. СЕКЦИОННЫЕ МП

Одной из движущих сил развития МП является совершенствование технологической базы. Стремление улучшить быстродействие МП за счет биполярной технологии: транзисторно-транзисторной логики с диодами Шоттки (ТТЛШ), интегрально-инжекционной логики (И²Л), эмиттерно-связанной логики (ЭСЛ) — привело к появлению в 1974 г. первого секционного МП на основе ТТЛШ. Выполнение МП на многокристальной основе связано с внутренней функциональной сложностью МП и ограничениями на плотность компонентов на одном кристалле, с числом выводов корпуса, допустимым размером кристалла для биполярной технологии, с рассеиваемой мощностью.

Функционально секционные МП разбиты на ряд модулей, конструктивно реализованных в виде отдельных кристаллов БИС, типичными представителями которых являются: арифметико-логическое устройство и регистры общего назначения с возможностью наращивания длины слова с кратностью, соответствующей разрядности БИС; контроллер последовательности микрокоманд (КПМК); память микрокоманд (ПМК).

Контроллер последовательности микрокоманд и память микрокоманд образуют микропрограммное устройство управления. По сравнению с однокристалльными МП секционные МП позволяют разрабатывать более быстродействующие ЭВМ не только за счет биполярной технологии, но и за счет гибкой адаптации структуры, длины слова, специального набора команд проектируемого процессора к классу решаемых задач. Кроме того, иногда только за счет микропрограммирования критичных по времени функций или алгоритма решаемой задачи можно повысить быстродействие в несколько раз по сравнению с его реализацией на командном уровне. Перечисленные достоинства не даются даром, так как, например разработка собственной системы команд требует также разработки программного обеспечения для моделирования, редактирования и загрузки микропрограмм, интерпретирующих систему команд. Кроме того, сам процесс микропрограммирования даже при использовании микроассемблера является более трудоемким из-за более сложных форматов микрокоманд по сравнению с командами, необходимости учета временных диаграмм микрокомандного цикла и изучения других деталей на аппаратном уровне. Однако, несмотря на отмеченные трудности, секционные биполярные МП с микропрограммированием представляют мощное средство для построения микро- и мини-ЭВМ средней и высокой производительности, замены быстродействующих контроллеров, ранее реализованных на основе схем с жесткой логикой, создания спецпроцессоров для обработки сигналов (быстрое преобразование Фурье, цифровая фильтрация), процессоров систем передачи данных и других применений.

Структурная схема типичного секционного МП представлена на рис. 3-17. МП состоит из двух функциональных модулей: микропрограммного устройства управления (МПУУ) и операционного устройства (ОУ), построенного из отдельных секций. МПУУ включает в себя память микрокоманд (ПМК) и контроллер последовательности микрокоманд (КПМК), основным назначением которого является реализация управляющих структур (фрагментов), встречающихся в микропрограммах: линейной последовательности, структуры вида «если p , то X , иначе Y » и структуры вида «пока p , делай X ». Для этого контроллер обеспечивает дешифрацию кода операции команды для обращения к первой микрокоманде микропрограммы, интерпретирующей данную команду, формирует адреса следующих микрокоманд: как линейной последовательности микрокоманд в

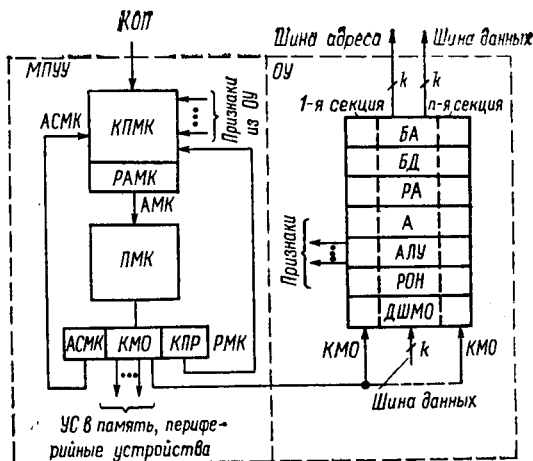


Рис. 3-17. Структурная схема секционного МП

микропрограмме, так и безусловных и условных переходов, обращения к микроподпрограммам. Кроме того, некоторые контроллеры могут хранить признаки переходов, управлять прерываниями на микропрограммном уровне. Как правило, в микропроцессорный комплект входят БИС контроллеров последовательности микрокоманд.

Память микрокоманд предназначена для хранения микрокоманд, ее емкость и разрядность однозначно определяются набором реализуемых микропрограмм. Путем изменения набора микропрограмм можно гибко ориентировать функциональную направленность МП.

Микропрограммное устройство управления функционирует следующим образом. КОП с регистра команд поступает на вход КПМК, и на выходе регистра адреса микрокоманд (РАМК) контроллера формируется адрес первой микрокоманды выполняемой микропрограммы. Микрокоманда, подлежащая реализации в текущем микрокомандном цикле, считывается из памяти на регистр микрокоманд (РМК). Микрокоманда содержит следующие три основных поля: 1) поле кода микрооперации (КМО), определяющее микрооперации, выполняемые в одном из устройств микроЭВМ (АЛУ, память, периферийные устройства); 2) поле, в котором закодированы признаки (КПР), поступающие из операционного устройства в контроллер и анализируемые контроллером при выполнении микрокоманд условного пе-

рехода по данным признакам; 3) поле, в котором содержится код адреса для формирования адреса следующей микрокоманды (АСМК).

После выполнения считанной микрокоманды микрокомандный цикл повторяется.

Обрабатывающая часть МП, или операционное устройство (ОУ), предназначено для выполнения всех арифметических и логических операций. ОУ собирается из секций процессорных элементов (ПЭ), каждый из которых содержит АЛУ, РОН, аккумулятор А. Кроме того, в процессорный элемент включается дешифратор микрооперации (ДШМО) с целью сокращения числа выводов корпуса и более простого объединения секций. Большинство ПЭ являются 2- и 4-битными, но есть и 8-битные секции. ПЭ работают параллельно, и длина слов может произвольно наращиваться. Одной из особенностей в организации ОУ секционного типа является их вертикальное разбиение вместо традиционного горизонтального. Вертикальное разбиение требует меньшего числа передач кодов между отдельными БИС при обработке данных, меньше микрокоманд, связей. Тип реализуемой микрооперации и регистры ПЭ, участвующие в ее выполнении в качестве источников операндов и приемников результата, указываются в коде микрокоманды. Объединение секций в ОУ иллюстрирует рис. 3-18. Последовательно объединяются входы и выходы цепей переносов, левого и правого сдвигов отдельных секций. Шины данных, адреса, кода микрооперации объединяются параллельно в общую магистраль. По шине данных в секцию передаются операнды из памяти и периферийных устройств ввода информации. Некоторые секции имеют отдельные шины данных для этих устройств, что упрощает задачу построения интерфейса. Из секций результаты выдаются по выходной шине данных, шина адреса используется

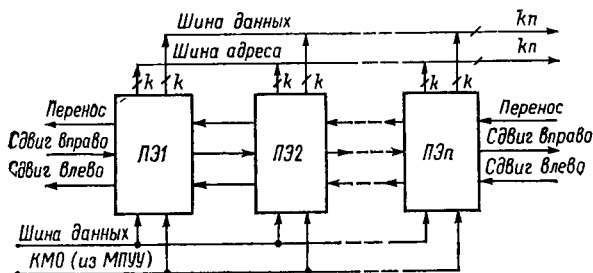


Рис. 3-18. Организация ОУ секционного типа

для выдачи адреса в память и периферийные устройства. Кроме представленных на рис. 3-17 блоков, для построения МП необходим регистр команд, программный счетчик и другие узлы. В зависимости от требований к быстродействию МП разработчик использует в качестве этих регистров РОН или вводит внешние дополнительные регистры. Кроме того, при организации ОУ решается задача выбора типа переноса (последовательный или ускоренный), если, конечно, из секции имеются выходы ускоренного переноса и в комплект входит БИС ускоренного переноса. Таким образом, при использовании МП секционного типа перед разработчиком микро-ЭВМ стоит сложная задача, связанная с выбором структуры, набора команд, разрядности МП, организацией операционного и управляющего устройств, разработкой микропрограмм, т. е. задача построения функционально и конструктивно законченного МП, который обычно содержит до 50—70 интегральных схем различной степени интеграции, и, по-видимому, точнее называть его одноплатным процессором.

Основными достоинствами МП секционного типа являются:

1. Разработка микро-ЭВМ с максимальным соответствием ее структуры характеру решаемых задач.
2. Выбор произвольной нестандартной разрядности МП.
3. Наличие отдельных независимых адресных шин, входных и выходных шин данных, позволяющее проще организовать сопряжение МП с памятью и периферийными устройствами без использования мультиплексирования.

3-7. ПРОЦЕССОРЫ МИНИ-ЭВМ СМ-3 И СМ-4

Мини-ЭВМ СМ-3 и СМ-4 являются дальнейшим развитием мини-ЭВМ М-400, которая была первой в семействе совместимых мини-ЭВМ СМ-3(4), «Электроника-100/25», «Электроника-100/79», микро-ЭВМ «Электроника-60». Одной из причин успеха малых ЭВМ данного семейства является то, что они внесли многие особенности в организацию современных мини- и микро-ЭВМ: использование интерфейса «общая шина», в котором подключение памяти, периферийных устройств и процедура обмена данными между процессором и этими блоками являются унифицированными. В «Электронике-60» используется Q-шина, которая отличается от общей шины логически, механически и электрически, но с точки зрения программиста является идентичной.

Содержимое регистров периферийных устройств может обрабатываться так же гибко и тем же набором команд, что и содержимое ячеек памяти. Кроме того, характерной особенностью ЭВМ данного семейства является большое разнообразие способов адресации, что позволяет гибко обрабатывать разнообразные структуры данных, и разнообразие форматов команд (безадресные, одноадресные и двухадресные). Например, в данных ЭВМ впервые были введены автоинкрементный и автодекрементный режимы адресации, относительная адресация команд и др. В данных ЭВМ для вызова подпрограмм и обработки прерываний введен указатель стека, использована многоуровневая система приоритетных прерываний. Одним словом, эти малые ЭВМ содержат много особенностей современных мини- и микро-ЭВМ, изучение которых является важным.

Базовая структура ЭВМ СМ-3 и СМ-4 приведена на рис. 3-19. Процессор связан с памятью и периферийными устройствами по «общей шине», которая содержит 16-битную шину данных, 18-битную шину адреса, шину управления. Адрес определяет один из 64К байт, который является минимальным адресуемым элементом. Команды могут выполнять обработку как отдельных байтов, так и двухбайтных слов. Память для программиста можно представить последовательностью байт с адресами 0—157 777₈ или последовательностью двухбайтных слов с адресами 0—157 776₈. Адреса слов всегда являются четными, младшему байту соответствует четный адрес, а старшему — нечетный. Старшие 8К байт адресного пространства памяти (160 000—177 777₈) зарезервированы для интерфейсов периферийных устройств.

Управление общей шиной разрешается не только процессору, но и некоторым другим периферийным устройствам. Устройство, управляющее шиной, называется ведущим, а устройство, с которым связано ведущее устройство, — ведомым. Всем устройствам, которые могут функционировать как ведущие, присвоены приоритеты. Связь между ведущим и ведомым устройствами осуществляется в асинхронном режиме с квитированием (подтверждением). Быстрота ответа определяется реакцией участвующих в обмене устройств, а не системным генератором синхронизации. Такой асинхронный режим связи дает возможность пользователю применять в ЭВМ процессор, память и периферийные устройства с различным быстродействием.

В ЭВМ СМ-4 имеется устройство расширения памяти (УРП), или диспетчер памяти, который выполняет функ-

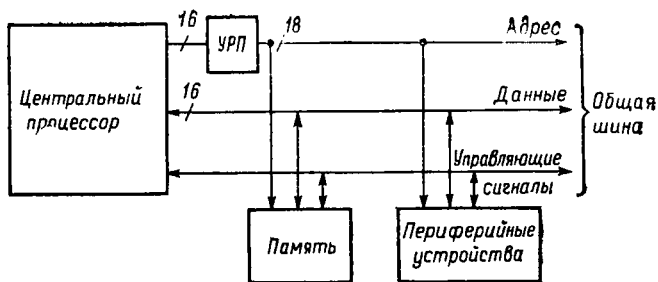


Рис. 3-19. Базовая структура ЭВМ СМ-3 и СМ-4

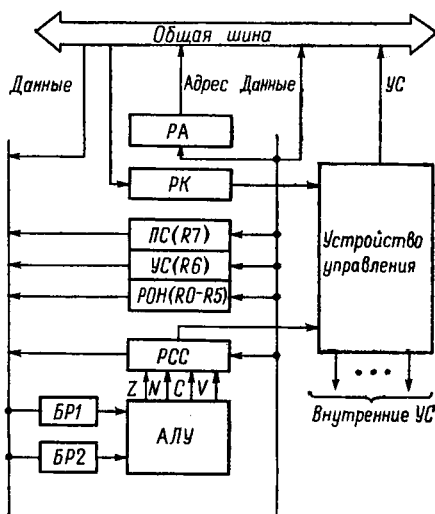


Рис. 3-20. Структура базового процессора СМ-3 и СМ-4

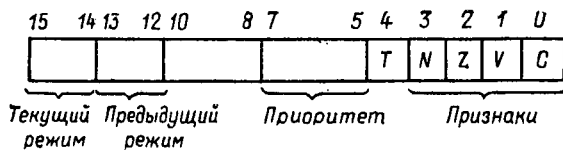


Рис. 3-21. Слово состояния процессора

цию преобразования 16-битного виртуального адреса, определяемого командой, в 18-битный физический адрес, выдаваемый на шины адреса. Это позволяет увеличить объем программной памяти до 124К слов.

Упрощенная структура базового процессора ЭВМ СМ-3 (4) приведена на рис. 3-20. Процессор подключается к общей шине как подсистема, управляет распределением ее между памятью и периферийными устройствами, выполняет логические, арифметические и другие операции. Процессор содержит набор быстродействующих РОН, которые могут быть использованы как аккумуляторы, указатели стека, индексные регистры и выполнять другие функции.

Функцию программного счетчика выполняет регистр R7, который содержит адрес следующей команды и используется только для этой цели, а не как аккумулятор для арифметических операций. Регистр R6 используется в качестве указателя стека, который в СМ-3(4) заполняется, начиная со старшего адреса, зарезервированного под стек. Стек используется автоматически при вызове подпрограмм, обработке прерываний. Когда процессор прерывается, слово состояния процессора из регистра слова состояния (РСС) и содержимое ПС запоминаются автоматически в стеке. Новое слово состояния программы загружается из области памяти, которая зарезервирована для обработки прерываний. Назначение отдельных полей 16-битного слова состояния процессора приведено на рис. 3-21. В разрядах 0—3 формируются признаки результатов C, V, Z, N. Признак «слежения» T может быть установлен или сброшен программно. Если данный бит установлен, будет иметь место внутреннее прерывание (через ячейку памяти с адресом 14) после завершения каждой команды и будет загружаться новое слово состояния. Признак T является особенно полезным при отладке программ, так как позволяет эффективно устанавливать в программе контрольные точки.

Биты 5—7 позволяют назначать процессору любой из восьми приоритетов, причем уровень 7 запрещает прерывать текущую программу периферийным устройством, только приоритет внепроцессорного запроса выше любого приоритета процессора.

Биты 12—15 используются только в СМ-4. Биты 14 и 15 определяют режим работы процессора: системный или режим пользователя, а биты 12, 13 — предыдущий режим.

Форматы команд и режимы адресации. Основные форматы базовой системы команд процессоров СМ-3 и СМ-4 приведены на рис. 3-22. Поля источника и приемника оп-

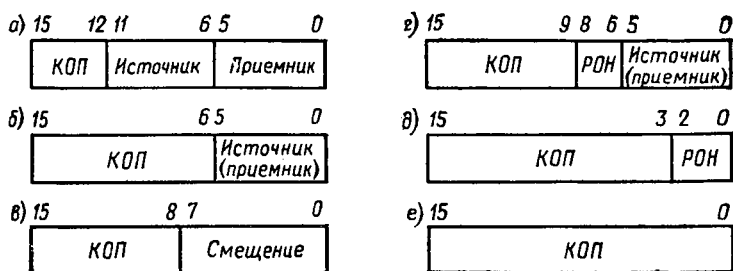


Рис. 3-22. Основные форматы команд СМ-3 и СМ-4: *а* — двухоперандная; *б* — однооперандная; *в* — разветвление; *г* — вызов подпрограммы; *д* — возврат из подпрограммы; *е* — безоперандная

ределяют операнды, используемые режимы адресации; команды с полями источника и приемника могут определять режим адресации без ограничений. Кроме того, почти каждая команда, оперирующая словами, может также оперировать байтами, что определяется старшим битом в команде, который должен быть установлен в 1 для указания команды обработки байта. Как видно из рис. 3-22, все базовые команды — длиной в одно слово. При использовании некоторых режимов адресации к команде добавляется одно слово, поэтому команды с полями «источник» и «приемник» могут содержать до двух слов адресной информации, а длина команды может варьироваться от одного до трех слов.

Базовые режимы адресации. Однооперандные и двухоперандные команды могут содержать операнды в памяти; операнд в команде определяется 6-битным полем, 3 бита которого указывают номер РОН, а другие 3 — один из 8 базовых режимов адресации:

1. Регистровая адресация. При данном режиме адресации (код РА=000) РОН может быть использован как приемник и источник информации. Например, команда ADD R2, R4 (код ее 060204) обеспечивает сложение содержимого R2 и R4 с записью результата в R4:

$$(R2) + (R4) \rightarrow (R4).$$

2. Косвенная регистровая адресация (код РА=001). Содержимое регистра в этом случае является адресом ячейки памяти. Команда $\bar{C}LR @ R3$ (005015) обеспечивает сброс ячейки памяти, адрес которой хранится в регистре R3.

3. Автоинкрементная адресация. Данный режим адресации (код $PA=010$) автоматически увеличивает содержимое регистра, выполняющего адресацию памяти, на 1 (при выборке байта) или на 2 (при выборке слова). Это обеспечивает последовательную адресацию ячеек памяти, что требуется при обработке массивов данных, организации стека и в других случаях. Например, при выполнении команды $ADD(R1)+, R2$ (062104) содержимое $R1$ используется для адресации операнда, который прибавляется к содержимому $R2$. После выборки операнда ($R1$) увеличивается на 2.

4. Косвенная автоинкрементная адресация ($PA=011$). Содержимое указанного в команде регистра используется как адрес адреса операнда. Например, команда $INC@(R2)+(005232)$ обеспечивает увеличение на 1 операнда, адрес которого указан в ячейке памяти, адресуемой регистром $R2$. С выполнением команды ($R2$) вырастет на 2.

5. Автодекрементная адресация (код $PA=100$). При использовании данного режима адресации, например при выполнении команды $DEC-(R1)$ (005241), содержимое $R1$ вначале уменьшается на 2, а затем используется как адрес для выборки операнда, который уменьшается на 1.

6. Косвенная автодекрементная адресация ($PA=101$). Содержимое регистра уменьшается на 2, а затем используется как адрес адреса операнда. Например, команда $SOM, @-(R0)$ (005150) обеспечивает вычитание 2 из $R0$, и затем содержимое $R0$ становится адресом адреса числа, которое инвертируется под действием данной команды.

7. Индексная адресация ($PA=110$). При данном способе адресации содержимое указанного в команде регистра суммируется с индексным словом, которое следует за первым словом команды. При этом содержимое регистра и индексное слово не изменяются. Например, команда $CLR 300(R4)$ выполняет очистку содержимого ячейки, адрес которой определяется путем сложения индексного слова 300 и содержимого $R4$.

8. Косвенная индексная адресация ($PA=111$). Для этого режима адресации сформированный код используется как адрес адреса операнда. Например, команда $ADD@1000(R2)$, $R1$ выполняет сложение операнда из $R1$ с операндом, адрес адреса которого формируется при сложении индекса 1000 и содержимого $R2$; результат запоминается в $R1$.

В режимах адресации 3—6 содержимое регистра увеличивается или уменьшается на 1 или на 2. Следовательно, их можно использовать для прохождения вперед или назад по таблицам байтов или слов без дополнительных команд на модификацию указателей адреса. Автоинкрементные и автодекрементные режимы позволяют также использовать любой регистр как указатель стека со стандартными характеристиками: УС указывает на верхний элемент стека.

При использовании R7, выполняющего функцию ПС, с некоторыми базовыми режимами адресации возможны четыре дополнительных режима: непосредственной, абсолютной, относительной и косвенной адресации:

9. Непосредственная адресация (ПА=010). Данная адресация имеет место при использовании автоинкрементной адресации для R7. Например, рассмотрим выполнение команды $ADD\#10, R0$. Непосредственный операнд 10 размещается за первым словом команды и суммируется с содержимым R0. После выборки этой команды содержимое R7 увеличивается на 2, что обеспечивает выборку операнда 10 по адресу из регистра R7, и затем R7 увеличивается на 2 для выборки следующей команды.

10. Абсолютная адресация (ПА=011). Этот режим ставляет собой фактически режим косвенной непосредственной адресации, когда слово, следующее за первым словом команды, является абсолютным адресом операнда. Например, команда $CLR\#1000$ выполняет очистку ячеек памяти с адресом 1000.

11. Относительная адресация (ПА=110). При сложении индекса с содержимым R7 формируется исполнительный адрес по отношению к ПС R7. Этот режим адресации очень эффективен для построения позиционно независимых программ, допускающих перемещение в памяти. Когда команды программы перемещаются, операнд передвигается на то же самое число ячеек. Например, команда $INC\ A$ (005267) обеспечивает увеличение содержимого ячейки A на 1. Адрес ячейки A определяется путем суммирования слова, следующего за командой, и содержимого ПС R7.

12. Относительная косвенная адресация (ПА=111). При использовании данного режима адресации, например при выполнении команды $CLR\@A$, слово, следующее за первым словом команды, складывается с содержимым ПС RC для формирования косвенного адреса ячейки памяти.

Система команд. В мини-ЭВМ СМ-3(4), микро-ЭВМ «Электроника-60» команды классифицированы на следую-

Двухоперандные команды СМ-3(4)

Мнемоника	Операнд	Описание
MOV (B)	src, dst	Операнд источника src пересылается по адресу операнда приемника dst
ADD	src, dst	Операнд источника src прибавляется к операнду приемника dst, и результат записывается в dst
SUB	src, dst	Из операнда приемника dst вычитается операнд источника src, и результат записывается по адресу dst
CMP (B)	src, dst	Сравнение операндов источника и приемника и установка признаков N, Z, V, C
BIS (B)	src, dst	Логическая операция ИЛИ содержимого src и dst, результат записывается по адресу dst
BIC (B)	src, dst	Сброс разрядов операнда src, соответствующих установленному разряду операнда dst
BIT (B)	src, dst	Установка признаков N, Z в соответствие с результатом логической операции И над содержимым src и dst, значение операндов не изменяется
XOR	red, dst	Логическая операция «исключающее ИЛИ» над содержимым регистра и dst, результат записывается в dst

Примечание. Формат команд см. на рис. 3-22, а.

щие типы: двухоперандные, однооперандные, управления программой и дополнительные.

Двухоперандные и однооперандные команды приведены в табл. 3-3 и 3-4 соответственно. Почти все операции в соответствии с данными командами можно выполнять над словами и байтами, что указывается символом В, например MOV передает 16-битное слово, а MOVВ — байт. В зависимости от режима адресации один или оба операнда для команды MOV(B), которая наиболее часто встречается в программах СМ ЭВМ, могут храниться в памяти. Поэтому MOV выполняет функции команд загрузки LD и запоминания ST, а также PUSH и POP с автодекрементным и автоинкрементным режимами адресации. Эта команда допускает также передачи типа память — память без промежуточного запоминания данных в регистрах.

Среди группы команд управления программой наиболее широко используются условные и безусловные переходы, составляющие 20 % команд. Если переход выполняется, то смещение умножается на 2 и результат прибавляется к со-

Таблица 3-4

Однооперандные команды СМ-3(4)

Мнемоника	Операнд	Описание
CLR (B)	dst	Сброс приемника
COM (B)	dst	Инверсия приемника
INC (B)	dst	Инкремент приемника на 1
DEC (B)	dst	Декремент приемника на 1
NEG (B)	dst	Содержимое приемника заменяется дополнительным кодом
TST (B)	dst	Установка признаков N, V, Z, C по содержимому источника
ASR (B)	dst	Арифметический сдвиг приемника вправо
ASL (B)	dst	Арифметический сдвиг приемника влево
ROR (B)	dst	Циклический сдвиг приемника вправо с признаком C
ROL (B)	dst	Циклический сдвиг приемника влево с признаком C
ADC (B)	dst	К содержимому приемника прибавляется C
SBC (B)	dst	Из содержимого приемника вычитается C
SWAB	dst	Обмен байтами в слове приемника
SXT	dst	Расширение знака N в слове источника

Примечание. Формат команд см. на рис. 3-22, б.

Таблица 3-5

Команды управления программой СМ-3(4)

Мнемоника	Операнд	Формат на рис. 3-22	Описание
BR	rels	в	Безусловное разветвление
Bcc	rels	в	Условное разветвление
JMP	dst	г	Безусловный переход по исполнительному адресу, указанному в источнике
JSR	reg, dst	г	Переход к подпрограмме
RTS	reg	д	Возврат из подпрограммы
MARK	nn	б	Восстановление указателя стека
SOB	reg, nn	г	Вычитание 1 и разветвление, если результат не равен 0
EMT	code	в	Командное прерывание для системных программ
TRAP	code	в	Командное прерывание
BPT		е	Командное прерывание для отладки
RTI		е	Возврат из прерывания
IOT		е	Командное прерывание для ввода-вывода
RTT		е	Возврат из прерывания, запрещающий прерывание по разряду слежения ССП

Таблица 3-6

Дополнительные команды СМ-3(4)

Мнемоника	Описание
NOP	Нет операции
CL (C, V, N, Z)	Сброс битов слова состояния
SE (C, V, N, Z)	Установка битов слова состояния
RESET	Сброс содержимого внешних устройств
HALT	Останов процессора
WAIT	Ожидание прерывания

Примечание. Формат команд см. на рис. 3-22, е.

держимому ПС. Когда выполняется сложение, ПС уже хранит адрес, увеличенный по отношению к выбранной команде, поэтому смещение —1, а не 0 вызывает переход к самой команде перехода. В табл. 3-5 приведены команды управления программой. Команды JMP и JSR переходят по исполнительному адресу, определенному в поле источника (регистровый режим не допускается).

Дополнительные команды иллюстрируются табл. 3-6.

При описании системы команд СМ-3(4), приведенных в табл. 3-3—3-6, используются следующие обозначения: В — указание на передачу байта, например команда MOV передает 16-битное слово, а команда MOVB — байт; src — источник, определяемый 6-битной спецификацией адреса; dst — приемник, определяемый 6-битной спецификацией адреса; reg — 3-битный адрес регистра; rels — целевой адрес команды передачи управления; cc — одно из условий; pp — 6-битное беззнаковое целое; code — произвольный 8-битный код.

3-8. УСТРОЙСТВО УПРАВЛЕНИЯ

Устройство управления (УУ) формирует распределенную во времени и в пространстве последовательность внешних и внутренних УС, обеспечивающих выборку и выполнение команд. На этапе цикла выборки команды УУ интерпретирует КОП команды, выбранной из программной памяти. На этапе выполнения команды в соответствии с

типом реализуемой операции УУ формирует требуемый набор микроопераций, таких, как передача кодов, сдвиг кодов, анализ признаков, запоминание результатов и др. Порядок выполнения микрокоманд определяется микропрограммой реализации команды и может изменяться в зависимости от признаков операций, вырабатываемых в АЛУ и являющихся входными сигналами УУ. Одной из важнейших характеристик, определяющих гибкость УУ, является способность изменения последовательности УС. По этому критерию УУ подразделяют на УУ с «жесткой» логикой, или специализированные УУ, и на универсальные, или микропрограммные, УУ. Данные варианты организации УУ, кроме гибкости их к изменению последовательности микрокоманд, возможности замены микропрограмм и тем самым системы команд, отличает принцип построения, способ хранения микропрограмм, аппаратные затраты, время реализации микропрограмм. Специализированные УУ (рис. 3-23) формируют неизменную последовательность УС и включают в свой состав запоминающие и комбинационные схемы, выполняющие функции запоминания текущего состояния, определяющего совокупность УС, и формирования следующего состояния в соответствии с входными признаками. Микропрограмма в таком автомате хранится за счет системы связей между элементами УУ, основой для построения которого являются логические схемы, счетчики, регистры, дешифраторы. Специализированные УУ не нашли широкого распространения, используются в основном микропрограммные УУ, структурная схема которых приведена на рис. 3-24. Микропрограмма хранится в памяти микрокоманд

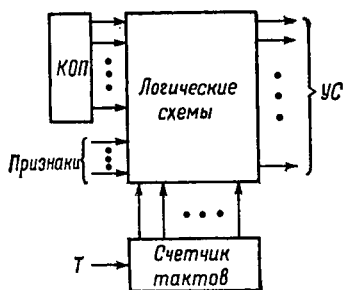


Рис. 3-23. Специализированное УУ

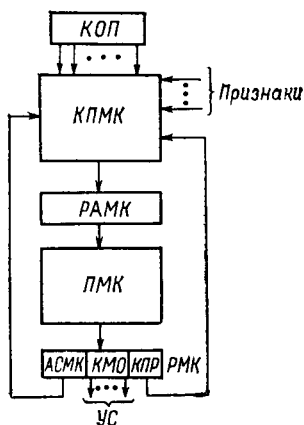


Рис. 3-24. Микропрограммное УУ

ПМК, адрес микрокоманды формируется контроллером последовательности микрокоманд КПМК, микрокоманды считываются на регистр микрокоманд РМК. Микрокоманда содержит три основных поля: код микрооперации КМО, адрес следующей микрокоманды АСМК, поле кода признаков КПР, в котором указывается, какой признак разветвления в микропрограмме необходимо анализировать КПМК. Адрес первой микрокоманды определяет КОП, АСМК может указываться в микрокоманде явным образом или формироваться естественным путем, как это имеет место при выборке команд. После выполнения выбранной микрокоманды микрокомандный цикл повторяется.

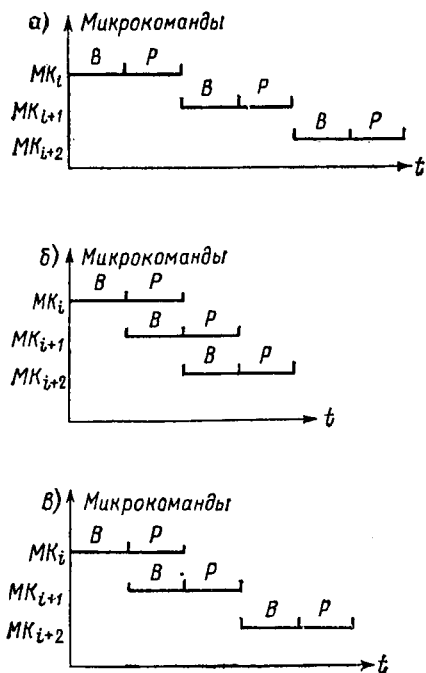
Основными вопросами при проектировании микропрограммных устройств управления, которые приходится решать с целью достижения оптимальных параметров УУ, являются следующие: определение совместимости во времени этапов выборки и выполнения микрокоманд; анализ способа формирования адреса следующей микрокоманды; выбор способа кодирования микрокоманд; выбор типа синхронизации при формировании микроопераций.

Выборка и выполнение микрокоманд. С точки зрения совместимости выборки и выполнения микрокоманд можно выделить следующие способы: последовательный, параллельный и последовательно-параллельный.

При последовательном способе (рис. 3-25, а) выборка следующей микрокоманды $МК_{i+1}$ не инициируется до момента завершения предыдущей $МК_i$. Достоинством данного подхода является простота организации микрокомандного цикла.

Рис. 3-25. Выборка и реализация микрокоманды: последовательная (а), параллельная (б), последовательно-параллельная (в)

В — выборка; Р — реализация



Во втором варианте (рис. 3-25, б) имеет место совмещение этапов выборки $МК_{i+1}$ и выполнения $МК_i$. В этом случае при одинаковом времени выборки и выполнения микрокоманд достигается фактическое сокращение микрокомандного цикла в два раза. Однако при выполнении микрокоманд условной передачи управления адрес следующей микрокоманды зависит от результата выполнения предыдущей, и в этом случае используют последовательно-параллельный способ выборки и выполнения микрокоманд (рис. 3-25, в). Здесь выборка микрокоманды условного перехода $МК_{i+2}$ начинается только после завершения выполнения предыдущей микрокоманды $МК_{i+1}$. Однако иногда с целью сокращения времени реализации микрокомандного цикла для третьего способа направление перехода выбирается вероятностно, и если оно является корректным, то потери быстродействия не будет, иначе цикл выборки повторяется.

Способы формирования адреса следующей микрокоманды. Способ адресации микрокоманд определяет правило формирования адреса следующей микрокоманды. В ЭВМ используются два основных способа адресации: принудительная и естественная. Принудительная адресация сводится к указанию в каждой микрокоманде адреса следующей

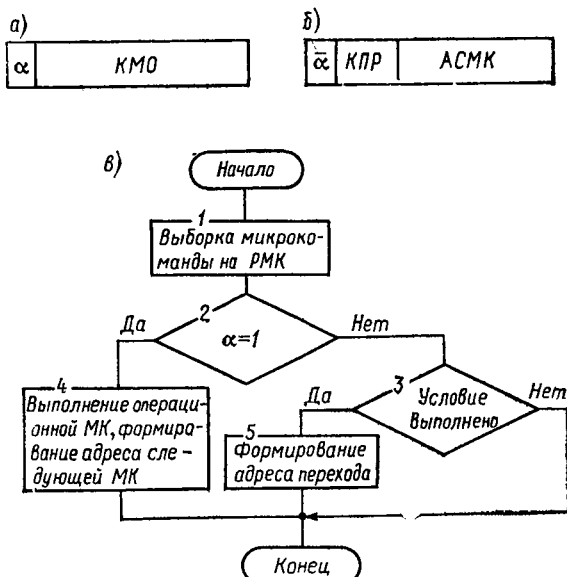


Рис. 3-26. Форматы микрокоманды: а — операционная МК; б — управляющая МК; в — микрокомандный цикл при естественной адресации

за нею микрокоманды, а при естественной адресации адрес следующей микрокоманды образуется путем приращения адреса предыдущей, как это имеет место при формировании адреса следующей команды. Это позволяет за счет исключения поля адреса из операционных микрокоманд уменьшить разрядность памяти. Для выполнения условных и безусловных переходов в микропрограмме используются управляющие микрокоманды, содержащие адрес перехода и поле признаков. На рис. 3-26, а и б приведены форматы операционной и управляющей микрокоманд, а рис. 3-26, в иллюстрирует выполнение микрокомандного цикла при естественной адресации; признак α определяет тип микрокоманды: $\alpha=1$ — операционная микрокоманда, $\alpha=0$ — управляющая.

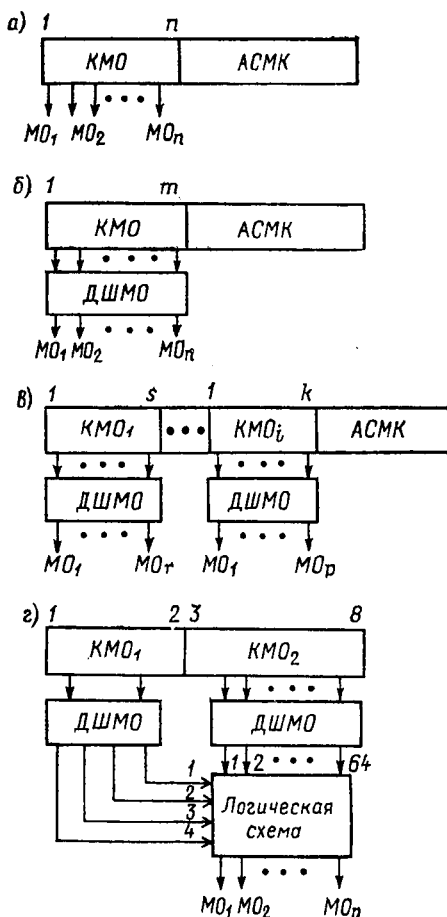
Кодирование микрокоманд. Выбор способа кодирования микрокоманд определяется системой команд, структурой ЭВМ, требованиями к быстродействию и стоимости и представляет собой очень сложную задачу. Ниже рассматриваются основные способы кодирования микрокоманд.

Горизонтальное кодирование. Простейшим вариантом кодирования микрокоманд является горизонтальное кодирование, при котором каждый разряд поля кода микрооперации однозначно определяет управляющий сигнал для выполнения микрооперации (рис. 3-27, а). Из-за большого набора микро-

операций однозначно определяет управляющий сигнал для выполнения микрооперации (рис. 3-27, а). Из-за большого набора микро-

Рис. 3-27. Кодирование МК:

а — горизонтальное; б — вертикальное; в — смешанное; г — косвенное



операций (от нескольких десятков до нескольких сотен) горизонтальное кодирование может потребовать большой разрядности памяти микрокоманд, что и является основным его недостатком. Достоинством горизонтального кодирования является возможность параллельной работы нескольких операционных устройств ЭВМ, что позволяет существенно повысить быстродействие и приводит к высокой степени загрузки оборудования.

Вертикальное кодирование. Другим, противоположным, подходом к кодированию микрокоманд с целью максимального сокращения разрядности полей микрокоманды является вертикальное кодирование (рис. 3-27, б). При вертикальном кодировании дополнительно требуется дешифратор микроопераций, а кроме того, увеличивается время выполнения микрокоманды за счет временных задержек в дешифраторе и отсутствует возможность одновременного формирования нескольких микроопераций.

Смешанное кодирование. Развитием способов кодирования микрокоманд с целью устранения основных недостатков, присущих горизонтальному и вертикальному способам, является горизонтально-вертикальное, или смешанное, кодирование микрокоманд (рис. 3-27, в). При данном варианте кодирования в отдельных полях кода микроопераций объединяют взаимоисключающие наборы микроопераций для обеспечения их параллельного выполнения, как это имеет место при чисто горизонтальном кодировании. Дешифраторы, декодирующие код микроопераций отдельных полей, образуют уровень схем дешифрации микроопераций, реализуемых в каждом поле в течение одного микрокомандного цикла. Данный способ кодирования находит широкое применение в микропрограммных УУ.

Косвенное кодирование. В рассмотренных выше вариантах кодирования микрокоманд каждое поле КМО формирует определенные фиксированные управляющие сигналы и интерпретируется всегда одинаковым образом. Однако в некоторых ЭВМ с целью дальнейшего сокращения разрядности микрокоманды одно и то же поле может быть использовано для формирования УС для различных блоков и его функции в этом случае определяются другим полем. Пример кодирования микрокоманд с использованием данного способа приведен в табл. 3-7.

Двухбитное поле КМО₁ кодирует одну из четырех групп микроопераций, а шестибитное поле КМО₂ определяет реализуемую в данной группе микрооперацию. Переменные форматы сокращают разрядность микрокоманды, но требу-

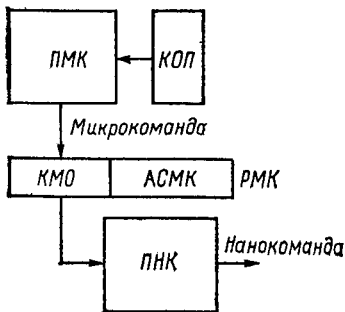
Таблица 3-7

Косвенное кодирование микрокоманды

КМО ₁		КМО ₂						Назначение полей МК
1	2	3	4	5	6	7	8	
0	0	X	X	X	X	X	X	МО АЛУ
0	1	Y	Y	Y	Y	Y	Y	МО памяти, периферийных устройств
1	0	Z	Z	Z	Z	Z	Z	МО безусловных и условных переходов
1	1	K	K	K	K	K	K	Константы для загрузки регистров, счетчиков

ют введения дополнительных дешифраторов, логических схем и приводят к временным задержкам из-за коммутации полей микрокоманды (рис. 3-27, з).

Двухуровневое кодирование. В современных микро- и мини-ЭВМ широко используется двухуровневая структура микрокоманд для интерпретации команды. Например, в МП 68 000 фирмы Motorola первый уровень микрокоманд использует вертикальное кодирование микрокоманд, выбирающих широкие горизонтальные микрокоманды второго уровня, называемые нанокомандами. Основой для использования данного принципа кодирования является то, что в случае, например, горизонтального кодирования часто для реализации микропрограмм используется небольшая часть комбинаций параллельно выполняемых микроопераций из максимального возможного их числа. Поэтому за счет хранения данных комбинаций в отдельной памяти нанокоманд (ПНК) небольшой емкости и разрядности, требуемой для горизонтального кодирования микрокоманды, можно достигнуть значительного сокращения общей емкости памяти с максимальным параллелизмом при выполнении микроопераций. На рис. 3-28 представлена структурная схема УУ с двухуровневым кодированием микрокоманд, в котором на



достигнуть значительного сокращения общей емкости памяти с максимальным параллелизмом при выполнении микроопераций. На рис. 3-28 представлена структурная схема УУ с двухуровневым кодированием микрокоманд, в котором на

Рис. 3-28. УУ с двухуровневым кодированием МК

первом уровне используется вертикальное кодирование, а на втором — горизонтальное. Выполнение микрокоманд осуществляется следующим образом. Первая микрокоманда с вертикальным кодированием извлекается из памяти микрокоманд ПМК, поле кода микрооперации КМО которой является адресом нанокоманды. Выбранная нанокоманда и определяет множество микроопераций, реализуемых в текущем микрокомандном цикле. Например, память для хранения микропрограмм объемом 4К 32-битных микрокоманд при числе нанокоманд 256 может быть реализована как совокупность памяти микрокоманд емкостью 4К 8-битных слов и памяти нанокоманд емкостью 256 32-битных слов. Это позволяет сократить емкость памяти примерно в 3 раза. Использование ПНК целесообразно в случае многократного повторения в микропрограммах микрокоманд. Одним из путей практической реализации такой двухуровневой памяти является использование программируемых логических матриц.

Синхронизация микрокоманд. В основе синхронизации микрокоманды лежит число тактирующих сигналов, необходимых для ее реализации. С этой точки зрения выделяют одноктактные микрокоманды (рис. 3-29, а) и многотактные, для реализации которых требуется последовательность тактирующих сигналов (рис. 3-29, б). При этом в микрокоманду может быть включен дополнительный разряд, который определяет тип синхронизации. Многотактная синхронизация позволяет минимизировать число микрокоманд в памяти, но требует большего объема оборудования для управления временными интервалами отдельных фаз сложной микрокоманды. Многотактная синхронизация упрощает параллельную выборку и выполнение микрокоманд, а также связи между источниками и приемниками информации при реализации микрокоманд. Достоинством же одноктактной

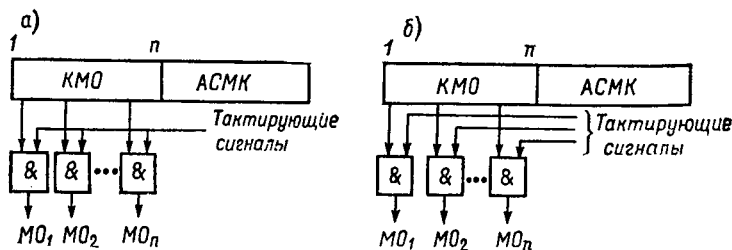


Рис. 3-29. Синхронизация МК: а — одноктактная; б — многотактная

синхронизации является простота ее технической реализации.

В заключение следует отметить, что микропроцессорные комплекты содержат БИС контроллеров последовательно-сти микрокоманд. Некоторые контроллеры имеют регистровый стек на несколько уровней, который используется для запоминания адресов возврата при вызове микроподпрограмм. Другой отличительной чертой выпускаемых контроллеров является возможность конвейерного принципа реализации микрокоманд, что позволяет повысить быстродействие процессора за счет совмещения в одном микрокомандном цикле выборки и выполнения микрокоманды на линейном участке микропрограммы при отсутствии условных переходов. Однако данный подход несколько усложняет процесс микропрограммирования.

С точки зрения наращивания памяти микропрограмм известные БИС контроллеров можно подразделить на две группы: с фиксированным числом адресуемых ячеек памяти и с наращиваемой по секционному принципу разрядностью адреса. К первой группе относится, например, контроллер микропроцессорного комплекта К589, ко второй группе — контроллер комплекта 1804. Однако следует отметить, что емкость адресуемой памяти при использовании контроллера комплекта К589 также можно наращивать при использовании принципа страничной организации памяти микрокоманд.

3-9. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ СИСТЕМ ПОЛУПРОВОДНИКОВОЙ ПАМЯТИ

ЗУ, используемые в ЭВМ, во многом определяют стоимость машины, поэтому к выбору памяти необходим обоснованный подход, который должен учитывать следующие факторы: структуру данных реализуемого алгоритма (биты, байты, слова фиксированной и переменной длины, векторы, массивы и др.), которая должна быть эффективно упорядочена в памяти для экономичного хранения и достижения максимального быстродействия при обработке данных с учетом их выборки из памяти; требования к временным и энергетическим характеристикам памяти, обусловленным требованиями к проектируемой ЭВМ; стоимость памяти; планируемый объем производства ЭВМ, что определяет соотношение между аппаратными и программными средствами, требования к минимизации объема памяти при программировании.

Полупроводниковая память имеет большое число характеристик и параметров, которые необходимо учитывать при проектировании системы:

1. Емкость памяти определяется числом бит хранимой информации. Емкость кристалла обычно выражается также в битах и составляет 1024 бита, 4К бит, 16К бит, 64К бит и т. п. Важной характеристикой является информационная организация кристалла памяти $M \times N$, где M — число слов, N — разрядность слова. Например, кристалл емкостью 16К бит может иметь различную организацию: $16K \times 1$, $4K \times 4$, $2K \times 8$. Разрядность слова кристалла памяти является достаточно важной характеристикой, так как при одинаковом времени обращения память с большей шириной выборки обладает большей информационной емкостью.

2. Время обращения к памяти определяется как время с момента подачи сигнала на запись или считывание до того момента, когда закончатся все действия, связанные с выполняемой операцией, и устройство будет готово реализовать следующую операцию. Это время называют также временем обращения к памяти. Длительность цикла есть время, которое требуется для полного обращения к памяти. В течение цикла можно выбрать информацию (считывание), ввести информацию (запись), выполнить считывание и запись или модифицировать состояние некоторой ячейки памяти. Таким образом, в отличие от времени обращения к памяти время выборки — это время от момента подачи сигнала записи или считывания до завершения соответствующей операции.

На рис. 3-30 приведена типичная структура кристалла запоминающего устройства с произвольной выборкой (ЗУПВ) с организацией 1024×4 , а рис. 3-31 иллюстрирует временные диаграммы операций считывания (а) и записи информации (б). Четыре двунаправленные шины D_3-D_0

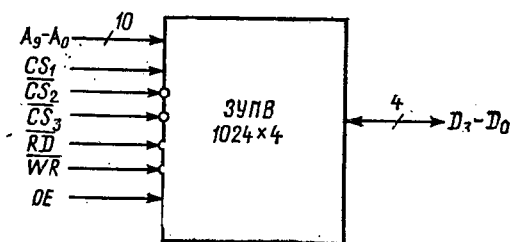


Рис. 3-30. Типичная структура кристалла ЗУПВ

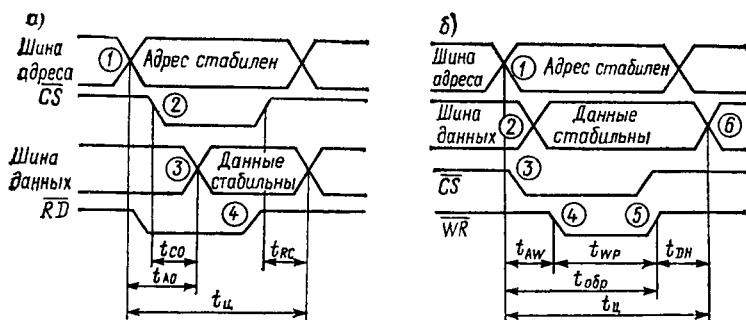


Рис. 3-31. Временные диаграммы операций считывания (а) и записи (б)

предназначены для обмена данными между памятью и остальными компонентами системы. Они подключаются к системной шине данных и работают как входные линии при записи и как выходные линии при считывании информации. Сигналы $\overline{RD}$ (чтение) и $\overline{WR}$ (запись) определяют выполняемую операцию, иногда линии $\overline{RD}$ и $\overline{WR}$ объединяются в одну линию $\overline{RD}/\overline{WR}$. Десять входных линий A_9-A_0 используются для адресации ячеек памяти. Обычно они подсоединяются к младшим линиям шины адреса системы и являются входными для внутреннего дешифратора адреса. Линии выбора кристалла $\overline{CS}$ разрешают выполнение операций записи или чтения. В кристалле имеется логическая схема И, срабатывающая при одной определенной комбинации сигналов на линиях $\overline{CS}$. Если комбинация совпадает с требуемой для данного кристалла, то он выбирается для выполнения нужной операции. В противном случае кристалл является невыбранным, выходные буферы линий шины данных переводятся в состояние высокого сопротивления и кристалл отключается от шины данных. Наличие входов $\overline{CS}$, число которых обычно не превышает трех, позволяет подключать параллельно большое число кристаллов к одним и тем же шинам данных и адреса. Выборка требуемого кристалла осуществляется в соответствии со значениями сигналов на «старших» шинах адреса. Иногда кристаллы имеют сигнал разрешения выдачи $\overline{OE}$ (output enable), который стробирует выходные усилители при чтении информации.

Операция считывания начинается с установления на шине адреса сигналов требуемой ячейки памяти (1), после этого подается сигнал выбора кристалла $\overline{CS}$ (2). Через не-

которое время содержимое целевой ячейки появляется на шине данных (3) и под действием сигнала $\overline{RD}$ информация с шины данных передается, например, в процессор. После передачи данных можно снимать сигналы с шины адреса и входа $\overline{CS}$. Отметим, что сигнал $\overline{RD}$ можно формировать одновременно с установлением адреса.

Время обращения при считывании равно большему из интервалов t_{CO} и t_{AO} , где t_{CO} — задержка сигналов считываемых данных по отношению к $\overline{CS}$; t_{AO} — задержка сигналов считываемых данных от момента установления стабильных сигналов адреса.

Продолжительность цикла считывания, равная интервалу между последовательными операциями считывания, имеет минимальное значение $t_{AO} + t_{RC}$, где t_{RC} — задержка восстановления выходных сигналов после снятия сигнала $\overline{CS}$.

Операция записи характеризуется большим числом параметров, приведенных на рис. 3-31, б. Вначале на шине адреса устанавливаются сигналы адреса целевой ячейки (1), а затем (или одновременно) на шине данных устанавливаются сигналы записываемого слова (2). После этого подаются импульсы выбора кристалла $\overline{CS}$ (3) и записи $\overline{WR}$ (4). Отсчетной точкой почти всех параметров операции записи является момент окончания сигнала $\overline{WR}$.

Время обращения при записи равно $t_{AW} + t_{WP}$, где t_{AW} — задержка между началом импульса $\overline{WR}$ и установлением стабильных сигналов адреса; t_{WP} — длительность импульса записи $\overline{WR}$.

Минимальный цикл записи равен $t_{AW} + t_{WP} + t_{DH}$, где t_{DH} — интервал времени между окончанием импульса $\overline{WR}$ и снятием данных.

Кристаллы динамической памяти имеют дополнительно еще две важные характеристики: t_H — максимальное время сохранения данных без регенерации; t_R — время регенерации, равное временному интервалу регенерации информации во всех ячейках кристалла.

3. Удельная стоимость ЗУ определяется отношением его стоимости к информационной емкости, т. е. определяется стоимостью бита хранимой информации.

4. Потребляемая энергия (или рассеиваемая мощность) приводится для двух режимов работы кристалла: режима

пассивного хранения информации и активного режима, когда операции записи и считывания выполняются с номинальным быстродействием. Кристаллы динамической МОП-памяти в резервном режиме потребляют примерно в десять раз меньше энергии, чем в активном режиме. Наибольшее потребление энергии, не зависящее от режима работы, характерно для кристаллов биполярной памяти.

5. Плотность упаковки определяется площадью запоминающего элемента и зависит от числа транзисторов в схеме элемента и используемой технологии. Наибольшая плотность упаковки достигнута в кристаллах динамической МОП-памяти. Кристаллы с большей плотностью упаковки обеспечивают экономию при монтаже печатных плат. Сокращение длины проводников на платах и в межплатных соединениях уменьшает паразитные емкости и позволяет несколько увеличить быстродействие.

6. Допустимая температура окружающей среды обычно указывается отдельно для активной работы, для пассивного хранения информации и для нерабочего состояния с отключенным питанием. Указывается тип корпуса, если он стандартный, или чертеж корпуса с указанием всех размеров, маркировкой и нумерацией контактов, если корпус новый. Приводятся также условия эксплуатации: рабочее положение, механические воздействия, допустимая влажность и др.

3-10. ПОСТОЯННЫЕ ЗАПОМИНАЮЩИЕ УСТРОЙСТВА

Основными требованиями, предъявляемыми к ПЗУ, являются неразрушаемость хранимой информации и энергонезависимость, т. е. способность сохранять информацию при отключении питания.

ПЗУ, программируемые маской. Самым простым видом ПЗУ является диодное ПЗУ, элемент памяти которого приведен на рис. 3-32, а. Выбор требуемого слова производится подачей сигнала на соответствующую шину адреса. При этом диод, подключенный в точке пересечения матрицы и соединяющий разрядную линию и линию адреса, будет проводить, устанавливая соответствующий уровень на разрядной линии. Если диода в точке пересечения нет, адресный сигнал не будет проходить на разрядную линию. Таким образом, наличие диода соответствует записи в элементе памяти 1, а отсутствие — записи 0. Недосток диодных ПЗУ заключается в сравнительно невысоком быстродействии, так как в элементах памяти отсутствуют активные элемен-

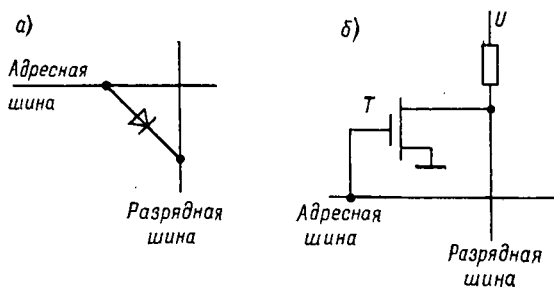


Рис. 3-32. Диодный (а) и транзисторный (б) элементы памяти ПЗУ

ты, ускоряющие заряд емкостей разрядных линий. Запись информации (программирование) в диодные ПЗУ осуществляется путем выжигания ненужных диодов с помощью лазерного или электронного луча.

Наибольшее распространение получили ПЗУ с транзисторными элементами памяти: биполярными и на МОП-транзисторах (рис. 3-32, б). Преимуществом биполярных ПЗУ является высокое быстродействие (время обращения 20—50 нс), а ПЗУ на МОП-транзисторах имеет высокую плотность компонентов, более низкую удельную стоимость, время обращения составляет 200—600 нс.

При выборе слова подключенные к адресной линии транзисторы переводятся во включенное состояние. Если сток транзистора подключен к земле, то на связанной с истоком данного транзистора разрядной линии появляется низкий уровень. Когда же сток транзистора изолирован от земли, то на соответствующей разрядной линии высокий уровень не изменяется. Эти два уровня и используются для кодирования 0 и 1. Запись информации в ПЗУ осуществляется подключением или неподключением МОП-транзистора к адресной линии, что осуществляется путем металлизации стока транзистора или отсутствием металлизации в соответствующих точках матрицы. При программировании маской на кристалле формируется базовая матрица без всякой металлизации в областях стока, затвора и истока. Когда требуемый набор слов определен, то изготавливается маска для металлизации и либо производится металлизация при необходимости формирования 0, либо транзистор остается неподключенным, если требуется написать 1. Очевидно, что если транзистор не используется для хранения бита, то металлизация стока и затвора не нужна. Поэтому с помощью

металлизации подключаются только те транзисторы, которые должны обеспечивать уровень 0. Поскольку этап металлизации является составной частью последнего этапа изготовления ИС, то такое ПЗУ можно изготовить и запрограммировать только на заводе-изготовителе. Изготовленное таким образом ПЗУ является специализированным модулем, что обуславливает его относительно высокую стоимость при малом тираже.

Имеется ряд других способов программирования ПЗУ с помощью технологических масок, однако общие принципы их организации и экономические аспекты являются общими.

Для построения памяти необходимо ячейки памяти определенным образом объединить, добавить также схемы, выполняющие дешифрацию адреса, формирование и усиление сигнала, обеспечить совместимость входов и выходов кристалла. Структурная схема типичного полупроводникового ПЗУ со словарной организацией емкостью 4К бит (512×8) представлена на рис. 3-33.

Элементы памяти объединяются в квадратную матрицу 64×64 , адресные линии A_0 — A_5 используются для выборки 64 элементов памяти одной из строк матрицы, а старшие адресные линии A_6 — A_8 , управляющие мультиплексорами (МПЛ), выбирают 8-битное слово из данной строки. Сигналы на входах выборки кристалла $\overline{CS}_1$ и $\overline{CS}_2$ обеспечивают

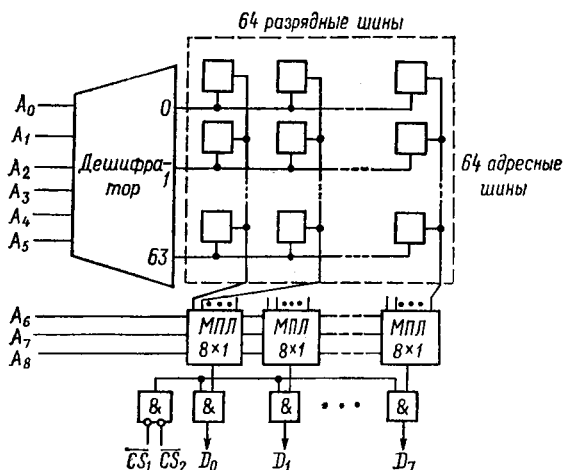


Рис. 3-33. Структурная схема ПЗУ 512×8

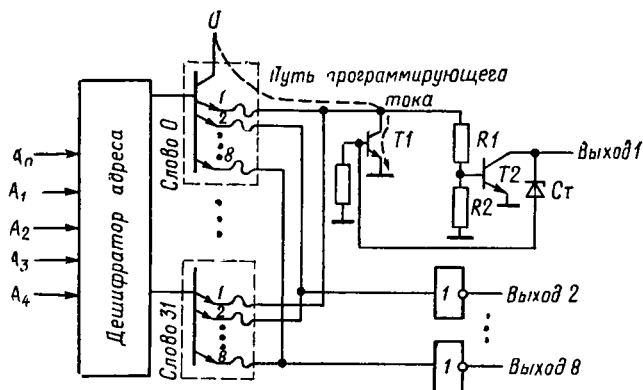


Рис. 3-34. Структурная схема ПЗУПП 32×8

управление передачей 8-битного слова с выхода мультиплексоров на выходы кристалла ПЗУ D₀—D₇.

ПЗУ, программируемые пользователем (ПЗУПП). Наибольшее распространение получили биполярные ПЗУПП с плавкими нихромовыми или титано-вольфрамовыми перемычками. В последнее время появились сообщения о производстве ПЗУПП, выполненных по комплементарной МОП-технологии. В матрице ПЗУПП содержатся все элементы связи между словарными и разрядными линиями, последовательно с каждым элементом связи включается тонкая проводящая перемычка. В процессе программирования в цепи перемычки действует импульс тока, и металлическая перемычка расплавляется. После размыкания связи импульс продолжается еще некоторое время, чтобы предотвратить отверждение металлизации и восстановление расплавленной связи.

На рис. 3-34 приведена структурная схема ПЗУПП с организацией 32×8. Для хранения одного 8-битного слова используется многоэмиттерный транзистор, в каждой эмиттерной цепи которого находится плавкая перемычка. При выборке дешифратором слова на соответствующей разрядной линии при наличии перемычки устанавливается высокий уровень и выходной транзистор T₂ включается (логический 0). Если перемычка расплавлена, то соответствующая разрядная линия оказывается «плавающей» и выходной транзистор T₂ выключен (логическая 1).

Для программирования некоторого разряда на шины подаются сигналы для выборки требуемого адреса, и на

базе выбранного многоэмиттерного транзистора появляется сигнал высокого уровня. Внешней схемой на выходе программируемого разряда устанавливается напряжение около 8 В. Этого напряжения достаточно для включения стабилитрона Ст (с напряжением стабилизации около 7 В), что приводит к включению транзистора Т1 и заземлению правого вывода плавкой перемычки. Напряжение питания многоэмиттерных транзисторов увеличивается до 12 В, и между выводами перемычки оказывается приложенным напряжение около 5 В. Если перемычка имеет сопротивление 100 Ом, то через нее проходит ток 50 мА, что вызывает расплавление перемычки. В том случае, когда состояние некоторого разряда программировать не нужно, на выход его 8 В не подается, транзистор Т1 выключен, ток в цепи перемычки ограничивается резистором и ее разрушения не происходит.

Характерным типом отказа в данном ПЗУПП является повторное замыкание разрушенной перемычки. Исследования показали, что чем продолжительнее программирующий импульс, тем выше вероятность повторного замыкания. Под воздействием протекающего тока нихром переходит в жидкое состояние и образует очень узкий зазор. Фотографии показывают, что в зазоре видны дендриты («усы»), которые могут в дальнейшем привести к повторному замыканию из-за явления роста кристаллов.

Самые простые программаторы рассчитаны на программирование ПЗУПП одного типа. Обычно они имеют шестнадцатиричную клавиатуру для установки адресов и данных, а также схему управления и формирования необходимой последовательности сигналов. Имеются более универсальные программаторы, на которых с помощью съемных плат или изменений в программе встроенной машины можно программировать разнотипные ПЗУ. Почти во всех современных программаторах предусматривается буферное ЗУПВ, в которое загружаются, например с перфоленты, данные, подлежащие записи в ПЗУПП. Наличие встроенного ЗУПВ позволяет реализовать в программаторе режим редактирования и режим контроля программирования. Наиболее сложными являются так называемые оперативные программаторы, которые подключаются к терминалу большой ЭВМ. В таких программаторах выполняются следующие функции: загрузка данных в буферное ЗУПВ с терминала; проверка того, что до программирования все биты ПЗУ находятся в незапрограммированном состоянии; программирование: данные из буферного ЗУПВ записыва-

ются в кристалл ПЗУ с соблюдением необходимой последовательности программирующих сигналов; контроль программирования.

Программаторы, подключаемые к терминалу, выполняют указанные функции автоматически после загрузки данных в буферное ЗУПВ. Громоздкая процедура программирования оправдана только на первых этапах проектирования опытного образца или при короткой предпроизводственной проверке работы.

Перепрограммируемые ПЗУ относятся к классу стираемых программируемых ПЗУ (СППЗУ), так как после стирания хранимой в ПЗУ информации возможно занесение новых данных в тот же накопитель.

Стирание очень удобно при проектировании экспериментальных программных ПЗУ, поскольку ошибка в программном ПЗУ не разрушает устройства, а приводит к его стиранию и перепрограммированию.

СППЗУ можно выполнить в виде МОП-матриц двух типов или в виде полупроводникового ПЗУ в корпусе со стеклянной прозрачной крышкой. В СППЗУ второго типа стирание информации осуществляется ультрафиолетовым облучением кристалла.

СППЗУ первого типа по МОП-технологии изготавливается как обычная МОП-матрица, но между металлическим затвором и слоем изолирующего оксида осаждается тонкий слой нитрида кремния. Слой нитрида кремния имеет свойство сохранять электрический заряд (положительный или отрицательный в зависимости от материала МОП-матрицы), когда на затвор транзистора подается высоковольтный программирующий импульс. Амплитуда этого импульса в несколько раз превышает обычный для памяти уровень напряжения. При отсутствии дополнительных сигналов программирования или выключений питания заряд в слое нитрида кремния будет сохраняться весьма долго. Гарантированное время сохранения информации для ряда приборов составляет 10 лет.

Такие БИС программируются подачей соответствующих сигналов на внешние контакты их корпусов, а стирание их содержимого осуществляется подачей на те же контакты стирающего сигнала.

Программирующий импульс на затворе программируемого транзистора заставит обычно не проводящий транзистор включиться, что приведет к появлению на выходе нулевого бита. Транзистор, который не программировался, будет иметь в слое нитрида кремния нулевой заряд и оста-

нется выключенным, приводя к появлению на выходе единицы.

Главное достоинство такого СППЗУ заключается в возможности перепрограммирования устройства с помощью подачи на затвор транзистора импульса напряжения противоположной полярности, чтобы удалить заряд, захваченный в слое нитрида кремния. Такое действие возвращает переход транзистора в исходное непроводящее состояние. Рассмотренное СППЗУ может перепрограммироваться в микро-ЭВМ.

Второй тип СППЗУ образуется матрицей рМОП-транзисторов с плавающим затвором, и все транзисторы вначале находятся в непроводящем состоянии. Подача большого напряжения между истоком и стоком удаляет положительные носители от плавающего затвора, и изолированные отрицательные носители остаются захваченными на затворе. Результирующий отрицательный заряд порождает проводящий канал между истоком и стоком транзистора. В этом отношении устройство напоминает рассмотренное выше СППЗУ. Однако содержимое СППЗУ второго типа нельзя стереть подачей обратного импульса напряжения между истоком и стоком. Вместо этого для разряда затвора через кварцевую крышку подается УФ-излучение, которое переводит все транзисторы устройства в непроводящее состояние. Комнатное освещение, солнечный свет и флюоресцентные лампы не оказывают заметного влияния на хранимые данные даже при длительной экспозиции. УФ-излучение в продолжение 10—30 мин полностью возвращает все запрограммированные биты в исходное состояние.

3-11. ЗАПОМИНАЮЩИЕ УСТРОЙСТВА С ПРОИЗВОЛЬНОЙ ВЫБОРКОЙ

По принципу действия различают два типа ЗУПВ: статические, которые могут быть выполнены по любой технологии, и динамические, которые выполняются только по МОП-технологии. Сейчас наибольшее распространение получили кристаллы МОП-памяти, характеризующиеся большой плотностью упаковки и малой потребляемой энергией, хотя по быстродействию они уступают биполярным устройствам. Однако разработанная в последнее время совершенная МОП-технология обеспечивает приближение к быстродействию биполярных схем.

В зависимости от принципа построения массива запоминающих элементов ЗУПВ имеют словарную или матрич-

ную организацию; последняя характерна для динамических ЗУПВ. При одинаковой емкости память с матричной организацией имеет несколько преимуществ по сравнению с памятью со словарной организацией. В ней, например, гораздо более простым оказывается дешифратор адреса и значительно уменьшается число линий выборки.

Статические ЗУПВ реализуются на триггерах с непосредственной связью, которые при включенном питании могут хранить информацию неограниченно долго без дополнительных управляющих сигналов. Они обладают хорошей помехоустойчивостью, не требуют сложных схем управления и имеют простой интерфейс с процессором. Благодаря этим качествам достигается умеренная сложность и стоимость подсистемы памяти. Основным недостатком статического ЗУПВ является сравнительная сложность запоминающего элемента, из-за чего не удается достичь большой плотности упаковки. Кроме того, по сравнению с динамическими ЗУПВ они имеют сравнительно большое потребление энергии.

Динамические ЗУПВ. В основе работы динамических ЗУПВ лежит хранение двоичной информации в виде заряда на емкости между затвором информационного МОП-транзистора и общей точкой схемы — землей. Емкость образована собственно емкостью затвор — сток и паразитной емкостью. В запоминающих элементах затвор информационного транзистора обычно связан с истоком другого МОП-транзистора. Так как входное сопротивление выключенного МОП-транзистора составляет 10^9 — 10^{10} Ом, а затвора 10^{12} — 10^{13} Ом, то накопленный на емкости заряд сохраняется в течение продолжительного времени. При комнатной температуре это время достигает нескольких десятков и даже сотен миллисекунд. С увеличением температуры ток разряда быстро увеличивается, поэтому допустимое время сохранения заряда при температуре $+100^\circ\text{C}$ составляет примерно 2 мс. Для восстановления заряда на емкости ее необходимо подключать к источнику питания, т. е. производить регенерацию хранимых данных. Регенерация обычно производится по строкам или столбцам массива памяти.

¹ Динамическая память позволяет значительно уменьшить потребляемую кристаллом энергию, особенно в режиме пассивного хранения информации. Кроме того, число транзисторов в запоминающем элементе можно сократить даже до одного, что приводит к значительному увеличению плотности упаковки.

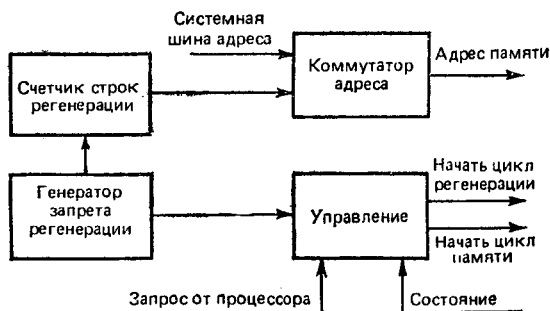


Рис. 3-35. Контроллер динамической памяти

К недостатку динамической памяти относится усложнение схемы управления, в которую необходимо вводить схему регенерации и разрешать конфликты между обращениями к памяти от процессора и от схемы регенерации. Это приводит к усложнению интерфейса динамической памяти по сравнению со статической.

Для регенерации системы динамической памяти используется один из следующих трех способов:

1. Асинхронная регенерация не связана с действиями процессора и не зависит от его состояния, т. е. память имеет автономную схему управления (контроллер) и может работать независимо от процессора. Контроллер выполняет необходимые для регенерации действия самостоятельно и связывается с процессором только при выполнении операций записи и считывания, инициируемых процессором. Для процессора асинхронная память не отличается от статической, но иногда его запросы задерживаются, если память занята регенерацией. Асинхронная регенерация обеспечивает наибольшую модульность проектирования подсистемы памяти, но контроллер оказывается довольно сложным (рис. 3-35). Надежный и быстродействующий контроллер должен разрешать конфликты между запросами регенерации и обращениями к памяти от процессора. Начало циклов памяти приходится задерживать до полного разрешения конфликтов и стабилизации сигналов адреса и управления. Разрешение конфликтов несколько увеличивает время обращения, что приводит к уменьшению производительности системы.

2. Синхронная регенерация выполняется синхронно с действиями процессора. Для регенерации выбираются те циклы, в которых процессор не обращается к памяти, и

поэтому конфликтов между обращениями к памяти от процессора и схем регенерации не возникает. Основой эффективной синхронной регенерации является схема «планировщика» регенерации. Она воспринимает сигналы состояния процессора и запросы таймера регенерации и «планирует» циклы регенерации в те же фазы машинных циклов, когда процессор не обращается к памяти. Конкретная реализация планировщика определяется особенностями конкретного процессора, поэтому модули памяти теряют универсальность по сравнению с модулями, в которых применяется асинхронная регенерация.

3. Полусинхронная регенерация, являющаяся комбинацией первых двух способов, выполняется синхронно с сигналами главного генератора синхронизации, но асинхронно с действиями процессора. Здесь несколько упрощается логика разрешения конфликтов: если запрос к памяти от процессора инициируется синхронно с фронтом сигналов главной синхронизации, а запрос регенерации — со спадом, то два запроса одновременно не возникают. В остальном полусинхронная регенерация аналогична асинхронной.

При выборе способа регенерации необходимо учитывать особенности процессора, например, организацию состояний ожидания, особенности машинных циклов, а также кристаллов динамической памяти. Наиболее сложной для проектирования оказывается синхронная регенерация. Следует отметить, что при использовании асинхронной регенерации потери производительности обычно невелики. Если, например, цикл памяти составляет 500 нс, то в интервале регенерации 2 мс укладывается 4000 циклов памяти. В этом же интервале для памяти 4К с организацией 64×64 необходимо выполнить 64 цикла регенерации, что составляет примерно 2 %. В настоящее время многие микропроцессорные комплекты имеют в своем составе БИС контроллеров динамической памяти.

Организация системы памяти. Адресуемое пространство памяти с 16-битной шиной адреса составляет 64К байт. Емкость отдельных кристаллов памяти гораздо меньше, поэтому память ЭВМ может состоять из десятков кристаллов. Для построения памяти разрядностью M бит на основе m -битных кристаллов памяти необходимо объединить M/m кристаллов, входы адресных и управляющих сигналов которых объединяются параллельно. Поэтому далее при рассмотрении организации системы памяти на 64К байта будем предполагать, что в качестве базовых кристаллов используются кристаллы с организацией $N \times 8$, где N — чис-

ло ячеек памяти кристалла. На рис. 3-36 приведена типичная схема модуля статической памяти емкостью 4К байт, построенного на основе 8 кристаллов с организацией 512×8 . Обычно память большой емкости разделяют на взаимосвязанные модули (платы) емкостью 4К—8К байт, содержащие необходимые обслуживающие схемы: дешифраторы, буферные схемы, схемы управления и регенерации. Для простоты построения схемы будем считать, что все активные сигналы управления соответствуют высокому уровню.

Одноименные линии данных всех кристаллов системы памяти объединяются параллельно и подключаются к системной шине данных. При обращении к памяти с шиной данных электрически связан только один выбранный кристалл, а тристабильные буферы линий данных остальных, невыбранных, кристаллов переведены в состояние высокого сопротивления, т. е. отключены от шины данных. При параллельном включении многих кристаллов памяти могут быть превышены нагрузочные способности процессора, поэтому между памятью и процессором обычно включают шинные усилители-формирователи.

Линии адреса разделяются на четыре группы и выполняют следующие функции.

Линии адреса A_{15} — A_{12} используются для выбора модуля емкостью 4К байт в адресуемом пространстве памяти, 16 выходных линий ДШ подключены к отдельным модулям.

Три линии адреса A_{11} — A_9 дешифруются на плате модуля дешифратором ДШ2, выходные сигналы которого выбирают один из 8 кристаллов ЗУПВ, участвующий в обмене данными с процессором. Кристалл выбирается посредством входов CS.

Младшие 9 линий адреса A_8 — A_0 используются для выборки адресуемой ячейки памяти на кристалле. Во многих системах рабочее пространство памяти оказывается гораздо меньшим потенциально адресуемого пространства. Наличие у кристаллов памяти нескольких входов выбора кристалла позволяет упростить дешифрацию старших линий адреса.

Для расширения системы памяти можно использовать принцип страничной организации памяти. В этом случае прямо адресуемая память рассматривается как одна страница и используется для обращения к другим страницам расширенной памяти. Для обращения к памяти емкостью, например, в 1М байт необходим 20-битный адрес, сформированный из

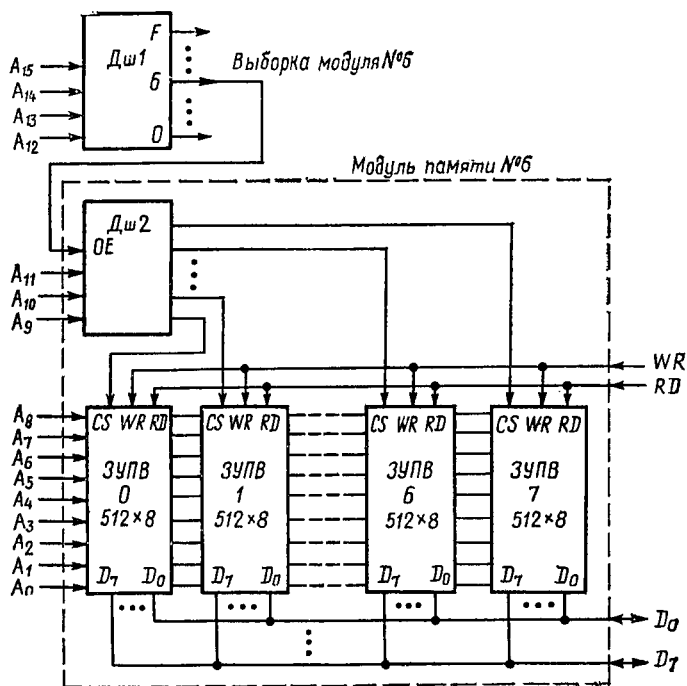


Рис. 3-36. Модуль памяти емкостью 4К байт

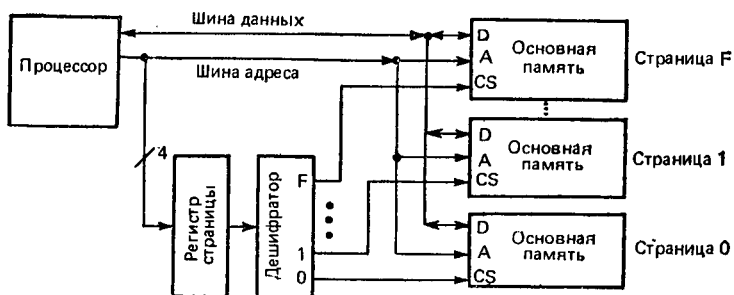


Рис. 3-37. Выбор страницы с помощью дешифратора

ровать который без дополнительных аппаратных средств невозможно. Поэтому обращение к расширенной памяти реализуется в два этапа: выбор страницы и обращение к ячейке в заданной странице. Если память разделена на 16 страниц по 64К байта в каждой, то адрес страницы содержит 4 бита, а адрес в пределах страницы 16 бит. Для определенности положим, что обычная память считается нулевой страницей. Будем называть ее локальной памятью, а остальную часть расширенной системы памяти — основной или главной памятью. Для выбора целевой страницы необходимо предусмотреть 4-битный порт вывода, называемый портом выбора страниц и определяющий одну из 16 выбираемых страниц (рис. 3-37). При включении системы следует произвести загрузку порта выбора страницы, чтобы процессор начал выполнение программы с известной ячейки.

3-12. БУФЕРНЫЕ ЗУПВ

В вычислительных системах используются подсистемы с различным быстродействием, и в частности с различной скоростью передачи данных (рис. 3-38, а). Обычно обмен данными между такими подсистемами реализуется с использованием прерываний или канала прямого доступа к памяти. В первом случае подсистема 1 формирует запрос на обслуживание по мере готовности данных к обмену. Однако обслуживание прерываний связано с непроизводительными потерями времени и при пакетном обмене производительность подсистемы 2 заметно уменьшается. При обмене данными с использованием канала прямого доступа к памяти подсистема 1 передает данные в память подсистемы 2. Данный способ обмена достаточно эффективен с точки зрения быстродействия, но для его реализации необходим довольно сложный контроллер прямого доступа к памяти.

Наиболее эффективно обмен данными между системами с различным быстродействием реализуется при наличии между ними специальной буферной памяти. Данные от подсистемы 1 временно запоминаются в буферной памяти до готовности подсистемы 2 принять их. Емкость буферной памяти должна быть достаточной для хранения тех блоков данных, которые подсистема 1 формирует между считываниями их подсистемой 2. Отличительной особенностью буферной памяти является запись данных с быстродействием и под управлением подсистемы 1, а считывание — с

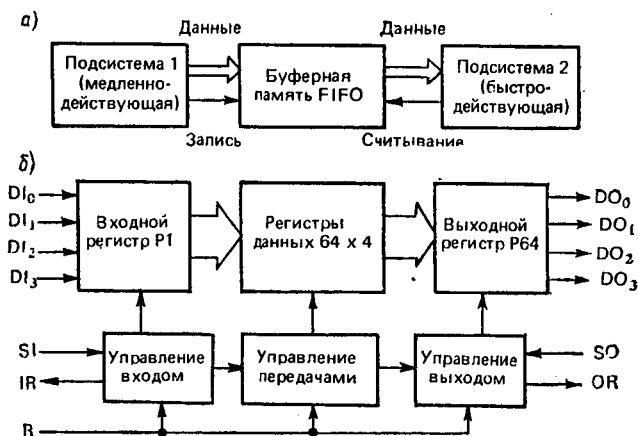


Рис. 3-38. Применение буферной памяти (а); структурная схема буфера 64×4 (б)

быстродействием и под управлением подсистемы 2 («эластичная» память). В общем случае память должна выполнять операции записи и считывания совершенно независимо и даже одновременно, что устраняет необходимость синхронизации подсистем. Разумеется, буферная память должна сохранять порядок поступления данных от подсистемы 1, т. е. работать по принципу «первое записанное слово считывается также первым» (FIFO). Таким образом, под буферной памятью типа FIFO понимается ЗУПВ, которое автоматически следит за порядком поступления данных и выдает их в том же порядке, допуская выполнение независимых и одновременных операций записи и считывания. На рис. 3-38, б приведена структурная схема буферной памяти типа FIFO емкостью 64×4 . На кристалле размещены 64 4-битных регистра с независимыми цепями сдвига, организованных в 4 последовательных 64-битных регистра данных, 64-битный управляющий регистр, а также схема управления. Входные данные поступают на линии $DI_0—DI_3$, а вывод данных осуществляется через контакты $DO_0—DO_3$. Ввод (запись) данных производится управляющим сигналом SI (shift in), а вывод (считывание) — сигналом вывода SO (shift out). Ввод данных осуществляется только при наличии сигнала готовности ввода IR (input ready), а вывод — при наличии сигнала готовности вывода OR (output ready). Управляющий сигнал R (reset) производит сброс содержимого буфера.

При вводе 4-битного слова под действием сигнала SI оно автоматически передвигается в ближайший к выходу свободный регистр. Состояние регистра данных отображается в соответствующем ему управляющем триггере, совокупность триггеров образует 64-битный управляющий регистр. Если регистр содержит данные, то управляющий триггер находится в состоянии 1, а если регистр не содержит данных, то триггер находится в состоянии 0. Как только управляющий бит соседнего справа регистра изменяется на 0, слово данных автоматически сдвигается к выходу. Перед началом работы в буфер подается сигнал сброса R и все управляющие триггеры переводятся в состояние 0 (все регистры буфера свободны). На выводе IR формируется логическая 1, т. е. буфер готов воспринимать входные данные. При действии сигнала ввода SI входное слово загружается в регистр P1, а управляющий триггер этого регистра устанавливается в состояние 1: на входе IR формируется логический 0. Связи между регистрами организованы таким образом, что поступившее в P1 слово «спонтанно» копируется во всех регистрах данных FIFO и появляется на выходных линиях DO_0 — DO_3 . Теперь все 64 регистра буфера содержат одинаковые слова, управляющий триггер последнего регистра P64 находится в состоянии 1, а остальные управляющие триггеры сброшены при передаче данных в соседние справа регистры. Состояние управляющего триггера P64 выведено на линию готовности выхода OR; OR принимает значение 1, когда в триггер записывается 1. Процесс ввода может продолжаться до полного заполнения буфера; в этом случае все управляющие триггеры находятся в состоянии 1 и на линии IR сохраняется логический 0.

При подаче сигнала SO производится восприятие слова с линией DO_0 — DO_3 , управляющий триггер P64 переводится в состояние 1, на линии OR появляется логическая 1, а управляющий триггер P64 сбрасывается в 0. Затем этот процесс повторяется для остальных регистров и нуль в управляющем регистре перемещается ко входу по мере сдвига данных вправо.

В некоторых кристаллах буфера FIFO имеется дополнительная выходная линия флажка заполнения наполовину. На ней формируется сигнал 1, если число слов составляет более половины емкости буфера.

Рассмотренный принцип организации FIFO допускает выполнение записи и считывания данных независимо и одновременно. Скорость ввода определяется временным интервалом, необходимым для передачи данных из P1, а вы-

водить данные можно с такой же скоростью. Единственным ограничением является время распространения данных через FIFO, равное времени передачи входного слова на выход незаполненного буфера FIFO. Оно равняется произведению времени внутреннего сдвига и числа регистров данных. В буферах FIFO, выполненных по МОП-технологии и имеющих емкость 64 слова, время распространения составляет примерно 30 мкс, а в биполярных FIFO такой же емкости — примерно 2 мкс.

Данные буферы можно наращивать как по числу слов, так и по их длине.

3-13. СТЕКОВАЯ ПАМЯТЬ

Стековой называют память, доступ к которой организован по принципу: «последним записан — первым считан» (last input first output — LIFO). Использование принципа доступа к памяти на основе механизма LIFO началось с больших ЭВМ. Использование стековой памяти оказалось очень эффективным при построении компилирующих и интерпретирующих программ, при вычислении арифметических выражений с использованием польской инверсной записи. В малых ЭВМ стековая память стала широко использоваться в связи с удобствами реализации процедур вызова подпрограмм и при обработке прерываний.

Принцип работы стековой памяти поясняется на рис. 3-39. Когда слово А помещается в стек, оно располагается в первой свободной ячейке памяти. Следующее записываемое слово перемещает предыдущее на одну ячейку вверх и занимает его место и т. д. Запись 8-го кода, после Н, приводит к переполнению стека и потере кода А. Считывание кодов из стека осуществляется в обратном порядке, т. е. начиная с кода Н, который был записан последним. Необходимо отметить, что выборка, например, кода Е невозможна до выборки кода F, что определяется используемым механизмом обращения при записи и чтении типа LIFO. Для фиксации момента переполнения стека в принципе желательно формировать признак переполнения.

Перемещение данных при записи и считывании информации в стековой памяти может быть фактическим подобно тому, как это имеет место, например, в сдвигающих регистрах, или может моделироваться. С точки зрения реализации механизма доступа к стековой памяти выделяют аппаратный и аппаратно-программный (внешний) стек.

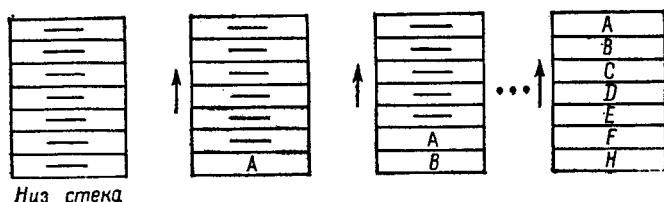


Рис. 3-39. Принцип работы стековой памяти

Аппаратный стек представляет собой совокупность регистров, связи между которыми организованы таким образом, что при записи и считывании данных содержимое стека автоматически сдвигается. Обычно емкость аппаратного стека ограничена диапазоном от нескольких регистров до нескольких десятков, поэтому в большинстве МП такой стек используется для хранения содержимого программного счетчика и его называют стеком команд. Основное достоинство аппаратного стека — высокое быстродействие, а недостаток — ограниченная емкость.

Аппаратно-программный, или внешний, стек. Наиболее распространенным в настоящее время и, возможно, лучшим вариантом организации стека в ЭВМ является использование области ЗУПВ. Для адресации стека используется указатель стека, который предварительно загружается в регистр и определяет адрес последней занятой ячейки. Помимо команд CALL и RET, которые используют стек для спасения и восстановления содержимого программного счетчика, имеются также команды PUSH и POP, которые используются для временного запоминания в стеке содержимого регистров и их восстановления соответственно. В некоторых МП запоминание содержимого регистров в стеке осуществляется автоматически при прерывании программ. Принцип организации аппаратно-программного стека иллюстрирует рис. 3-40. При выполнении записи содержимого регистра (Pr) под действием команды PUSH реализуются следующие микрооперации:

$$(UC) := (UC) - 1; ((UC)) := (Pr).$$

Таким образом, после записи очередного слова в стек указатель стека хранит адрес ячейки, в которую было произведено последнее обращение. Содержимое UC при записи уменьшается, что является характерным для МП КР580 и процессоров СМ.

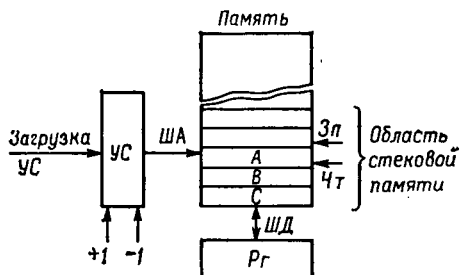


Рис. 3-40. Принцип организации аппаратно-программного стека

Чтение информации из ячейки памяти стека реализуется при выполнении команды POP:

$$(Pr) := ((UC)); (UC) := (UC) + 1.$$

Таким образом, механизм доступа к памяти по принципу FIFO в аппаратно-программном стеке реализуется за счет уменьшения или увеличения содержимого УС на 1 при выполнении команд PUSH и POP соответственно. Следует отметить, что в данном типе стека не формируются признаки переполнения результата аппаратным путем.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие факторы влияют на структуру процессоров, реализованных на основе БИС?
2. Что составляет понятие архитектуры ЭВМ?
3. Перечислите состав и назначение типичных функциональных блоков процессора.
4. Объясните функциональное назначение управляющих сигналов в процессоре.
5. Какие достоинства и недостатки присущи однокристалльным и секционнным микропроцессорам?
6. Сравните различные варианты организации шин в процессоре.
7. Какие способы кодирования микрокоманд используются в микропрограммном устройстве управления?
8. Каково минимальное число состояний для выполнения машинного цикла в МП КР580 и почему?
9. Какие действия выполняются в каждом из состояний машинного цикла для МП КР580?
10. Назовите основные способы программирования постоянной памяти.
11. Какое назначение в БИС памяти имеет вход выборки кристалла?
12. В чем отличие памяти статического и динамического типов?
13. С какой целью в ЭВМ применяется стековая память?
14. Каким образом осуществляется запись и чтение информации из стековой памяти?

ОРГАНИЗАЦИЯ ИНТЕРФЕЙСА В МИКРО- И МИНИ-ЭВМ**4-1. ОСНОВНЫЕ ПОНЯТИЯ И ТРЕБОВАНИЯ
К ИНТЕРФЕЙСУ**

ЭВМ осуществляют обмен информацией с внешним миром посредством таких периферийных устройств, как теле-тайпы, электрические пишущие машинки, дисплеи, магнитные диски, ленты, аналого-цифровые и цифро-аналоговые преобразователи и др. Однако необходимо иметь в виду, что к числу периферийных устройств относятся также и более простые цифровые устройства: триггеры, регистры и т. п. Все периферийные устройства требуют для функционирования определенного набора управляющих сигналов, протокола обмена и способа обмена с ЭВМ, вида используемого кода, и поэтому шины обмена информацией подключаются не непосредственно к периферийному устройству, а через интерфейс, структура и принцип работы которого в сильной степени определяется совместимостью сопрягаемых компонентов. Для подключения периферийных устройств к ЭВМ используют специальные электронные схемы, называемые интерфейсными модулями. Сложность интерфейса определяется типом периферийных устройств, их числом, расстоянием между ЭВМ и периферийными устройствами, физической природой устройств (электронные, электромеханические) и их архитектурой. В ряде случаев компоненты интерфейса разбивают на две группы и одну относят к интерфейсу ЭВМ, а другую — к периферийным устройствам и называют контроллером. Вообще термины «интерфейс» и «контроллер» строго не определены и их иногда используют как синонимы.

Кроме аппаратной части для организации интерфейса необходимо разрабатывать некоторое программное обеспечение, которое включает в себя программы идентификации типа информации (данные, управляющие символы и т. п.), программы преобразования форматов, программы-драйверы для управления обменом информацией, программы обработки запросов прерываний и др. Следует отметить, что при использовании специальных модулей программируемых периферийных интерфейсов, например типа КР580ИК55, их требуется предварительно программировать с целью определения режима и направления обмена по каждому из каналов, необходимости использовать прерывания и т. д.

Под интерфейсом будем понимать совокупность программных и аппаратных средств, с помощью которых компоненты системы объединяются таким образом, чтобы она могла решать требуемую задачу. Сложность интерфейса в значительной мере определяется степенью совместимости ЭВМ и периферийных устройств, т. е. сложностью необходимых преобразований. Под совместимостью будем понимать возможность объединения отдельных компонентов системы в единую операционную сеть посредством программных и аппаратных средств. При решении данной задачи почти всегда необходимо принимать компромиссные сложные решения при выборе аппаратных или программных средств, так как часто желание минимизировать аппаратную часть интерфейса приводит к существенному увеличению стоимости и сроков разработки программного обеспечения интерфейса.

Совместимость определяется четырьмя основными признаками: быстродействием, кодами, используемыми для обмена, архитектурой процессора, электрическими характеристиками. Если объединяемые компоненты не соответствуют друг другу по одному или нескольким признакам, то они не могут быть объединены без интерфейсных модулей. Остановимся на рассмотрении этих признаков совместимости. Различное быстродействие ЭВМ и периферийных устройств требует контроля над состоянием признаков готовности (флажков) периферийных устройств к обмену и введения буферных регистров для временного хранения передаваемых кодов с целью согласования работы ЭВМ и периферийных устройств по быстродействию. Для передачи данных используются специальные стандартные коды обмена информацией, содержащие алфавитно-цифровые символы, специальные символы и управляющие символы, которые должны интерпретироваться компонентами системы одинаковым образом с точки зрения их семантики. Например, если необходимо некоторое перекодирование, то эта функция выполняется программными или аппаратными средствами интерфейса. Архитектура процессора с точки зрения набора и организации шин для приема и передачи данных, адресных шин, набора управляющих сигналов определяет протокол или метод синхронизации процессора и периферийных устройств, и все передачи данных, кодов признаков состояния, управляющих сигналов должны подчиняться этому протоколу обмена. Электрические характеристики процессора должны быть совместимы с характеристиками логических схем интерфейса, которые, в свою

очередь, должны быть согласованы с периферийными устройствами.

Решение проблемы разработки интерфейса зависит от конкретных применений и требований к нему и, хотя в настоящее время не существует формализованной процедуры для проектирования интерфейса, в общем случае можно выделить определенный набор и последовательность действий:

1. Изучение структуры шин процессора, используемых для обмена информацией, и характеристик периферийного устройства.

2. Выбор способа адресации периферийных устройств.

3. Анализ необходимости буферирования передаваемых кодов и требований к процедуре управления приемом и выдачей кодов.

4. Определение функций, предписываемых интерфейсному модулю помимо передачи кодов, например анализ готовности к приему или передаче, дополнительные функции в периферийном устройстве (сброс, перемотка, инициирование и др.), генерация временных задержек.

5. Анализ целесообразности обслуживания периферийных устройств в режиме с прерыванием.

6. Выбор готовых модулей программируемого периферийного интерфейса или разработка специализированных модулей.

Некоторые вспомогательные этапы проектирования в данной последовательности не выделены в силу их специфичности для конкретных разработок, и, кроме того, разбиение на этапы имеет несколько условный характер, так как часто между ними трудно провести четкую границу.

В соответствии с выделенными двумя наиболее распространенными типами архитектуры ЭВМ рассмотрим особенности организации интерфейса процессора с памятью и периферийными устройствами.

4-2. ИНТЕРФЕЙС С ИЗОЛИРОВАННОЙ И ОБЩЕЙ СИСТЕМОЙ ШИН

Обмен информацией между процессором, памятью и периферийными устройствами осуществляется по системе шин, состав и назначение которых, правила использования, функциональное и физическое разделение при передаче данных, адресов, управляющих сигналов различны для разных ЭВМ. Однако есть принципиально общие закономерности в организации интерфейса, процедуре обмена

информацией между двумя устройствами, одно из которых является источником, а другое — приемником информации. Некоторые устройства могут являться приемниками и источниками информации, что требует наличия шин данных для приема и передачи. Передача данных между приемником и источником информации инициируется управляющими сигналами, которые обеспечивают выдачу и прием кодов, проверку состояния готовности устройств, формирование сигналов для выбранного режима обмена. Необходимые сигналы инициируются командами обмена информацией между процессором и периферийными устройствами, процессором и памятью. Если устройство, например источник информации, связано более чем с одним приемником, то также необходимо адресовать выбираемое устройство. Таким образом, можно выделить типичный набор внешних шин процессора, показанных на рис. 4-1. Данный рисунок отражает логическую структуру шин и сигналов. Организация физических шин в процессоре с точки зрения их совмещения при передаче данных, адресов, управляющих сигналов в различных процессорах, естественно, может отличаться. При использовании группы шин для приема и выдачи данных, выдачи адреса в режиме разделения времени требуется дополнительный внешний буферный регистр для хранения адреса, что также увеличивает машинный цикл. В настоящее время наиболее распространенным является вариант использования отдельных однонаправленных адресных и двунаправленных шин данных. Однако в некоторых 16-битных однокристальных МП имеет место исполь-



Рис. 4-1. Внешние шины и сигналы процессора

зование шин данных для передачи полного адреса или старших разрядов адреса. Это позволяет использовать освободившиеся выводы для увеличения числа входов обработки запросов прерываний, организации каналов ввода и вывода информации последовательным кодом. Многообразие в организации и использовании шин данных, адресов и управляющих сигналов является не принципиальным, а только следствием ограничений технологии и числа выводов интегральных схем.

Для обмена информацией в общем случае необходимо осуществить следующие основные действия: адресацию периферийного устройства, с которым необходимо произвести обмен, и области памяти, если это требуется; анализ состояния периферийного устройства с целью определить его готовность к обмену для обеспечения совместимости по быстродействию; выполнение требуемых управляющих функций в периферийном устройстве, например, начальный сброс, перемотку, старт, останов и др.; передачу и буферирование данных, если необходимо; окончание процесса обмена путем засылки в процессор сигнала прерывания.

Несмотря на многообразие типов ЭВМ, можно выделить две основные организации интерфейса между процессором, памятью и периферийными устройствами:

1. Двухшинную организацию, или интерфейс с изолированной системой шин, который используется в мини-ЭВМ «Электроника-100», М-6000 и др. Название «двухшинная» отражает тот факт, что с функциональной точки зрения есть два такта передачи кодов, процессор — память и процессор — периферийные устройства, обращение к которым осуществляется отдельными группами команд. Однако можно утверждать, что в такой организации имеются четыре шины для передачи кодов в обоих направлениях, или еще можно дополнительно учесть адресные группы шин. Эта неоднозначность часто проистекает из-за того, что разработчики по-своему описывают уже известные организации, делая акцент на отличиях в непринципиальных моментах. Ниже используется понятие «интерфейс с изолированной шиной».

2. Одношинную организацию интерфейса, или интерфейс с общей шиной. Он использован в отечественных мини-ЭВМ М-400, СМ-3(4) и других. Впоследствии многие разработчики микро-ЭВМ стали широко использовать данный способ построения интерфейса. Например, в микро-ЭВМ «Электроника-60» используется модифицированный

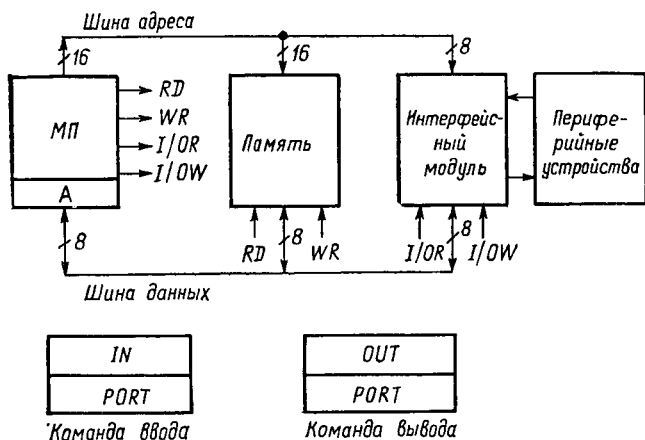


Рис. 4-2. Интерфейс с изолированной шиной

вариант данной организации, отличительными особенностями которого являются 38 линий вместо 56, передача адреса и данных по общим линиям, сокращенное число входов прерываний и некоторые другие.

Интерфейс с изолированной шиной. Данный тип организации интерфейса иллюстрируется на рис. 4-2. Характерной его особенностью является раздельная адресация памяти и периферийных устройств при обмене информацией. Это осуществляется путем использования отдельных групп команд для обмена информацией с периферийными устройствами и памятью. Рассмотрим организацию обмена в микро-ЭВМ на основе МП КР580. Для обмена с периферийными устройствами используются двухбайтные команды ввода IN PORT и вывода OUT PORT, формат которых приведен на рис. 4-2, внизу. Данные команды позволяют МП адресовать 256 портов ввода и 256 портов вывода информации. Управление обменом выполняется под действием управляющих сигналов ввода I/OR и вывода I/OW, формируемых системным контроллером при выполнении соответствующих команд. Адрес из МП передается по 8 младшим и 8 старшим линиям шины адреса, передача данных выполняется между аккумулятором А и буферным регистром интерфейсного модуля (портом) периферийного устройства.

Обмен информацией с памятью выполняется путем использования команд передачи данных. Команда MOV r,

М выполняет передачу из ячейки памяти, адресуемой 16-битной регистровой парой (H, L), в аккумулятор или любой регистр общего назначения, кодируемый в поле г. Команда MOV М, г выполняет передачу из регистра МП в ячейку памяти. Максимальная емкость адресуемой памяти при такой организации обмена для 16-битной шины адреса составляет 64К байт.

Недостатками организации интерфейса с изолированной шиной являются следующие:

1. Обмен выполняется только через аккумулятор МП, и для передачи данного в регистр общего назначения, если аккумулятор занят, требуется выполнение четырех команд. Рассмотрим пример ввода кода из периферийного устройства 1 на регистр В.

Метка	Код	Операнд	Комментарий
	MOV	С, А	; Временное запоминание содержимого аккумулятора в регистре С
	IN	001 Q	; Ввод кода из периферийного устройства в аккумулятор
	MOV	В, А	; Передача содержимого аккумулятора в регистр В
	MOV	А, С	; Восстановление аккумулятора

Аналогичная по числу команд программа будет иметь место при выводе кода из регистра МП.

2. Число подключаемых периферийных устройств ввода или вывода информации не более 256.

3. Для обработки содержимого буферного регистра периферийного устройства, например для анализа готовности к обмену, его необходимо передать в МП.

Рассмотрим пример организации интерфейса с изолированной шиной между аналого-цифровым преобразователем (АЦП) с 10-битным выходом и МП КР580 (рис. 4-3). АЦП выполняет преобразование напряжения с аналогового датчика в цифровой код. Современные АЦП обеспечивают выход, совместимый с ТТЛ-схемами, в цифровой двоичной форме. Типичный АЦП имеет аналоговый вход, цифровой выход, вход запуска преобразования информации, после завершения которого формируется признак готовности (ПГ). Так как 10-битный код не может быть в параллельной форме введен в 8-битный МП, то ввод осуществляется последовательно-параллельно из многоцелевых буферных регистров (МБР) типа К589ИР12, имеющих трехстабильный выход. Сигнал на входе АЦП «Запуск» инициирует преобразование;

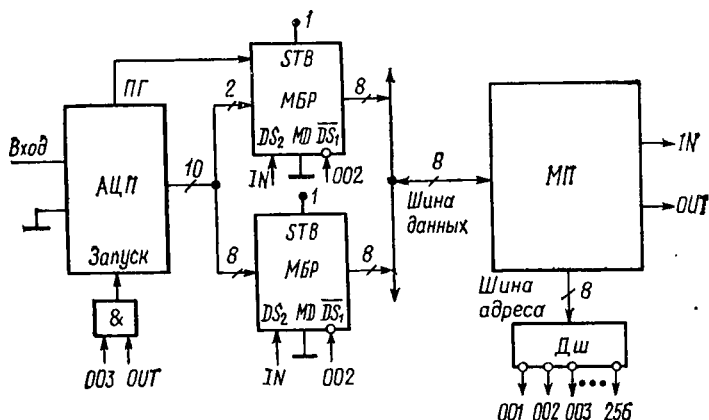


Рис. 4-3. Интерфейс микропроцессора с АЦП

он может быть сформирован при выполнении команды вывода по адресу, присвоенному данному входу (в рассматриваемом примере 003).

ПГ принимает значение 1, когда преобразование завершено, и может анализироваться МП или использоваться в качестве сигнала прерывания.

Подпрограмма в системе команд МП КР580 имеет вид:

Метка	Код	Операнд	Комментарий
	PUSH	PSW	; Временное запоминание содержимого
			; аккумулятора и флажков в стеке
	OUT	003 Q	; Запуск преобразования
TEST	IN	002 Q	; Ввод ПГ и двух старших бит кода
	ADI	200 Q	; Формирование ПГ на триггере С
	JNC	TEST	; Если ПГ=0, переход к TEST
	MOV	B, A	; Передача 2-битного кода в регистр В
	IN	001 Q	; Ввод в аккумулятор восьми младших
			; бит кода
	MOV	C, A	; Передача 8-битного кода в регистр С
	POP	PSW	; Восстановление аккумулятора и флаж-
			; ков
	RET		; Возврат из подпрограммы

Таким образом, 10-битный код с АЦП после выполнения данной программы передается в регистровую пару В, С. Как видно из рис. 4-3, для подключения АЦП к МП требуется три кода адреса. При рассмотренной организации ввода МП может много времени проводить в цикле ожидания

ПГ и поэтому целесообразно использовать режим ввода с прерыванием. В этом случае ПГ будет использоваться в качестве сигнала прерывания работы МП.

Команда CALL CONV из основной программы обеспечивает запуск АЦП в соответствии с подпрограммой CONV:

Метка	Код	Операнд	Комментарий
CONV:	EI		; Разрешить прерывание
	OUT	003 Q	; Запустить АЦП
	RET		; Возврат из подпрограммы

После завершения преобразования формируется ПГ=1 и вызывает прерывание основной программы для обращения к подпрограмме INP, которая обеспечивает ввод 10-битного кода в память ЭВМ по адресу, хранящемуся в ячейке памяти POINT:

Метка	Код	Операнд	Комментарий
INP:	PUSH	PSW	; Запоминание аккумулятора в стеке
	PUSH	H	; Запоминание (H, L) в стеке
	LHLD	POINT	; Загрузка в H, L адреса памяти
	IN	002 Q	; Ввод в МП 2-битного кода
	MOV	M, A	; Передача кода в память
	INX	H	; Увеличение адреса H, L
	IN	001 Q	; Ввод в МП 8-битного кода
	MOV	M, A	; Передача кода в память
	INX	H	; Увеличение адреса H, L
	SHLD	POINT	; Запоминание адреса в ячейке POINT
	POP	PSW	; Восстановление слова состояния и
			; (H, L)
	POP	H	
	RET		; Возврат из подпрограммы

Таким образом, 10-битный код после реализации подпрограммы INP загружается в две последовательные ячейки памяти.

Интерфейс с общей шиной. При данной организации интерфейса часть общего адресного пространства отводится для периферийных устройств, регистры которых адресуются так же, как и ячейки памяти (рис. 4-4, а). Обращение к периферийным устройствам осуществляется посредством набора команд, используемых для передачи данных с памятью. При этом команды ввода и вывода не используются, а в некоторых МП и процессорах вовсе отсутствуют. Такой принцип построения интерфейса реализуется в МП КР580 путем выделения, например, старшего разряда шины адре-

са A_{15} в качестве признака обращения к памяти (0—32К) или периферийным устройствам (32К—64К):

$$A_{15} = \begin{cases} 0, & \text{обращение к памяти;} \\ 1, & \text{обращение к периферийным устройствам.} \end{cases}$$

На рис. 4-4, б показано распределение адресного пространства в интерфейсе с общей шиной по сравнению с интерфейсом с изолированной шиной. Под периферийные устройства можно выделить любое адресное пространство при использовании в качестве признака обращения вместо A_{15} нескольких старших разрядов кода адреса.

В качестве управляющих сигналов для обращения к памяти используются сигналы чтения RD и записи WR, а к периферийным устройствам — сигналы I/OR для вывода и I/OW для ввода информации, формируемой в соответствии с рис. 4-4, а.

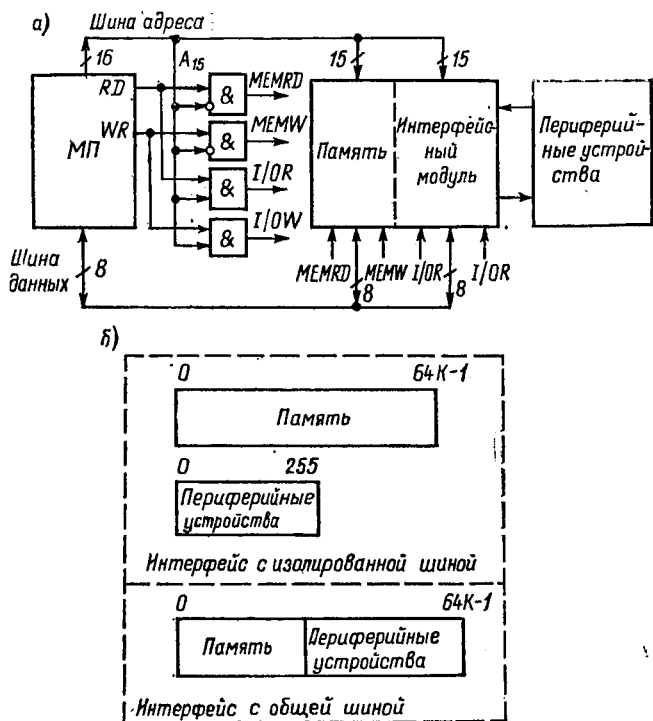


Рис. 4-4. Интерфейс с общей шиной (а); распределение адресного пространства (б)

К достоинствам интерфейса с общей шиной можно отнести следующие:

1. Расширение набора команд для обращения к периферийным устройствам, что позволяет сократить объем программ и повысить быстродействие. Примерами дополнительных команд для МП КР580 являются следующие:

MOV r, M — ввод кода в любой регистр;

MOV M, r — вывод кода из любого регистра;

MVI M, Data — загрузка в буферный регистр периферийного устройства 8-битного кода;

LHLD — ввод 16-битного кода в регистровую пару;

SHLD — вывод 16-битного кода из регистровой пары;

ANA M — логическое умножение содержимого аккумулятора и буферного регистра периферийного устройства;

ADD M — сложение содержимого аккумулятора и буферного регистра периферийного устройства.

2. Значительное увеличение числа подключаемых к МП периферийных устройств.

3. Возможность внепроцессорного обмена данными между периферийными устройствами, если в системе команд имеются команды передачи между ячейками памяти.

4. Возможность обмена информацией не только с аккумулятором, но и с любым регистром МП.

В качестве недостатка интерфейса с общей шиной можно отметить сокращение области памяти и усложнение дешифрующих схем.

В настоящее время структура интерфейса с общей шиной является очень распространенной в мини- и микро-ЭВМ.

4-3. КОМПОНЕНТЫ ДЛЯ ОРГАНИЗАЦИИ ИНТЕРФЕЙСА

Для построения интерфейса МП с периферийными устройствами используются такие компоненты общего назначения, как буферные регистры, дешифраторы, шинные формирователи, интервальные таймеры, параллельные и последовательные адаптеры, модули обработки запросов прерываний и др. Все эти компоненты с точки зрения их функциональной ориентации на различные режимы работы можно подразделить на программируемые и непрограммируемые. Быстрое развитие интерфейсных компонентов различного назначения определяется увеличением плотности упаковки, снижением стоимости интегральных схем, повышением быстродействия, гибкостью компонентов за счет их программирования, простотой подключения периферийных

устройств, упрощением программирования процедур обмена информацией. Одним из первых интерфейсных компонентов для микро-ЭВМ был 8-битный многорежимный буферный регистр К589ИР12, реализованный на основе технологии ТТЛШ и содержащий порядка 100 транзисторов. Плотность и сложность интерфейсных компонентов росли очень быстро, и, например, контроллер гибкого диска содержит более 20 000 транзисторов. При реализации такого контроллера на основе схем с малой и средней степенью интеграции требуется около 200 корпусов. Программный интерфейс стал упрощаться за счет использования сложных «интеллектуальных» контроллеров периферийных устройств с высокой степенью аппаратной реализации функций. Ниже рассматриваются вопросы организации некоторых из перечисленных компонентов.

Многорежимный буферный регистр К589ИР12 (МБР) при построении интерфейсов используется в качестве буферного регистра периферийного устройства с формированием запросов прерываний для организации мультиплексоров, стробируемых буферов, двунаправленных магистралей и других целей. Таким образом, данный компонент выполняет все основные функции, необходимые для построения интерфейса. Функциональная схема регистра К589ИР12 приведена на рис. 4-5. К589ИР12 — это 8-битный регистр на основе D-триггеров с выходными тристабильными схемами. Регистр имеет встроенную логическую схему для управления режимами работы: записью, считыванием, формированием сигнала прерывания. Высокое сопротивление позволяет подсоединять регистр непосредственно к шине данных.

Запись информации в регистр обеспечивается одной из следующих комбинаций управляющих сигналов: $\overline{DS}_1 \cdot DS_2 \cdot MD \vee STB \cdot \overline{MD}$, где DS_1 , DS_2 — входы выборки кристалла; когда $\overline{DS}_1 = 0$ и $DS_2 = 1$, регистр выбирается; STB — вход для подачи стробирующего сигнала при записи; MD — вход выбора режима, используется для определения тактирующего сигнала на входе С триггеров регистра ($\overline{DS}_1 \cdot DS_2$ при $MD = 1$ или STB при $MD = 0$) и для управления состоянием выходного буфера.

Чтение информации с регистра осуществляется одной из двух комбинаций входных сигналов: $\overline{DS}_1 \cdot DS_2 \vee MD$. Формирование сигнала прерывания $\overline{INT}$ возможно при записи или чтении информации с регистра: $\overline{DS}_1 \cdot DS_2 \vee STB$. Сброс запроса прерывания $\overline{INT}$ осуществляется сигналом очистки

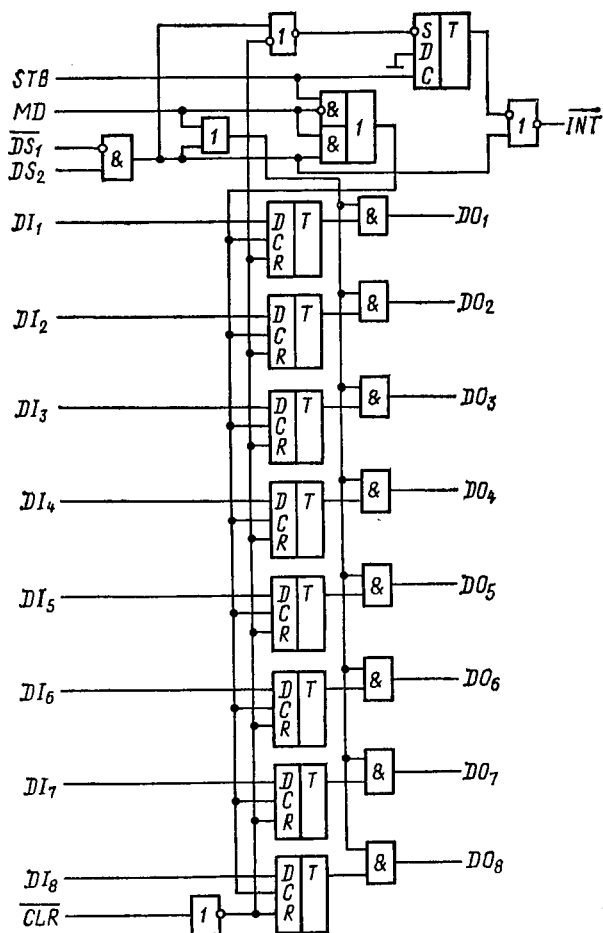


Рис. 4-5. Многорежимный буферный регистр К589ИР12

регистра $\overline{CLR}=0$, или при $DS_1 \cdot DS_2=1$ триггер прерывания устанавливается в 1.

Рассмотрим некоторые примеры использования регистра К589ИР12 в микро-ЭВМ.

Простейшее использование регистра — стробируемый буфер (рис. 4-6, а). Сигналы логической 1 (высокий потенциал) на входе STB и логического 0 (низкий потенциал) на входе MD обеспечивают постоянный прием данных в регистр. При $\overline{DS}_1 \cdot DS_2=1$ содержимое регистра передается

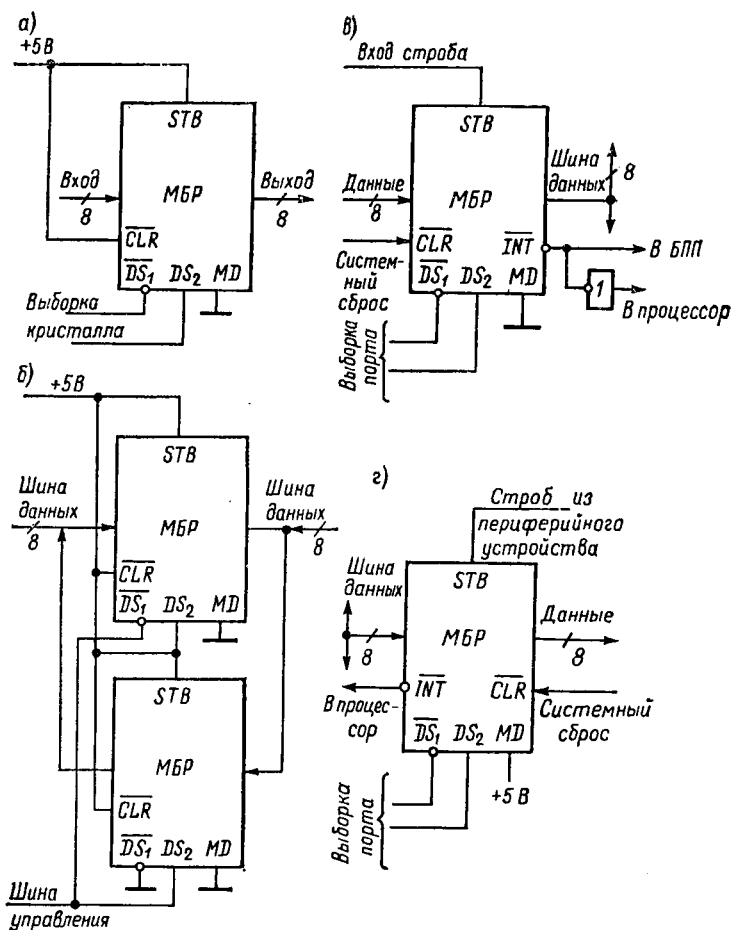


Рис. 4-6. Примеры использования МБР в интерфейсных схемах

на выход ($DO_1—DO_8$), при $\overline{DS}_1 \cdot DS_2 = 0$ выход регистра соответствует высокому сопротивлению.

Два регистра К589ИР12 могут быть использованы для организации двунаправленного шинного формирователя (рис. 4-6, б). В данной схеме один из регистров работает в режиме стробируемого буфера, а другой находится в состоянии высокого сопротивления. Такой вариант целесообразно использовать в простых микро-ЭВМ.

Использование регистра в качестве входного порта периферийного устройства приведено на рис. 4-6, в. При за-

грузке данного в порт ($STB=1$, $MD=0$) формируется сигнал прерывания $\overline{INT}$, процессор идентифицирует порт и программа ввода обеспечивает передачу кода по шине данных в процессор при $\overline{DS}_1 \cdot DS_2 = 1$.

Регистр K589IP12 может быть использован как выходной порт при передаче данных из процессора в периферийное устройство (рис. 4-6, з). Сигнал на входе STB выполняет функцию подтверждения получения данного периферийным устройством, и, кроме того, под действием этого сигнала вырабатывается сигнал прерывания $\overline{INT}$ в процессор для разрешения передачи следующего кода. Выборка порта обеспечивается сигналами на входах $\overline{DS}_1$ и DS_2 .

Блок приоритетных прерываний K589ИК14 (БПП) обеспечивает взаимодействие ЭВМ с внешней средой посредством обслуживания источников запросов в режиме прерывания (рис. 4-7). Рассмотрим вначале функциональное назначение выводов БПП:

$\overline{R}_0$ — $\overline{R}_7$ — входы запросов прерывания, низкий потенциал соответствует активному запросу. Вход $\overline{R}_7$ имеет высший приоритет, а $\overline{R}_0$ — низший;

$\overline{B}_0$ — $\overline{B}_2$ — входы для установки кода текущего приоритета;

$\overline{SGS}$ — вход выборки группы, может использоваться для формирования сигнала прерывания из БПП только на ос-

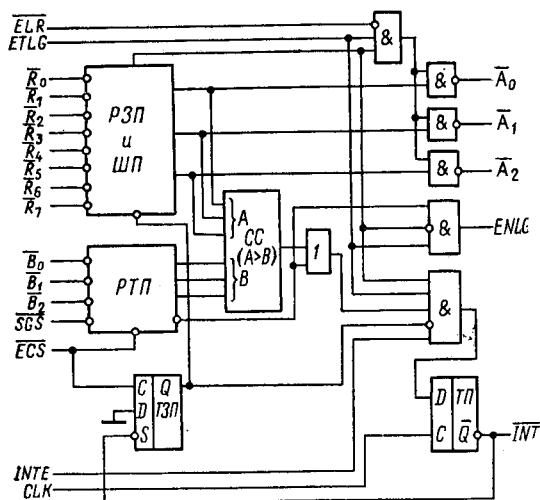


Рис. 4-7. Блок приоритетных прерываний K589ИК14

нове сравнения приоритетов запросов на входах $\bar{R}_0$ — $\bar{R}_7$, без учета кода текущего приоритета;

$\overline{ECS}$ — вход сигнала разрешения выборки кристалла, разрешает загрузку кода текущего приоритета по линиям $\bar{V}_0$ — $\bar{V}_2$;

$\overline{INTE}$ — входной сигнал разрешения прерывания, нулевой сигнал на данном входе запрещает формирование сигнала прерывания из БПП;

$\overline{CLK}$ — вход для подачи тактирующего сигнала, синхронизирующей выдачу сигнала прерывания из БПП с работой процессора;

$\bar{A}_0$ — $\bar{A}_2$ — 3-битный код для формирования вектора-адреса программы обработки прерывания;

$\overline{ELR}$ — вход для сигнала разрешения выдачи кода $\bar{A}_0$ — $\bar{A}_2$;

$\overline{INT}$ — выход сигнала прерывания из БПП;

$\overline{ETLG}$ — вход сигнала разрешения группы прерывания данного БПП; при использовании одного БПП на этот выход подается логическая 1, а при объединении нескольких БПП — сигнал с вывода $\overline{ENLG}$ БПП с более высоким приоритетом;

$\overline{ENLG}$ — выход сигнала разрешения группы прерываний БПП с более низким приоритетом.

Сигналы $\overline{ETLG}$ и $\overline{ENLG}$ обеспечивают объединение БПП с целью расширения числа уровней запросов прерывания.

Запросы прерывания поступают на входы $\bar{R}_0$ — $\bar{R}_7$ (низкий потенциал соответствует активному запросу) регистра запросов прерываний (РЗП) и шифратора приоритетов (ШП). Данная схема определяет запрос с наивысшим приоритетом среди поступивших в данный момент и формирует 3-битный код, поступающий на схему сравнения (СС). РЗП управляется триггером запрета прерываний (ТЗП) таким образом, что после формирования сигнала прерывания из БПП триггер запрета прерывания устанавливается в 1, запрещая поступление сигналов запроса прерываний в РЗП. Только загрузка кода текущего приоритета по линиям $\bar{V}_0$ — $\bar{V}_2$ на регистр текущего приоритета (РТП) при подаче сигнала на вход разрешения выборки кристалла $\overline{ECS}$ сбрасывает триггер ТЗП. При необходимости 3-битный код с РЗП сравнивается на СС с кодом текущего приоритета. СС формирует выходной сигнал только в том случае, если приоритет запроса прерывания выше текущего приоритета. Нулевой сигнал на входе $\overline{SGS}$ разрешает фактически не учиты-

вать это сравнение и формировать сигнал на входе триггера прерывания (ТП) при наличии на входе схемы И сигнала разрешения прерывания INTE, тактирующего сигнала CLK, а также сигналов ETLG и с выхода РЗП. Сигнал прерывания с БПП $\overline{INT}$ поступает на вход процессора, и после завершения текущей команды процессор переходит к программе обработки запроса прерывания. Выход элементов, формирующих код вектора-адреса $\overline{A}_2\text{—}\overline{A}_0$ и сигнал прерывания $\overline{INT}$, — с открытым коллектором, что обеспечивает возможность объединения нескольких БПП.

На рис. 4-8 приведен пример сопряжения БПП К589ИК14 с МП КР580. Для обращения к программе обслуживания прерывания в МП КР580 используется команда повторного старта RST, которая формируется на основе 3-битного вектора-адреса $\overline{A}_2\overline{A}_1\overline{A}_0$. В табл. 4-1 показано соответствие между формируемыми командами RST и входами запросов прерывания БПП $\overline{R}_0\text{—}\overline{R}_7$.

Команда RST обеспечивает запоминание содержимого программного счетчика в стеке и формирует на программном счетчике адрес первой ячейки прерывания. Команда RST формируется в регистре К589ИР12, выход которого связан с шиной данных процессора. Источники запросов прерываний подсоединяются ко входам $\overline{R}_0\text{—}\overline{R}_7$ в соответствии с назначенными приоритетами, входы $\overline{B}_0\text{—}\overline{B}_2$ подключе-

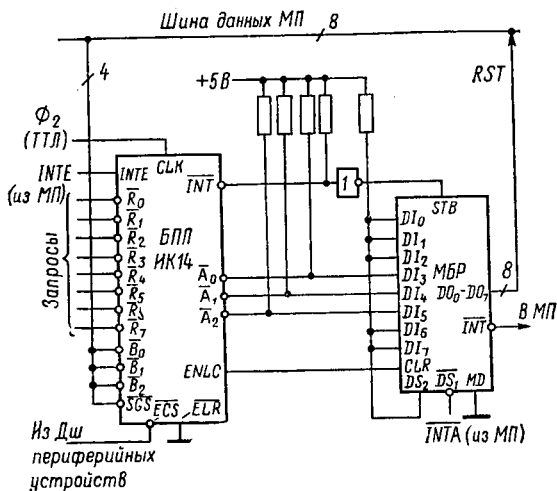


Рис. 4-8. Сопряжение БПП с микропроцессором КР580

Формирование команд RST

Вход БПП	RST	Разряды шины данных							
		D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
$\bar{R}_0$	RST7	1	1	1	1	1	1	1	1
$\bar{R}_1$	RST6	1	1	1	1	0	1	1	1
$\bar{R}_2$	RST5	1	1	1	0	1	1	1	1
$\bar{R}_3$	RST4	1	1	1	0	0	1	1	1
$\bar{R}_4$	RST3	1	1	0	1	1	1	1	1
$\bar{R}_5$	RST2	1	1	0	1	0	1	1	1
$\bar{R}_6$	RST1	1	1	0	0	1	1	1	1
$\bar{R}_7$	RST0	1	1	0	0	0	1	1	1

ны к шине данных, по которой в РТП программным путем загружается код текущего приоритета. Сигнал прерывания $\overline{INT}$ из БПП фиксируется в триггере прерывания регистра K589IP12. Команда RST передается по шине данных в регистр команд процессора с МБП при поступлении из процессора сигнала подтверждения прерывания $\overline{INTA}$.

При необходимости обслуживания более восьми приоритетных источников прерывания можно объединять БПП с целью увеличения числа входов. На рис. 4-9 показан пример организации системы прерываний на 16 входов. Для объединения БПП выход ENLG БПП1 подсоединяется ко входу ETLG БПП2. Так как требуется обращаться к 16 программам, то использование команды RST невозможно и поступаем следующим образом. При получении процессором сигнала прерывания $\overline{INT}$ следует выполнить команду ввода в аккумулятор процессора содержимого регистра K589IP12, рассматривая его как порт периферийного устройства, и использовать этот код как 8-битное смещение, загружаемое в программный счетчик в соответствии с нижеприведенной программой:

Метка	Код	Операнд	Комментарий
	PUSH	PSW	; Запоминание содержимого аккумулятора в стеке
	IN	PORT	; Ввод в аккумулятор кода с МБП (PORT — адрес, присвоенный МБП)
	MOV	L, A	; Передача содержимого аккумулятора в регистр L
	MVI	H, BASE	; Загрузка в регистр H базового адреса
	PCHL		; Передача (H, L) в программный счетчик

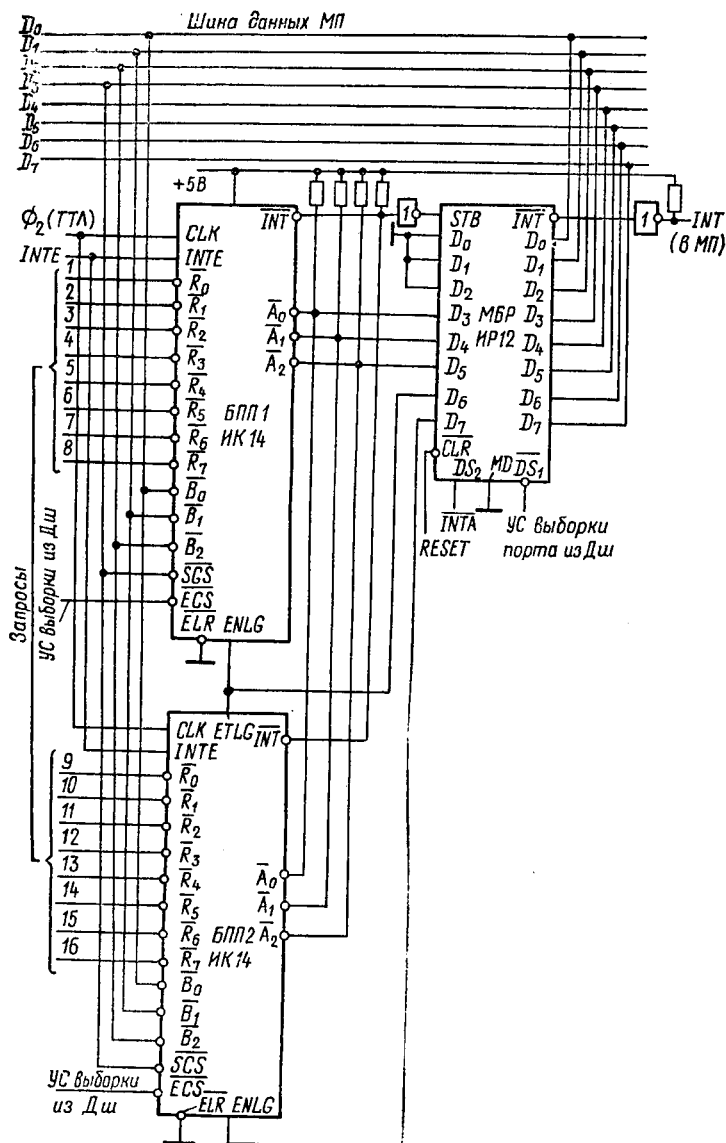
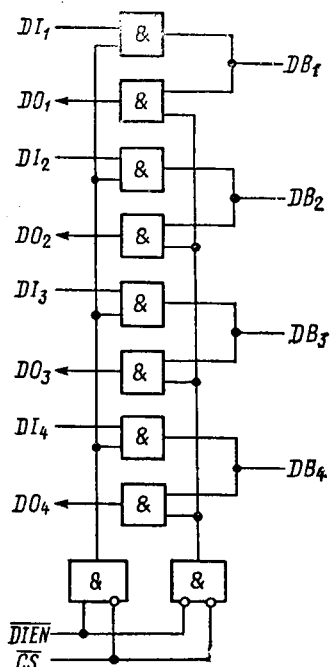


Рис. 4-9. Объединение БПП типа К589ИК14

Рис. 4-10. Шинный формирователь

Аналогичным образом можно объединять большее число БПП для увеличения числа входов системы обработки прерываний. Необходимо заметить, что есть другие варианты объединения БПП, которые рассмотрены в литературе по проектированию микропроцессорных систем.

Шинные формирователи K589АП16 и K589АП26. Для увеличения нагрузочной способности шин в микропроцессорные комплекты входят шинные формирователи, которые могут выполнять также дополнительные функции: осуществление электрической совместимости компонентов, отключение источников и приемников информации от шин, так как обычно шинные формирователи имеют три выходных состо-



яния, обеспечение двунаправленного обмена информацией. Типичными представителями шинных формирователей являются 4-битные двунаправленные формирователи K589АП16 и K589АП26; причем последний имеет инверсные выходы. Структурная схема шинного формирователя K589АП16 приведена на рис. 4-10. Каждая линия шинного формирователя содержит две логические схемы И с тремя выходными состояниями. С одной стороны шинного формирователя входы и выходы схем И объединены (DB_1 — DB_4), эти линии используются для интерфейса формирователя со схемами, совместимыми с ТТЛ. С другой стороны входы (DI_1 — DI_4) и выходы (DO_1 — DO_4) шинного формирователя являются отдельными с целью получения максимальной гибкости при использовании данной схемы. Они, конечно, могут быть соединены, например, при использовании шинного формирователя с двунаправленной шиной МП КР580. Вход $\overline{CS}$ управляет выборкой кристалла, при $\overline{CS}=0$ (низкий потенциал) формирователь выбирается и в зависимости от управляющего сигнала $\overline{DIEN}$ определяется направление передачи информации: при $\overline{DIEN}=1$ содержимое ли-

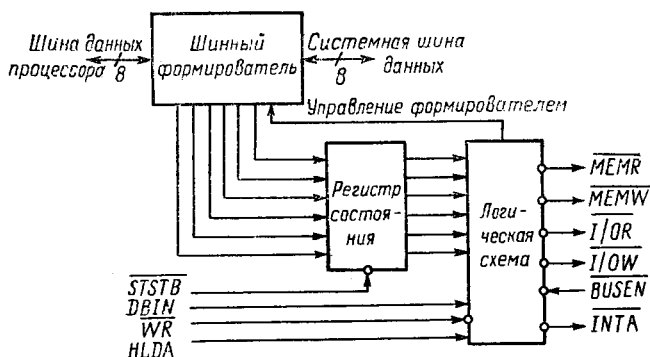


Рис. 4-11. Системный контроллер и шинный формирователь

ний DI_1 — DI_4 передается на линии DB_1 — DB_4 , а при $\overline{DIEN}=0$ имеет место передача с линией DB_1 — DB_4 на линии DO_1 — DO_4 .

Системный контроллер и шинный формирователь БГ28 (СКШФ) предназначен для формирования системных управляющих сигналов (СУС), для непосредственного сопряжения МП КР580 с памятью и интерфейсными модулями периферийных устройств. Двухнаправленный ШФ включен в БИС для увеличения нагрузочной способности шин данных МП. Структурная схема СКШФ приведена на рис. 4-11.

В начале каждого машинного цикла микропроцессор по шине данных выдает код, определяющий тип машинного цикла, этот код по шине данных загружается в регистр состояния (РС) БГ28 при поступлении сигнала $\overline{STSTB}$ из генератора тактовых сигналов. РС связан с логической схемой, которая формирует следующие управляющие сигналы. $\overline{MEMR}$ — сигнал чтения информации из памяти; $\overline{MEMW}$ — сигнал записи информации в память; $\overline{I/OR}$ — сигнал чтения информации из устройства ввода; $\overline{I/OW}$ — сигнал записи информации в устройство вывода; $\overline{INTA}$ — сигнал подтверждения прерывания.

Входными сигналами в СКШФ являются: $\overline{DBIN}$ — сигнал приема информации в МП; $\overline{WR}$ — сигнал выдачи информации из МП; $\overline{HLDA}$ — сигнал подтверждения захвата МП; $\overline{BUSEN}$ — сигнал, соответствующий 1, который переводит шину данных и линии управляющих сигналов в состояние высокого сопротивления для отключения МП от системных шин при организации обмена в режиме прямого доступа к памяти.

4.4. ПРОГРАММИРУЕМЫЙ АДАПТЕР ПОСЛЕДОВАТЕЛЬНОГО ИНТЕРФЕЙСА

В некоторых применениях мини- и микро-ЭВМ необходимо осуществлять обмен данными в последовательном формате. Такой обмен характерен для интерфейсов медленноредействующих или удаленных периферийных устройств, а также для интерфейсов распределенных систем обработки. В большинстве микропроцессорных семейств имеются программируемые БИС, предназначенные для интерфейса параллельного процессора с последовательным периферийным оборудованием. Эти БИС называются универсальными приемопередатчиками, программируемым связным интерфейсом или адаптерами последовательной связи.

Единицей обмена в последовательном формате является символ, представленный в одной из систем кодирования и содержащий 5—8 бит. Примером 5-битного кода служит международный телеграфный код № 2. Биты кодируются наличием тока в линии (1 или MARK) либо отсутствием тока (0 или SPACE).

На практике применяются два режима последовательного обмена: асинхронный (старт-стопный) и синхронный. В первом из них каждый символ передается автономно и передача может быть начата в любой момент времени. Стандартный формат символа в асинхронной связи приведен на рис. 4-12, а. Передача начинается со стартового бита, т. е. прекращения тока в линии связи. Затем в зависимости от применяемого кода передаются 5—8 бит собственно символа. Передача символа завершается необязательным битом четного или нечетного паритета и одним, полутора или двумя стоповыми битами. После этого может быть начат цикл передачи следующего символа. Скорость пере-

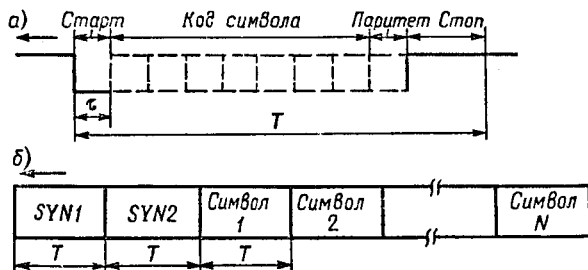


Рис. 4-12. Формат асинхронной передачи (а), формат синхронной передачи (б)

дачи измеряется либо числом символов в секунду ($1/T$), либо числом битовых посылок в секунду ($1/\tau$).

Синхронная передача начинается с одного или двух специальных символов синхронизации SYN1 и SYN2, после которых последовательно без всяких разделителей передаются 5—8-битные коды символов с обязательными битами четного или нечетного паритета (рис. 4-12, б).

В обоих режимах необходимо контролировать передачу по битам паритета (если они предусмотрены в формате), выдерживать необходимые временные соотношения, а в асинхронном режиме дополнительно контролировать установленный формат (кадр) символа.

Из рис. 4-12, а видно, что асинхронному режиму свойственна значительная избыточность. Если, например, код символа содержит 5 бит, то вместе с ним могут передаваться до 4 служебных бит. Непроизводительное использование линии связи доходит до 44 %, поэтому асинхронный режим применяется в системах с небольшой скоростью передачи или с нерегулярным обменом.

Некоторые БИС последовательного интерфейса могут работать только в синхронном (универсальные синхронные приемопередатчики — УСПП) или в асинхронном (универсальные асинхронные приемопередатчики — УАПП) режиме. Наибольшей же гибкостью отличаются БИС универсальных синхронно-асинхронных приемопередатчиков (УСАПП), которые допускают программирование любого режима работы. Типичным примером УСАПП является микросхема КР580ВВ51, реализующая практически любой способ последовательной связи. Далее ради краткости она называется адаптером последовательного интерфейса или просто адаптером.

Адаптер должен автоматически производить параллельно-последовательные (при передаче) и последовательно-параллельные (при приеме) преобразования, а для процессора выглядеть устройством параллельного ввода и вывода. Он может работать в полудуплексном или дуплексном режимах, обеспечивающих одновременную одностороннюю или двухстороннюю связь. Наличие в схеме двойных буферов приводит к тому, что для реакции на запрос прерывания адаптера в распоряжении процессора имеется временной интервал T передачи одного символа. Адаптер также генерирует и воспринимает сигналы управления модемом.

На рис. 4-13 приведены структурная схема и программная модель адаптера, а на рис. 4-14 подключение его к системной шине. Рассмотрим основные компоненты адаптера.

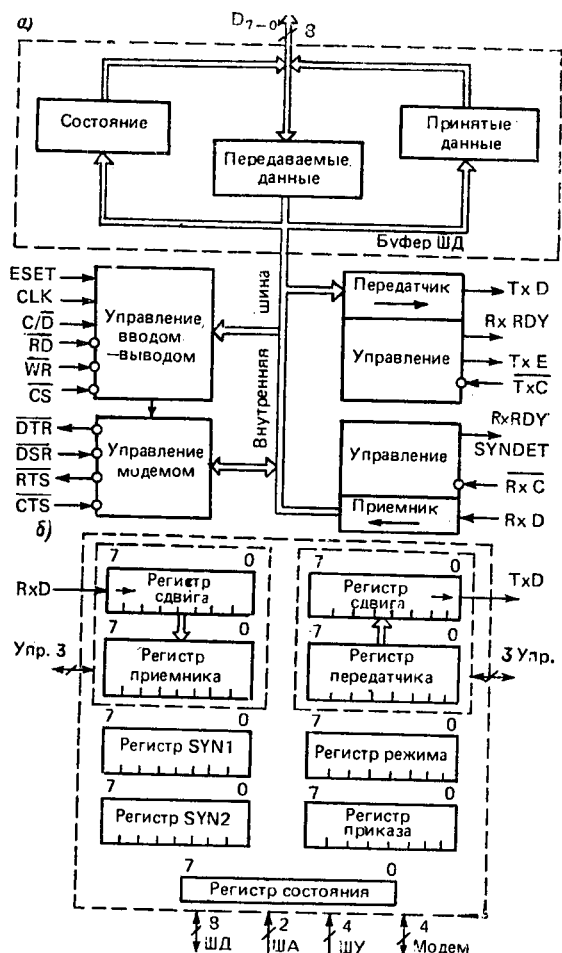


Рис. 4-13. Структурная схема (а) и программная модель (б) микросхемы KP580BB51

Параллельный 8-битный двунаправленный буфер шины данных D_{7-0} с тристабильными каскадами служит для передачи собственно данных, управляющих слов и информации состояния. Обычно обмен инициируется командами ввода IN и вывода OUT.

Схема управления воспринимает сигналы с шины управления и генерирует внутренние управляющие сигналы. Все

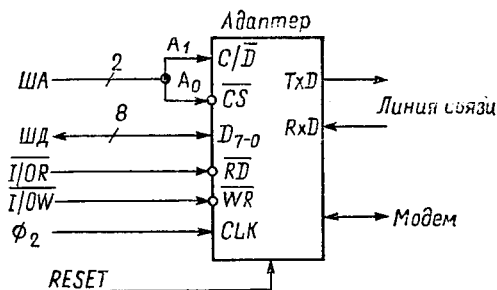


Рис. 4-14. Подключение адаптера к системной шине

составе имеются регистр слова режима и регистр слова приказа, которые хранят управляющие слова функционального определения адаптера. В адаптер подаются шесть входных сигналов:

RESET (сброс) — Н-активный сигнал сброса с минимальной длительностью 6 периодов синхронизации. После действия этого сигнала адаптер переводится в «холостой» режим и остается в нем до загрузки управляющих слов.

CLK (синхронизация), подключается ко второй фазе системного генератора синхронизации. Частота сигналов CLK минимум в 30 раз больше максимальной скорости обмена данными.

$\overline{RD}$ (считывание) — L-активный сигнал, инициирующий передачу данных или состояния из адаптера на шину данных.

$\overline{WR}$ (запись) — L-активный сигнал загрузки информации с шины данных в адресуемый регистр адаптера.

$C/\overline{D}$ (управление/данные) — сигнал идентификации передачи данных или управляющих слов.

$\overline{CS}$ (выбор кристалла) — L-активный сигнал, разрешающий связь между адаптером с шиной данных.

В табл. 4-2 приведены функции, выполняемые при различных комбинациях управляющих сигналов. Отметим, что состояние $\overline{RD}=0$ и $\overline{WR}=0$ является запрещенным и дает непредсказуемый результат. Схема управления адаптера содержит цепи синхронизации, поэтому сигналы $\overline{RD}$ и $\overline{WR}$ могут быть несинхронными со входом CLK.

Узел передатчика со схемой управления выполняет все функции, связанные с передачей последовательных данных: воспринимает параллельные коды символов от процессора,

Функции, соответствующие комбинациям УС

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	Действие
0	0	0	1	Считывание символа из адаптера
0	1	0	1	Считывание из адаптера слова состояния
0	0	1	0	Запись символа в адаптер
0	1	1	0	Запись в адаптер управляющего слова
0	×	1	1	Адаптер отключен от шины данных
1	×	×	×	

автоматически вводит необходимые служебные биты и символы синхронизации и выдает последовательный поток на выход TxD. К этому узлу относятся следующие внешние сигналы:

TxD (выход передатчика) — выходная линия, по которой действуют сигналы передаваемых данных.

$\overline{TxC}$ (синхронизация передатчика) — входной сигнал, управляющий скоростью передачи данных. Спад $\overline{TxC}$ «выдвигает» последовательные биты на выход TxD. В синхронном режиме скорость передачи соответствует частоте сигнала $\overline{TxC}$, а в асинхронном режиме программируется как 1, 1/16 или 1/64 частоты сигнала $\overline{TxC}$.

TxE (пустой передатчик) — Н-активный выходной сигнал, обозначающий отсутствие в адаптере символа для передачи. Его можно использовать для идентификации в полудуплексной связи окончания передачи и коммутации линии на прием. В синхронном режиме Н-уровень TxE указывает, что символ вовремя не загружен в адаптер и в качестве «заполнителей» автоматически передаются символы синхронизации. Сигнал TxE сбрасывается при загрузке символа в адаптер.

TxRDY (готовность передатчика) — Н-активный выходной сигнал, определяющий готовность передатчика к восприятию символа. Используется для прерывания процессора или проверяется при считывании состояния адаптера (бит 0). При загрузке в передатчик нового символа сигнал TxRDY сбрасывается.

Узел приемника с автономной схемой управления воспринимает последовательные данные со входа RxD , преобразует их в параллельный формат, контролирует и исключает служебные биты и символы синхронизации, а затем передает «собранный» символ в процессор. К приемнику относятся следующие внешние сигналы:

$\overline{RxD}$ (вход приемника) — входная линия, по которой передаются сигналы принимаемых последовательных данных.

$\overline{RxC}$ (синхронизация приемника) — входной сигнал, определяющий скорость приема символов. В синхронном режиме скорость приема соответствует частоте сигнала $\overline{RxC}$, а в асинхронном частота $\overline{RxC}$ кратна скорости приема. Данные вводятся в адаптер по фронту $\overline{RxC}$. Обычно передача и прием осуществляются с одинаковой скоростью, поэтому сигналы $\overline{TxC}$ и $\overline{RxC}$ должны иметь одинаковую частоту; для этого они объединяются и подключаются к одному генератору синхронизации.

$RxRDY$ (готовность приемника) — выходной сигнал, уровень которого свидетельствует о наличии в адаптере принятого символа. Его можно подключить на вход прерывания процессора или проверить значение при считывании состояния адаптера (бит 1). Когда процессор вводит символ из адаптера, сигнал $RxRDY$ сбрасывается.

$SYNDET$ (обнаружение синхронизации) — Н-активный сигнал синхронного режима, который может быть запрограммирован как выходной или входной. Если он запрограммирован как выходной, то при обнаружении символа SYN на выходе $SYNDET$ формируется высокий уровень в момент времени, соответствующий середине последнего бита (в случае работы с двумя символами синхронизации это относится к $SYN2$). При считывании состояния адаптера сигнал сбрасывается. Когда сигнал $SYNDET$ определен как входной, подача на него высокого уровня фиксирует момент начала приема символа. Адаптер обслуживает четыре линии управления модемом, сигналы которых при необходимости могут использоваться и для других функций. Выходными сигналами можно управлять с помощью слов приказов, а входные сигналы считываются в слове состояния.

$\overline{DSR}$ (готовность модема) — входной сигнал от модема, означающий его готовность к работе (бит 7 в слове состояния).

$\overline{DTR}$ (готовность терминала) — выходной сигнал (бит 1 слова приказа).

RTS (запрос передачи) — выходной сигнал (бит 5 слова

приказа).

$\overline{CTS}$ (передача) — входной сигнал, разрешающий адаптеру передавать данные; обычно генерируется модемом в ответ на $\overline{RTS}$.

Для программирования адаптера необходимо загрузить несколько управляющих слов, определяющих скорость передачи, длину символа, число стоповых бит, режим работы и условия контроля (четный или нечетный паритет). Кроме того, для синхронного режима требуется определить внешнюю или внутреннюю синхронизацию, а также символ (или два символа) синхронизации. После программирования адаптер готов выполнять свои функции. На входе $TxRDY$ формируется высокий уровень, сигнализирующий о готовности передатчика принять символ для передачи. Приемник сдвигает последовательные биты с линии RxD и после приема всего символа генерирует сигнал $RxRDY$. Адаптер не начинает передачу до тех пор, пока не установлен бит разрешения передатчика $TxEN$ в слове приказа. На выходе TxD передатчика до сброса формируется высокий уровень.

Управляющие слова, определяющие функциональную конфигурацию адаптера, должны загружаться сразу после операции сброса. Структурная схема инициализации и типичный блок данных приведены на рис. 4-15, а. Управляющие слова имеют два формата: слово режима и слово приказа. Слово режима задает общие рабочие характеристики адаптера и обязательно загружается первым, так как оно необходимо для коммутации схем прибора. После слова режима загружаются один или два символа SYN , если был определен синхронный режим. Символ $SYN2$ пропускается, если адаптер программируется для работы с одним символом синхронизации. Когда же в слове режима определен асинхронный обмен, пропускаются оба символа SYN . Последним в адаптер загружается слово приказа, определяющее его конкретные действия в соответствии со словом режима.

Для правильной работы адаптера управляющие слова должны следовать в определенном порядке. Слово режима записывается в адаптер сразу после сброса прибора, иницируемого сигналом $RESET$ или специальным битом в слове приказа. Слово приказа подается за словом режима или после символа (символов) синхронизации. Загрузка всех управляющих слов производится командами OUT при следующих значениях управляющих сигналов: $C/D=1$, $\overline{CS}=0$, $\overline{RD}=1$, $\overline{WR}=0$.

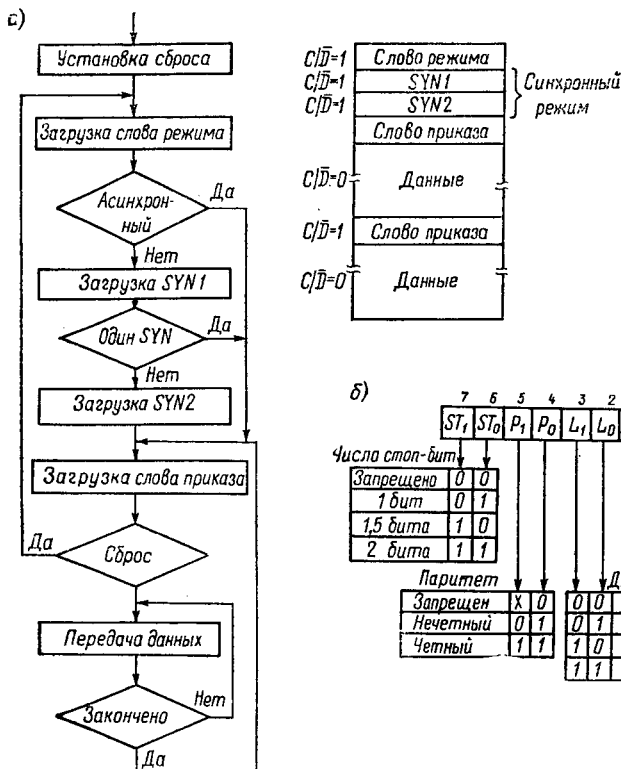


Рис. 4-15. Структурная схема программирования и блок данных адаптера (а), формат слова режима адаптера (б)

Слово режима, формат которого приведен на рис. 4-15, б, имеет четыре двухбитных поля и загружается во внутренний регистр режима. Первое поле $D_{1,0}$ определяет синхронный или асинхронный режим работы, а в асинхронном режиме содержит множитель 1, 16 или 64, соответствующий коэффициенту деления частоты сигналов синхронизации. Второе поле $D_{3,2}$ задает число бит в коде символа. При программировании длины символа менее 8 бит данные выравниваются вправо, т. е. при записи символа в адаптер неиспользуемые биты не определены, а при считывании символа они содержат нули. Третье поле $D_{5,4}$ управляет формированием бита паритета. Бит D_4 называется битом разрешения паритета PEN, а бит D_5 — битом четного паритета

ЕР. Бит паритета, если он запрограммирован, в длине символа не учитывается. В зависимости от режима поле $D_{7,6}$ имеет двоякий смысл. В асинхронном режиме оно определяет число стоповых бит, передаваемых с символом (приемник всегда работает с одним стоповым битом), а в синхронном режиме управляет процессом синхронизации. Бит D_6 программирует сигнал SYNDET как входной ($D_6=1$) или выходной ($D_6=0$), а бит D_7 определяет один ($D_7=1$) или два ($D_7=0$) символа синхронизации.

После того как слово режима запрограммирует общие функции адаптера и будут загружены один или два символа синхронизации синхронного режима, адаптер готов к обмену данными. Загружаемое после этого слово приказа задает действительную операцию адаптера. В слове приказа задаются разрешение передачи или приема, сброс ошибок, управление модемом и некоторая другая информация. Отдельные биты приказа имеют следующий смысл:

D_0 (TxEN — разрешение передачи); единичное значение разрешает передачу данных, а нулевое — запрещает.

D_1 (DTR); если этот бит содержит единицу, на выходе $\overline{\text{DTR}}$ адаптера формируется низкий уровень.

D_2 (RxЕ — разрешение приема) действует как маска сигнала готовности приемника RxRDY (1 — прием разрешен, 0 — прием запрещен). Когда устанавливается бит RxЕ, приемник может содержать один или два символа.

D_3 (SBRK — разрыв) при единичном значении на линии TxD формирует низкий уровень, а нулевое значение соответствует нормальной работе.

D_4 (ER — сброс ошибок) при единичном значении форсирует сброс ошибок паритета, переполнения и нарушения кадра.

D_5 (RTS); если этот бит содержит единицу, на выходе $\overline{\text{RTS}}$ адаптера формируется низкий уровень.

D_6 (IR — внутренний сброс) при единичном значении инициирует внутренний сброс адаптера, после которого должно загружаться слово режима.

D_7 (EH — режим поиска); в синхронном режиме единичное значение этого бита вводит режим поиска символа синхронизации.

При организации последовательного интерфейса возникает необходимость проанализировать состояние адаптера. Состояние адаптера можно считать в любой момент времени посредством команды ввода IN, обеспечивающей формирование сигнала $C/\overline{D}=1$. Кроме уже рассмотренных сиг-

налов DSR(D₇), SYNDET(D₆), TxE(D₂), RxRDY(D₁) и TxRDY(D₀) слово состояния содержит следующие три бита, или флажка, ошибок, формируемых в процессе приема данных:

D₅(FE — ошибка кадра) устанавливается в асинхронном режиме, если в конце любого символа не обнаружен стоповый бит.

D₄(OE — ошибка переполнения) устанавливается в любом режиме, когда процессор вовремя не считал символ (этот символ теряется).

D₃(PE — ошибка паритета) устанавливается при обнаружении в принятом символе нарушения паритета.

Все флажки ошибок сбрасываются, когда бит D₄ слова приказа установлен в единицу. Следует специально подчеркнуть, что возникновение любого ошибочного условия не останавливает работу адаптера.

Остановимся несколько подробнее на работе прибора в обоих режимах. В асинхронном режиме, когда процессор загружает символ, адаптер автоматически вводит стартовый бит, заданное число стоповых бит и бит паритета, если он запрограммирован. Затем символ в последовательном формате выводится на контакт TxD. Битовые послышки выдвигаются по спаду сигнала $\overline{\text{TxC}}$ со скоростью, определенной множителем 1, 16 или 64 в слове режима. До тех пор пока в адаптер не загружен символ, на выходе TxD действует высокий уровень, если не был установлен бит D₃ слова приказа.

При приеме нормально на входе RxD действует высокий уровень. Спад считается началом стартового бита. Достоверность его проверяется повторным опросом (стробированием) сигнала RxD в центре битовой посылки. Если здесь снова фиксируется низкий уровень, то посылка считается достоверным стартовым битом и в адаптере запускается специальный счетчик, обеспечивающий опрос сигнала на линии RxD в центрах битовых посылок. Когда же второй опрос линии RxD фиксирует высокий уровень, это означает появление на линии связи импульса помехи либо разрешение приема данных в середине символа. В любом случае приемник прекращает свои действия и готовится к восприятию нового символа.

При обнаружении ошибки паритета или низкого уровня на месте стопового бита устанавливаются соответствующие флажки ошибок. После приема всего символа он загружается в параллельный буфер и формируется высокий уровень сигнала RxRDY готовности приемника. Если процес-

сор не считал из буфера предыдущий символ, он замещается принятым символом и устанавливается флажок ошибки переполнения.

В синхронном режиме до загрузки в адаптер первого символа, которым обычно является символ SYN, на выходе TxD действует высокий уровень. Передача битовых посылок инициируется по спаду сигнала $\overline{\text{TxC}}$, а скорость передачи соответствует частоте этого сигнала. После начала передачи на выходе TxD должен формироваться непрерывный поток данных. Если же процессор не успел вовремя записать символ в адаптер, на выходе автоматически появляются символы SYN (один или два), а на выходе TxE устанавливается высокий уровень.

Синхронизация символов при приеме в синхронном режиме может быть внутренней или внешней. Если словом режима запрограммирована внутренняя синхронизация (линия SYNDET — выход), приемник начинает работу в режиме поиска. Содержимое буфера приемника непрерывно сравнивается с символом SYN до достижения соответствия. Если же адаптер запрограммирован на два символа SYN, то сравнивается и следующий принимаемый символ. После обнаружения одного или двух символов SYN адаптер заканчивает режим поиска, входит в синхронизм и принимает собственно данные. Момент достижения синхронизации фиксируется установкой высокого уровня на линии SYNDET, который автоматически сбрасывается при считывании состояния.

Внешняя синхронизация (линия SYNDET — вход) достигается подачей высокого уровня на вход SYNDET. Его можно снять спустя один период сигнала RxC.

Ошибки паритета и переполнения контролируются так же, как в асинхронном режиме. Если синхронизация потеряна, процессор специальным словом приказа ($D_7=1$) может ввести режим поиска.

В заключение приведем простые подпрограммы инициализации адаптера, записи и считывания символов, которые составлены при следующих предположениях:

символический адрес CMD обеспечивает загрузку в адаптер управляющих слов ($\overline{\text{CS}}=0$, $\text{C}/\overline{\text{D}}=1$), адрес CNTRL — считывание из адаптера слова состояния ($\overline{\text{CS}}=0$, $\text{C}/\overline{\text{D}}=1$), адрес DATIN — считывание принятых данных ($\overline{\text{CS}}=0$, $\text{C}/\overline{\text{D}}=0$) и адрес DOUT — запись передаваемых данных ($\overline{\text{CS}}=0$, $\text{C}/\text{D}=0$);

используется дуплексный асинхронный режим со следующим форматом: 8 бит кода символа, бит нечетного паритета, два стоповых бита и множитель 16X; эту конфигурацию определяет слово режима 11011110 (DE);

реагируя на прерывание, процессор вводит состояние адаптера и проверяет необходимость ввода или вывода символа; если идентифицируется прерывание от другого устройства, осуществляется переход к программе опроса с начальным адресом POLL;

при обнаружении сшибок вызывается подпрограмма ERROR;

принятый символ передается в буферную область памяти, текущий адрес которой хранится в двух смежных ячейках с адресом RBUF;

очередной передаваемый символ берется из буферной области памяти с указателем в ячейках TBUF.

Метка	Код	Операнд	Комментарий

; Подпрограмма инициализации адаптера

DI			; Запрещение прерываний
XRA	A		; Сброс аккумулятора
OUT	CMD		; Вывод три раза для надежного сброса
OUT	CMD		; адаптера
OUT	CMD		
MVI	A, 40H		; Вывод слова приказа внутреннего сброса
OUT	CMD		; са адаптера
MVI	A, 0DEH		; Вывод слова режима — программирование адаптера
OUT	CMD		; вание адаптера
MVI	15H		; Вывод слова приказа
OUT	CMD		
EI			; Разрешение прерываний
RET			; Возврат

; Подпрограмма обслуживания прерываний адаптера

PUSH	PSW		; Запоминание модифицируемых регистров процессора
PUSH	H		; ров процессора
IN	CNTRL		; Ввод состояния адаптера
RRC			; Бит переноса фиксирует готовность приемника
RRC			; ;
JC	RCV		; Переход к обслуживанию приема
RLC			; Бит переноса фиксирует готовность передатчика
RLC			; редатчика
JNC	POLL		; Прерывание не от адаптера
LHLD	TBUF		; Указатель области передачи
MOV	A, M		; Очередной передаваемый символ
OUT	DOUT		; Загрузка символа в адаптер
INX	H		; Продвижение и запоминание указателя
JHLD	TBUF		
JMP	EXIT		; Конец передачи
RCV: LHLD	RBUF		; Указатель области приема

	IN	DATIN	; Ввод принятого символа
	MOV	M, A	; Передача в область приема
	INX	H	; Продвижение и запоминание указателя
	SHLD	REUF	
	IN	CNTRL	; Ввод состояния адаптера
	ANI	38H	; Проверка ошибок
	CNZ	ERROR	; Обработка ошибочной ситуации
EXIT:	POP	H	; Восстановление регистров процессора
	POP	PSW	
	EI		; Разрешение прерываний
	RET		; Возврат

4.5. ПРОГРАММИРУЕМЫЙ АДАПТЕР ПАРАЛЛЕЛЬНОГО ИНТЕРФЕЙСА

В любой ЭВМ необходимы средства обмена данными с разнообразными периферийными устройствами. В зависимости от условий конкретного применения и характеристик периферийного оборудования передача данных производится в параллельном или последовательном формате. Общность функций ввода-вывода стимулировала разработку БИС периферийных адаптеров, представляющих собой гибкие программируемые приборы, ориентированные исключительно на ввод-вывод. В настоящее время такие адаптеры стали обязательными компонентами практически всех микропроцессорных семейств. Как пример типичного параллельного периферийного адаптера, или программируемого периферийного интерфейса, ниже рассмотрена микросхема KP580BB55.

На рис. 4-16 приведена структурная схема адаптера и его программная модель. Подключение периферийного оборудования производится через три двунаправленных 8-битных порта (или канала) А, В и С. Интерфейс с системной шиной осуществляется с помощью 14 линий:

D_{7-0} — двунаправленная шина данных с трехстабильными каскадами.

A_1, A_0 — линии адреса, которые выбирают внутренний регистр адаптера, коммутируемый на шину данных: 00 — порт А, 01 — порт В, 10 — порт С и 11 — регистр управления.

$\overline{CS}$ — L-активный вход выбора кристалла; высокий уровень запрещает, а низкий разрешает связь прибора с системной шиной данных.

$\overline{RD}$ — L-активный вход считывания информации из адресуемого по линиям $A_{1,0}$ регистра на шину данных.

$\overline{WR}$ — L-активный вход записи информации с шины данных в адресуемый внутренний регистр адаптера.

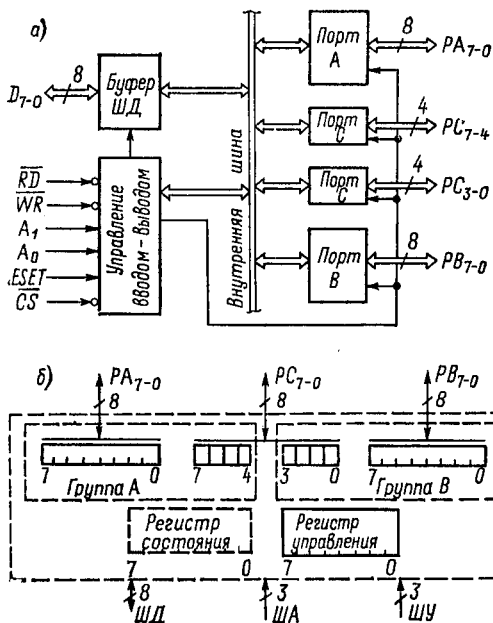


Рис. 4-16. Структурная схема (а) и программная модель (б) микросхемы КР580BB55

RESET — Н-активный сигнал сброса для приведения прибора в начальное состояние; при действии сброса регистр управления обнуляется, а все три порта переводятся в режим ввода.

Отметим, что считывание из регистра управления ($\overline{RD}=0$, $A_{1,0}=11$) не допускается, а одновременные запись и считывание ($\overline{RD}=0$ и $\overline{WR}=0$) приводят к непредсказуемому результату.

Программирование и обмен данными с адаптером осуществляются командами ввода IN и вывода OUT, при выполнении которых на линиях A_{7-0} (и одновременно A_{15-8}) находится адресная информация. Приемником и источником данных в микропроцессоре является аккумулятор. Входы $A_{1,0}$ адаптера обычно подключаются к младшим линиям шины адреса, а подключение входа $\overline{CS}$ зависит от принятого способа выбора адаптеров, если их несколько. В линейном выборе с кодами адреса 011111XX, 101111XX, ..., 111110XX входы $\overline{CS}$ подключаются без дополнительного

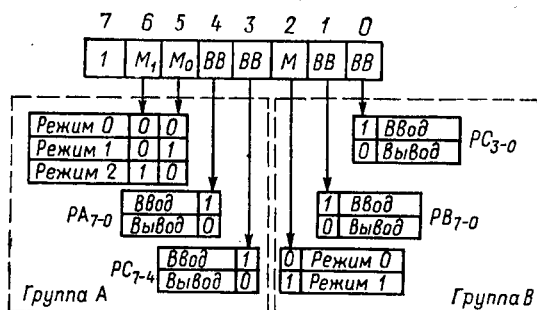


Рис. 4-17. Формат приказа определения режима

дешифратора к соответствующим (нулевым) линиям адреса A₇₋₂. В этом способе система может иметь до шести адаптеров; при большем числе адаптеров потребуется дешифратор с L-активными выходами, вход которого подключается к линиям A₇₋₂.

Программирование адаптера заключается в загрузке управляющего слова (приказа) в регистр управления. Формат приказа определения режима, идентифицируемого условием D₇=1, приведен на рис. 4-17. Отметим, что 8-битные порты А и В не разделены, а линии порта С разделены на две 4-битные группы, представляющие собой два независимых порта. Например, приказ конфигурации порта А на ввод в режиме 0, порта В на вывод в режиме 1, бита 3 порта С на ввод и бита 7 порта С на вывод имеет код 10010101. Этот код загружается в аккумулятор командой MVI и выводится в регистр управления адаптера командой OUT.

Приказ с нулевым старшим битом D₇=0 используется для установки или сброса любого бита порта С. Биты D₆₋₄ не используются и обычно содержат нули; биты D₃₋₁ содержат двоичный номер (адрес) модифицируемого бита порта С, а бит D₀ задает установку (D₀=1) или сброс (D₀=0) адресуемого бита. Например, приказ сброса бита 2 порта С имеет код 00000100. Приказы данного формата используются для формирования сигналов квитирования в режимах 1 или 2. Формирование положительного импульса на выходе бита 3 порта С реализуется следующими командами:

```
MVI    A, 00000111B    ; Установка бита 3
OUT     CWR
MVI    A, 00000110B    ; Сброс бита 3
OUT     CWR
```

Здесь CWR — символический адрес регистра управления, который определяется интерфейсом адаптера с системной шиной. Например, им может быть $CWR = 0F7H$. Отметим, что в приведенном фрагменте можно сэкономить один байт, заменив вторую команду MVI на команду DCR A.

Аналогичная функция установки и сброса отдельных бит портов А и В реализуется в три этапа: содержимое порта вводится в аккумулятор, командами ORI и ANI с соответствующими операндами-масками модифицируется нужный бит, а затем измененный результат выводится в тот же порт.

Как видно из приказа определения режима, адаптер имеет три режима работы. Рассмотрим некоторые особенности каждого из режимов.

В режиме 0 базового ввода-вывода могут работать все три порта, причем порт С разделяется на два независимых 4-битных порта. Всего, таким образом, получается два 8-битных и два 4-битных порта параллельного ввода или вывода, что обеспечивает 16 возможных конфигураций адаптера в режиме 0. Выводимые данные фиксируются в регистрах-защелках, входящих в состав всех портов, а вводимые данные не запоминаются, т. е. в операции считывания входного порта в аккумулятор передается текущее состояние входных линий.

Режим 0 применяется в программно-управляемом вводе-выводе с медленнодействующими периферийными уст-

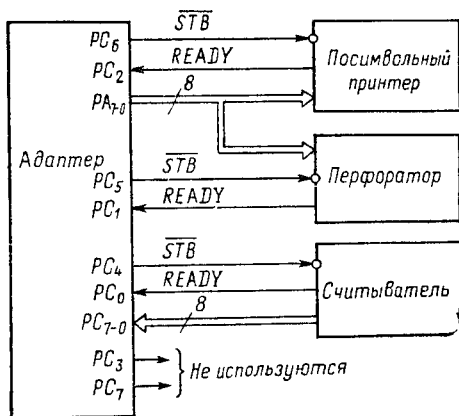


Рис. 4-18. Интерфейс адаптера с принтером и перфоратором-считывателем

ройствами. На рис. 4-18 приведен интерфейс адаптера с символьным принтером и ленточным перфатором-считывателем. Два устройства вывода (принтер и перфатор) разделяют общую шину данных порта А. Будем считать, что состояние готовности всех устройств представлено Н-активными сигналами READY и собственно передача данных сопровождается L-активными стробирующими сигналами STB. Приказ режима должен определить следующую конфигурацию адаптера: порт А — вывод, порт В — ввод, порт С РС₃₋₀ — ввод, порт С РС₇₋₄ — вывод. Данную конфигурацию определяет приказ 10000011 (83H).

Ради простоты воспользуемся символическими адресами регистров адаптера: PORT А — порт А, PORT В — порт В, PORT С — порт С и CWR — регистр управления. Действительные значения этих адресов зависят от подключения адаптера к системной шине адреса и должны быть определены в программе.

Для организации ввода-вывода необходима подпрограмма инициализации адаптера и три аналогичные подпрограммы ввода и вывода для каждого периферийного устройства. Каждая из них выполняет следующие действия: ввод состояния устройства, проверку готовности, вывод или ввод данных и формирование сопровождающего stroba. Если устройство не готово к обмену, микропроцессор входит в цикл ожидания.

Режим 1 стробируемого ввода-вывода предназначен для однонаправленных передач данных, инициируемых прерываниями. Собственно передача слов данных осуществляется через порты А и В, а шесть линий порта С используются для управления обменом. Данный режим предоставляет пользователю следующие возможности: запрограммировать один или два параллельных порта с линиями квитирования и прерывания, каждый из которых может работать на ввод или вывод; при использовании только одного порта остальные 13 линий запрограммировать в режиме 0; при определении двух портов в режим 1 оставшиеся 2 линии использовать для ввода или вывода.

На рис. 4-19 показана одна из возможных конфигураций адаптера, в которой оба порта А и В работают в режиме 1. Приказ определения этой конфигурации имеет вид 1011D₃10X, где D₃ задает использование линий РС_{6,7} для ввода (1) или вывода (0).

Для операции ввода определяются следующие управляющие сигналы:

STB (строб ввода) — импульсный входной сигнал от

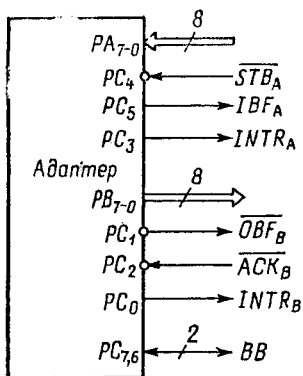


Рис. 4-19. Конфигурация адаптера в режиме 1

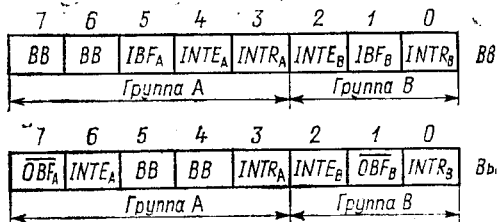


Рис.4-20. Форматы слов состояния режима 1 (ввод и вывод)

устройства ввода, загружает данные, вводимые во входной регистр-зашелку.

IBF (входной буфер загружен) — выходной сигнал подтверждения восприятия данных; устанавливается спадом $\overline{STB}$ и сбрасывается фронтом $\overline{RD}$.

INTR (запрос прерывания) — входной сигнал прерывания микропроцессора; устанавливается при следующих условиях: $\overline{STB}=1$, $IBF=1$, $INTE=1$; сбрасывается спадом сигнала $\overline{RD}$.

Сигналом INTE разрешения прерываний порта А можно управлять с помощью установки и сброса бита PC_4 , а порта В — с помощью установки и сброса бита PC_2 .

Операцию вывода характеризуют также три сигнала:

$\overline{OBF}$ (выходной буфер загружен) — входной сигнал в устройство вывода о наличии данных в порту; формируется после завершения фронта $\overline{WR}$ и оканчивается L-уровнем $\overline{ACK}$.

$\overline{ACK}$ (подтверждение) — входной сигнал от периферийного устройства, подтверждающий восприятие данных из адаптера.

INTR (запрос прерывания) — выходной сигнал прерывания микропроцессора; устанавливается при выполнении условий $\overline{ACK}=1$, $\overline{OBF}=1$, $INTE=1$ и сбрасывается спадом $\overline{WR}$, когда микропроцессор записывает в адаптер новые данные.

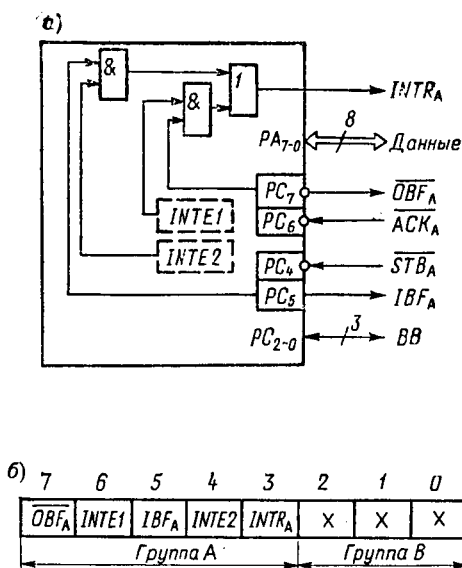


Рис. 4-21. Конфигурация портов А и С режима 2 (а), формат слова состояния режима 2 (б)

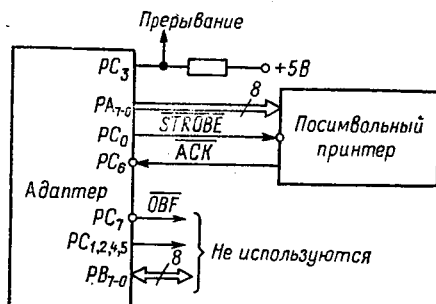


Рис. 4-22. Структурная схема интерфейса адаптера с принтером

Сигналом INTE разрешения прерываний порта А можно управлять установкой и сбросом бита PC_6 , а порта В — установкой и сбросом бита PC_2 .

Значения приведенных выше управляющих сигналов становятся доступными микропроцессору при выполнении обычного считывания порта С.

Форматы слов состояния режима 1 для ввода и вывода приведены на рис. 4-20.

В режиме 2 двунаправленной шины, обеспечивающей ввод и вывод данных, может работать только группа А. Порт А используется для передач собственно 8-битных данных, а пять линий порта С выполняют функции квитирования и прерываний. Общая дисциплина квитирования аналогична режиму 1, но имеются отдельные триггеры разрешения прерываний по выводу INTE 1 (управляется установкой и сбросом бита PC₆) и по вводу INTE 2 (управляется через бит PC₄). Вводимые и выводимые данные фиксируются в регистрах-зашелках порта А. На рис. 4-21, а показана подобная конфигурация портов А и С в режиме 2, определяемая приказом 11XXD₂D₁D₀, где биты D₂₋₀ задают режим группы В (допускаются режимы 0 и 1). Если адаптер запрограммирован в режиме 2, то при считывании из порта С выдается слово состояния, формат которого приведен на рис. 4-21, б. Значения бит D₂₋₀ зависят от запрограммированного режима группы В.

Как пример использования адаптера в режиме 1 рассмотрим вывод текста в посимвольный принтер по прерываниям. Структурная схема интерфейса показана на рис. 4-22. Код выводимого символа передается через порт А, по линии PC₀ генерируется L-активный строб сопровождения STROBE, а сигнал OBF не используется. Сигнал ACK от принтера означает, что он отпечатал символ и готов к приему следующего; по сигналу ACK адаптер инициирует прерывание микропроцессора.

Приказ определения режима должен задать следующую конфигурацию адаптера: порт А — вывод в режиме 1, порт С PC₃₋₀ — вывод, порт С PC₇₋₄ — ввод, порт В не используется и программируется на ввод. Следовательно, приказ режима имеет код 10101010 (ААН). В приводимых далее программах сделаны следующие предположения:

выводимый текст хранится в буферной области памяти, и адрес текущего байта находится в паре смежных ячеек с адресом CHAR;

текст заканчивается символом-терминатором *;

при подтверждении микропроцессором прерывания аппаратно формируется команда RST7, а в ячейках 0038H — 003AH находится команда JMP LPOUT перехода к подпрограмме обслуживания прерываний принтера;

после записи символа в порт А генерируется строб STROBE;

символические адреса PORT А, PORT С, CWR зависят от схемы интерфейса адаптера;

по адресу POLL находится подпрограмма опроса других периферийных устройств (если они есть), а по адресу ENDT — подпрограмма реакции на окончание выводимого текста.

Ниже приведены подпрограммы инициализации адаптера и обслуживания прерываний принтера:

Метка	Код	Операнд	Комментарий
; Подпрограмма инициализации адаптера			
	MVI	A, 0AAH	; Приказ конфигурации
	OUT	CWR	; адаптера
	MVI	A, 01H	; Сброс строба
	OUT	CWR	
	MVI	A, 0C3H	; Загрузка команды JMP
	STA	0038H	LPOUT в ячейки 0038H—
	LXI	H, LPOUT	; 003AH
	LHLD	0039H	
	RET		; Окончание инициализации
; Подпрограмма обслуживания прерываний принтера			
LPOUT:	PUSH	PSW	; Запоминание модифициру-
	PUSH	H	; емых регистров
	IN	PORT C	; Считывание состояния
			; адаптера
	ANI	08H	; Проверка прерывания
			; принтера
	JZ	POLL	; Опрос других устройств
	LHLD	CHAR	; Адрес выводимого символа
	MOV	A, M	; Символ текста в аккумуля-
			; торе
	CPI	'*'	; Текст закончен?
	JZ	ENDT	; Да, переход по адресу
			; ENDT
	OUT	PORT A	; Вывод символа в порт A
	MVI	A, 00H	; Генерирование сопровож-
	OUT	CWR	; дающего строба
	MVI	A, 01H	
	OUT	CWR	
	INX	H	; Продвижение указателя
	SHLD	CHAR	; текста
	POP	H	; Восстановление содержи-
	POP	PSW	; мого регистров
	EI		; Разрешение прерываний
	RET		; Возврат

4-6. ИНТЕРВАЛЬНЫЙ ТАЙМЕР

В некоторых применениях микро-ЭВМ встречаются такие простые функции управления, как генерация сигналов переменной частоты, формирование временных задержек,

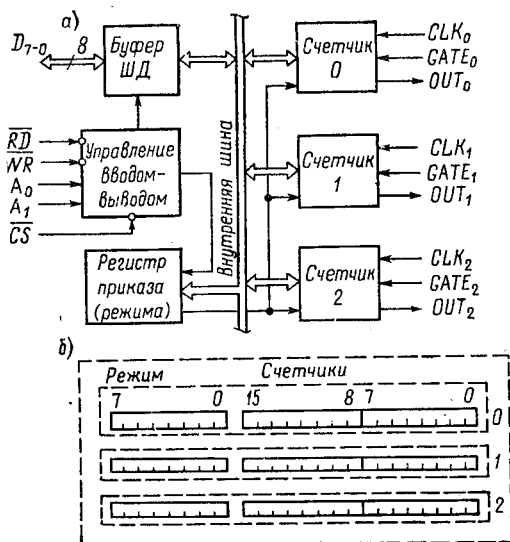


Рис. 4-23. Структурная схема (а) и программная модель (б) таймера

подсчет числа внешних событий и др. С целью максимального упрощения реализации таких функций в большинстве микропроцессорных семейств выпускаются БИС программируемых счетчиков — таймеров. Одним из примеров их служит микросхема КР580ВИ53, входящая в семейство микропроцессора КР580ИК80. БИС выполнена по НМОП-технологии, имеет одно напряжение питания +5 В и максимальную частоту сигналов синхронизации до 2 МГц.

На рис. 4-23 показаны организация и программная модель таймера. Основными компонентами ее являются три независимых вычитающих 16-битных счетчика со своими схемами управления. В любой из счетчиков допускается загрузка начального значения счетчика с системой шины, а текущее содержимое можно считать, не прекращая счета. Собственно входные сигналы CLK_0 — CLK_2 счетчиков подаются через схемы с внешними управляющими входами $GATE_0$ — $GATE_2$. С каждым счетчиком ассоциируется выходной сигнал OUT . Остальные компоненты схемы предназначены для установки режима и интерфейса с системной шиной.

Стандартное подключение таймера к системной шине приведено на рис. 4-24. Выбирающие входы $A_{1,0}$, соединен-

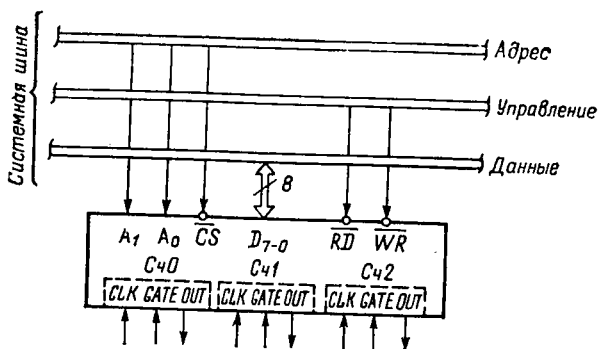


Рис. 4-24. Интерфейс таймера с системной шиной

ные с соответствующими линиями шины адреса, определяют целевой (адресуемый) регистр, вход выбора кристалла $\overline{CS}$ подключается к одной из линий шины адреса, когда применяется линейный выбор, либо к выходу дешифратора адреса. Сигнал записи $\overline{WR}$ загружает байт с шины данных в целевой регистр, а сигнал считывания $\overline{RD}$ передает содержимое целевого регистра на шину данных. В зависимости от используемого способа ввода-вывода в качестве $\overline{WR}$ и $\overline{RD}$ служат либо сигналы записи-считывания ввода-вывода, либо соответствующие сигналы памяти. Все операции, определяемые сигналами $\overline{CS}$, $\overline{WR}$, $\overline{RD}$ и адресными входами $A_{1,0}$, сведены в табл. 4-3. Отметим невозможность считывания содержимого регистра приказа.

Для функционирования таймера процессор должен загрузить управляющие слова (приказы), определяющие режим каждого счетчика, и необходимые начальные значения. Приказы, формат которых приведен на рис. 4-25, программируют режим, последовательность загрузки и считывания, а также используемый код (двоичный или десятичный).

Двухбитное поле $SC1$ — $SC0$ выбора счетчика адресует счетчик, режим которого определяет текущий приказ. Следующее поле $RL1$ — $RL0$ считывания-загрузки задает порядок считывания содержимого счетчика или загрузки начального значения. Трехбитное поле режима $M2$ — $M0$ идентифицирует один из шести режимов работы адресуемого полем $SC1$ — $SC0$ счетчика. Младший бит устанавливает используемый код.

Таблица 4-3

Функции управляющих и адресных входов

Сигнал на входе					Функция
$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Загрузка Сч0 с шины данных
0	1	0	0	1	Загрузка Сч1 с шины данных
0	1	0	1	0	Загрузка Сч2 с шины данных
0	1	0	1	1	Загрузка регистра приказа с шины данных
0	0	1	0	0	Считывание Сч0 на шину данных
0	0	1	0	1	Считывание Сч1 на шину данных
0	0	1	1	0	Считывание Сч2 на шину данных
0	0	1	1	1	Высокое сопротивление буфера
1	×	×	×	×	То же
0	1	1	×	×	» »

При рассмотрении каждого из режимов необходимо учитывать сигнал GATE соответствующего счетчика. Функции сигнала GATE приведены в табл. 4-4. Под счетом понимается декремент счетчика, производимый спадающим фронтом сигнала CLK.

Режим 0 — прерывание по окончании счета или программируемая задержка. По окончании операции задания режима на выходе OUT формируется низкий уровень, кото-

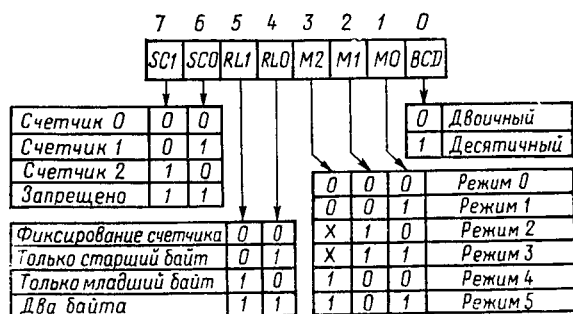


Рис. 4-25. Формат приказа таймера

Функции сигнала GATE

Режим	Состояние входа		
	Низкий уровень или спад	Фронт	Высокий уровень
0	Запрещает счет	—	Разрешает счет
1	—	Иницирует счет; сбрасывает $OUT=0$ в следующем такте синхронизации	—
2	Запрещен счет; устанавливает $OUT=1$	Иницирует счет	Разрешает счет
3	То же	То же	
4	Запрещает счет	—	
5	—	Иницирует счет	—

рый сохраняется и после загрузки в счетчик начального значения N . Начало работы счетчика иницируется высоким уровнем на входе GATE. Когда содержимое счетчика достигает нуля, на выходе OUT устанавливается высокий уровень, используемый для прерывания микропроцессора и сохраняющийся до нового задания режима. Когда в процессе работы осуществляется повторная загрузка счетчика, загрузка первого байта прекращает счет, а загрузка второго байта начинает новый цикл. Если в процессе счета устанавливается низкий уровень сигнала GATE, содержимое счетчика сохраняется и после восстановления высокого уровня счет продолжается от запомненного значения. Отметим, что $N=0$ соответствует значению 2^{16} в двоичном коде и 10^4 в десятичном коде.

Режим 1 — программируемого одновибратора. Если в счетчик загружено начальное значение N , то после первого сигнала CLK, следующего за установлением высокого уровня GATE, на выходе OUT формируется отрицательный импульс продолжительностью N периодов τ сигналов CLK. Загрузка нового значения N при низком уровне OUT не влияет на длительность импульса до последующего запуска. Одновибратор является «перезапускаемым», т. е. отри-

цательный импульс на выходе OUT сохраняет продолжительность $N\tau$ от любого фронта на входе GATE.

Режим 2 — генератора программируемой частоты, или счетчика с коэффициентом деления N . После установки высокого уровня GATE на выходе OUT формируется периодический сигнал, продолжительность высокого уровня которого составляет $(N-1)\tau$, а низкого τ . Когда производится перезагрузка счетчика, текущий период не изменяется, а следующий будет соответствовать новому значению N . Вход GATE используется для аппаратной синхронизации, так как низкий уровень GATE устанавливает высокий уровень OUT, а с последующим переходом GATE к высокому уровню счет иницируется с начального значения. При задании режима 2 высокий уровень OUT сохраняется до загрузки N , чем обеспечивается программная синхронизация генератора.

Режим 3 — генератора прямоугольных сигналов со скважностью 2 (меандр). Данный режим напоминает предыдущий, так как период выходного сигнала равен $N\tau$, но выход OUT имеет высокий уровень в течение $N\tau/2$, если N четное, или $(N+1)\tau/2$, если N нечетное. При перезагрузке счетчика новое значение N учитывается в следующем периоде выходного сигнала.

Режим 4 — формирования одиночного строба с программным запуском. После задания режима на выходе OUT формируется высокий уровень и иницирование счета осуществляется загрузкой в счетчик значения N . При достижении счетчиком нуля на выходе OUT формируется отрицательный импульс длительностью τ . Если производится перезагрузка счетчика между выходными импульсами, текущий период не изменяется, а следующий соответствует новому значению N . Низкий уровень GATE запрещает работу счетчика; если при действии его происходит перезагрузка счетчика, счет начинается с нового значения N .

Режим 5 — формирование одиночного строба с аппаратным запуском. Начало счета иницируется фронтом сигнала GATE. При достижении нулевого состояния счетчика на выходе OUT формируется отрицательный импульс длительностью τ . Как и одновибратор в режиме 1, счетчик является перезапускаемым, т. е. любой фронт сигнала GATE иницирует полный цикл работы.

Для правильной работы таймера необходимо запрограммировать каждый счетчик на требуемый режим и загрузить в него начальное значение. Регистры режима счетчиков адресуются полем (SC1, SC0) приказа, поэтому про-

граммирование режима может осуществляться в любом порядке, но загрузка начального значения должна производиться в соответствии с полем RL приказа каждого счетчика.

Команды, осуществляющие загрузку начальных значений, например команды вывода OUT, должны адресовать целевой счетчик по линиям $A_{1,0}$. Примерная последовательность программирования, когда в приказах всех трех счетчиков определяется загрузка двух байт, такова (в скобках указаны состояния линий $A_{1,0}$):

1. Вывод приказа режима Сч0 (11).
2. Вывод приказа режима Сч2 (11).
3. Вывод приказа режима Сч1 (11).
4. Загрузка младшего байта Сч1 (01).
5. Загрузка старшего байта Сч1 (01).
6. Загрузка младшего байта Сч0 (00).
7. Загрузка старшего байта Сч0 (00).
8. Загрузка младшего байта Сч2 (10).
9. Загрузка старшего байта Сч2 (10).

Считывание, т. е. ввод в микропроцессор, содержимого любого счетчика для последующего анализа выполняется двумя способами. Первый способ реализуется обычными командами передачи данных в процессор, например командами ввода IN, посредством которых адресуется нужный счетчик. Недостаток данного способа заключается в том, что на операцию считывания работу счетчика необходимо останавливать с помощью входа GATE или прекращения подачи сигналов CLK. Считывание должно обязательно выполняться в соответствии с полем RL приказа режима счетчика.

Второй способ (считывание «на лету») не нарушает работы счетчика, но требует предварительной загрузки в регистр режима специального кода SC1, SC0, 0,0, X, X, X, X. Этот код вызывает фиксацию текущего содержимого целевого счетчика в отдельном регистре.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что понимается под интерфейсом?
2. Какие основные функции выполняет интерфейс?
3. В чем принципиальное отличие интерфейса с изолированной шиной от интерфейса с общей шиной?
4. В чем состоят достоинства и недостатки интерфейса с общей шиной?
5. Чем обеспечивается многофункциональность в использовании регистра K589IP12?

6. Какие функции выполняет блок приоритетных прерываний?
7. Каким образом осуществляется увеличение числа входов запроса прерываний в блоке K589ИК14?
8. Каково назначение шинных формирователей?
9. Какую информацию необходимо передавать в системный контроллер из процессора?
10. Какие функции выполняет программируемый адаптер последовательного и параллельного интерфейса?
11. Каким образом программируется адаптер на различные режимы работы?
12. На какие режимы обмена информацией можно запрограммировать параллельный адаптер?
13. Чем отличается асинхронный режим работы от синхронного в последовательном адаптере?
14. Как организовать интерфейс между адаптером и процессором?

ГЛАВА ПЯТАЯ

ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА

5-1. РЕЖИМЫ И СТРУКТУРА КОМАНД ВВОДА-ВЫВОДА

Вводом-выводом (ВВ) называются передачи данных между ядром ЭВМ, включающим в себя процессор и основную память, и периферийными устройствами. Он представляет собой единственное средство взаимодействия ЭВМ с «внешним миром», и архитектура ВВ (режимы работы, форматы команд, особенности прерываний, скорость обмена и др.) непосредственно влияет на эффективность всей системы. За время эволюции ЭВМ подсистема ВВ претерпела наибольшие изменения благодаря расширению сферы применения ЭВМ и появлению новых периферийных устройств. Особенно важную роль средства ВВ играют в управляющих ЭВМ. Разработка аппаратных средств и программного обеспечения ВВ является наиболее сложным этапом проектирования новых систем на базе ЭВМ, а возможности ВВ серийных машин представляют собой один из важных параметров, определяющих выбор машины для конкретного применения.

Несмотря на разнообразие периферийных устройств, разработано несколько стандартных способов подключения их к ЭВМ и программирования ВВ. Основные из них рассматриваются в настоящей главе.

В ЭВМ применяются три режима ВВ: программный ВВ (называемый также программно-управляемым или нефор-

сированным ВВ), ВВ по прерываниям (форсированный ВВ) и прямой доступ к памяти. Первый из них характеризуется тем, что инициирование и управление ВВ осуществляются процессором, а периферийные устройства играют сравнительно пассивную роль и сигнализируют только о своем состоянии, в частности о готовности к операциям ВВ. Во втором режиме ВВ иницируется не процессором, а периферийным устройством, генерирующим специальный сигнал прерывания. Реагируя на этот сигнал готовности устройства к передаче данных, процессор переключается на подпрограмму обслуживания устройства, вызвавшего прерывание. Действия, выполняемые этой подпрограммой, определяются пользователем, а непосредственными операциями ВВ управляет процессор. Наконец, в режиме прямого доступа к памяти, когда пропускной способности процессора недостаточно, действия процессора приостанавливаются, он отключается от системной шины и совершенно не участвует в передачах данных между основной памятью и быстродействующим периферийным устройством.

Рассмотрение ВВ усложняется тем, что в каждой ЭВМ применяются особые способы ВВ, различные конфигурации схем и типы устройств. Но можно выделить следующие общие принципы, характерные для большинства машин:

- передача ВВ осуществляется отдельными битами (последовательный интерфейс) либо полными словами (параллельный интерфейс);

- при наличии нескольких периферийных устройств необходимо генерировать специальный сигнал выбора, идентифицирующий активное устройство;

- ВВ в медленнодействующих устройствах обычно организуется по прерываниям, а в быстродействующих — на основе прямого доступа к памяти;

- операции ВВ иницируются только в том случае, если периферийное устройство готово к ним;

- управляющие сигналы ВВ от процессора синхронизируются с сигналами генератора тактовых сигналов.

Собственно передача данных осуществляется либо по общей системной шине, либо по специальной шине ВВ. Первый вариант характерен для большинства микро-ЭВМ и мини-ЭВМ младших моделей, а второй — для мини-ЭВМ старших моделей и больших ЭВМ. Иногда отдельная быстродействующая шина ВВ выделяется только для передач с прямым доступом к памяти.

Подключение периферийных устройств к системной ши-

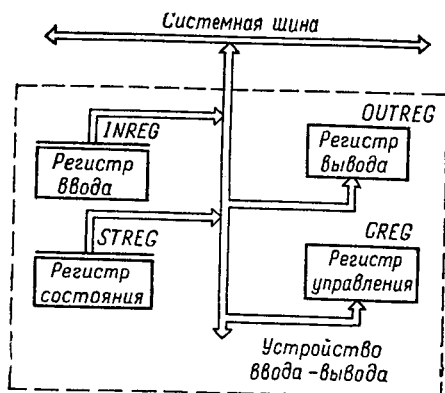


Рис. 5-1. Общая программная модель периферийного устройства

не осуществляется посредством электронных схем, называемых интерфейсами ВВ. Они согласуют уровни электрических сигналов, а также преобразуют машинные данные в любой формат, необходимый устройству, и наоборот. Обычно интерфейсы ВВ конструктивно оформляются вместе с процессором и основной памятью в виде интерфейсных плат (карт).

В процессе ВВ передается информация двух видов: управляющие данные (слова) и собственно данные, или данные-сообщения. Управляющие данные от процессора, называемые также командными словами или приказами, инициируют действия, не связанные непосредственно с передачей данных, например запуск устройства, запрещение прерываний и т. п. Управляющие данные от периферийных устройств называются словами состояния; они содержат информацию об определенных признаках, например о готовности устройства к передаче данных, о наличии ошибок при обмене и т. п. Состояние обычно представляется в декодированной форме — один бит для каждого признака.

Группа бит, к которым процессор обращается в операциях ВВ, образует регистр (порт) ВВ. Таким образом, наиболее общая программная модель периферийного устройства, которое может выполнять ввод и вывод, содержит четыре регистра ВВ: регистр выводимых данных (выходной порт), регистр вводимых данных (входной порт), регистр управления и регистр состояния (рис. 5-1). Каждый из этих регистров должен иметь однозначный адрес, который идентифицируется дешифратором адреса. На рис. 5-1 показаны

символические адреса регистров. В зависимости от особенностей устройства общая модель конкретизируется, например отдельные регистры состояния и управления объединяются в один регистр, в устройстве ввода (вывода) имеется только регистр ввода (вывода), для ввода и вывода используется двунаправленный порт.

Подключение регистров ВВ к шине данных осуществляется с помощью логических схем, формирователей с открытым коллектором и тристабильных буферов. Наиболее удобно в качестве регистров ВВ использовать многорежимный буферный регистр К589ИР12, имеющий тристабильные выходные каскады, встроенную логику выбора и вход аппаратного программирования на ввод или вывод. В обычной конфигурации на вход $\overline{DS1}$ подается управляющий сигнал ввода или вывода, а на вход $DS2$ — сигнал выбора от дешифратора адреса.

Для упрощения дешифрирующей логики в системах с небольшим числом устройств применяется линейный выбор, когда адреса регистров ВВ представляются кодами, содержащими только по одной единице.

Непосредственные действия, связанные с вводом-выводом реализуются одним из двух способов, различающихся адресацией регистров ВВ. Изолированный ВВ предполагает наличие специальных команд ВВ, общий формат которых показан на рис. 5-2. При выполнении команды ввода IN содержимое адресуемого входного порта PORT передается во внутренний регистр REG процессора, а при выполнении команды вывода OUT содержимое регистра REG передается в выходной порт PORT. В процессоре могут быть также и другие команды, относящиеся к ВВ и связанные с проверкой и модификацией содержимого регистра управления и состояния.

Нетрудно заметить, что в этом способе адресное пространство портов ввода и вывода изолировано от адресного пространства памяти, т. е. в ЭВМ могут быть входной порт, выходной порт и ячейка памяти с одинаковыми адресами.

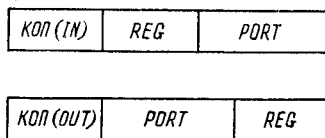


Рис. 5-2. Команды ввода-вывода (общий формат)

Разделение адресных пространств осуществляется с помощью управляющих сигналов, относящихся к системам ВВ и памяти.

В ЭВМ, рассчитанной на изолированный ВВ, нетрудно перейти к ВВ, отображенному на память. Если, например, адресное пространство памяти составляет 64К байт, а для программного обеспечения достаточно 32К байт, то область адресов от 0 до 32К—1 используется для памяти, а область от 32К до 64К—1 используется для ВВ. При этом признаком, дифференцирующим обращения к памяти и портам ВВ, становится старший бит A_{15} адреса.

В операционных системах ЭВМ имеется набор подпрограмм (драйверов ВВ), управляющих операциями ВВ стандартных периферийных устройств. Благодаря им пользователь может не знать многих особенностей периферийных устройств и интерфейсов ВВ, а применять четкие программные протоколы.

Если к ЭВМ подключено несколько идентичных устройств, например видеотерминалов, каждое из них имеет свой интерфейс ВВ. Аппаратные средства интерфейсов одинаковы, но регистры ВВ имеют различные адреса. Вместо отдельных драйверов ВВ для каждого устройства обычно используется разделенный драйвер, который обслуживает все идентичные устройства. Как параметры ему передаются базовый адрес таблицы адресов регистров ВВ и смещение (индекс), определяемое конкретным обслуживаемым устройством.

Разделенный драйвер легко программируется при использовании ВВ, отображенного на память, так как для динамического вычисления адресов регистров ВВ доступны режимы косвенной и базовой адресации. В микропроцессоре КР580 абсолютный адрес регистра ВВ является частью команд IN и OUT, поэтому запрограммировать разделенный драйвер без модификации программы невозможно.

5-2. ПРОГРАММНЫЙ ВВОД-ВЫВОД

Данный режим характеризуется тем, что все действия по ВВ реализуются командами прикладной программы. Наиболее простыми эти действия оказываются для «всегда готовых» периферийных устройств, например индикатора на светодиодах. При необходимости ВВ в соответствующем месте программы используются команды IN или OUT. Такая передача данных называется синхронным или безусловным ВВ.

Однако для большинства периферийных устройств до выполнения операций ВВ необходимо убедиться в их готовности к этим операциям, т. е. ВВ оказывается асинхронным. Общее состояние устройства характеризуется флажком готовности **READY**, называемым также флажком готовности/занятости **READY/BUSY**. Иногда состояния готовности и занятости идентифицируются отдельными флажками **READY** и **BUSY**, входящими в слово состояния устройства. Интерпретация флажка **READY** в устройствах ввода и устройствах вывода различна. Для устройств ввода обычно принимается следующая интерпретация флажка **READY** и выполняемых с ним операций:

- состояние **READY**=0 означает отсутствие входных данных в регистре ввода;

- состояние **READY**=1 определяет наличие входных данных;

- подпрограмма инициализации сбрасывает флажок **READY**;

- при загрузке входных данных в регистр данных флажок **READY** автоматически устанавливается;

- при вводе (считывании) данных процессором флажок **READY** сбрасывается.

Для устройств вывода соответствующие состояния и действия имеют несколько другой смысл:

- состояние **READY**=0 означает недоступность регистра вывода для приема данных от процессора;

- состояние **READY**=1 сигнализирует о доступности регистра вывода;

- подпрограмма инициализации устанавливает флажок **READY**;

- при восприятии выходных данных из регистра вывода непосредственно в устройство флажок **READY** автоматически устанавливается;

- при выводе (записи) данных процессором флажок **READY** сбрасывается.

Процессор проверяет флажок готовности с помощью одной или нескольких команд. Если флажок установлен, инициируется собственно ввод или вывод одного или нескольких слов данных. Когда же флажок сброшен, процессор выполняет цикл из 2—3 команд с повторной проверкой флажка **READY** до тех пор, пока устройство не будет готово к операциям ВВ. Данный цикл называется циклом ожидания и реализуется в различных процессорах по-разному. Например, для процессоров **СМ** цикл ожидания устройства, флажок готовности которого находится в старшем, 7-м

бите байта состояния из регистра с адресом CSR, имеет следующий вид:

Метка	Код	Сперанд	Комментарий
WAIT:	TSTB BPL	CSR WAIT	; Проверка бита готовности ; Зацикливание, если уст- ; ройство не готово

Команда TSTB проверки байта устанавливает признак N=1, когда бит 7 содержит единицу. Если признак N=0, команда BPL (перейти, если плюс) передает управление команде с меткой WAIT, что и обеспечивает зацикливание. Следующая за циклом ожидания команда MOV осуществляет непосредственную передачу данных.

В микропроцессоре KP580 аналогичные действия выполняет следующий фрагмент:

Метка	Код	Сперанд	Комментарий
WAIT:	IN ORA JP	CSR A WAIT	; Ввод состояния ; Установка признаков ; Зацикливание до готов- ; ности

Предполагается, что CSR — 8-битный адрес регистра управления и состояния, а флажок READY находится в старшем бите регистра. Следующая за циклом ожидания команда IN может произвести ввод данных из регистра ввода с другим адресом DATA, а для вывода данных необходимо сначала передать их в аккумулятор и затем вывести с помощью команды OUT.

Чтобы проиллюстрировать некоторые особенности программирования ВВ, приведем простую программу для процессоров СМ ввода данных с перфоленточного считывателя. Программная модель считывателя приведена на рис. 5-3. Биты регистра управления и состояния имеют следующий смысл:

RDR ENB (разрешение считывания) — единичное состояние инициирует считывание с перфоленты одного символа;

INT ENB (разрешение прерываний) — единичное состояние разрешает прерывание от считывателя по флажку READY или ERROR;

READY (готовность) — флажок готовности, фиксирующий состоянием READY=1 наличие считанного символа в регистре данных;

BUSY (занятость) — флажок занятости, идентифицирующий состоянием BUSY=1 механическое движение перфоленты;

ERROR (ошибка) — флажок ошибки, сигнализирующий условием ERROR=1 неправильное считывание с перфоленты.

Вначале иницируется считывание символа посредством установки бита RDR ENB. Это действие вызывает установку бита BUSY, сброс бита READY и считывание символа в регистр данных. Когда символ загружается в регистр данных, устанавливается бит READY, а бит RDR ENB сбрасывается. Приводимая ниже программа вводит восемь символов в область памяти с начальным адресом AREA=17400.

Метка	Код	Операнд	Комментарий
AREA	=	17 400	; Начальный адрес
	.=	17 744	; буфера данных
			; Начало программы
START:	MOV	READER, R0	; Адрес регистра
			; управления и
	MOV	# 0, R1	; состояния
			; Смещение в регистре R1
READ:	INC	(R0)	; Иницирование
			; считывания
			; символа
WAIT:	TSTB	(R0)	; Считыватель готов?
	BPL	WAIT	; Нет, цикл ожидания
			; дания
	MOVB	2(R0), AREA (R1)	; Ввод символа
	INC	R1	; Продвижение
			; смещения
	CMP	# 10, R1	; Введено восемь
			; символов?
	BNE	READ	; Нет, повторение
			; ввода
READER:	.WORD	177 560	; Адрес регистра
			; управления и
			; состояния
	.END	START	; Конец программы

Основной недостаток программного ВВ связан с непроизводительными потерями времени процессора в циклах ожидания. Например, при выводе данных на перфоленту (скорость до 150 символов/с) или на пишущую машинку

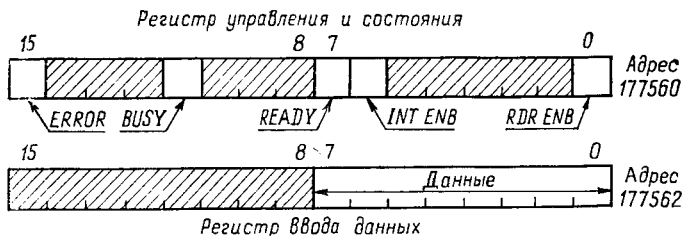


Рис. 5-3. Программная модель перфоленточного считывателя

(скорость до 25 символов/с) практически все время процессора приходится на бесполезные циклы ожидания, в которых он не выполняет никакой полезной работы. К достоинствам программного ВВ относят простоту его реализации, не требующей дополнительных аппаратных средств.

5-3. ВВОД-ВЫВОД В РЕЖИМЕ ПРЕРЫВАНИЙ

При организации в ЭВМ системы (или подсистемы) прерываний неизбежные потери времени процессора в циклах ожидания резко сокращаются.

Последовательность событий при прерываниях. Каждое периферийное устройство может посылать в процессор сигнал INT запроса прерывания или запроса на обслуживание, когда оно готово к операциям ВВ, т. е. требует немедленной реакции процессора. По существу, этот сигнал представляет собой выходной сигнал триггера, фиксирующего флажок готовности READY. Сигнал INT появляется в произвольные моменты времени, асинхронно по отношению к действиям процессора, и управлять его появлением программа не может. Следовательно, заранее неизвестно, в какой точке программы и какие периферийные устройства инициируют прерывания, поэтому непосредственно в программе команды ВВ использовать нельзя. Остается одно: реагируя на сигнал INT, процессор должен прервать, т. е. временно приостановить, текущую программу, идентифицировать прерывающее устройство, перейти к подпрограмме обслуживания прерываний работы этого устройства, а после ее завершения возобновить выполнение прерванной программы. Подпрограмме обслуживания потребуются внутренние регистры процессора: аккумулятор, программный счетчик, некоторые РОН, и их текущее содержимое будет модифицировано. Но прерванная программа должна

возобновляться так, как будто прерывания вообще не было. Кроме как увеличением времени выполнения факт обслуживания прерывания не должен влиять на прерванную программу. Следовательно, содержимое всех регистров, необходимых подпрограмме обслуживания прерывания, следует временно запоминать. Удобно и просто использовать в качестве такого временного «хранилища» стек.

Практически в каждом процессоре реализована особая структура системы прерываний, а программируемые БИС управления прерываниями еще более увеличивают число разновидностей этой структуры. Однако общая последовательность реакции процессора на сигнал прерывания примерно одинакова и содержит следующие действия:

периферийное устройство генерирует сигнал прерываний, который подается на вход INT процессора; на этой линии по схеме ИЛИ объединяются запросы всех устройств, работающих в режиме прерываний;

процессор завершает текущую команду и, если прерывания разрешены (не замаскированы), формирует сигнал INTA (или INT ACK) подтверждения прерывания; до получения этого сигнала устройство сохраняет активный уровень сигнала INT;

осуществляется запоминание содержимого РС и некоторых других внутренних регистров в стеке, причем содержимое РС обычно запоминается автоматически;

процессор идентифицирует прерывающее устройство для перехода к соответствующей подпрограмме обслуживания (наиболее распространенные способы идентификации прерывающих устройств рассмотрены ниже);

выполняется короткая (30—50 байт) подпрограмма обслуживания прерывания, в которой запрограммированы действия по передаче данных, модификации указателей, проверке окончания операций ВВ и др.;

восстанавливается состояние прерванной программы, для чего запомненное содержимое регистров извлекается из стека;

возобновляется выполнение прерванной программы; это действие инициируется командой возврата из прерывания RTI, являющейся последней командой подпрограммы обслуживания прерывания.

Системы прерываний можно классифицировать по объему перечисленных действий, которые реализуются аппаратно или программно. Разумеется, чем большее их число реализовано аппаратно, тем быстрее реакция процессора на прерывание.

После окончания каждой команды программный счетчик адресует ту команду программы, которая выполнялась бы при отсутствии прерывания, поэтому после обслуживания прерывания программа продолжается обычным образом. Объем временно запоминаемой информации определяется особенностями подпрограммы обслуживания прерывания. Как минимум, необходимо запоминать содержимое РС и регистра флажков (признаков), а в предельном случае потребуется запоминать содержимое всех регистров программной модели процессора. Процессы запоминания содержимого внутренних регистров после восприятия сигнала прерывания и восстановления его по окончании подпрограммы обслуживания прерывания называются контекстным переключением (процессора). Скорость контекстного переключения оказывает заметное влияние на производительность ЭВМ, особенно в условиях интенсивных прерываний. Поэтому во многих процессорах предусматриваются средства ускорения контекстного переключения, например команды, которые загружают в стек и извлекают из стека содержимое группы регистров. В идеальном случае контекстное переключение должно требовать минимум команд и времени.

Из анализа действий процессора при восприятии сигнала прерывания следует, что они очень похожи на действия при вызове подпрограммы (рис. 5-4). Однако вызов подпрограммы запрограммирован и полностью предсказуем, а переход к обслуживанию прерывания инициируется внешним сигналом, момент появления которого предсказать невозможно. Тем не менее внешняя аналогия реакции на прерывание и вызова подпрограммы позволяет считать прерывание аппаратным вызовом подпрограммы. Выполняемые при этом вызове действия процессора, связанные с вводом-выводом, ограничиваются только его программными возможностями.

Виды прерываний. Действия процессора при обслуживании запроса прерываний схематически показаны на рис. 5-5. Так как команды ВВ в основной программе отсутствуют, процессор после завершения каждой команды должен проверять наличие сигнала прерывания до перехода к следующей команде. Например, в микропроцессоре КР580 анализ состояния входной линии INT осуществляется в такте T2 последнего машинного цикла каждой команды. Переход к подпрограмме обслуживания прерывания происходит, если только $INT=1$ и прерывания разрешены.

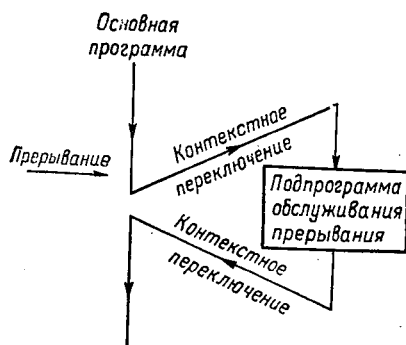


Рис. 5-4. Общая реакция на прерывание

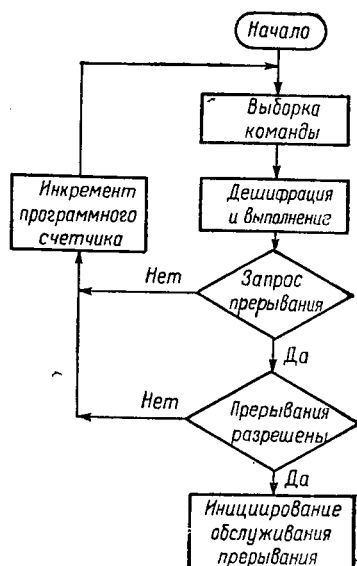


Рис. 5-5. Процедура восприятия прерывания

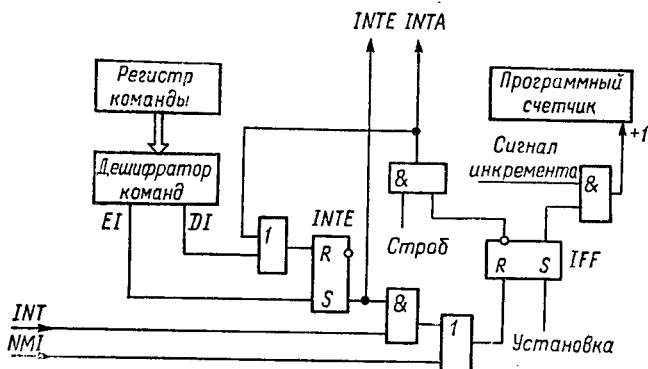


Рис. 5-6. Внутренняя схема прерываний

Процессор реагирует на запросы маскируемых прерываний по линии INT, если установлен внутренний триггер разрешения прерываний INTE, называемый также маской (рис. 5-6). Состояние этого триггера идентифицирует выходной сигнал разрешения прерываний с такой же мнемоникой INTE. Если $INTE=0$, прерывания запрещены (замас-

кированы) и процессор не реагирует на сигнал $INT=1$. С помощью команд разрешения EI и запрещения DI прерываний можно программно управлять состоянием триггера INTE, и пользователь может защитить от прерываний критические сегменты прикладной программы.

При восприятии прерывания триггер прерываний IFF переводится в нулевое состояние, что приводит к запрещению инкремента программного счетчика и генерированию сигнала подтверждения прерывания INTA. Отметим, что при этом сбрасывается триггер INTE и в дальнейшем разрешить прерывания можно только командой EI. После выполнения команды EI процессор обязательно выполняет еще одну команду, даже если на входе INT действует сигнал прерывания $INT=1$.

В мини-ЭВМ для программного управления прерываниями каждого периферийного устройства в их регистрах управления и состояния предусмотрен специальный бит INT ENB разрешения прерываний (маска). Иногда биты масок всех устройств объединяются в специальной регистр. Наконец, в некоторых процессорах бит маскирования прерываний входит в слово состояния процессора PSW.

Запросы немаскируемых прерываний по входу NMI процессор воспринимает всегда, независимо от того, разрешены прерывания или нет. Однако до реакции на сигнал NMI процессор все же завершает выполнение текущей команды. Обычно на вход NMI подается сигнал от схемы, фиксирующей уменьшение напряжения сети до некоторого критического уровня. Быстродействие процессора достаточно велико, чтобы за время от момента восприятия сигнала NMI до уменьшения напряжения питания ниже минимального допустимого выполнить несколько команд. Этими командами содержимое всех внутренних регистров и другая важная информация записываются в энергонезависимую память с резервным аккумуляторным питанием.

Подпрограммы обслуживания прерываний. Рассмотрим несколько подробнее организацию подпрограммы обслуживания прерывания, используя в качестве простого примера вывод текста на электрифицированную пишущую машинку (ЭПМ). Номинальная скорость печати ЭПМ не превышает 25 символов/с, т. е. печать одного символа длится 40 000 мкс.

Будем считать, что ЭПМ определяет возможность передачи ей символа сигналом «готовности печати», линия которого подключена на вход INT процессора. Пусть в некоторый момент времени ЭПМ формирует запрос на обслу-

живание; другими словами, сигнал прерывания становится активным ($INT=1$). Предположим, что, реагируя на прерывание, процессор сбросил триггер INTE разрешения прерываний, автоматически включил содержимое программного счетчика в стек и каким-либо образом загрузил в программный счетчик начальный адрес START подпрограммы обслуживания прерываний ЭПМ.

Выводимый текст, т. е. последовательность 8-битных кодов символов, находится в некоторой области памяти, называемой выходным буфером, и признаком его окончания является некоторый особенный символ, например звездочка (*). Указатель выходного буфера, т. е. адрес текущего выводимого символа, находится в ячейке памяти с символическим адресом PTR. Символический адрес регистра данных ЭПМ обозначим PRINT. С учетом всех сделанных предположений подпрограмма обслуживания, рассчитанная на микропроцессор КР580, принимает следующий вид:

Метка	Код	Операнд	Комментарий
START:	PUSH	PSW	; «Освобождение» аккумулятора
	PUSH	H	; тора и основного указателя
			; памяти
	LHLD	PTR	; Указатель выходного буфера
	MOV	A, M	; Символ в аккумуляторе
	CPI	' '	; Конец текста?
	JZ	ENDT	; Да, вывод закончен
	OUT	PRINT	; Вывод символа в ЭПМ
	INX	H	; Продвижение указателя бу-
			; фера
	SHLD	PTR	; Запоминание указателя
	POP	H	; Восстановление регистров
	POP	PSW	; процессора
	EI		; Разрешение прерываний
	RET		; Возврат в прерванную про-
			; грамму

Здесь ENDT обозначает символический начальный адрес подпрограммы, к которой переходит микропроцессор при завершении вывода текста из буфера.

Отметим, что разрешение прерываний осуществляется в конце подпрограммы командой EI, поэтому при выводе символа процессор игнорирует прерывания других периферийных устройств, если они возникают. Кроме того, при выполнении команды OUT флажок готовности ЭПМ сбрасывается, для чего используется сигнал загрузки кода символа с шины данных в регистр с адресом PRINT.

Длина подпрограммы составляет 21 байт, а время выполнения эквивалентно 122 тактам синхронизации, что при частоте $f_c = 2$ МГц соответствует 61 мкс. Сравнение этой величины с продолжительностью вывода одного символа в цикле ожидания (40 000 мкс) показывает, насколько увеличивается эффективность процессора при организации прерываний. Когда ЭПМ работает с максимальной скоростью, на управление выводом расходуется всего 0,15 % времени процессора.

Идентификация прерывающего устройства. Когда в ЭВМ имеется несколько периферийных устройств, работающих в режиме прерываний, сигналы их запросов на обслуживание объединяются по схеме ИЛИ и подаются на вход INT процессора. Следовательно, при наличии активного сигнала $INT = 1$ без дополнительных действий и аппаратных средств невозможно определить, какое устройство должен обслуживать процессор, тем более, что запросы на обслуживание могут формироваться одновременно несколькими устройствами. Другими словами, возникает проблема идентификации прерывающего устройства, т. е. однозначного перехода к определенной подпрограмме обслуживания. Разработано несколько способов решения этой проблемы, различающихся скоростью реакции процессора и объемом дополнительных аппаратных средств. При реализации любого способа необходимо назначить устройствам определенные приоритеты и учитывать их таким образом, чтобы процессор, реагируя на сигнал прерывания, выбирал для обслуживания запрашивающее устройство с максимальным приоритетом.

Программный полинг. Простейшее решение проблемы идентификации, почти не требующее дополнительных аппаратных средств, заключается в программном опросе (полинге) флажков готовности (т. е. сигналов прерываний) периферийных устройств. Реагируя на прерывание, процессор переходит к подпрограмме полинга, находящейся в фиксированной области памяти с известным начальным адресом. Этот начальный адрес загружается в программный счетчик после запоминания его предшествующего содержимого в стеке.

На рис. 5-7 приведен типичный вариант программного полинга. Первым проверяется флажок готовности устройства У1 с наибольшим приоритетом. Если оно не запрашивало обслуживания, опрашивается следующее устройство и т. д. Когда встречается первое устройство, готовое к операциям ВВ, управление передается подпрограмме обслужива-

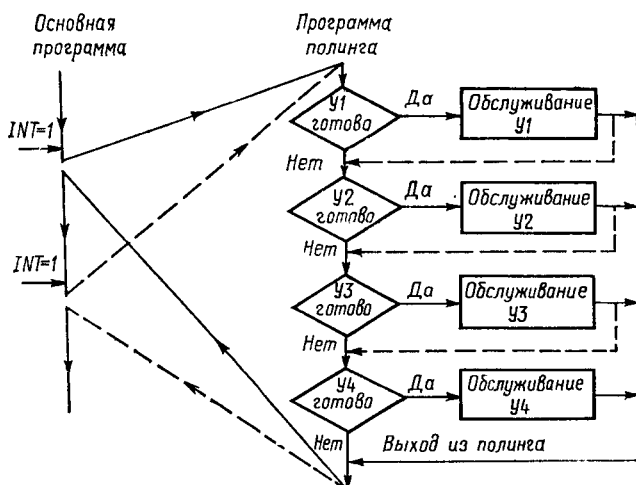


Рис. 5-7. Программный полинг

живания этого устройства, которая определяет необходимые операции ВВ. По завершении обслуживания в полинге может быть запрограммировано одно из следующих действий:

1. Управление возвращается в основную программу без проверки готовности остальных устройств. Здесь гарантируется обязательная проверка в каждом цикле полинга устройств с высоким приоритетом, так как обслуживание их блокирует обслуживание устройств с меньшими приоритетами.

2. Управление возвращается к программе полинга, т. е. в точку проверки прерывания следующего устройства (на рис. 5-7 показано штриховыми линиями). Этот способ гарантирует проверку в каждом цикле полинга всех устройств.

Конкретная реализация полинга зависит от особенностей системы команд процессора и конфигурации аппаратных средств. Например, для ускорения полинга сигналы прерываний всех устройств подключаются к специальному регистру. Полинг реализуется посредством ввода в процессор содержимого этого регистра и анализа состояний отдельных бит с помощью команд сдвига или маскирования. Последовательность анализа определяется приоритетами устройств, и при обнаружении первого установленного бита осуществляется переход к соответствующей подпрограмме обслуживания.

Удобно реализовать программный полинг с помощью команды условного пропуска. При ее выполнении адресуется регистр управления и состояния определенного устройства, проверяется состояние флажка готовности и, если флажок установлен, выполняется следующая по порядку команда перехода к подпрограмме обслуживания адресованного устройства. Если же флажок сброшен, следующая команда пропускается и аналогичным образом проверяется запрос прерывания очередного устройства.

Недостаток программного полинга заключается в необходимости проверки всех устройств, даже тех из них, которые не требуют обслуживания. Каждая проверка представляет собой последовательность команд, которые должен выполнять процессор. При увеличении числа устройств быстро увеличивается и число команд, что приводит к непроизводительным потерям времени процессора и замедлению его реакции на запросы устройств. В результате полинг вводит значительную задержку между моментом, когда устройство сигнализирует о своей готовности к операциям ВВ, и моментом собственно передачи данных.

Основное достоинство программного полинга, как уже отмечалось, заключается в простоте его реализации, почти не требующей дополнительных аппаратных средств.

Аппаратный полинг. В аппаратном полинге (синонимы: дейзи-цепь, приоритетная цепочка, гирляндное или каскадное включение устройств) процессор и все периферийные устройства соединяются таким образом, что процессор может осуществить автоматический опрос с целью идентификации прерывающего устройства (рис. 5-8). Когда процессор реагирует на запрос прерывания, он формирует сигнал подтверждения прерывания INT ACK на линии, которая последовательно проходит через все устройства. При прохождении сигнала по цепочке проверяется состояние флажков готовности устройств. Если устройство не формирует прерывания, сигнал INT ACK проходит в следующее устройство, пока не встретится прерывающее (активное) устройство. Оно блокирует дальнейшее распространение сигнала INT ACK по цепочке. Таким образом, здесь приоритеты устройств определяются их физической близостью к процессору по линии INT ACK.

Затем активное устройство передает по шине данных свой адрес (идентификатор или вектор) вместе со стробирующим (синхронизирующим) импульсом STROBE. Последний информирует процессор о нахождении на шине данных адреса устройства с наибольшим приоритетом, за-

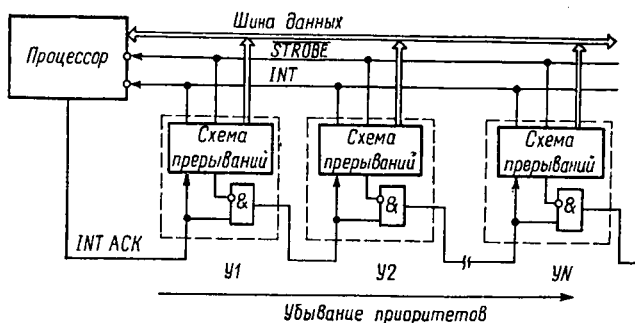


Рис. 5-8. Схема аппаратного полинга

прашивающего обслуживания. Этот адрес имеет однозначное соответствие с начальным адресом подпрограммы обслуживания прерывания данного устройства, в простейшем случае равен начальному адресу. Иногда возвращаемый адрес представляет собой индекс, прибавляемый к содержанию базового регистра процессора.

Адрес, возвращаемый прерывающим устройством, обычно встраивается в интерфейсную плату, и его с помощью переключек или переключателей может изменять пользователь. Приоритет устройства определяется размещением его интерфейсной платы в разъеме, занимающем фиксированное положение в схеме.

При наличии нескольких одновременных запросов прерываний аппаратный полинг обеспечивает быструю идентификацию устройства с наибольшим приоритетом. По линии прерывания к процессору можно подключить больше периферийных устройств. Основной недостаток этого способа связан с задержкой распространения сигнала INT ACK в цепочке. Однако общее время идентификации прерывающего устройства оказывается намного меньше, чем в программном полинге, поэтому аппаратный полинг в мини- и микро-ЭВМ распространен довольно широко.

Векторные и вложенные прерывания. Запрещение прерываний на время обслуживания любого периферийного устройства может привести к потере запросов прерываний быстродействующих, высокоприоритетных устройств, появляющихся при обслуживании устройств с меньшими приоритетами. Чтобы исключить такую ситуацию и обеспечить быструю реакцию процессора на прерывание, необходимо решить две проблемы:

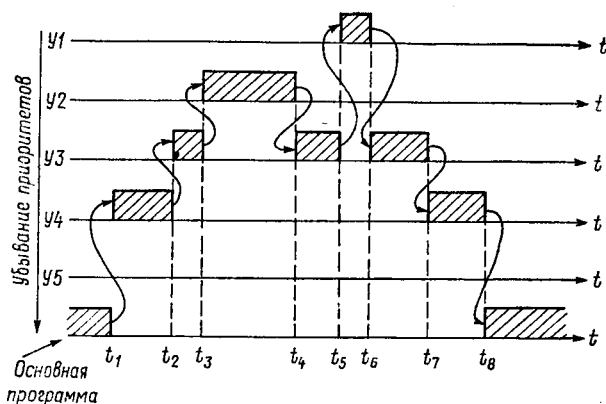


Рис. 5-9. Вложение прерываний

каким-либо образом очень быстро (в пределах с быстродействием комбинационных схем) идентифицировать запрос прерывания устройства с максимальным приоритетом из имеющихся запросов прерываний;

фиксировать текущий приоритет (порог) любой выполняемой процессором программы (в том числе и всех подпрограмм обслуживания прерываний) и разрешать ее прерывание только при возникновении запроса прерывания с большим приоритетом.

Прерывание подпрограмм обслуживания прерываний называется вложением прерываний (рис. 5-9). До момента t_1 выполняется основная программа, которой назначается наименьший приоритет, чтобы процессор реагировал на любые прерывания. В момент t_1 запрашивает обслуживания устройство $У4$ и процессор переходит на его подпрограмму обслуживания. В свою очередь, эта подпрограмма прерывается в момент t_2 запросом от устройства $У3$ с более высоким приоритетом. Подпрограмма обслуживания $У3$ в момент t_3 прерывается запросом устройства $У2$ с еще более высоким приоритетом, и по завершении обслуживания $У2$ в момент t_4 управление возвращается к продолжению обслуживания устройства $У3$. В интервале $t_5 - t_6$ аналогичным образом обслуживается запрос устройства $У1$ с максимальным приоритетом, после чего управление последовательно возвращается к прерванным подпрограммам обслуживания устройства $У3$ и $У4$. Наконец, в момент t_8 возобновляется выполнение основной программы. Разумеется, чтобы про-

цессор реагировал на запросы прерываний, в начале каждой подпрограммы обслуживания их необходимо разрешать командой EI.

В микро-ЭВМ указанные выше проблемы обычно решаются с помощью специальных БИС приоритетных прерываний, типичным примером которых является микросхема K589ИК14. Она рассчитана на подключение сигналов запросов прерываний восьми периферийных устройств, но допускает простое увеличение числа запросов до 64, для чего потребуется 9 таких микросхем без дополнительных элементов. Запросы устройств фиксируются в регистре-защелке, а подключенный к нему шифратор приоритетов формирует 3-битный код запроса с максимальным приоритетом из имеющихся запросов. Этот код при помощи комбинационного компаратора сравнивается с содержимым 3-битного регистра текущего приоритета. Этот регистр также входит в состав микросхемы и допускает программную загрузку. Схема формирует сигнал прерывания INT в микропроцессор, если среди имеющихся запросов прерываний присутствует запрос, приоритет которого выше текущего приоритета. По сигналу подтверждения прерывания INTA на выходе схемы появляется 3-битный код XXX приоритета запроса, на который должен реагировать микропроцессор. Этот код с помощью регистра K589ИР12 трансформируется в однобайтную команду повторного старта RST, имеющую код операции 11XXX111.

Когда микропроцессор КР 580 в конце текущей команды воспринимает запрос прерывания, он сбрасывает внутренний триггер INTE разрешения прерываний и вводит особый машинный цикл прерывания, напоминающий цикл M1 выборки кода операции следующей команды. В такте T₁ на шину адреса выдается содержимое программного счетчика, а на шину данных — байт состояния, в котором бит MEMR считывания из памяти сброшен, а бит INTA подтверждения прерывания установлен. В такте T₂ запрещается инкремент программного счетчика и в счетчике сохраняется адрес команды, которая выполнялась бы при отсутствии прерывания (адрес возврата).

Посредством объединения по схеме И сигнала DBIN приема с шиной данных и бита состояния INTA формируется импульсный сигнал INTA. Этот сигнал через схему приоритетных прерываний выдает на шину данных код 11XXX111, а микропроцессор загружает его в регистр команды. Таким образом, в данном цикле схема priori-

тетных прерываний как бы заменяет программную память.

Команда рестарта RST инициирует выполнение двух действий, аналогичных действиям команды CALL вызова подпрограммы:

текущее содержимое программного счетчика включается в стек, для чего вводятся два цикла M2 и M3 записи в стек;

в программный счетчик загружается адрес ячейки повторного старта 0000000000XXX000, причем биты XXX берутся из регистра команды.

Таким образом, в программном счетчике оказывается один из восьми адресов 0000, 0008, ..., 0038, определяемый кодом XXX, который, в свою очередь, зависит от устройства, вызвавшего прерывание программы. Указанные адреса являются начальными адресами восьми подпрограмм обслуживания прерываний. На каждую подпрограмму резервируется всего 8 байт, но с помощью безусловного перехода она может быть продолжена в любой области памяти. Реакция микропроцессора (выполнение команды RST) длится 11 тактов синхронизации или 5,5 мкс при частоте синхронизации 2 МГц.

Для организации вложенных прерываний в каждой подпрограмме обслуживания прерываний необходимо выполнить следующие действия: разрешить прерывания; временно запомнить приоритет прерываемой программы; загрузить в схему приоритетных прерываний новый текущий приоритет; собственно обслужить прерывание, т. е. выполнить передачу данных; восстановить прежний приоритет; возобновить прерванную программу с помощью команды возврата RET.

Двумерные многоуровневые векторные прерывания. В мини-ЭВМ старших моделей, например в процессорах CM, для которых характерно значительное число периферийных устройств с различным быстродействием, реализуется более сложная, но и более гибкая система прерываний. Она рассчитана на обработку не только внешних аппаратных прерываний от периферийных устройств, но и внутренних аппаратных прерываний, инициируемых при выполнении программы в случае возникновения особых ситуаций (переполнение, нереализованная операция и др.).

Система прерываний называется многоуровневой, если процессор имеет несколько линий (уровней) запросов прерываний и если подпрограммы обслуживания одного уровня могут прерываться запросами на другом уровне. При наличии n уровней допустимая глубина вложений преры-

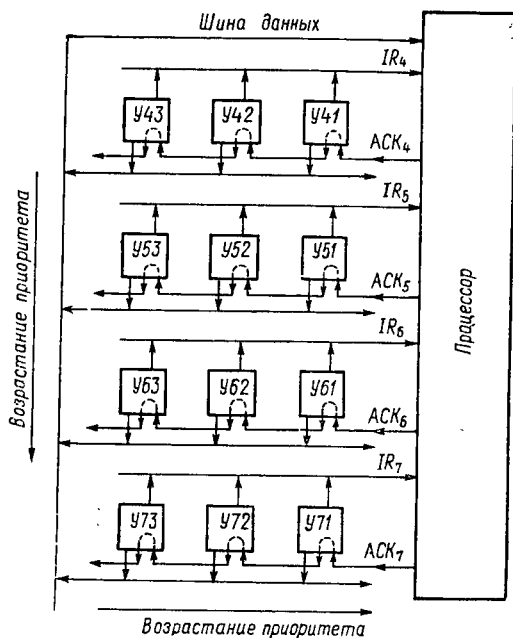


Рис. 5-10. Организация многоуровневых прерываний

ваний равна n . Многоуровневые системы прерываний удобны для группирования периферийных устройств с аналогичными характеристиками (и одинаковыми приоритетами) на одном и том же уровне. Например, все накопители на дисках и ленте подключаются к одному уровню, а все терминалы — к другому, с меньшим приоритетом.

Общая организация системы внешних прерываний показана на рис. 5-10. Каждому периферийному устройству назначается аппаратный приоритетный уровень между 4 и 7, причем высший уровень соответствует 7. Обычно приоритеты назначаются при подключении устройств следующим образом: уровень 4 имеет клавиатура, перфоленточный считыватель и перфоратор, принтер; уровень 5 назначается накопителям на магнитных дисках и ленте; уровень 6 имеет таймер реального времени.

В микро-ЭВМ «Электроника-60» предусмотрен только один уровень, 4. Приоритеты служат характеристиками устройств, и программно их изменять нельзя. Уровень приоритета выполняемой программы от 0 до 7 определяется

содержимым трех бит в слове состояния процессора PSW. Разрешение и запрещение прерываний осуществляется в каждом периферийном устройстве, для чего в их регистрах управления и состояния имеется специальный бит разрешения прерываний (маски). Любое устройство с разрешенными прерываниями, приоритет которого выше приоритета выполняемой программы, может прервать процессор.

Так как данная система прерываний должна обеспечивать быструю реакцию процессора на запрос прерываний, в ней аппаратно реализуются следующие действия:

1. Аппаратный полинг периферийных устройств каждого уровня выполняется автоматически механизмом прерываний процессора. Когда на входах IR_4 — IR_7 появляется запрос прерывания, процессор сравнивает уровень запроса (выбирая из них максимальный в случае нескольких одновременных запросов) с уровнем приоритета текущей программы из PSW. Если уровень приоритета поступившего запроса выше, процессор реагирует формированием соответствующего сигнала подтверждения ACK_4 — ACK_7 . Сигнал ACK подается в цепочку всех устройств данного уровня. Устройство с наибольшим приоритетом данного уровня, запрашивающее обслуживания, блокирует дальнейшее распространение сигнала ACK и идентифицирует себя, выдавая адрес (вектор) на шину данных и специальный квитный сигнал.

2. Содержимое регистров PSW и PC включается в аппаратный (системный) стек, указателем которого служит регистр R6. Это действие аналогично выполнению двух команд:

MOV PSW, — (R6)

MOV R7, — (R6).

Таким образом, на время обслуживания прерывания записываются флажки условий, приоритет прерванной программы и ее адрес возврата.

Затем процессор загружает в регистры PSW и PC содержимое двух смежных ячеек памяти, адрес которых был определен прерывающим устройством. Следовательно, в PC оказывается начальный адрес подпрограммы обслуживания прерывания нужного устройства, а в регистре PSW—состояние процессора для этой подпрограммы, в частности ее уровень приоритета. Обычно подпрограмма имеет такой же уровень приоритета, как и инициировавшее ее устройство. Причина такого назначения приоритета заключается

в том, чтобы предотвратить повторные прерывания до полного обслуживания устройства.

Заключительной командой подпрограммы обслуживания должна быть команда RTI возврата из прерывания. При ее выполнении из стека извлекаются запомненное в нем прежнее содержимое PC и PSW и прерванная программа продолжается из той точки, где она была прервана.

5-4. ОРГАНИЗАЦИЯ БЛОКОВ ОБРАБОТКИ ПРЕРЫВАНИЙ

Организация ввода-вывода с прерыванием, значительно сокращая непроизводительные потери времени процессора, связана с введением в ЭВМ дополнительных аппаратных средств. Благодаря совершенствованию технологии БИС сейчас значительная часть этих средств сосредоточена в микросхемах программируемых контроллеров прерываний и блоков приоритетных прерываний. В литературе достаточно подробно описан блок приоритетных прерываний К589ИК14, который может применяться в системах на базе микропроцессорных семейств К589 и КР580. Большими возможностями обладает микросхема КР580ВН59 программируемого контроллера прерываний, которая рассматривается в настоящем параграфе.

Контроллер обеспечивает управление 8-уровневыми векторными приоритетными прерываниями, число которых может быть легко расширено до 64. Допускается программное маскирование (запрещение) прерываний. Прибор может быть запрограммирован на следующие режимы работы.

Вложенные прерывания. Каждому из 8 входов запросов прерываний IR_{7-0} назначается фиксированный приоритет в порядке возрастания, и запрос с большим приоритетом прерывает обслуживание прерываний с меньшими приоритетами.

Круговой (циклический) приоритет. Как и в предыдущем режиме, каждому входу IR_{7-0} назначается приоритет, но теперь после запроса прерывания и выполнения соответствующей подпрограммы обслуживания приоритеты изменяются в круговом порядке таким образом, что последний обслуженный вход будет иметь низший приоритет. Этот режим характерен для таких применений, в которых периферийные устройства имеют одинаковый приоритет и ни одному из них нельзя отдать предпочтения.

Адресуемые приоритеты. Режим аналогичен второму режиму, но допускает программное определение входа IR , которому назначается низший приоритет.

Режим опроса. В этом режиме прерывания процессора запрещаются, а требующее обслуживания периферийное устройство идентифицируется с помощью считывания состояния контроллера.

При запросе прерывания от периферийного устройства выполняются следующие действия:

на одной или нескольких входных линиях IR_{7-0} формируется переход от низкого уровня к высокому, сигнализирующий о готовности устройства к вводу или выводу;

контроллер воспринимает запросы, учитывает их приоритеты и формирует сигнал INT прерывания процессора; процессор завершает текущую команду и, если прерывания разрешены ($INTE=1$), выдает в контроллер импульс подтверждения прерывания $\overline{INTA}$;

контроллер по шине данных вводит в процессор код операции команды $CALL$ вызова подпрограммы;

код операции $CALL$ заставляет процессор сформировать два дополнительных импульса $\overline{INTA}$, которые осуществляют ввод в процессор двух байт адресной части команды $CALL$, представляющие начальный адрес подпрограммы обслуживания запрашивающего устройства;

управление передается выбранной подпрограмме, а после ее завершения возвращается прерванной программе.

На рис. 5-11 приведена блок-схема контроллера и его программная модель. Основу контроллера составляют следующие узлы: регистр запросов прерываний IRR , регистр обслуживаемых прерываний ISR , регистр маски прерываний IMR , шифратор приоритетов PR , схема каскадирования, буфер шины данных и схемы управления.

Запросы прерываний обрабатываются тремя взаимосвязанными регистрами. Регистр IRR фиксирует все запросы прерываний, регистр ISR хранит обслуживаемые прерывания, а регистр IMR идентифицирует маскируемые (запрещаемые) входы прерываний. Шифратор приоритетов анализирует содержимое этих трех регистров и определяет, необходимо или нет генерировать сигнал INT прерывания процессора.

Входные цепи сигналов IR_{7-0} рассчитаны на восприятие фронта сигнала и сохранение высокого уровня до подтверждения запроса первым импульсом $\overline{INTA}$.

Пусть действует всего один сигнал запроса прерывания и других обслуживаемых прерываний нет. Тогда контроллер сразу формирует сигнал INT , а процессор, реагируя на прерывание, после завершения текущей команды отве-

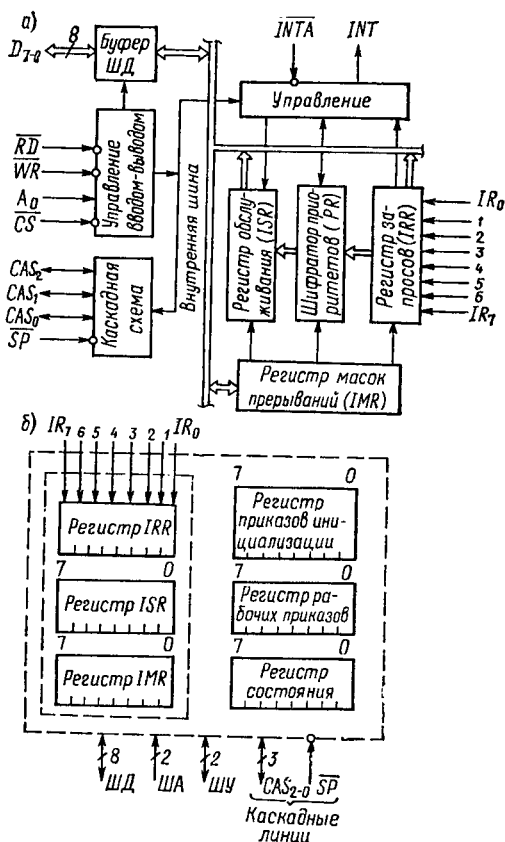


Рис. 5-11. Структурная схема (а) и программная модель (б) микро-схемы КР580ВН59

чает импульсом $\overline{INTA}$. Этот входной сигнал заставляет контроллер выдать на шину данных код операции команды CALL и зафиксировать состояние IRR. Следовательно, высокий уровень IR должен сохраняться минимум до первого сигнала $\overline{INTA}$. Код операции CALL вызывает формирование двух дополнительных импульсов $\overline{INTA}$, передающих из контроллера на шину данных два байта адреса подпрограммы обслуживания прерывания (вектор). Сразу после третьего импульса $\overline{INTA}$ шифратор приоритетов устанавливает в единицу соответствующий бит в регистре ISR и сбрасывает установленный запросом бит регистра IRR. Вход IR к этому моменту может еще находиться в активном со-

стоянии, что не вызывает повторного прерывания, так как последующие запросы прерываний данного входа запрещены. Установка бита ISR предотвращает прерывания от этого и всех других входов с меньшими приоритетами. Подпрограмма обслуживания прерывания при своем завершении должна сбросить бит ISR приказом окончания прерывания EOI, а чтобы повторить процедуру прерывания, вход IR должен вернуться в неактивное состояние.

Если какой-либо вход IR замаскирован, сигнал IR фиксируется в регистре IRR, но установленный бит регистра IMR не пропустит его в шифратор приоритетов. Когда выполняемая программа сбрасывает бит IMR, шифратор приоритетов распознает активный запрос и контроллер генерирует прерывание INT. Отметим, что маскирование входа с некоторым приоритетом не запрещает прерывания от входов с меньшими приоритетами.

Восьмибитный двунаправленный буфер шины данных с тристабильными каскадами используется для интерфейса с системной шиной данных по линиям D₇₋₀. Через него передаются управляющие слова (приказы), информация о состоянии контроллера и команда CALL.

Схемы управления выбирают адресуемые внутренние регистры контроллера, участвующие в его программировании. В них входят регистры приказов инициализации, рабочих приказов и регистр состояния.

Схема каскадирования позволяет объединить несколько контроллеров для расширения числа входов запросов прерываний. При этом один из контроллеров работает как ведущий (управляющий), а до 8 контроллеров — как ведомые (управляемые).

Контроллер имеет следующие управляющие сигналы (рис. 5-12):

$\overline{CS}$ (выбор кристалла) — L-активный входной сигнал, разрешающий связь контроллера с шиной данных;

$\overline{WR}$ (запись) — L-активный сигнал загрузки информации с шины данных в адресуемый регистр контроллера;

$\overline{RD}$ (считывание) — L-активный вход, инициирующий передачу на шину данных информации о состоянии контроллера;

INT (прерывание) — выходной H-активный сигнал, который подается на вход прерывания процессора;

$\overline{INTA}$ (подтверждение прерывания) — входной L-активный сигнал от процессора, на который контроллер реагирует выдачей команды CALL;

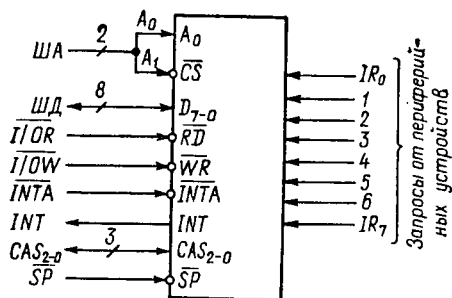


Рис. 5-12. Интерфейс контроллера с системной шиной

A_0 (адрес) — входной сигнал, адресующий внутренний регистр контроллера при загрузке приказов и считывании состояния; обычно подключается к младшей линии A_0 шины адреса;

CAS_{2-0} (линии каскадирования) — при наличии в системе нескольких контроллеров образуют локальную шину и являются выходными линиями ведущего контроллера и входными — ведомых контроллеров;

$\overline{SP}$ (ведомый) — линия определения контроллера как ведущего ($\overline{SP}=1$) или ведомого ($\overline{SP}=0$).

Контроллер воспринимает две разновидности приказов: приказы инициализации ICW и рабочие приказы OCW.

Приказы инициализации. После включения питания каждый контроллер необходимо инициализировать двумя или тремя приказами ICW, которые определяют наличие в системе одного или нескольких контроллеров, начальный адрес подпрограмм обслуживания прерываний и размещение подпрограмм в памяти через 4 или 8 байт.

Последовательность инициализации определяется первым приказом ICW_1 , и для каждого контроллера ее необходимо закончить до поступления от периферийных устройств запросов прерываний. Во время инициализации, которая выполняется с запрещенными прерываниями процессора, сбрасываются схемы фиксации переходов сигналов IR, регистры ISR и IMR, а также триггеры специальной маски и считывания состояния. Входу IR_7 автоматически присваивается низший приоритет, а входу IR_0 — высший.

С каждым входом IR ассоциируется адрес памяти, который выдается на шину данных в ответ на импульсы $INTA$

после выдачи кода операции CALL. Адреса для всех IR расположены равномерно через 4 или 8 байт и обычно содержат таблицу переходов (команды JMP) к соответствующим подпрограммам. Для каждого контроллера 32- или 64-байтные области памяти (иногда называемые страницами) могут находиться в любом месте памяти, начиная с четной границы. Биты A_{15-6} адреса передаются в ICW, а младшие биты A_{1-0} формируются в контроллере. Значение A_5 зависит от заданного интервала и для 4-байтного интервала программируется в ICW₁, а для 8-байтного интервала устанавливается контроллером. Формат младшего байта адреса для каждого входа IR приведен в табл. 5-1.

Таблица 5-1

Формат младшего байта адреса

Вход	Интервал 4 байта								Интервал 8 байт							
	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
IR ₇	A ₇	A ₆	A ₅	1	1	1	0	0	A ₇	A ₆	1	1	1	0	0	0
IR ₆	A ₇	A ₆	A ₅	1	1	0	0	0	A ₇	A ₆	1	1	0	0	0	0
IR ₅	A ₇	A ₆	A ₅	1	0	1	0	0	A ₇	A ₆	1	0	1	0	0	0
IR ₄	A ₇	A ₆	A ₅	1	0	0	0	0	A ₇	A ₆	1	0	0	0	0	0
IR ₃	A ₇	A ₆	A ₅	0	1	1	0	0	A ₇	A ₆	0	1	1	0	0	0
IR ₂	A ₇	A ₆	A ₅	0	1	0	0	0	A ₇	A ₆	0	1	0	0	0	0
IR ₁	A ₇	A ₆	A ₅	0	0	1	0	0	A ₇	A ₆	0	0	1	0	0	0
IR ₀	A ₇	A ₆	A ₅	0	0	0	0	0	A ₇	A ₆	0	0	0	0	0	0

Форматы всех приказов инициализации приведены на рис. 5-13. Отметим, что значение входа A_0 указывается явно, оно должно быть определено при выполнении команды OUT, загружающей приказ в контроллер. В приказе ICW₁ бит F (формат) определяет интервал адресов, а бит S идентифицирует наличие в системе одного или нескольких контроллеров. Поле A_{7-5} содержит биты загружаемого начального адреса. Приказ ICW₂ содержит старший байт начального адреса области памяти. В системе с несколькими контроллерами для ведущего и всех ведомых контроллеров необходим третий приказ ICW₃, определяющий их взаимодействие и зависящий от физического подключения контроллеров.

Приведем пример программы инициализации единственного имеющегося в системе контроллера ($S=1$), если начальный адрес подпрограмм обслуживания прерываний равен 4000 H и интервал соответствует 4 байтам ($F=1$).

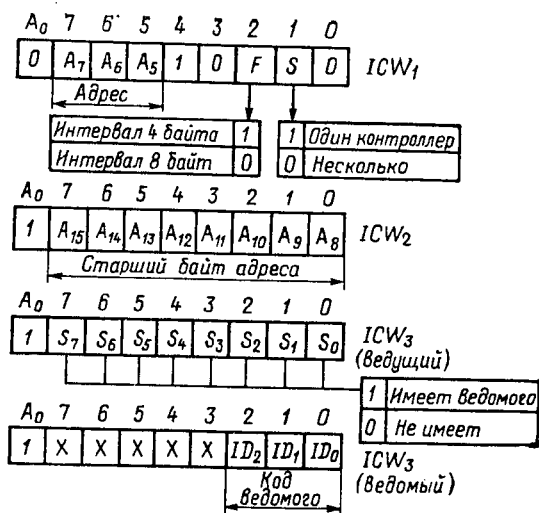


Рис. 5-13. Форматы приказов инициализации

Предполагается, что линия A₀ шины адреса подключена ко входу A₀ контроллера, символический адрес ADR1 имеет четное значение, а ADR2 — нечетное. Кроме того, в обоих адресах один бит имеет нулевое значение и соответствующая ему линия шины адреса подключена на вход $\overline{CS}$ контроллера.

Метка	Код	Операнд	Комментарий
	DI		; Запрещение прерываний процессора
	MVI	A, 16H	; Первый приказ инициализации
	OUT	ADR1	; Второй приказ инициализации
	MVI	A, 40H	; Разрешение прерываний процессора
	OUT	ADR2	
	EI		

После инициализации контроллер готов воспринимать запросы на входах IR₇₋₀ в режиме вложенных прерываний. Входы IR₇₋₀ имеют фиксированные приоритеты: IR₀ — высший, IR₇ — низший. При подтверждении прерывания идентифицируется запрос с наибольшим приоритетом и его вектор выдается на шину данных. Кроме того, устанавливает-

ся соответствующий бит в ISR, что блокирует восприятие прерываний от этого входа и всех входов с меньшими приоритетами. Запросы от входов с большими приоритетами генерируют сигналы INT, которые подтверждаются, если прерывания процессора разрешены командой EI в текущей подпрограмме. (Напомним, что процессор автоматически запрещает прерывания при восприятии сигнала INT, а после команды EI выполняет следующую команду, даже если на входе INT действует высокий уровень). Каждая подпрограмма обслуживания в режиме вложенных прерываний должна информировать контроллер о своем завершении посредством загрузки в него приказа неадресуемого конца прерывания EOI. Этот приказ выдается с $A_0=0$ и $D_{7-0}=20H$ и сбрасывает установленный бит регистра ISR с максимальным приоритетом.

При наличии в системе нескольких контроллеров каждый из них инициализируется автономно. Последовательность инициализации определяет адреса подпрограмм для всех входов IR и взаимодействия ведущего и ведомых контроллеров. Эту функцию выполняет приказ ICW₃; он сообщает ведущему контроллеру, какие его входы IR подключены к ведомым контроллерам, а каждому ведомому — к какому входу IR ведущего контроллера подключен его выход INT. Данный вход представлен полем идентификации ID₂₋₀ в приказе ICW₃ ведомого контроллера. Отметим, что каждый ведомый контроллер может быть запрограммирован на работу в любом режиме.

Рабочие приказы. Приказами инициализации без всякой дополнительной информации любой из контроллеров программируется для работы в режиме вложенных прерываний. Для задания других режимов работы в контроллер необходимо загрузить дополнительные рабочие приказы OCW, форматы которых приведены на рис. 5-14.

С помощью приказа OCW₁, который загружается после инициализации со значением $A_0=1$, в любой момент времени программно можно установить и сбросить отдельные биты регистра масок прерываний IMR. Состояния триггеров этого регистра действуют на выходы регистра IRR, поэтому даже при маскировании входа IR сигнал запроса может установить соответствующий бит IRR, но прерывание процессора не генерируется. После снятия маски запомненный запрос может привести к формированию сигнала INT.

С помощью приказа OCW₂ можно задать пять операций, приведенных в табл. 5-2. Приказ неадресуемого конца прерывания EOI дает возможность сбросить установлен-

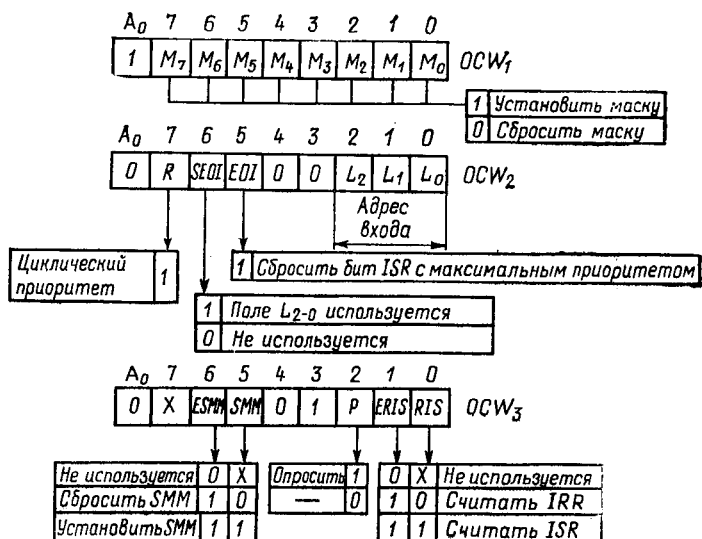


Рис. 5-14. Форматы рабочих приказов

Таблица 5-2

Варианты приказа OCW₂

Приказ	A ₀	7	6	5	4	3	2	1	0	Операция
Неадресуемый EOI	0	0	0	1	0	0	0	0	0	Сброс бита ISR с максимальным приоритетом
Адресуемый EOI	0	0	1	1	0	0	L ₂	L ₁	L ₀	Сброс бита ISR, определяемого полем L ₂₋₀
Установка приоритета	0	1	1	0	0	0	L ₂	L ₁	L ₀	Назначение входу IR, определяемому полем L ₂₋₀ , низшего приоритета
Автоцикл	0	1	0	1	0	0	0	0	0	Сброс бита ISR с максимальным приоритетом и назначение соответствующего входу низшего приоритета
Адресуемый цикл	0	1	1	1	0	0	L ₂	L ₁	L ₀	Сброс бита ISR, определяемого полем L ₂₋₀ , и назначение ему низшего приоритета

ный бит регистра ISR с наибольшим приоритетом, а код адресуемого EOI позволяет сбросить бит ISR, определяемый полем L_{2-0} . Первый из них рекомендуется для режима вложенных прерываний, а второй — для всех остальных режимов, что упрощает отладку системы.

Приказ установки приоритета позволяет программно задать устройство (вход IR) с низшим приоритетом независимо от EOI, т. е. без изменения регистра ISR. Поле L_{2-0} определяет вход IR, которому назначается низший приоритет, а приоритеты остальных входов назначаются в круговом порядке.

Приказ автоцикла сбрасывает бит ISR с наибольшим приоритетом и входу IR, соответствующему только что сброшенному биту, назначает низший приоритет. Приоритеты остальных входов IR модифицируются также в круговом порядке.

Наконец, приказ адресуемого цикла обеспечивает сброс в конце подпрограммы обслуживания бита ISR, адресуемого полем L_{2-0} . Дополнительно соответствующему входу IR назначается низший приоритет, а приоритеты остальных входов упорядочиваются также в круговом порядке.

В табл. 5-3 приведена сводка приказов OCW₃. Первый дополнительный режим называется режимом специальной маски SMM. Пусть в некоторой подпрограмме обслуживания имеется фрагмент, когда необходимо разрешить все прерывания, даже от входов с меньшими приоритетами. Этого можно достичь приказом EOI сброса соответствующего бита ISR. Однако сброс бита ISR необратим и прерывания от устройств с меньшими приоритетами остаются

Таблица 5-3

Варианты приказа OCW₃

Приказ	A ₀	7	6	5	4	3	2	1	0	Операция
Установка SMM	0	×	1	1	0	1	0	0	0	Установка режима специальной маски
Сброс SMM	0	×	1	0	0	1	0	0	0	Сброс режима специальной маски
Режим опроса	0	×	0	0	0	1	1	0	0	Опрос по следующему импульсу $\overline{RD}$
Считывание ISR	0	×	0	0	0	1	0	1	1	Считывание ISR по следующему импульсу $\overline{RD}$
Считывание IRR	0	×	0	0	0	1	0	1	0	Считывание IRR по следующему импульсу $\overline{RD}$

разрешенными и после окончания фрагмента, в котором они допустимы. Для временного прекращения действия бита ISR служит приказ установки режима SMM. Когда задан этот режим, он действует до его сброса специальным приказом.

Действие данного режима иллюстрирует следующая схема подпрограммы обслуживания прерывания от входа IR₄. Как и в предыдущем примере, символические адреса ADR1 и ADR2 обеспечивают выбор контроллера ($\overline{CS}=0$) и имеют, соответственно, нулевое и единичное значения младшего бита, а линия A₀ шины адреса подключена на вход A₀ контроллера:

Метка	Код	Операнд	Комментарий
IR ₄ :	EI		; Разрешение прерываний процессора
			; Здесь фрагмент, где прерывания с меньшими приоритетами запрещены
	DI		; Запрещение прерываний на приказы
	MVI	A, 10H	; Маскирование входа IR ₄
	OUT	ADR2	
	MVI	A, 68H	; Установка режима специальной маски
	OUT	ADR1	; Разрешение прерываний процессора
	EI		; Здесь фрагмент, где прерывания с меньшими приоритетами разрешены
	DI		; Запрещение прерываний на приказы
	MVI	A, 48H	; Сброс режима специальной маски
	OUT	ADR1	
	MVI	A, 00H	; Сброс маски на входе IR ₄
	OUT	ADR2	
	EI		; Разрешение прерываний процессора
			; Здесь фрагмент, где прерывания с меньшими приоритетами запрещены. Должен заканчиваться соответствующим приказом
	EOI		
	RET		; Возврат

Установка режима специальной маски относится ко всем замаскированным входам. Если, например, в приведенной подпрограмме возникает запрос IR₁, когда установлен режим специальной маски, а затем подпрограмма маскирует его, то запросы IR₂ и IR₃ с меньшими приоритетами будут разрешены.

Следующий приказ OCW_3 относится к режиму опроса, в котором для идентификации периферийного устройства, запрашивающего обслуживания, процессор опрашивает не сами устройства, а контроллер, который может работать в любом режиме. Приказ опроса с установленным битом P ($D_2=1$) выдается в контроллер командой OUT . Следующий импульс $\overline{RD}$, генерируемый при выполнении команды IN , выдает на шину данных слово состояний прерываний. В этом слове старший бит D_7 (I) фиксирует наличие прерывания, биты D_{6-3} не используются, а биты D_{2-0} (поле W_{2-0}) идентифицируют код имеющегося запроса прерывания с наибольшим приоритетом.

Обслуживание запросов устройств осуществляется программной дешифровкой считанного слова и переходом к соответствующей подпрограмме. Перед считыванием всегда необходимо загрузить приказ OCW_3 . В том случае когда прерывания отсутствуют, считанное слово содержит $I=0$ и $W_{2-0}=111$. Режим опроса может оказаться удобным в системах с большим числом контроллеров, каждый из которых должен быть правильно инициализирован (начальные адреса в приказах ICW могут быть фиктивными).

Последняя функция OCW_3 — считывание состояния контроллера, представленного содержимым регистров IRR , ISR и IMR . Регистры считываются при загрузке соответствующего приказа OCW_3 с последующим выполнением команды IN или подаче импульса $\overline{RD}$.

Не требуется записывать OCW_3 перед каждым считыванием состояния, если считывается тот же регистр, что и в предыдущей операции. Для считывания содержимого регистра IMR не нужен приказ OCW_3 , так как на линиях данных D_{7-0} контроллера находится это содержимое, если $\overline{RD}=0$ и $A_0=1$.

Каскадное включение контроллеров. Схема контроллера обеспечивает простое увеличение числа запросов прерываний до 64. На рис. 5-15 показана конфигурация с одним ведущим и двумя ведомыми контроллерами, имеющая 22 входа запросов прерываний. Ведущий контроллер определяется высоким уровнем сигнала на входе $\overline{SP}$, а у ведомых приборов этот вход заземлен. Выходы INT ведомых контроллеров подключаются ко входам IR ведущего контроллера. Все линии CAS_{2-0} соединяются параллельно и являются выходными у ведущего и входными у ведомых контроллеров. Сигналами на этих линиях ведущий контроллер вы-

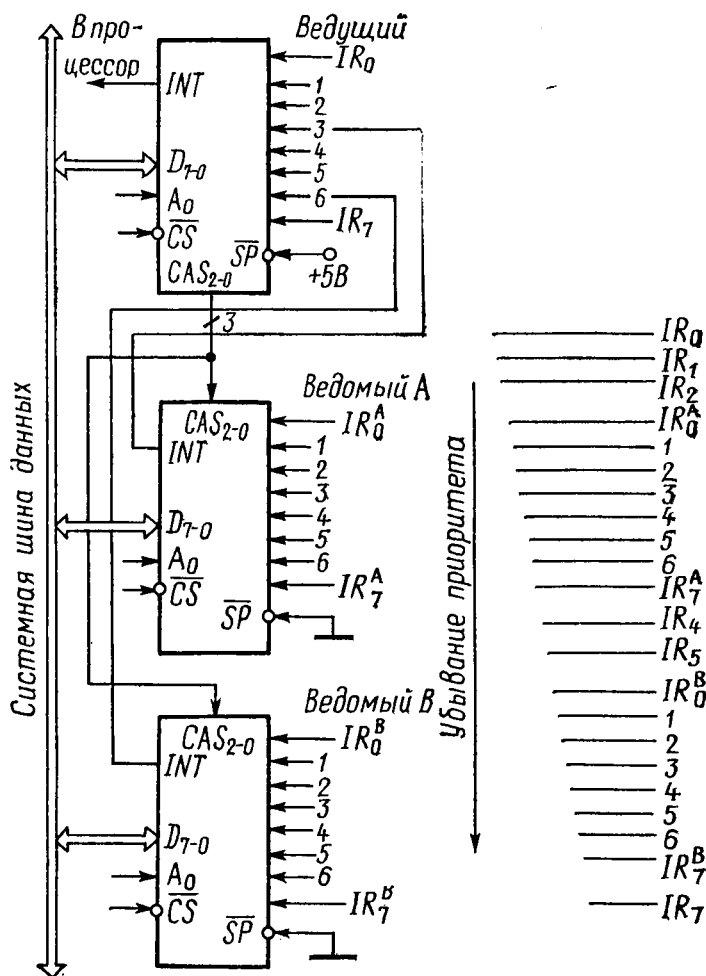


Рис. 5-15. Сопряжение трех контроллеров прерываний

бирает соответствующий ведомый контроллер, который передает в процессор адрес подпрограммы.

Когда на входе IR ведомого прибора появляется сигнал запроса прерывания, приоритет которого выше обслуживаемого им прерывания, на выходе INT формируется высокий уровень. С учетом приоритета прерывания, обслуживаемого ведущим контроллером, на его выходе INT появляется сигнал прерывания процессора. В ответ на первый им-

пульс $\overline{INTA}$ от процессора ведущий контроллер выдает на шину данных код операции $CALL$, а на линии CAS_{2-0} — идентификатор ID_{2-0} прерывающего ведомого контроллера. Поэтому на второй и третий импульсы $\overline{INTA}$ ведомый прибор выдает запрограммированный начальный адрес подпрограммы обслуживания. Кроме того, устанавливаются соответствующие прерывающим входам биты регистров ISR обоих контроллеров. Следовательно, при завершении подпрограммы обслуживания необходимо выдать два приказа EOI : по одному для ведущего и ведомого контроллеров.

При наличии в системе нескольких контроллеров каждый из них инициализируется специальной последовательностью приказов. В табл. 5-4 приведена последовательность инициализации контроллеров, соединенных в соответствии с рис. 5-15. Таблица переходов ведущего контроллера начинается по адресу $100H$, ведомого контроллера А — по адресу $120H$, ведомого контроллера В — по адресу $140H$, и все имеют 4-байтный интервал. Приказ ICW_3 ведущего контроллера показывает наличие ведомых контроллеров на входах IR_3 и IR_6 , а поля ID_{2-0} в приказах TCW_3 ведомых контроллеров содержат 011 и 110. Получающаяся структура приоритетов показана на рис. 5-15.

Разработка подпрограмм обслуживания прерываний для системы с несколькими контроллерами ставит ряд проблем. Пусть ведущий контроллер и ведомый контрол-

Таблица 5-4

Инициализация ведущего и ведомого контроллеров

Контроллер	Приказ	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Ведущий	ICW_1	0	0	0	0	1	0	1	0	0
	ICW_2	1	0	0	0	0	0	0	0	1
	ICW_3	1	0	1	0	0	1	0	0	0
Ведомый А	ICW_1	0	0	0	1	1	0	1	0	0
	ICW_2	1	0	0	0	0	0	0	0	1
	ICW_3	1	0	0	0	0	0	0	1	1
Ведомый В	ICW_1	0	0	1	0	1	0	1	0	0
	ICW_2	1	0	0	0	0	0	0	0	1
	ICW_3	1	0	0	0	0	0	1	1	0

лер А отреагировали на запрос прерывания IR_5^A (то есть установлены биты ISR_5^A и ISR_3) и формируется прерывание на входе IR_2^A . Ведомый контроллер А сформирует сигнал INT, но ведущий контроллер не воспримет его из-за установленного в регистре ISR бита ISR_3 до получения приказа EOI. Этим нарушается вложенность прерываний, так как приоритет запроса по входу IR_2^A выше приоритета обслуживаемого запроса IR_5^A .

Для решения этой проблемы подпрограмма обслуживания периферийного устройства, сигнал прерывания которого подключен к ведомому контроллеру, должна замаскировать у ведущего контроллера входы IR с меньшим приоритетом (в данном случае IR_{7-4}), а затем выдать в ведущий контроллер приказ EOI того входа, к которому подключен ведомый контроллер. Этими действиями обеспечивается восприятие ведущим контроллером прерываний с большим приоритетом (в рассматриваемом примере IR_{4-0}^A). Если входы ведущего контроллера с меньшими приоритетами не замаскировать, он будет реагировать на запросы IR_{7-4} , так как бит ISR_3 сброшен приказом EOI.

Подпрограмма должна также сохранять информацию о том, какой вход ведомого прибора с наименьшим приоритетом обслуживается. Маски входов ведущего контроллера должны сбрасываться при завершении подпрограммы обслуживания прерывания с наименьшим приоритетом ведомого контроллера. Если обслуживается только прерывание от входа IR_2^A , то маски должны сбрасываться соответствующей ему подпрограммой обслуживания. Но в рассмотренном выше примере подпрограмма входа IR_2^A возвращается к подпрограмме обслуживания прерывания IR_5^A . Следовательно, подпрограмма, инициированная прерыванием IR_2^A , должна не сбрасывать маски, а дать возможность сбросить их подпрограмме обслуживания IR_5^A . Это реализуется считыванием и временным запоминанием содержимого регистра IMR ведущего контроллера при вызовах подпрограмм прерываний ведомого контроллера с последующим восстановлением содержимого IMR.

Ниже приведен пример подпрограммы обслуживания прерывания IR_5^A . В ней по-прежнему используются символические адреса $ADR1 (A_0=0, \overline{CS}=0)$ и $ADR2 (A_0=1, \overline{CS}=0)$ ведущего контроллера и вводится адрес $ADRA (A_0=0, \overline{CS}=0)$ ведомого контроллера.

Метка	Код	Операнд	Комментарий
IR5A:	PUSH	B	; Запоминание всего сос-
	PUSH	D	; тояния процессора
	PUSH	H	
	PUSH	PSW	
	IN	ADR2	; Считывание IMR веду-
			; щего контроллера
	MOV	E, A	; Запоминание в регист-
			; ре E
	MVI	A, 0F0H	; Маскирование IR ₇ —IR ₄
	OUT	ADR2	
	MVI	A, 63H	; Адресуемый EOI бита
	OUT	ADR1	; ISR ₃
	EI		; Разрешение прерыва-
			; ний
	; Здесь находится собственно подпрограмма обслуживания		
	DI		; Запрещение прерыва-
			; ний для приказов
	MVI	A, 20H	; Неадресуемый EOI ве-
	OUT	ADRA	; домого контроллера
	MOV	A, E	; Восстановление IMR
	OUT	ADR2	; ведущего контрол-
			; лера
	POP	PSW	; Восстановление состоя-
	POP	H	; ния процессора
	POP	D	
	POP	B	
	EI		; Разрешение прерыва-
			; ний
	RET		; Возврат

5-5. ВВОД-ВЫВОД С ПРЯМЫМ ДОСТУПОМ К ПАМЯТИ

Обмен данными с медленнодействующими периферийными устройствами, например посимвольным принтером, организуется по прерываниям, инициирующим передачу каждого байта или слова данных. Если подпрограмма обслуживания запроса прерывания длится 50 мкс, а скорость передачи не превышает 100 символов/с, то на ввод-вывод расходуется не более 0,5 % времени процессора. Однако при передаче между основной и внешней памятью ЭВМ больших блоков данных (128—2048 байт) производительности (или пропускной способности) процессора в режиме прерываний недостаточно. Например, в накопителях на гибких дисках с одинарной плотностью записи скорость передачи после локализации требуемых данных составляет 31,25К байт/с (интервал между байтами 32 мкс), а при двойной плотности записи скорость передачи будет 62,5К

байт/с (интервал 16 мкс). Вместе с тем цикл полупроводниковой оперативной памяти составляет всего 0,2—2 мкс и, следовательно, скорость передачи ограничивается только процессором. Поэтому для передачи данных непосредственно между устройствами внешней памяти и оперативной (основной) памятью разработан специальный метод передачи данных без участия процессора, получивший название прямого доступа к памяти (ПДП). Аппаратные средства реализации канала ПДП называются контроллером ПДП (КПДП). Первые КПДП выполнялись на микросхемах с малой и средней степенью интеграции, а сейчас в большинстве микропроцессорных семейств выпускаются однокристальные контроллеры ПДП.

В идеальном случае ПДП совершенно не должен влиять на действия процессора, но для этого потребуются сложный и дорогой тракт в основную память. Поэтому в большинстве ЭВМ используется временное разделение (мультиплексирование) общей системной шины между процессором и КПДП. В обычных условиях системной шиной «распоряжается» процессор, управляющий всеми передачами по шине. Когда же иницируется режим ПДП, шиной «распоряжается» КПДП, управляя передачами данных непосредственно между основной и внешней памятью, а действия процессора приостанавливаются и он отключается от системной шины (рис. 5-16). Электрическое отключение достигается переводом тристабильных буферов на шине адреса, шине данных и некоторых линиях управления памятью и вводом-выводом в состояние высокого выходного сопротивления.

Разработано две разновидности ПДП. В режиме идентификации состояния памяти передачи ПДП выполняются без информирования процессора, для чего используются те интервалы машинных циклов, когда процессор не обращается к памяти, а выполняет внутренние преобразования

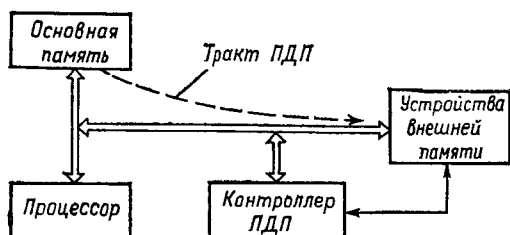


Рис. 5-16. Режим прямого доступа к памяти

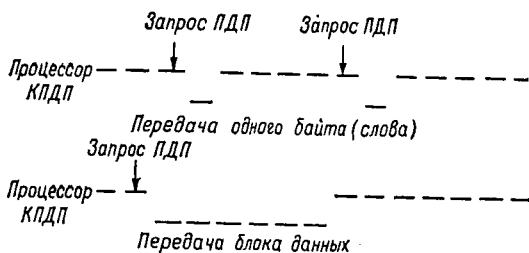
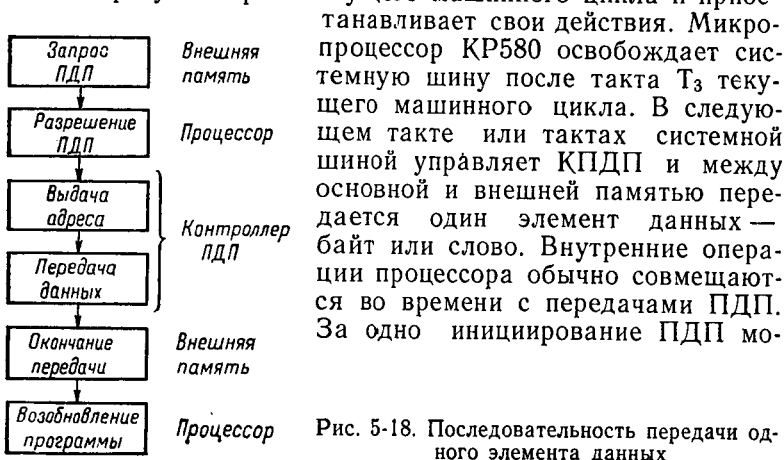


Рис. 5-17. Прямой доступ к памяти с пропуском тактов

данных. Такими интервалами в микропроцессоре КР580 являются такты T_4 и T_5 . Процессор (или дополнительная схема) идентифицирует эти интервалы для КПП специальным сигналом, означающим доступность системной шины. Производительность процессора в таком режиме не уменьшается, но для каждого процессора необходим КПП, а сами передачи носят нерегулярный характер, что ведет к уменьшению скорости передачи данных.

В режиме с пропуском тактов (точнее, с «займом» тактов) КПП при необходимости передачи данных сигналом запроса ПДП DMA REQ или HOLD заставляет процессор отключаться от системной шины на несколько тактов. В процессорах СМ сигнал запроса ПДП называется сигналом запроса внепроцессорной передачи NPR. После восприятия запроса ПДП процессор завершает текущую команду, а иногда фазу выборки текущего машинного цикла и приостанавливает свои действия. Микро-



жет передаваться также целый блок данных, например сектор гибкого диска, что характерно для пакетной передачи. Таким образом, передачи ПДП осуществляются путем пропуска тактов памяти в выполняемой программе. Иллюстрация данной разновидности ПДП приведена на рис. 5-17. При выполнении передач ПДП содержимое внутренних регистров процессора не модифицируется, поэтому его не нужно запоминать в памяти, а затем восстанавливать, как при обработке прерываний. Выполнение программы возобновляется сразу после снятия сигнала запроса ПДП. Тем не менее в условиях интенсивных передач ПДП эффективная производительность процессора уменьшается.

Аппаратная реализация каналов ПДП определяется особенностями ЭВМ и устройств внешней памяти, но можно сформулировать общие принципы работы большинства каналов ПДП. На рис. 5-18 показаны основные этапы передачи одного элемента данных в режиме ПДП, называемой также циклом ПДП. Последовательность передачи иницируется сигналом запроса от устройства внешней памяти, например накопителя на гибком диске; этот сигнал контроллер ПДП транслирует в запрос для процессора. Процессор реагирует сигналом подтверждения (разрешения) DMA АСК или HLDA, приостанавливает программу (но может продолжать внутренние операции) и отключается от системной шины. Для осуществления передачи КПДП выдает на шину адреса адрес ячейки памяти, к которой будет обращение для записи или считывания. После этого по шине данных выполняется собственно передача данных, для чего КПДП генерирует необходимые управляющие сигналы основной и внешней памяти. Затем устройство внешней памяти снимает запрос ПДП, что приводит к снятию соответствующего запроса в процессор, и он возобновляет приостановленную программу. Если в ЭВМ имеется несколько устройств внешней памяти, описанная последовательность должна в себя включать этап выборки устройства, инициировавшего запрос ПДП, и блокирования остальных устройств.

При передаче блока данных последовательность на рис. 5-18 необходимо дополнить действиями модификации адреса основной памяти и проверки окончания передачи блока. Для этого в КПДП загружаются начальный адрес области основной памяти и число передаваемых байт или слов. После каждой передачи в КПДП производится инкремент адреса, выдаваемого на шину адреса, и декремент счетчика байт или слов. Передача блока завершается, когда счет-

чик достигает нуля. Иногда вместо числа байт или слов загружается конечный адрес области основной памяти, и передача завершается, когда текущий адрес достигает конечного.

Типичным примером современного КПДП служит микросхема КР580ВТ57, входящая в состав микропроцессорного семейства КР580. Как показано выше, главные функции КПДП заключаются в выдаче адресов основной памяти и необходимых управляющих сигналов, обеспечивающих непосредственную передачу данных между основной и внешней памятью. Микросхема может управлять работой четырех независимых каналов ПДП с учетом приоритетов периферийных устройств. Запрос системной шины осуществляется через вход HOLD микропроцессора, и предусмотрены сигналы, обеспечивающие регулярное завершение передач ПДП. Для микропроцессора КПДП представляет несколько параллельных 8-битных портов ввода и вывода. После программной инициализации контроллер управляет передачей блока данных до 16К байт без вмешательства процессора. Допускается программирование любого из трех режимов работы: считывания (передача из основной памяти во внешнюю), записи (передача из внешней памяти в основную) и проверки канала ПДП. В последнем режиме физической передачи данных между основной и внешней памятью не производится.

На рис. 5-19 показана блок-схема контроллера и его программная модель. Каждый из четырех каналов ПДПК₃₋₀ имеет три регистра: 16-битный регистр адреса, 14-битный счетчик и 2-битный регистр режима канала. Последние два регистра объединяются в один 16-битный регистр, который далее ради простоты называется счетчиком. До выполнения любых операций каким-либо каналом ПДП необходимо инициализировать относящиеся к нему регистры: в регистр адреса загружается начальный адрес области основной памяти, в счетчик загружается число передаваемых байт¹, а в регистр режима канала — требуемый режим его работы (00 — проверка, 01 — запись, 10 — считывание, 11 — запрещенная комбинация).

Программная модель контроллера содержит 18 8-битных регистров, причем в регистр режима контроллера можно только записывать, а из регистра состояния — только считывать. Остальные регистры допускают и запись, и считывание.

¹ При необходимости передачи N байт в счетчик загружается двоичный код числа N—1.

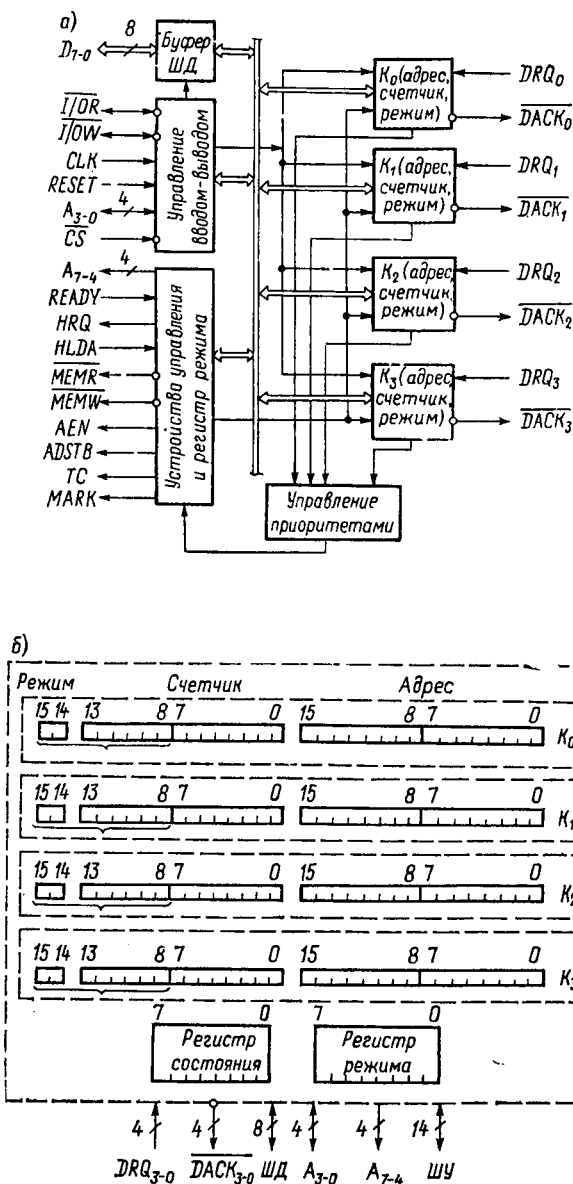


Рис. 5-19. Структурная схема (а) и программная модель (б) микро-
схемы КР580ВТ57

По четырем входным линиям DRQ_{3-0} запросов ПДП периферийные устройства сигнализируют об их готовности к передаче данных. Инициирование цикла ПДП осуществляется установкой Н-уровня на линии DRQ и сохранением его до подтверждения ПДП. В пакетной передаче Н-уровень DRQ сохраняется до подтверждения передачи последнего байта. Программно запросам DRQ могут быть назначены фиксированные приоритеты (DRQ_0 — высший, DRQ_3 — низший) или может быть задан режим циклического приоритета, т. е. после обслуживания некоторого канала ПДП ему назначается низший приоритет, а приоритеты остальных каналов изменяются в круговой последовательности. Например, после обслуживания канала CH_2 приоритеты каналов убывают в таком порядке: CH_3 , CH_0 , CH_1 , CH_2 . Данный режим приводит к тому, что приоритеты всех четырех периферийных устройств оказываются одинаковыми и ни одно из них не может блокировать запросы других.

Установкой L-уровня на одной из четырех выходных линий $DACK_{3-0}$ подтверждения ПДП контроллер извещает периферийное устройство о начале запрошенного им цикла ПДП.

Интерфейс КПДП с системной шиной данных D_{7-0} осуществляется через стандартный двунаправленный тристабильный буфер. При программировании процессор загружает через него байты в любой из 8-битных внутренних регистров, кроме регистра состояния, а при необходимости, например для контроля, считывает содержимое любого регистра, кроме регистра режима. Эти операции выполняются командами вывода OUT или ввода IN , вторым байтом которых адресуется целевой регистр, а при использовании ввода-вывода, отображенного на память, могут выполняться любыми командами с обращением к памяти.

Когда контроллер получает управление системной шиной, в начале цикла ПДП он выводит на линии D_{7-0} старшие 8 бит A_{15-8} адреса памяти из старшей половины регистра адреса работающего канала ПДП. Это действие идентифицируется Н-активным стробом адреса $ADSTB$, который должен загрузить байт с шины данных во внешний регистр-зашелку, например в многорежимный буферный регистр $K589IP12$. Выходы этого регистра подключаются к линиям A_{15-8} шины адреса.

Логическая схема управления считыванием и записью при программировании контроллера или при считывании содержимого внутреннего регистра (контроллер является ведомым) воспринимает входные сигналы I/OW или I/OR ,

дешифрует код на младших линиях адреса A_{3-0} и выполняет следующие действия: если сигнал $\overline{I/O\overline{W}}=0$, байт с шины данных загружается в адресуемый кодом A_{3-0} регистр; если сигнал $\overline{I/O\overline{R}}=0$, содержимое адресуемого регистра передается на шину данных. Когда же контроллер управляет системной шиной, т.е. является ведущим, эта же схема генерирует пары сигналов $\overline{I/O\overline{R}}-\overline{MEMW}$ при передаче данных в основную память и $\overline{I/O\overline{W}}-\overline{MEMR}$ при передаче данных в противоположном направлении.

К рассматриваемой схеме относятся следующие сигналы и соответствующие им линии:

$\overline{I/O\overline{R}}$ (считывание ввода-вывода) — L-активный сигнал, который в зависимости от состояния контроллера (ведомый — ведущий) может быть входным или выходным. Отрицательный импульс на входной линии $\overline{I/O\overline{R}}$ ведомого контроллера считывает на шину данных содержимое внутреннего регистра, а выходной отрицательный импульс ведущего контроллера используется для передачи байта из периферийного устройства на шину данных. Линия $\overline{I/O\overline{R}}$ имеет внутренний тристабильный буфер.

$\overline{I/O\overline{W}}$ (запись ввода-вывода) — L-активный сигнал, который также может быть входным или выходным. Отрицательным импульсом на входе $\overline{I/O\overline{W}}$ ведомого контроллера байт с шины данных загружается в адресуемый внутренний регистр, а выходной сигнал $\overline{I/O\overline{W}}$ ведущего контроллера записывает байт с шины данных в выбранное периферийное устройство. Эта линия также имеет внутренний тристабильный буфер.

A_{3-0} — четыре двунаправленные младшие линии адреса. Для ведомого контроллера они являются входными и адресуют один из внутренних регистров, участвующий в операциях записи (программирования) или считывания. Ведущий контроллер выдает, на них младшие биты адреса памяти и регистра адреса работающего канала ПДП.

$\overline{CS}$ (выбор кристалла) — входной L-активный сигнал, определяющий режим ведомого контроллера, когда он воспринимает входные сигналы $\overline{I/O\overline{W}}$ и $\overline{I/O\overline{R}}$. Вход $\overline{CS}$ ведущего контроллера автоматически запрещается, чтобы при адресации основной памяти контроллер не мог выбрать сам себя. Этот вход подключается к схеме дешифрации адреса.

RESET (сброс) — входной H-активный сигнал сброса, загружающий нулевой байт в регистр режима контролле-

ра. После действия сигнала сброса до программного разрешения контроллер не реагирует на сигналы DRQ запросов ПДП.

CLK (синхронизация) — входная линия, на которую подаются ТТЛ-сигналы второй фазы системной синхронизации.

Устройство управления с входящими в него регистрами режима и состояния контроллера выполняет действия в необходимой последовательности в передачах ПДП, генерируя требуемые управляющие сигналы и 16-битные адреса основной памяти. С ним ассоциируются следующие сигналы и соответствующие им линии:

A₇₋₄ — четыре выходные линии адреса, имеющие тристабильные буферы. На эти линии контроллер выдает соответствующие биты регистра адреса во всех передачах ПДП. Вместе с линиями A₃₋₀ определяют младший байт адреса основной памяти. Подключаются к аналогичным линиям шины адреса.

READY (готовность) — входной Н-активный сигнал, обеспечивающий введение состояний ожидания контроллера, если быстрое действие основной памяти недостаточно для синхронной работы.

HRQ (запрос ПДП или захват шины) — выходной Н-активный сигнал, который подается на вход HOLD микропроцессора и сигнализирует о необходимости его отключения от системной шины.

HLDA (подтверждение запроса ПДП или захвата шины) — входной Н-активный сигнал от процессора, свидетельствующий, что контроллер может управлять системной шиной.

MEMR (считывание из памяти) — выходной L-активный сигнал, инициирующий считывание из адресуемой ячейки основной памяти в передачах ПДП. Данная линия имеет внутренний тристабильный буфер.

MEMW (запись в память) — выходной L-активный сигнал, определяющий запись в адресуемую ячейку основной памяти в передачах ПДП. Имеется внутренний тристабильный буфер. Выходы MEMR и MEMW соединяются с соответствующими линиями схемы управления системой (системного контроллера).

ADSTB (строб адреса) — выходной Н-активный сигнал, используемый для загрузки старшего байта адреса основной памяти, который КПДП выдает на шину данных, во внешний регистр-защелку.

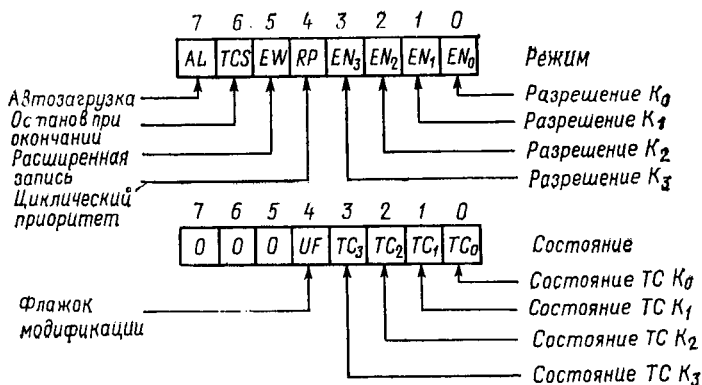


Рис. 5-20. Формат регистров режима и состояния

AEN (разрешение адреса) — выходной Н-активный сигнал, который используется для перевода шин адреса и данных и некоторых линий шины управления другими компонентами системы в состояние высокого выходного сопротивления. Подается на входы разрешения выхода OE или выбора устройства DS шинных усилителей-формирователей, связывающих процессор и другие устройства (кроме, разумеется, основной памяти) с системной шиной.

ТС (окончание счета) — выходной Н-активный сигнал, идентифицирующий последний цикл ПДП текущего блока данных. Формируется при достижении нуля (точнее, перехода через нуль) в 14-битном счетчике работающего канала ПДП.

MARK (маркер или метка) — выходной Н-активный сигнал маркера по mod 128. Сигнализирует в периферийное устройство, что текущая передача ПДП является 128-й от момента формирования предыдущего сигнала MARK. Если общее число N передач ПДП кратно 128 (напомним, что в счетчик загружается значение N—1), сигнал MARK появляется при 128-й передаче от начала блока данных.

Форматы регистров режима и состояния контроллера показаны на рис. 5-20. Регистр режима, определяющий общие функции контроллера, программируется после загрузки регистра адреса и счетчика канала ПДП. Он сбрасывается сигналом RESET, что приводит к запрещению использования всех каналов ПДП и предотвращает конфликты на шине при включении питания. Рекомендуется не разрешать работу канала ПДП до загрузки в его регистры ис-

ходных данных, чтобы случайные запросы ПДП не исказили содержимого основной памяти.

Четыре младших бита EN_{3-0} определяют состояние соответствующих каналов ПДП. Если $EN_i=0$, т. е. i -й канал запрещен, контроллер не реагирует на запрос DRQ_i ; когда же $EN_i=1$, т. е. i -й канал разрешен, контроллер удовлетворяет запрос DRQ_i .

Бит RP определяет фиксированный приоритет запросов ПДП ($RP=0$) или циклический приоритет ($RP=1$).

При установке бита EW расширенной (удлиненной) записи несколько увеличивается продолжительность сигналов $MEMW$ и $I/O\overline{W}$, генерируемых контроллером. Это позволяет не вводить дополнительного состояния ожидания через сигнал готовности $READY$ и несколько увеличивает скорость передачи данных.

Если установлен бит TCS разрешения останова при окончании передач ПДП, после формирования сигнала TC соответствующий канал ПДП запрещается, т. е. его бит EN сбрасывается. Дальнейшие передачи в этом канале возможны только после повторной загрузки регистра режима контроллера. Когда же $TCS=0$, появление сигнала TC не влияет на состояние канала ПДП и закончить операции ПДП должно периферийное устройство.

Бит AL автозагрузки позволяет использовать канал K_2 для повторных передач предыдущего блока данных или сцепленных блоков данных без программного вмешательства между передачами блоков. Для этого регистры K_2 инициализируются обычным образом, а в регистры K_3 загружаются параметры повторной инициализации K_2 . После завершения в K_2 передачи первого блока данных, о чем сигнализирует $TC=1$, в цикле модификации (обновления) параметры из регистров K_3 загружаются в регистры K_2 . При этом состояние бита TCS (если $AL=1$) не влияет на работу K_2 . Когда $AL=1$, при программировании K_2 его начальные параметры автоматически дублируются в регистрах K_3 , что обеспечивает повторяющиеся передачи одного и того же блока данных. Такое повторение удобно при использовании КПДП для регенерации видеодисплея. Но в регистры K_3 после загрузки регистров K_2 допускается загружать другие значения параметров.

Восьмибитный регистр состояния содержит четыре бита TC , отражающих состояния окончания счета соответствующих каналов ПДП, и бит UF флажка модификации. Биты TC устанавливаются одновременно с выдачей сигнала TC соответствующих каналов и сбрасываются сигналом

RESET или при считывании содержимого регистра состояния в процессор. Флажок UF устанавливается, когда в режиме автозагрузки параметры K_3 передаются в регистры K_2 , и не сбрасывается при считывании состояния. Его можно перевести в нулевое состояние посредством сигнала RESET или аннулирования режима автозагрузки. Кроме того, флажок автоматически сбрасывается после передачи первого блока вслед за модификацией регистров K_2 . Контролируя состояние UF, процессор определяет момент завершения повторной инициализации K_2 и, следовательно, возможность безопасной загрузки параметров нового блока данных.

Программирование КПП осуществляется командами OUT или любыми командами запоминания байта при использовании ввода-вывода, отображенного на память. При этом сигналы на линиях A_{3-0} определяют адрес целевого внутреннего регистра, а сигналы на линиях A_{15-4} после дешифрации используются для формирования сигнала CS. Собственно запись в целевой регистр выполняется сигналом I/OW. Выбор регистров контроллера определяется табл. 5-5.

Таблица 5-5

Адресация внутренних регистров контроллера

Регистр	Байт	Линии адреса				Триггер F/L	Линии шины данных							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Адрес K_0	Младший	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Счетчик K_0	Старший	0	0	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
	Младший	0	0	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	Старший	0	0	0	1	1	C_7	C_6	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
Адрес K_1	Младший	0	0	1	0	0	Аналогично K_0							
Счетчик K_1	Старший	0	0	1	0	1								
	Младший	0	0	1	1	0								
Старший	Старший	0	0	1	1	1								
	Младший	0	1	0	0	0								
Адрес K_2	Старший	0	1	0	0	1								
	Младший	0	1	0	1	0								
Счетчик K_2	Старший	0	1	0	1	1								
	Младший	0	1	0	1	1								
Адрес K_3	Старший	0	1	1	0	0								
	Младший	0	1	1	0	1								
Счетчик K_3	Старший	0	1	1	1	0								
	Младший	0	1	1	1	1								
Режим	—	1	0	0	0	0	AL	TCS	EW	RP	EN ₃	EN ₂	EN ₁	EN ₀
Состояние	—	1	0	0	0	0	0	0	0	UF	TC ₃	TC ₂	TC ₁	TC ₀

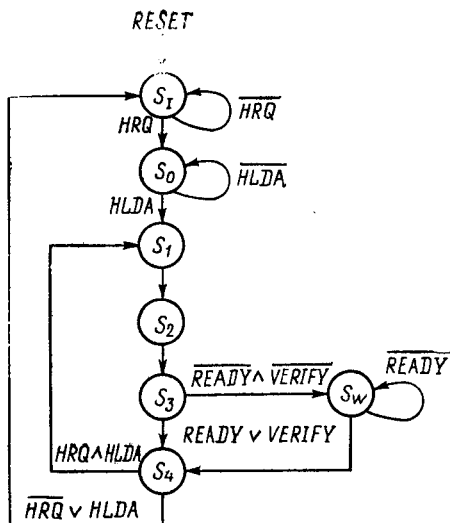


Рис. 5-21. Диаграмма состояний контроллера

Бит A_3 определяет обращение к канальным регистрам ($A_3=0$) или к регистрам режима и состояния ($A_3=1$). При выборе канального регистра бит A_0 идентифицирует регистр адреса ($A_0=0$) или счетчик ($A_0=1$). Оставшиеся входные линии адреса $A_{2,1}$ определяют один из четырех каналов.

Для загрузки и считывания содержимого каждого из 16-битных регистров необходимы две команды. В контроллере предусмотрен специальный триггер F/L (первый/второй, или младший/старший), состояние которого определяет старшую или младшую половину выбранного канального регистра. Триггер F/L сбрасывается при действии сигнала RESET и при загрузке регистра режима КППД. Для обеспечения правильной инициализации команды обращения к канальным регистрам должны быть парными и первая из них должна обращаться к младшей половине регистра. Нельзя формировать $\overline{CS}=0$ при активном уровне сигналов $\overline{I/O\overline{W}}$ или $\overline{I/O\overline{R}}$, так как это вызовет ошибочный переход триггера F/L. В системе с возможными прерываниями при программировании КППД или считывании содержимого его регистров прерывания следует запрещать.

Контроллер имеет семь внутренних состояний, связи между которыми показаны на рис. 5-21. В холостом состоянии S_1 он не выполняет операций ПДП. Цикл ПДП начи-

нается с установки H-уровня на любой из линий DRQ запроса ПДП. Контроллер переходит в состояние S_0 , формирует сигнал запроса HRQ в процессор и ожидает сигнала подтверждения HLDA. В этом состоянии продолжается опрос линий DRQ с учетом их приоритетов. При получении сигнала HLDA от процессора контроллер выдает сигнал подтверждения $\overline{DACK}$ в запрашивающее устройство с наивысшим приоритетом и переходит в состояние S_1 . H-уровень на линии DRQ должен сохраняться до получения сигнала $\overline{DACK}$, когда передается один байт, или до выполнения условия $\overline{DACK}=0$ и $TC=1$, свидетельствующего о передаче блока данных в пакетном режиме. Каждый цикл ПДП состоит минимум из четырех состояний ($S_1S_2S_3S_4$). Если быстроедействие основной или внешней памяти недостаточно для передачи во временном интервале, определяемом сигналами синхронизации, с помощью L-уровня на входе готовности READY между состояниями S_3 и S_4 вводится необходимое число состояний ожидания S_w . В режиме проверки (VERIFY) сигнал READY не используется.

При передаче данных в основную память выходной сигнал I/OR генерируется в начале состояния S_2 , а сигнал $\overline{MEMW}$ — в начале состояния S_3 . При передаче данных из основной памяти в начале состояния S_2 генерируется сигнал $\overline{MEMR}$, а сигнал I/OW — в начале состояния S_3 . Указанные последовательности формирования управляющих сигналов обеспечивают правильную передачу данных. Сигнал расширенной записи I/OW или $\overline{MEMW}$ формируется в состоянии S_2 . Управляющие сигналы записи и считывания при работе канала ПДП в режиме проверки не генерируются.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что понимается под архитектурой ввода-вывода?
2. Каковы достоинства и недостатки изолированного ввода-вывода и ввода-вывода, отображенного на память?
3. Каким образом в системе с вводом-выводом, отображенным на память, можно упростить дешифраторы адреса регистров периферийных устройств?
4. Какие режимы ввода-вывода используются в мини- и микро-ЭВМ? Сравнить их по быстрдействию и дополнительным аппаратным и программным средствам.
5. В чем главное отличие маскируемых и немаскируемых прерываний?
6. Можно ли использовать команду повторного старта RST в прикладных программах? Рассмотреть особенности ее выполнения и отличия от команды CALL.

7. В чем заключаются достоинства и недостатки программного и аппаратного полинга?

8. Какие характеристики периферийных устройств определяют назначаемые им приоритеты?

9. Какую информацию необходимо включать в стек при обслуживании прерываний?

10. Какие операции целесообразно предусмотреть в системе команд процессора для ускорения контекстного переключения?

11. В чем заключаются особенности всех режимов работы программируемого контроллера прерываний?

12. Сколько ведомых контроллеров можно подключить к одному ведущему контроллеру прерываний?

13. Какие функции выполняет контроллер прямого доступа к памяти?

14. Какие факторы определяют максимальную скорость передачи данных в режиме прямого доступа к памяти?

15. Каким образом операции прямого доступа к памяти влияют на реальную производительность системы?

ГЛАВА ШЕСТАЯ

ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

6-1. КЛАССИФИКАЦИЯ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

Периферийные (или внешние) устройства, обеспечивающие взаимодействие ядра ЭВМ с окружающей средой, играют важную роль во всех применениях ЭВМ, так как их технические характеристики в значительной степени определяют производительность и эффективность всего комплекса. Параллельно с эволюцией собственно ЭВМ непрерывно идет совершенствование периферийных устройств. Значение периферийного оборудования особенно возрастает в связи с массовым производством дешевых мини- и микро-ЭВМ. Их стоимость часто оказывается ниже стоимости традиционного периферийного оборудования средних и больших ЭВМ, рассчитанного на эксплуатацию в условиях вычислительных центров и имеющего значительные габариты и массу. Необходимы исследования и разработки новых малогабаритных, дешевых и надежных средств ввода, вывода и хранения цифровых данных. Без этих средств ЭВМ превращается в бесполезный набор электронных схем. За последние годы в области периферийного оборудования достигнут заметный прогресс, который в перспективе обещает быть еще более значительным.

В данной главе приводится классификация периферийного оборудования, дается обзор перспективных разработок и более подробно рассматриваются дисплеи и устройства массовой памяти.

По быстродействию периферийные устройства традиционно делятся на два больших класса: 1) устройства ввода и вывода данных со сравнительно невысокой скоростью обмена данными, достигающей 10К байт/с; 2) внешние запоминающие устройства (или устройства массовой памяти), обеспечивающие после локализации необходимых данных гораздо более высокую скорость обмена.

Такая классификация, являясь в общем правильной, оказывается неполной и требует дальнейшей детализации. Представляется удобным учитывать специфику разнообразных периферийных устройств, если придерживаться следующей классификации:

1. Периферийные устройства, предназначенные для связи с пользователем. Их, в свою очередь, можно разделить на устройства ввода (клавиатура, переключатели, перфоленточные считыватели и графические планшеты), устройства вывода (индикаторы, различные печатающие устройства, или принтеры, ленточные перфораторы и графопостроители) и интерактивные (диалоговые) устройства, включающие в себя дисплеи и принтеры с клавиатурой, устройства речевого ввода и вывода.

2. Устройства массовой памяти, представленные разнообразными накопителями на магнитных лентах и дисках, памятью на приборах с зарядовой связью и цилиндрических магнитных доменах.

3. Устройства связи с объектом управления, включающие в себя аналого-цифровые и цифро-аналоговые преобразователи, многочисленные датчики, цифровые регуляторы и исполнительные устройства (типа шаговых двигателей), сигнализаторы, реле, таймеры и счетчики событий и т. д. Этот класс периферийных устройств наиболее динамичен и широк, поскольку многие новые применения ЭВМ ассоциируются с разработкой новых типов таких периферийных устройств.

4. Средства передачи данных на большие расстояния (средства телекоммуникации), к которым относятся модемы, мультиплексоры, демультиплексоры и концентраторы данных.

6.2. УСТРОЙСТВА ДЛЯ СВЯЗИ С ПОЛЬЗОВАТЕЛЕМ

Устройства ввода. Наиболее широко для ручного ввода символьных данных в ЭВМ применяется клавиатура. При нажатии клавиши формируется 7- или 8-битный код соответствующего символа, который загружается во входной буфер в памяти, либо во внутренний регистр процессора. Для уменьшения числа клавиш каждая из них несет символы нижнего и верхнего регистров, а специальная клавиша выбирает используемый регистр. Имеются также клавиши управляющих символов: пробел, возврат каретки (CR), перевод строки (LF), возврат на шаг и др. Клавиатура должна удовлетворять требованиям высокой надежности, бесшумности, одинаковых усилий и одинакового хода каждой клавиши и др. Иногда используется цветное кодирование клавиш различного функционального назначения.

Разработано много конструкций переключателей, которые могут использоваться в клавиатуре. В простых механических переключателях при движении плунжера, связанного с клавишей, замыкается электрический контакт. Недостаток таких переключателей состоит в явлении дребезга контакта, приводящем к тому, что в течение нескольких миллисекунд формируется нестабильный сигнал. Более совершенными, надежными и быстродействующими являются герметизированные магнитоуправляемые контакты (герконы). В бесконтактных переключателях используется эффект Холла, явление насыщения магнитного материала и изменение емкости при перемещении пластинки, связанной с плунжером.

Однополюсные переключатели, связанные с каждой клавишей, организуются в матрицу, число строк и столбцов которой определяется общим числом клавиш. При нажатии клавиши переключатель замыкает соответствующие строку и столбец (рис. 6-1). Для сканирования матрицы и формирования кодов символов разработаны специальные микросхемы шифраторов клавиатуры (ШК). Шифратор сканирует клавиатуру, поочередно устанавливая высокий уровень на одной из 9 выходных линий X_i и последовательно проверяя сигналы на 10 входных линиях Y_j . Скорость сканирования определяется частотой внешних сигналов синхронизации CLK. Если, например, частота синхронизации составляет 30 кГц, шифратор обнаруживает замыкание клавиши в течение 3 мс. Совокупность активных сигналов линий X_i и Y_j используется для адресации внутреннего по-

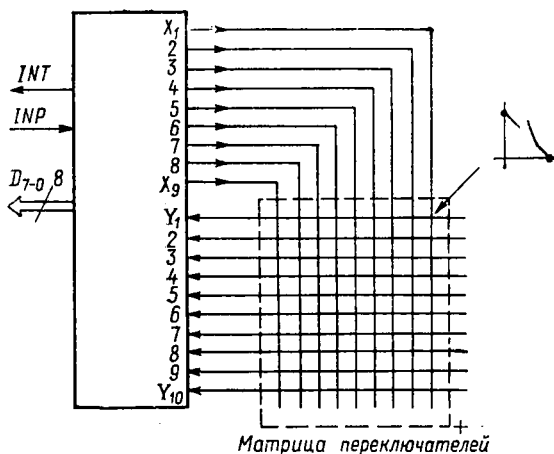


Рис. 6-1. Схема подключения клавиатуры

стоянного запоминающего устройства, хранящего коды символов. Сигнал прерывания INT свидетельствует о нажатии клавиши и формировании кода символа, а сигнал ввода INP передает код символа на линии данных D_{7-0} .

Обработка одновременного нажатия нескольких клавиш осуществляется в одном из двух режимов. В режиме блокировки фиксируется первая нажатая клавиша, а восприятие остальных нажатых клавиш блокируется до отпущения первой. В режиме запоминания клавиши, нажимаемые одна после другой, идентифицируются в определенной последовательности, если каждое нажатие происходит в различных циклах сканирования. Когда фиксируется нажатие клавиши, оно будет игнорироваться в последующих циклах до тех пор, пока эта клавиша не будет нажата повторно. Если в одном цикле оказываются нажатыми две или более клавиш, будет обнаружена первая сканируемая клавиша, затем вторая и т. д. Для такого режима необходимы однонаправленные переключатели, что достигается включением последовательно с контактами диодов.

В схеме шифратора клавиатуры предусматривается подавление дребезга контактов. Когда обнаруживается нажатие клавиши, цикл приостанавливается на несколько миллисекунд и состояние клавиатуры устройством не принимается.

Перфоленточное оборудование. В большинстве ЭВМ имеются традиционные устройства вывода на перфоленту

(выходной перфоратор) и ввода с перфоленты (ленточный считыватель). Оба устройства допускают применение перфолент с любым числом дорожек и работают в start-стопном режиме. Перфоратор ПЛ-150 имеет скорость перфорации до 150 строк/с, а считыватель FS-1501 вводит до 1500 символов в секунду.

Индикаторы. В простейшем случае данные от ЭВМ индицируются в двоичной форме «включено — выключено», для чего применяются сигнальные лампы, светодиоды и другие индикаторы. Для пользователей гораздо удобнее выводить данные в привычной буквенно-цифровой (символьной) форме. Выходной алфавит включает в себя цифры, буквы (обычно только прописные) и специальные знаки. Изображения отдельных символов формируются либо на точечной матрице 5×7 , 7×9 или 10×14 элементов, либо на сегментном поле с числом сегментов 7, 9 или 14.

В накаливаемых сегментных индикаторах для каждого сегмента имеется отдельная нить накала. При пропуске через нить электрического тока соответствующий сегмент начинает светиться.

Газоразрядные (или плазменные) индикаторы основаны на свечении разряда, возникающего в газовой смеси под воздействием электрического напряжения. Основным компонентом смеси является неон с небольшими добавками аргона и криптона. Разработаны плоские конструкции сегментных газоразрядных индикаторов.

Наибольшее распространение получили индикаторы на светодиодах, в которых используется эффект видимого свечения при прямом смещении $p-n$ -перехода некоторых полупроводников, например фосфида галлия. Светодиодные индикаторы отличаются небольшими габаритами, малой потребляемой энергией, высокой надежностью и большим сроком службы. Имеются плоские сегментные индикаторы на светодиодах.

Действие сегментных жидкокристаллических индикаторов основано на изменении оптических свойств некоторых жидкостей под воздействием электрического поля. Жидкость находится между двумя параллельными стеклянными пластинками, внутренняя поверхность которых покрыта прозрачными пленочными электродами. Напряжение, действующее между электродами, создает управляющее электрическое поле. В пропускающем индикаторе формируется темный символ на светлом фоне, а в отражательном индикаторе наблюдается светлый символ на темном

фоне. В основном жидкокристаллические индикаторы применяются в калькуляторах.

Для работы индикатора внутренний 7- или 8-битный код символа преобразуется в управляющий код отдельных элементов матрицы или сегментов. Длина управляющего кода определяется числом элементов матрицы или сегментов. Необходимое преобразование осуществляется программно либо аппаратно, с помощью преобразователей кода на базе постоянного запоминающего устройства. В цифровых сегментных индикаторах, рассчитанных на вывод только шестнадцатиричных цифр, преобразователь может быть встроен в корпус индикатора.

Светодиодные, газоразрядные и жидкокристаллические индикаторы объединяются в одну конструкцию и обеспечивают вывод строки (а иногда и целой страницы) символов. Чтобы уменьшить число усилителей-формирователей, работа многосимвольного индикатора организуется с временным мультиплексированием. Принцип мультиплексирования 4-символьного 7-сегментного светодиодного индикатора показан на рис. 6-2. Одноименные сегментные катоды всех индикаторов объединены и подключены к выходам семи усилителей-формирователей, сигналы которых определяют выводимый символ. На аноды подаются распределенные во времени импульсы, которые как бы включают индикатор, когда на выходах усилителей имеются напряжения соответствующего сегментного кода. Если частота коммутации превышает 30 Гц, глаз человека перестает замечать мерцание изображений символов. Разумеется, средняя яркость свечения индикаторов в мультиплексном режиме соответственно уменьшается.

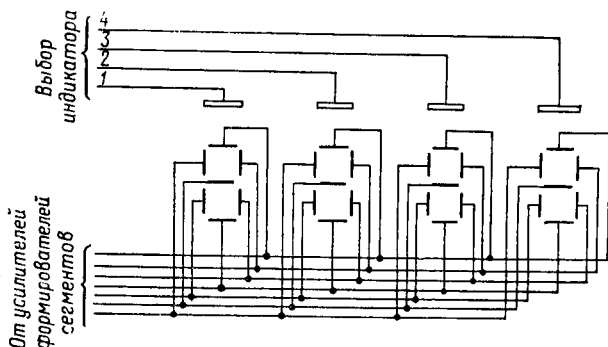


Рис. 6-2. Мультиплексирование индикаторов

Большими возможностями по выводу информации обладают газоразрядные индикаторные панели. Они представляют собой матрицу 128×128 , 256×256 или 512×512 элементов (газоразрядных ячеек), состоянием которых управляют сигналы, действующие на двух ортогональных системах электродов. Суммарное напряжение двух электродов больше потенциала ионизации, что инициирует в ячейке видимый разряд. Выводимые данные можно сохранять, подавая переменное напряжение, или модифицировать импульсами по координатным электродам.

Принтеры предназначены для вывода и документирования символьных данных и ограниченных графических изображений. Широко распространенные электрические пишущие машинки «Консул» обеспечивают также ввод данных с клавиатуры.

Многочисленные конструкции принтеров классифицируются следующим образом. В посимвольных (последовательных) принтерах символы строки текста печатаются последовательно, один за другим, а в построчных (параллельных) принтерах одновременно формируются символы целой строки. В ударных принтерах используется традиционный способ печати посредством удара печатающей головки или рычага с литерой по красящей ленте и переноса краски на бумагу. В безударных принтерах формирование изображений символов не связано с механическим движением печатающей головки. Наконец, собственно изображения символов могут быть слитными или точечными. В последнем случае принтеры называются знаковсинтезирующими.

Кроме собственно печати посимвольные принтеры выполняют управляющие операции, основными из которых являются: возврат каретки — перемещение каретки или печатающей головки в левую позицию, перевод строки — транспорт бумаги на новую строку, пробел — перемещение на одну позицию вправо без печати, возврат на шаг — перемещение на одну позицию влево без печати и перевод формата — транспорт бумаги на новую страницу.

Один из недостатков рычажных пишущих машинок, определяющих их сравнительно небольшую скорость печати, до 25 симв/с, заключается в наличии громоздкой и инерционной каретки, которую необходимо перемещать на одну позицию после печати каждого символа и возвращать в исходное состояние после печати строки текста. Чтобы уменьшить инерцию движущихся узлов, используется подвиж-

ная сферическая печатающая головка с литерами символов выходного алфавита. Головка может поворачиваться относительно двух осей для перевода в ударную позицию литеры нужного символа. Собственно печать осуществляется ударом всей головки по красящей ленте. Набор сменных головок обеспечивает быстрое изменение выходного алфавита.

Знакосинтезирующие посимвольные ударные принтеры выпускаются в виде игольчатых принтеров или принтеров с баллистической печатающей головкой. Изображения символов формируются на матрице 5×7 или 7×9 элементов (точек). Код выводимого символа подается в знакогенератор, который последовательно формирует сигналы, соответствующие столбцам матрицы. Эти сигналы вызывают срабатывание электромагнитов, которые, в свою очередь, приводят в движение семь или девять игл (штифтов). Они ударяют по красящей ленте, и на бумаге печатаются точки одного столбца. Затем головка передвигается на следующую позицию, печатается очередной столбец и т. д. После печати одного символа головка движется в следующую символьную позицию и процесс повторяется до завершения печати всей строки текста. В баллистической печатающей головке срабатывание электромагнитов вызывает удар молоточков по печатающим иглам. Движение головки производится с помощью ходового винта. Скорость печати в таких принтерах доведена до 180 симв/с.

Ударные построчные принтеры, называемые обычно алфавитно-цифровыми печатающими устройствами (АЦПУ), в мини- и микро-ЭВМ применяются редко из-за громоздкости конструкции и высокой стоимости. Скорость печати их достигает 1200 строк/мин, а число символов в строке обычно составляет 128.

Из безударных знаковинтезирующих устройств следует отметить термографические принтеры. В основе их действия лежит локальный нагрев специальной термочувствительной бумаги, приводящий к изменению ее цвета. Печатающая головка имеет небольшие нагревательные элементы, которые формируют сегментные или точечные изображения символов, когда головка прижимается к бумаге и необходимые элементы нагреваются до температуры около 200°C . Скорость печати посимвольных принтеров достигает 30 симв/с, а построчных (с несколькими печатающими головками) доходит до 1000 строк/мин.

Безударные построчные принтеры (электроискровые, электрографические, магнитографические и др.) в мини- и

микро-ЭВМ не используются по тем же причинам, что и ударные построчные принтеры.

Многие современные принтеры имеют встроенный микропроцессорный контроллер. Он позволяет организовать печать в обоих направлениях с сокращением числа возвратов каретки, совмещать печать текущей и прием следующей строки текста, программно управлять вертикальным форматированием, обеспечивать прямую адресацию печатающей головки по горизонтали и вертикали, изменять размеры символов в знаковосинтезирующих принтерах, выполнять диагностические подпрограммы, вызывающие печать всего алфавита или любого выбранного символа и др.

Передача данных от ЭВМ в медленнодействующие принтеры осуществляется в последовательном формате, а в быстродействующие принтеры — в параллельном формате.

Устройства речевого ввода и вывода. Исследования речевого (голосового) взаимодействия человека с ЭВМ ведутся в течение долгого времени, но только за последние годы в этом направлении достигнуты заметные успехи. Речевой ввод освобождает человека от клавишного ввода, увеличивает скорость и уменьшает число ошибок при вводе, а речевой вывод дает возможность оператору одновременно выполнять другие функции.

Имеется два основных метода формирования распознаваемой речи. Метод прямого синтеза речи опирается на управление шумовым сигналом в соответствии с параметрами, которые передают сущность фонем — наименьших элементов речи. Параметры определяются физическими процессами речеобразования и правилами лингвистики. Достоинство метода заключается в неограниченном словаре, поскольку число фонем конечно. Память необходима только для хранения фонем и программы, осуществляющей поиск необходимых фонем в требуемой последовательности. Однако практическая реализация этого метода усложняется такими факторами, как искажения на границах слов, изменения высоты и громкости звука и др.

Второй метод заключается в воспроизведении записанных элементов (слов и коротких фраз), которые выбираются в нужной последовательности управляющим процессором. Воспроизведение осуществляется путем сцепления отдельных элементов, поэтому данный метод называется еще синтезом с помощью конкатенации. Словарь такого устройства ограничен емкостью и быстродействием памяти, предназначенной для хранения элементов сообщений.

Поскольку некоторые синтезируемые сообщения лишены смысла, число полезных сообщений намного меньше общего числа комбинаций хранимых элементов.

Речевой ввод (точнее, распознавание речи) подразумевает, что ЭВМ распознает и соответственно реагирует на произносимые слова, хотя бы в пределах запрограммированного словаря. Распознавание опирается на набор признаков, идентифицирующих произносимые слова. В качестве признаков используются две или три интенсивные форманты (частотные диапазоны с наибольшей акустической энергией), формантные переходы между согласными и гласными звуками и др. Алгоритм распознавания с аппаратной или программной реализацией включает в себя точное количественное измерение и взвешивание признаков. Работа большинства устройств речевого ввода опирается на спектральный анализ входного сигнала. Словарь либо фиксируется в схеме устройства, либо формируется самим устройством в режиме обучения. В новых устройствах распознавания речи имеются микропроцессоры, память значительной емкости и специализированные БИС. Например, в устройстве со словарем в 100 слов, имеющем режим обучения, емкость оперативной памяти для хранения эталонных признаков и программного постоянного запоминающего устройства составляет 4К байт.

6.3. УСТРОЙСТВА МАССОВОЙ ПАМЯТИ

В настоящее время основными устройствами массовой памяти являются накопители на магнитных лентах и дисках. Для мини- и микро-ЭВМ разработано несколько новых конструкций накопителей, которые рассмотрены в § 6-6 и 6-7. Недостаток магнитных накопителей заключается в необходимости механического движения носителя, что уменьшает их быстродействие и несколько снижает надежность. Альтернативой магнитных накопителей считаются устройства на цилиндрических магнитных доменах (ЦМД) и приборах с зарядовой связью (ПЗС), однако их широкому применению пока препятствует высокая стоимость и сложная технология.

ЦМД-память. Действие ЦМД-памяти основано на образовании в тонких пленках некоторых магнитных материалов (в основном гранатов редкоземельных элементов) крошечных цилиндрических доменов («пузырьков») и на передвижении их внешним магнитным полем. Когда на пленку действует поле смещения H , домены, направление собствен-

ных полей которых противоположно направлению H , сокращаются в размере и принимают форму цилиндров с диаметром в несколько микрометров. Поле H создается постоянным магнитом.

Домены передвигаются вращающимся магнитным полем вдоль пермалловых доменопредвигающих структур (аппликаций), имеющих Т-образную и шевронную конфигурацию. Вращающееся с частотой около 100 кГц магнитное поле создается парой сдвинутых по фазе токов I_x, I_y . Токи подаются в катушки, охватывающие корпус микросхемы.

Таким образом, появляется возможность построить память в виде длинного регистра сдвига, в котором наличие домена означает единицу, а отсутствие — нуль. Как и все магнитные устройства, такая память оказывается энерго-независимой и не требует непрерывной регенерации: токи I_x, I_y можно выключить и содержимое памяти сохраняется неизменным.

Генерирование доменов осуществляется с помощью проводника, изогнутого по форме шпильки. При пропускании через него импульса тока высотой порядка 1 А возникает магнитное поле, напряженности которого достаточно для инверсии ориентированных доменов в магнитном слое. Подключение генератора на вход регистра сдвига позволяет вводить в регистр последовательные двоичные данные. Импульс тока противоположного направления вызывает исчезновение (аннигиляцию) домена, что при необходимости реализуется операцией стирания. Для восприятия домена используется магниторезистивный эффект в некоторых материалах, например в пермаллое. Вначале с помощью расширителя осуществляется вытягивание домена, чтобы увеличить выходной сигнал детектора. Когда растянутый домен проходит под столбцом последовательно включенных пермалловых шевронов, сопротивление столбца изменяется, что приводит к формированию считанного сигнала в несколько милливольт. Подавление помех производится с помощью мостовой схемы, в два плеча которой включены рабочий и фиктивный магниторезисторы. В некоторых структурах памяти необходима операция дублирования (раздвоения, или репликации) домена, которая реализуется импульсом тока, подаваемым в специальный проводник.

Простейшая структура памяти имеет вид последовательной петли, показанной на рис. 6-3. Основу прибора составляет длинный регистр сдвига емкостью 16К бит и более, в котором данные циркулируют под воздействием вращаю-

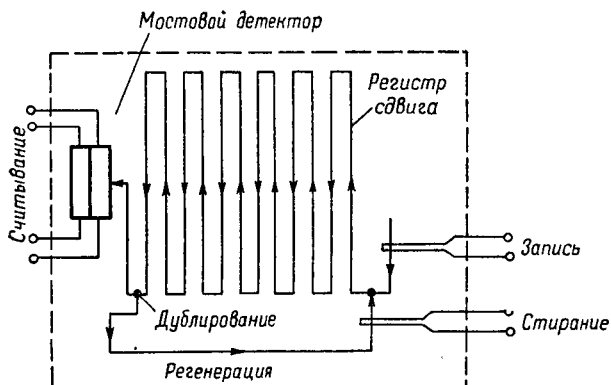


Рис. 6-3. Последовательная ЦМД-память

щегося магнитного поля. Ввод данных осуществляется с помощью проводников стирания и записи. Первый из них уничтожает домены в тракте рециркуляции, а второй генерирует домены при записи единиц. Для считывания с помощью репликатора создаются копии доменов, находящихся в регистре сдвига, и направляются в магниторезистивный детектор. Недостаток такой структуры заключается в сравнительно большом времени обращения к произвольному биту, которое в среднем равно времени сдвига домена на половину длины регистра.

ПЗС-память. Действие ПЗС-памяти основано на передаче зарядовых пакетов по цепочке емкостей (конденсаторов), образованной в МОП-структуре. Передачей управляют внешние сигналы синхронизации. Прибор аналогичен набору регистров сдвига, хранящиеся в которых биты представлены наличием или отсутствием зарядов. ПЗС-память оказывается энергозависимой и требует непрерывной регенерации.

Металлические электроды на поверхности подложки представляют собой линейный массив близко расположенных МОП-емкостей, выполняющих функции запоминающих элементов. Когда на электрод подается положительное напряжение, основные носители в р-легированной подложке отталкиваются и под электродом образуется обедненный слой. На границе диэлектрик — кремний создается потенциальная яма для неосновных носителей (электронов). Распространению потенциальной ямы по поверхности кремния препятствуют специальные области полупроводника.

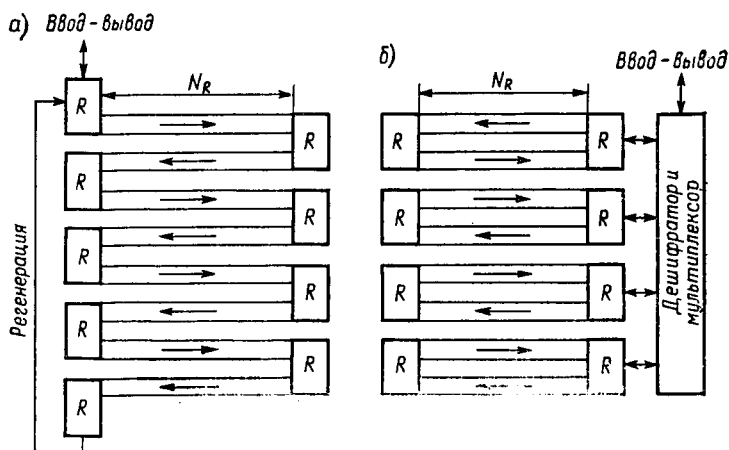


Рис. 6-4. Структуры приборов ПЗС-памяти

Обедненный слой заполняется электронами от соседних областей канала или от схемы инжекции заряда. Если разместить МОП-емкости так близко друг от друга, что их потенциальные ямы смогут перекрываться, то заряд неосновных носителей накапливается в том месте, где поверхностный потенциал выше. В соответствии с гидравлической аналогией можно считать, что заряд (как жидкость) течет в наиболее глубокую часть потенциальной ямы.

В синхронной серпантинной структуре, приведенной на рис. 6-4, а, все хранимые биты сдвигаются по одному тракту и синхронизируются одной и той же частотой f_c . Запоминающие элементы образуют один регистр сдвига, в который включены промежуточные усилители восстановления R . Число бит N_R между усилителями зависит от эффективности передачи и минимальной частоты синхронизации в пассивном режиме. Среднее время обращения составляет $MN_R/(2f_c)$, где M — число сегментов серпантина. Время обращения в операции считывания сокращается, если каждый сегмент имеет отдельный выход. Недосток серпантинной структуры заключается в том, что все биты сдвигаются с одной частотой синхронизации, определяемой скоростью передачи данных. При работе на высокой частоте кристалл потребляет значительную энергию, а источник синхроимпульсов должен иметь мощный выход. Кроме того, большое число усилителей сокращает полезную площадь кристалла и емкость прибора. В некоторых приборах име-

ется несколько серпантинов, например девять, по 1024 бита каждый. Ввод и вывод осуществляются по девяти двуполупроводниковым линиям с тристабильными буферами.

На рис. 6-4, б показана ПЗС-память с петлевой организацией, в которой все запоминающие элементы по-прежнему работают с одинаковой частотой синхронизации. Каждая автономная петля имеет усилитель регенерации. Содержимое одной петли считывается последовательно, и для получения высокой скорости передачи данных требуется мощный источник синхриимпульсов. Среднее время обращения к памяти составляет N_R/f_c . Если, например, $N_R = 128$ бит и $f_c = 2$ МГц, среднее время обращения равно 64 мкс. На кристалле должен размещаться дешифратор адреса, выбирающий для записи или считывания адресуемую петлю.

Широкое применение ПЗС-памяти пока сдерживается сложной технологией и высокой стоимостью. В некоторых случаях отрицательную роль играет энергозависимость ПЗС-памяти.

6-4. УСТРОЙСТВА СВЯЗИ С ОБЪЕКТАМИ УПРАВЛЕНИЯ И ПЕРЕДАЧИ ДАННЫХ

Устройства связи с объектами управления. Как упоминалось выше, рассматриваемый класс периферийных устройств отличается наибольшим разнообразием, которое является следствием многочисленных применений мини- и микро-ЭВМ в системах управления реальными объектами. Наиболее распространенными представителями устройств этого класса считаются входные аналого-цифровые преобразователи (АЦП) и выходные цифро-аналоговые преобразователи (ЦАП). На рис. 6-5 показана структурная схема системы с многоканальным аналоговым вводом-выводом. Аналоговый мультиплексор МПЛ коммутирует на вход формирующего усилителя ФУ сигналы n входных ка-

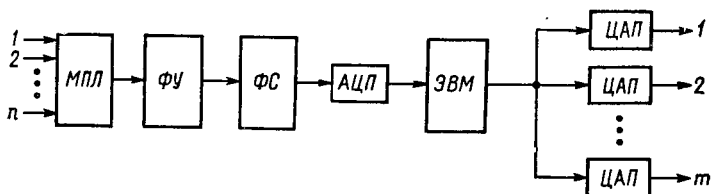


Рис. 6-5. Система управления с многоканальным аналоговым вводом-выводом

налов. Усилитель приводит уровни сигналов к шкале АЦП. Схема выборки и хранения (называемая далее фиксирующей схемой ФС) обеспечивает постоянный уровень отсчета сигнала на входе АЦП во время преобразования. ЭВМ воспринимает от АЦП дискретные величины, обрабатывает их по программе и выдает цифровые результаты обработки, которые с помощью ЦАП превращаются в аналоговые сигналы и распределяются между исполнительными устройствами.

Устройства передачи данных. Проблема передачи значительных объемов данных на большие расстояния включает в себя такие вопросы, как ввод данных в ЭВМ с удаленных терминалов и вывод в обратном направлении, создание сетей ЭВМ, использование разделенных баз данных, организация распределенной обработки. Когда машины находятся на большом расстоянии друг от друга, естественно использовать для их взаимодействия существующие телеграфные и особенно телефонные линии. Однако при этом необходимо учитывать следующие обстоятельства: телеграфные и телефонные системы при их создании не рассчитывались на передачу цифровых данных, и требуется разработка специальных методов передачи таких данных; при передаче неизбежно появляются ошибки, которые устройства передачи должны обнаруживать и каким-либо образом исправлять; стоимость средств передачи данных достаточно высока, и следует стремиться к максимальному использованию их возможностей.

Телефонные линии, имея полосу пропускания 300—3300 Гц, рассчитаны на передачу аналоговых сигналов, и для передачи цифровых данных необходимы специальные устройства, называемые модемами (модулятор-демодулятор). По существу модемы представляют собой сложный цифро-аналоговый и аналого-цифровой преобразователь. Тракт связи, показанный на рис. 6-6, содержит модем-передатчик, который преобразует цифровые сигналы от ЭВМ в аналоговые сигналы, предназначенные для передачи по

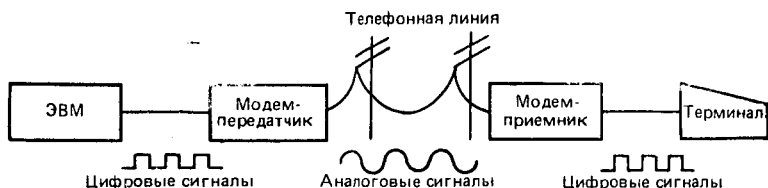


Рис. 6-6. Общая схема связи ЭВМ с удаленным терминалом

телефонным линиям, и модем-приемник, выполняющий обратную функцию. В идеальном случае модемы неощутимы для пользователя и нормально ЭВМ и дистанционный терминал работают так, как будто они находятся рядом и имеют непосредственную связь.

Условия конкретного применения определяют скорость передачи данных, режим передачи (односторонний или двухсторонний), вид передачи (небольшие группы или большие пакеты) и некоторые другие характеристики. По скорости передачи модемы классифицируются на низкоскоростные (до 600 бит/с), среднескоростные (1200—2400 бит/с) и высокоскоростные (4800 бит/с и выше). Необходимо различать скорость передачи в битах в секунду и в бодах. Первая соответствует фактической скорости передачи данных, а вторая характеризует число сигнальных элементов, передаваемых в одну секунду. В медленноразнодействующих модемах эти величины совпадают, но в быстродействующих модемах скорость в бодах меньше. Такое отличие объясняется тем, что для достижения высокой скорости передачи в модемах применяются сложные методы модуляции, в которых один сигнальный элемент содержит несколько бит.

При заданной ширине полосы пропускания линии и отношении сигнал-шум существует теоретический предел скорости передачи информации, который для телефонных линий составляет примерно 10К бит/с. Поэтому максимальная скорость передачи по телефонным линиям равна 9600 бит/с. Специальные линии обеспечивают скорость передачи 19,2К бит/с.

Различают три режима передачи данных: симплексный, полудуплексный и дуплексный. Симплексный режим обеспечивает одностороннюю связь между двумя точками, в одной из которых находится передатчик, а в другой — приемник. Полудуплексная связь обеспечивает двухсторонний обмен данными между двумя точками, в каждой из которых имеются передатчик и приемник, но одновременная передача в двух направлениях невозможна. Для изменения направления передачи требуется некоторое время переключения (коммутации). В дуплексном режиме допускается одновременная передача данных в двух направлениях, для чего необходимы четыре телефонные линии или специальный модем с частотным мультиплексированием. Каждый из режимов имеет разновидности, например, в полудуплексном режиме вводится обратный канал, по которому терминал передает подтверждения приема или диагностическую

информацию с низкой скоростью (к примеру, 50 бит/с), а принимает данные от ЭВМ по прямому каналу с более высокой скоростью (скажем, 2400 бит/с).

Синхронизация передаваемых данных осуществляется двумя способами. В большинстве низко- и среднескоростных модемов применяется асинхронная передача, в которой каждый символ окаймляется стартовым и стоповыми битами. При обнаружении стартового бита приемник воспринимает определенное число следующих бит как данные и заканчивает прием при распознавании стопового бита. Таким образом, стартовый и стоповые биты обеспечивают посимвольную синхронизацию передатчика и приемника, но значительно уменьшают скорость передачи данных. Например, при использовании одного стартового, двух стоповых бит и 7-битных кодов символов 30 % передаваемых бит предназначены только для синхронизации. Высокоскоростные модемы работают в синхронном режиме, когда большие блоки данных передаются без стартовых и стоповых бит. Синхронизация обеспечивается специально установленными символами синхронизации.

Модемы и интерфейсные схемы обеспечивают передачу бит, но для передачи осмысленных сообщений необходим более высокий уровень управления, называемый протоколом линии связи, который выполняет следующие функции: установление связи между терминалами, управление потоком данных, обнаружение и исправление ошибок, синхронизацию. Подробное рассмотрение протоколов линий выходит за рамки настоящей книги.

Стремление увеличить эффективность использования линий связи привело к разработке специальных устройств мультиплексирования данных. Мультиплексирование заключается в объединении потоков данных от нескольких источников, работающих с невысокой скоростью, в одну быстродействующую линию связи.

На приемной стороне другое устройство, называемое демультимплексором, осуществляет разделение медленных потоков.

В мультиплексорах с частотным разделением каждому медленнодействующему каналу выделяется гармоническая несущая определенной частоты. Для надежной работы необходимо достаточное различие между частотами несущих, что ограничивает число мультиплексируемых каналов. Например, мультиплексор, работающий на линию со скоростью 2400 бит/с, может объединить 12 потоков данных, имеющих скорость 110 бит/с.

1

Работа мультиплексора с временным разделением основана на выделении каждому медленнодействующему каналу фиксированных временных интервалов. В составе устройства имеется несколько регистров, по одному на каждый медленнодействующий канал. Символы от источника принимаются в соответствующий регистр, а мультиплексор последовательно сканирует регистры и с большой скоростью направляет содержащиеся в них символы в модем. Если опрашиваемый регистр не содержит кода символа, в соответствующем временном интервале передается пустой символ. Демльтиплексор разделяет принимаемый поток с учетом порядка сканирования.

Техника средств связи постоянно совершенствуется и усложняется, что особенно стимулируется созданием сетей ЭВМ. На смену простым мультиплексорам идут статистические мультиплексоры, сложные концентраторы данных и связные процессоры на базе мини-ЭВМ.

6-5. СИМВОЛЬНЫЕ И ГРАФИЧЕСКИЕ ДИСПЛЕИ

Современные периферийные устройства с электронно-лучевыми трубками (ЭЛТ), получившие общее название дисплеев, предназначены в основном для вывода информации в удобной для пользователя форме. При оснащении их другими устройствами типа клавиатуры, светового пера и другими пользователь может осуществлять и ввод информации, поэтому большинство дисплеев относится к интерактивным устройствам ввода-вывода. Дисплей с клавиатурой называется также видеомонитором и видеодисплеем, а комплекс устройств на базе дисплея, включающий клавиатуру, принтер, перфоленточное оборудование, память и средства связи с ЭВМ, называется видеотерминалом или интеллектуальным терминалом. В состав таких терминалов входит микропроцессор или микро-ЭВМ и предусматривается довольно обширный набор программных средств.

На экран символьного (алфавитно-цифрового) дисплея выводятся только буквенно-цифровые тексты, а графический дисплей позволяет выводить информацию в графической форме (чертежи, схемы, диаграммы и т. п.). Графические дисплеи намного сложнее символьных и рассчитаны на подключение к довольно мощной ЭВМ.

Символьный дисплей выполняет следующие функции: вывод (индикация) содержимого текстовых файлов, например, исходных программ на языке ассемблера, интерактивное взаимодействие с пользователем посредством тек-

стовых сообщений (например, в обучающих системах) и редактирование текстов с клавиатуры. По существу, эти функции дисплея напоминают функции быстродействующего принтера, не имеющего движущихся частей и не требующего бумаги.

В прошлом было разработано много способов формирования изображений символов на экране ЭЛТ (функциональное управление лучом, кусочно-линейная аппроксимация контуров символов, локальный микрорастр и др.). Сейчас в символьных дисплеях наиболее распространено растровое сканирование с формированием точечных изображений символов. С помощью двух ортогональных управляющих воздействий, подаваемых в отклоняющую систему ЭЛТ, электронный луч перемещается на экране по регулярной траектории, состоящей из отдельных строк (или линий) раstra. Обычно луч движется из верхнего левого угла экрана слева направо и сверху вниз, как показано на рис. 6-7, а. Управляющими воздействиями являются напряжения U_x , U_y в электростатических отклоняющих системах или токи I_x , I_y в электромагнитных системах. Сами управляющие воздействия могут быть непрерывными или дискретными. В первом случае эти воздействия формируются генераторами разверток, а во втором — с помощью двух счетчиков и цифро-аналоговых преобразователей.

Когда луч достигает конца строки раstra, он быстро возвращается в начало следующей строки. Во время строчного обратного хода, определяемого сигналом HSYNC горизонтальной, или строчной, синхронизации, луч выключается

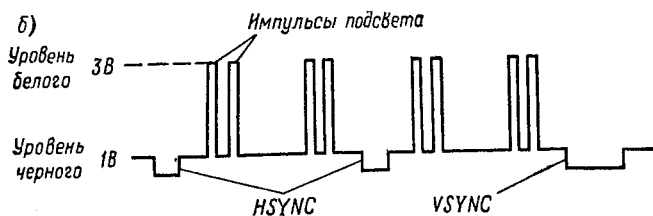
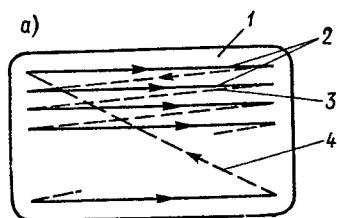


Рис. 6-7. Формирование раstra на экране ЭЛТ (а), типичный видеосигнал (б)

1 — экран; 2 — строки раstra; 3 — строчный обратный ход; 4 — кадровый обратный ход

и на экране обратный ход невидим. По прохождении последней строки луч смещается в начало самой первой строки. Это движение, во время которого луч также выключается, называется кадровым обратным ходом и вызывается сигналом VSYNC вертикальной, или кадровой, синхронизации.

Для образования на экране четких, немерцающих изображений построение раstra должно осуществляться периодически с частотой 40—60 Гц. Во многих дисплеях эта частота регенерации установлена равной 50 Гц, т. е. частоте напряжения сети.

Управление символьной информацией производится по цепи модулятора, на котором напряжение видеосигнала определяет интенсивность электронного луча и яркость свечения экрана. Типичный вид составного видеосигнала, в котором объединены сигналы данных и сигналы горизонтальной и вертикальной синхронизации, приведен на рис. 6-7, б. Такой сигнал можно передавать по одному коаксиальному кабелю на большие расстояния.

Рабочее поле экрана, т. е. его центральная часть, где нелинейные искажения минимальны, разбивается на большое число точек или элементов изображения. Путем подсвета требуемых комбинаций элементов на экране формируется любое изображение. В простых черно-белых изображениях каждому элементу экрана соответствует один бит в оперативной памяти: если он содержит единицу, элемент подсвечивается (белый), а если нуль — элемент остается неподсвеченным (черный).

Точечные изображения символов формируются на матрице, имеющей 5×7 , 7×9 или 10×14 элементов (рис. 6-8). Число символов в строке текста, занимающей соответственно 7, 9 или 14 строк раstra, обычно составляет 80, что отвечает емкости одной перфокарты, но иногда доходит до

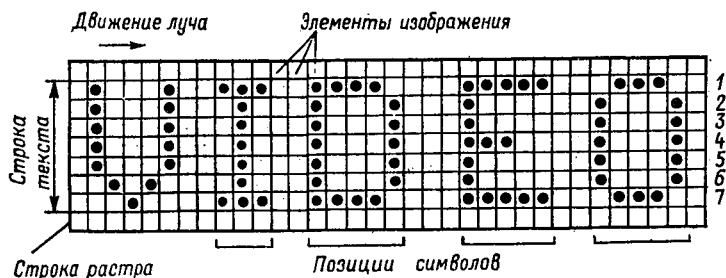


Рис. 6-8. Точечные изображения сигналов

132 символов. Число строк текста варьируется от 12 до 40 (типичное число 16—24), поэтому общее число выводимых на экран символов, называемое информационной емкостью экрана, оказывается равным 960—2560. Выводимый на экран текст называется кадром, или страницей. Для регенерации в дисплее предусматривается оперативная память соответствующей емкости, называемая видео-ЗУПВ. В некоторых дисплеях видео-ЗУПВ организуется в основной памяти ЭВМ, а регенерация осуществляется по методу прямого доступа к памяти. При этом в дисплее потребуются буферная память с минимальной емкостью в одну строку текста.

При движении электронного луча по первой строке раstra последовательно формируются верхние ряды матрицы 5×7 каждого символа, находящегося в строке текста. Когда луч перемещается по второй строке раstra, аналогичным образом формируются следующие ряды матриц тех же символов. Такие же действия повторяются для всей строки текста. После промежутка, составляющего 5—15 строк раstra, производится построение следующей строки текста.

Семи- или восьмибитные коды символов индицируемой страницы хранятся в видео-ЗУПВ дисплея. Чтобы обеспечить формирование видеосигнала, в дисплее предусматривается знакогенератор (генератор символов). Он представляет собой полупроводниковое ПЗУ, на адресные входы которого подаются код символа и код текущей строки раstra. По этому адресу из ПЗУ считывается соответствующий 5-битный код, например для третьей строки раstra буквы V считывается код 10001, буквы I — код 00100 и т. д. Параллельные 5-битные коды загружаются в регистр сдвига и преобразуются в последовательную форму, биты которой объединяются с сигналами синхронизации и образуют видеосигнал.

На рис. 6-9 приведена структурная схема простого контроллера символьного дисплея. Генератор синхронизации вырабатывает непрерывные импульсные сигналы, период которых соответствует отдельным элементам изображения на каждой строке раstra. Они используются для управления сдвигом и подаются в счетчик элементов, коэффициент пересчета которого определяется числом столбцов в матрице изображений символов плюс число элементов, отводимых на промежутки между символами. Выходной сигнал счетчика элементов подается в счетчик символьных позиций, кодом которого определяется номер индицируемого

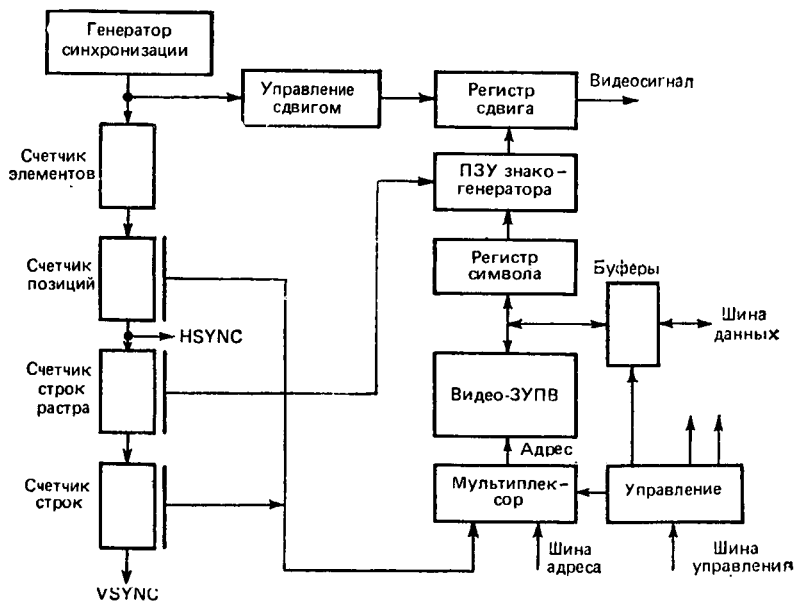


Рис. 6-9. Простой контроллер символьного дисплея

символа в текущей строке текста. Коэффициент пересчета этого счетчика устанавливается на 20—25 % превышающим число символов в строке текста, чтобы отвести достаточное время на строчный обратный ход. Содержимое счетчика позиций используется для адресации видео-ЗУПВ.

Выход счетчика символьных позиций подключен на вход счетчика строк раstra, содержимое которого определяет номер строки раstra в текущей строке текста, а коэффициент пересчета равен сумме числа рядов в матрице символов и числа строк раstra, приходящихся на промежутков между строками текста. Код из этого счетчика управляет входом адреса ПЗУ знакогенератора, определяя текущий ряд матрицы символов.

Последний счетчик контроллера — счетчик строк текста — фиксирует номер текущей строки текста и используется для адресации видео-ЗУПВ. Его коэффициент пересчета на 20—25 % больше действительного числа строк текста на экране дисплея, чтобы обеспечить временной интервал для кадрового обратного хода. Таким образом, общий адрес видео-ЗУПВ определяется кодами в счетчиках символьных позиций и строк текста, которые однозначно

идентифицируют слово видео-ЗУПВ, считываемое в регистр символа.

Графический дисплей. Во многих применениях ЭВМ вывод данных в символьной форме оказывается малоэффективным и неудобным. Примерами могут служить машинное проектирование топологии БИС и разводки печатных плат, анализ сетевых графиков, автоматизированное конструирование и др. В то же время для большинства невычислительных применений ЭВМ весьма удобно графическое представление выводимых данных на экране ЭЛТ или на бумаге (в графопостроителях). Графический дисплей намного сложнее символьного дисплея, тем более, что графические изображения сопровождаются символьными данными в виде поясняющих надписей, размеров и т. п.

Качество изображений на экране зависит от разрешающей способности дисплея, т. е. от числа адресуемых элементов изображения (точек) на экране. Чем выше разрешающая способность, тем более мелкие детали может идентифицировать дисплей, точнее передаются геометрические соотношения и наблюдается более плавная передача цветовых оттенков или изменений яркости. В графических дисплеях со средней разрешающей способностью число элементов изображения составляет 512×512 , а в высококачественных дисплеях 1024×1024 и более.

В графических дисплеях применяются главным образом три способа формирования изображений: графические элементы, векторное сканирование и растровое сканирование (битовое отображение).

Первый способ, по существу, напоминает формирование изображений символов и используется в дисплеях со сравнительно малой разрешающей способностью. Он заключается в построении сложного изображения из простых графических элементов (отрезки горизонтальных, вертикальных и наклонных прямых, а также сегменты различных кривых на точечной матрице, например 8×10). Графические элементы имеют однозначное кодирование и хранятся в постоянном запоминающем устройстве, называемом графическим генератором. Последовательность кодов графических элементов в буферной памяти дисплея определяет формируемое на экране изображение. На рис. 6-10 показан принцип формирования изображений с малой матрицей 3×3 . В графическом генераторе может быть предусмотрено несколько наборов графических элементов, например 6 наборов по 190 элементов. Следует подчеркнуть, что 1140

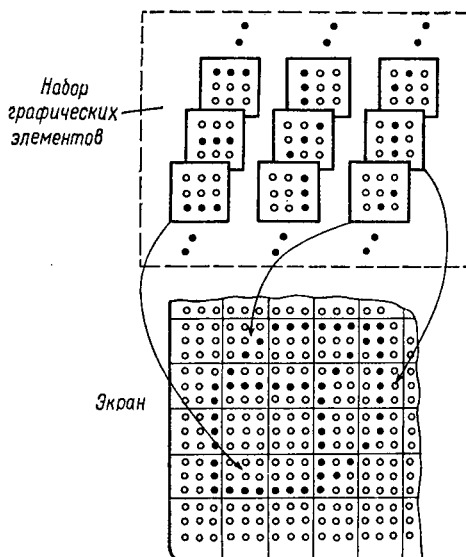


Рис. 6-10. Формирование изображения из графических элементов

элементов составляют ничтожную часть от 2^{80} возможных комбинаций на матрице 8×10 .

В другой разновидности дисплея матрица 8×9 разделена на шесть адресуемых графических элементов 4×3 , каждый из которых может быть включен или выключен. На рис. 6-11 показано, что управление состоянием элементов осуществляется входным 6-битным кодом, однозначно связанным с нумерацией элементов. Старший бит b_7 содержит нуль, следующий бит b_6 определяет графическое изображение ($b_6=0$) или символ ($b_6=1$), биты b_5-b_0 представляют либо состояние элементов (1 — включен, 0 — вы-

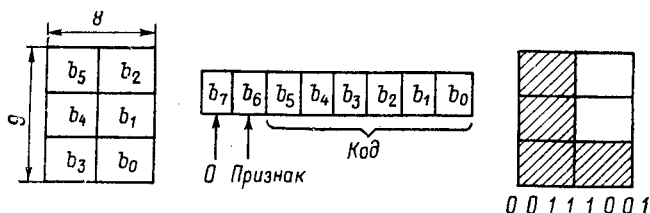


Рис. 6-11. Графические элементы на матрице 8×9

ключен), либо код символа. Последовательность входных кодов определяет изображение, формируемое на экране дисплея. Дополнительные управляющие коды задают цвет, мерцание и другие атрибуты изображения.

Дисплей с векторным (произвольным) сканированием имеет специализированный дисплейный процессор, выполняющий разнообразные функции формирования изображений, интерфейса с ЭВМ и взаимодействия с пользователем. Если в растровом дисплее луч в продолжение кадра последовательно проходит все элементы экрана, то при векторном сканировании луч движется только по отрезкам прямых или кривых (дуги окружностей или эллипсов), образующим собственно выводимое изображение. Одним из основных узлов дисплейного процессора является вектограф — устройство для формирования на экране отрезков прямых.

Экраны, чувствительные к прикосновениям. Стремление упростить взаимодействие пользователя с дисплеем привело к разработке экранов, чувствительных к прикосновениям. Имеется несколько конструкций таких экранов.

Акустическое перо при касании экрана формирует ультразвуковой сигнал. Микрофоны, расположенные по периферии экрана, воспринимают ультразвук, который распространяется в воздухе со скоростью 330 м/с. Время, необходимое для достижения микрофонов, пропорционально расстоянию пера до каждого из них. Место касания пера определяется посредством вычисления координат источника ультразвука. Так как положение индицируемых данных в эквивалентных координатах также известно, программа идентифицирует символьный или графический элемент, выбираемый акустическим пером.

В дисплеях, опирающихся на свойство отражения ультразвука, пьезоэлектрические преобразователи на краях экрана генерируют ультразвуковые колебания. Отраженные от зонда сигналы улавливаются микрофонами и преобразуются в координаты зонда.

Наконец, емкостные датчики положения представляют собой матрицу оптически прозрачных емкостных площадок, каждая из которых образует электрически изолированную поверхность. Контакт пальца с площадкой изменяет ее емкость. Сигнал об этом передается в контроллер по тонким проводникам, встроенным в специальное покрытие экрана. Информация о месте касания закодирована в изменении емкости одной или нескольких площадок.

В заключение относительно дисплеев отметим следую-

щее. Один из недостатков дисплея заключается в наличии самой достаточно сложной и громоздкой ЭЛТ, требующей для работы высоких напряжений и мощных отклоняющих сигналов. Периодически в литературе появляются прогнозы о замене ЭЛТ более удобными и компактными газоразрядными или жидкокристаллическими индикаторами. Однако многие специалисты придерживаются того мнения, что до конца XX века дисплей будет наиболее распространенным интерактивным периферийным устройством.

6.6. НАКОПИТЕЛИ НА МАГНИТНЫХ ДИСКАХ

Накопители на магнитных дисках (НМД) широко применяются во всех современных ЭВМ в качестве массовой памяти со сравнительно небольшим временем обращения, которое составляет в зависимости от типа накопителя несколько десятков или сотен миллисекунд. Такое время обращения позволяет называть НМД памятью с прямым, или непосредственным, доступом.

Данные хранятся в последовательной форме на концентрических дорожках, число которых варьируется от 40 до 815. Дорожки

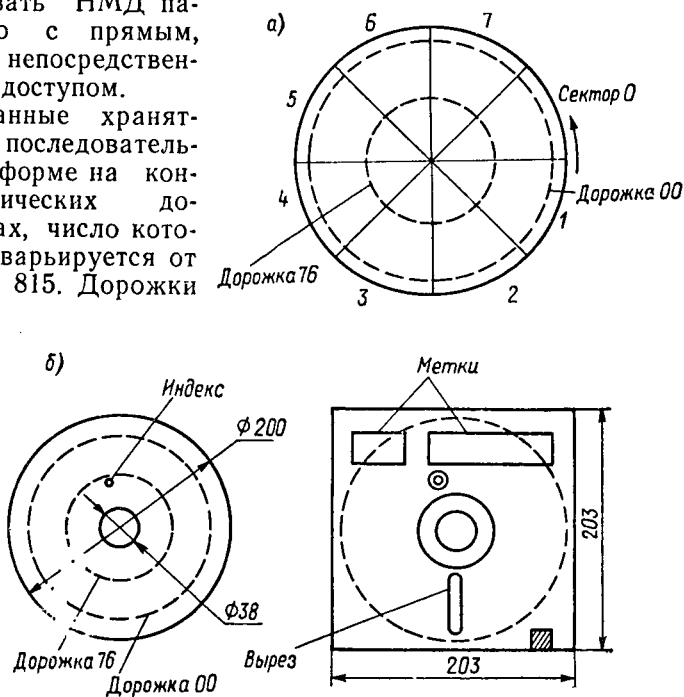


Рис. 6-12. Формат магнитного диска (а), гибкий магнитный диск (б)

нумеруются от периферии к центру диска. Диск разделен на отдельные секторы числом от 8 до 50 (рис. 6-12, а). Таким образом, любой блок данных, определяемый емкостью сектора 128, 256, 512 или 1024 байта, идентифицируется номером (адресом) дорожки и номером (адресом) сектора.

Имеются две основные разновидности магнитных дисков: металлические диски из алюминия с ферролаковым или никель-кобальтовым покрытием и гибкие диски из синтетического материала с ферролаковым покрытием. Накопители с металлическими дисками имеют значительно более высокие технические характеристики по сравнению с накопителями на гибких дисках, но оказываются более дорогими и более критичными к условиям эксплуатации. Поэтому накопители на металлических дисках ориентированы на большие и средние ЭВМ, а накопители на гибких дисках применяются в мини- и микро-ЭВМ.

В больших ЭВМ наибольшее распространение получили накопители со сменными пакетами дисков, емкость которых составляет 7,25М, 29М и 100М байт. В новейших накопителях емкость пакета из 12 дисков доведена до 300М байт. Бесконтактная запись (считывание) на каждую рабочую поверхность осуществляются индивидуальными подвижными магнитными головками. В мини-ЭВМ старших моделей применяются накопители комбинированного типа с несъемным (фиксированным) диском и съемной однодисковой кассетой (так называемый картридж-диск). Несъемные диски иногда оснащаются неподвижными (фиксированными) магнитными головками, по одной для каждой дорожки. Такая конструкция уменьшает время обращения, поскольку отпадает необходимость в механическом перемещении головки на нужную дорожку.

В последнее время за рубежом появились так называемые винчестерские диски, считающиеся третьим поколением НМД и имеющие близкие к предельным характеристики. Типичный винчестерский накопитель имеет герметичный несъемный пакет из пяти дисков с общей емкостью 70М байт, плотностью записи 236 бит/мм и скоростью передачи данных 1 Мбит/с.

Ниже более подробно рассмотрены накопители на гибких магнитных дисках, которые в течение некоторого времени будут основными НМД в мини- и микро-ЭВМ.

Общая характеристика накопителя. Появлению и распространению накопителей на гибких дисках, называемых также флоппи-дисками или дискетами, способствовали та-

кие их положительные качества, как небольшие габариты и масса, дешевизна, экономичность, энергонезависимость и достаточно высокая надежность. Кроме того, емкость накопителей постоянно возрастает благодаря увеличению плотности записи, плотности дорожек, использованию в качестве рабочих обеих сторон диска и компоновке в одном корпусе двух приводов. Дальнейший материал в основном относится к стандартному одностороннему диску с одинарной плотностью записи и программным разделением на секторы.

Гибкий диск представляет собой майларовый диск толщиной 0,125 мм и диаметром 200 мм с отверстием в центре (рис. 6-12, б). Для синхронизации в диске имеется еще одно небольшое отверстие, обычно называемое индексным отверстием или просто индексом. С одной стороны диск покрыт износоустойчивым ферролаком. Информация размещается на 77 дорожках, представляющих собой концентрические окружности. Обычно внешняя дорожка 00 содержит идентифицирующую информацию для всего диска, две дорожки являются запасными, а 74 дорожки используются как рабочие. Запись и считывание данных осуществляются одной универсальной магнитной головкой.

Диск постоянно находится в защитном чехле (конверте), внутренняя поверхность которого обеспечивает поглощение инородных частиц. В чехле имеется прорезь для подвижной магнитной головки и индексное отверстие. При восприятии индексного отверстия фотооптический датчик формирует импульс, сигнализирующий о начале дорожки. На чехле предусматривается светоотражающая этикетка или вырез для защиты от записи.

Когда диск помещается в приемник накопителя, при закрывании дверцы пружинный захват прижимает диск к поверхности шпинделя, вращающегося от электродвигателя. Частота вращения диска внутри чехла составляет 360 об/мин с допуском $\pm 2,5\%$, поэтому номинальное время одного оборота равно 166,7 мс.

Разработано несколько конструкций механической части накопителя. Наиболее распространена конструкция с одной подвижной магнитной головкой и контактным способом записи и считывания. Головка находится на подвижной платформе, которая при позиционировании головки на нужную дорожку перемещается шаговым двигателем. Время на перемещение головки с одной дорожки на другую варьируется от 5 до 10 мс. Нормально магнитная головка отведена от поверхности диска и только во время операций записи и

считывания с небольшим усилием контактирует с диском. Время приведения головки в рабочее положение изменяется от 5 до 20 мс. Рабочий контакт головки и носителя неизбежно вызывает их износ, зависящий от геометрической формы и абразивных свойств головки. Обычно гарантируется не менее одного миллиона проходов головки по одной дорожке.

Способы записи. В накопителях на гибких дисках применяется несколько способов записи (или кодирования) данных. Отличительной чертой их является запись вместе с сигналами данных сигналов синхронизации. Объясняется это сравнительно широкими допусками на скорость вращения диска, что приводит к изменениям моментов появления («плаванию») считываемых сигналов. Последовательный поток данных преобразуется в импульсы тока для магнитной головки и разделяется на битовые элементы, каждый из которых кодирует двоичный ноль или единицу. В накопителях с одинарной плотностью записи длительность импульса битового элемента составляет 4 мкс, а с двойной плотностью записи — 2 мкс. Это соответствует скорости обмена данными 31,25К и 62,5К байт/с соответственно.

Различие способов записи заключается в формировании сигналов собственно данных D и синхронизации C в битовых элементах. В большинстве способов записи для гибких дисков бит данных D кодируется одинаково: единице соответствует изменение (переход) магнитного потока в середине элемента, а нулю — отсутствие такого перехода.

В накопителях с одинарной плотностью применяется запись с частотной модуляцией, называемая также ЧМ-кодированием или записью с удвоением частоты. Этот способ характеризуется записью бита синхронизации как изменения потока в начале импульса каждого битового элемента. При считывании переходы потока от битов синхронизации и двоичных единиц данных вызывают появление на выходе усилителя считывания импульсных сигналов. На рис. 6-13 показаны разделенные сигналы C и D. Нетрудно заметить, что при считывании одних нулей формируется поток импульсов с частотой f , а при считывании одних единиц — поток с частотой $2f$, что объясняет название способа.

В ЧМ-кодировании более половины переходов магнитного потока вызываются битами синхронизации и для увеличения плотности записи необходимо применять более эффективные способы кодирования. Биты C предназначены для синхронизации в схеме восстановления данных, поэтому можно увеличить плотность записи, если удалить из за-

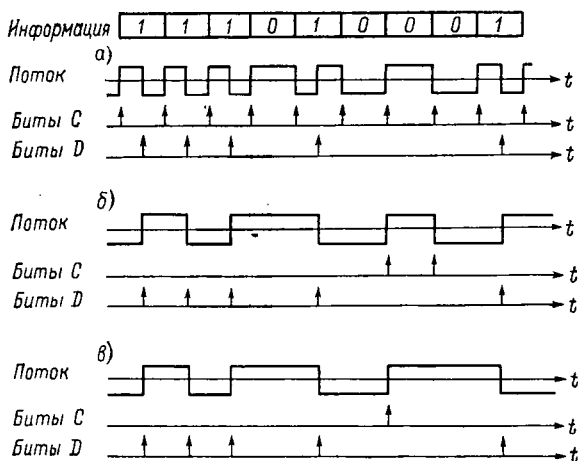


Рис. 6-13. Способы записи информации на гибкий диод; а — ЧМ-кодирование; б — МЧМ-кодирование; в — М²ЧМ-кодирование

писываемого потока часть переходов, вызываемых сигналами синхронизации. Другими словами, при сохранении плотности переходов потока можно уменьшить длительность импульса битовых элементов. В модифицированном ЧМ-кодировании (МЧМ-кодировании) запись битов С, т. е. изменения потока в начале импульса битовых элементов, производится только в том случае, если текущий битовый элемент и предыдущий битовый элемент содержат нули. Модифицированное МЧМ-кодирование (дважды модифицированное, или М²ЧМ-кодирование) отличается тем, что запись бита С производится только тогда, когда текущий битовый элемент содержит нуль, а в предыдущем битовом элементе не было изменений потока ни от данных, ни от синхронизации.

Из диаграмм на рис. 6-13, б и в видно, что в модифицированных способах записи, сохранив плотность переходов потока обычного ЧМ-кодирования, можно в два раза увеличить плотность записи, если уменьшить длительность битовых элементов с 4 до 2 мкс. Однако сокращение числа сигналов синхронизации приводит к усложнению схем записи и восстановления данных. Наиболее распространенный способ разделения считываемых сигналов заключается в использовании генератора с фазовой подстройкой частоты. После вхождения в синхронизм генератор формирует

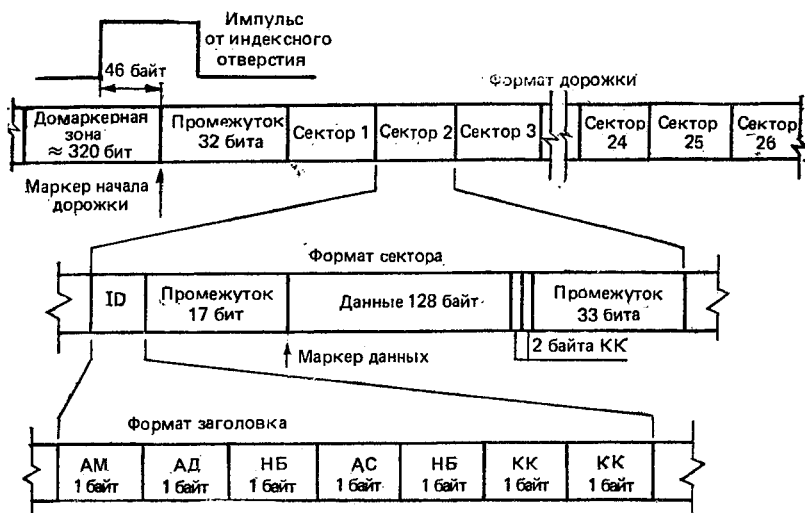


Рис. 6-14. Формат гибкого диска

стролирующие сигналы («окна») для разделения сигналов синхронизации и данных. Генератор устойчиво работает при медленных изменениях скорости вращения диска.

Формат гибкого диска. В гибких дисках с программным разделением на секторы применяются довольно сложные форматы, приведенные на рис. 6-14. Физическое начало всех дорожек отмечается импульсом, формируемым фотодатчиком индексного отверстия, и специальным байтом маркера начала дорожки. Этот маркер, как и другие рассматриваемые ниже маркеры, представляет собой специальный байт, в котором содержится особый набор сигналов данных и синхронизации. Например, маркер начала дорожки имеет код данных FC и код синхронизации D7. Маркеры выделяются схемой контроллера и используются для управления и синхронизации.

После маркера начала дорожки идет послеиндексный промежуток в 32 байта, а затем следует 26 секторов с одинаковым форматом. Заканчивается дорожка предындexным промежутком в 320 байт, называемым также домаркерной зоной. Промежутки предназначены для образования временных интервалов, необходимых для выполнения требуемой операции. Они же в некоторой степени компенсируют изменение скорости вращения и деформацию диска. Соответствующий интервал равен $32N$ мкс, где N — число

байт в промежутке. Обычно промежуток содержит нулевые наборы данных (нулевые байты — НБ).

Сектор начинается с поля идентификации ID, называемого также заголовком и состоящего из семи байт. Первый байт представляет собой адресный маркер АМ, второй байт содержит двоичный адрес дорожки АД, четвертый — двоичный адрес сектора АС, а в третьем и пятом байтах в качестве кодов данных записаны нули (НБ). Заканчивается поле заголовка двумя байтами контрольного кода (КК), называемыми также символами циклического избыточного контроля. Они содержат остаток от деления кода предыдущих пяти байт на специальный 16-битный код, называемый порождающим полиномом. В операции записи контроллер автоматически вычисляет байты КК и помещает их на дорожку. При выполнении операции считывания контроллер вновь вычисляет байты КК и сравнивает их с записанными на дорожке. Если они не совпадают, фиксируется ошибка и автоматически инициируется повторное считывание той же самой информации. Такие попытки производятся до десяти раз, и только после этого генерируется сигнал о неправильной работе.

За полем заголовка следует промежуток длиной 17 байт, а затем — поле собственно данных сектора. Оно начинается с байта маркера данных, который содержит код данных FВ и код синхронизации С7, но маркер данных может быть установлен пользователем. Следующие 128 байт сектора являются рабочими, т. е. доступны пользователю. Завершается поле данных двумя байтами КК. В конце сектора находится промежуток 33 байта. Таким образом, всего сектор содержит 188 байт, из которых 128 байт отведены для данных пользователя, а 60 — для идентификации, контроля и формирования служебных временных интервалов.

Содержимое одного сектора называется записью, длина ее составляет 128 байт. Набор (файл) данных состоит из любого числа последовательных записей, вплоть до емкости всего диска. Дорожка 00 обычно используется как оглавление (таблица) текущего содержимого диска. Секторы 01—06 этой дорожки являются резервными, сектор 07 идентифицирует весь диск, а в каждом из секторов 08—26 описывается один набор данных. В них содержится 8-символьное имя набора, введенное пользователем и рабочая емкость каждого из входящих в набор секторов и экстенгов каждого набора. Экстент набора представляет собой адреса первого и последнего секторов набора вместе с адресами дорожек, на которых они размещены.

Программное разделение на секторы приводит к малоэффективному использованию поверхности гибкого диска. Из общего числа 385392 байта доступными для пользователя оказываются только 246272 байта, и коэффициент использования поверхности составляет всего 65 %.

В некоторых зарубежных накопителях применяется формат с аппаратным разделением на секторы. На гибком диске имеются 32 равномерно расположенных отверстия, отмечающие начало секторов, а одно индексное отверстие идентифицирует начало дорожек. В данном формате на служебную информацию отводится гораздо меньше байт, чем в предыдущем формате, что улучшает использование поверхности диска: из общего числа 369408 байт рабочими являются 312576 байт или 85 %. Кроме того, здесь несколько упрощаются схема контроллера и схема управления в самом накопителе. Такие накопители рассчитаны на более жесткие условия эксплуатации в системах производственного управления. Обычно гибкие диски поставляются размеченными, с записанной идентифицирующей информацией.

Приводим основные технические характеристики накопителя на гибком диске «Электроника ГМД-70»:

Число приводов	2
Число рабочих поверхностей на диске	1
Число дорожек	77
Емкость, байт	512K
Число секторов на дорожке	26
Емкость сектора, байт	128
Среднее время обращения, мс	380
Скорость обмена данными, кбайт/с	50
Потребляемая энергия, В·А	460

6-7. НАКОПИТЕЛИ НА МАГНИТНОЙ ЛЕНТЕ

Накопители на магнитной ленте (НМЛ) представляют собой вторую распространенную разновидность массовой памяти с движущейся магнитной поверхностью. Основными достоинствами их являются небольшие габариты и масса, большая емкость, малая удельная стоимость, а также характерная для всех магнитных запоминающих устройств энергонезависимость. Однако НМЛ относятся к устройствам последовательного действия и среднее время обращения у них значительно больше, чем у накопителей на магнитных дисках.

Магнитная лента представляет собой синтетическую основу, покрытую тонким слоем ферролака. Ширина ленты изменяется от 3,8 до 25,4 мм, но наибольшее распростране-

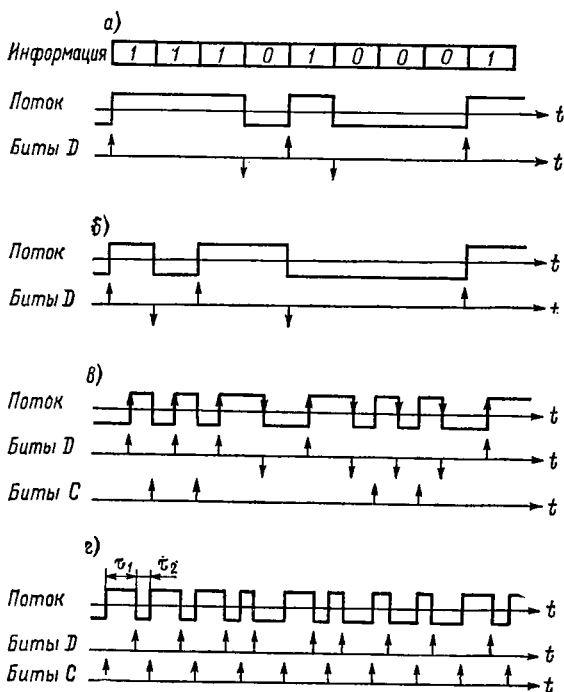


Рис. 6-15. Способы записи на магнитную ленту: а — без возврата к нулю; б — то же с инверсией; в — с фазовой модуляцией; г — с временной модуляцией

ние в вычислительной технике получила стандартная лента шириной 12,7 мм. Контактные запись и считывание осуществляются в процессе движения ленты относительно блока магнитных головок, выполняющих операции параллельно на нескольких дорожках. Число дорожек стандартной магнитной ленты равно 9.

Способы записи. На рис. 6-15 показаны некоторые современные способы записи двоичной информации на магнитную ленту. Во всех способах, как и в накопителях на дисках, запись осуществляется полным током, т. е. в течение всей операции записи в головке действует ток, достаточный для насыщения магнитного материала в том или ином направлении. В этом случае исходное состояние носителя не играет роли. Ток записи варьируется от 10 до 200 мА, а амплитуда считываемых сигналов — от 100 мкВ до 100 мВ.

Самый простой способ записи — без возврата к нулю

(NRZ), или, точнее, без возврата к нейтральному состоянию носителя. Здесь при записи единицы носитель насыщается с одним знаком, а при записи нуля — с противоположным. Нетрудно заметить, что полярность считываемого сигнала определяет начало последовательности одинаковых бит данных, которые должны восстанавливаться с помощью сигналов синхронизации даже при отсутствии последующих считанных импульсов.

Разновидностью способа NRZ является запись без возврата к нулю с инверсией (NRZI). Здесь в начале импульса битового элемента направление тока записи изменяется на противоположное, если записывается двоичная единица, и не изменяется при записи двоичного нуля. Считываемый импульс любой полярности фиксирует двоичную единицу.

Отметим, что в обоих способах длинные цепочки нулей (а в способе NRZ — и цепочки единиц) не вызывают формирования считываемых сигналов, что может привести к потере синхронизации. Кроме того, усилитель считывания должен иметь широкую полосу пропускания, что уменьшает отношение сигнал-шум.

Способ фазовой модуляции относится к самосинхронизирующемуся кодированию, так как в записываемом потоке скрыты сигналы синхронизации. Запись единицы производится изменением тока в середине импульса битового элемента в одном направлении, а запись нуля — в другом. Два соседних одинаковых бита требуют еще одного изменения тока в начале импульса битового элемента, которое при считывании приводит к формированию сигнала синхронизации. Данный способ записи применяется во многих современных НМЛ.

В некоторых простых НМЛ используется способ записи с временной модуляцией, в котором двоичная единица записывается импульсом тока с $\tau_1/\tau_2 > 1$, а ноль — импульсом с $\tau_1/\tau_2 < 1$, где $\tau_1 + \tau_2$ есть длительность импульса битового элемента.

При записи информации на двух дорожках применяется способ дополнения без возврата к нулю. Здесь единица записывается одним состоянием носителя на одной дорожке и противоположным состоянием на другой. Двоичный ноль кодируется инверсной комбинацией. Две оставшиеся комбинации резервируются для синхронизации.

В новых НМЛ используется запись с групповым кодированием.

НМЛ с двумя катушками. НМЛ больших и средних ЭВМ, а также мини-ЭВМ старших моделей обычно выпол-

няются по схеме с двумя съемными катушками диаметром 150—267 мм, которые приводятся во вращение двумя автономными двигателями. Ширина ленты составляет 12,7 мм, длина ее около 730 м, плотность записи варьируется от 8 до 246 бит/мм, а емкость катушки зависит от плотности записи и доведена до 180 Мбайт. Нормально лента неподвижна, а при выполнении операций записи и считывания транспортируется относительно магнитных головок с постоянной скоростью, от 320 до 5000 мм/с. Данные записываются параллельно блоком из 7 или 9 головок. В 9-дорожечной ленте строка содержит один байт (символ) и контрольный бит. Скорость передачи данных определяется плотностью записи и скоростью движения ленты и изменяется от 2500 байт/с до 1,25 Мбайт/с.

При выполнении операций записи и считывания лентопротяжный механизм должен быстро разгонять ленту до номинальной скорости и поддерживать постоянное натяжение ленты. Время разгона и останова зависит от конструкции лентопротяжного механизма и составляет от 1 до 30 мс. Для уменьшения ударных нагрузок при пуске и останове создаются буферные хранилища ленты с помощью вакуумных колонок в быстродействующих НМЛ или натяжных рычагов в медленнодействующих и компактных накопителях. Фотооптические датчики фиксируют запас ленты в хранилище, а система управления формирует управляющие воздействия для приводов катушек. Транспорт ленты с постоянной скоростью обеспечивается комбинацией ведущего и прижимного роликов.

Байты, записанные на магнитной ленте, группируются в блоки данных, содержащие 18—2048 байт. Блоки разделяются промежутками, размер которых составляет 7,6—19 мм. В начале и в конце блоков содержится идентифицирующая и управляющая информация, а также контрольные коды. Сочетание поперечного (вертикального) и продольного контроля позволяет обнаруживать и иногда исправлять ошибки. Блоки данных объединяются в записи, а группы записей образуют файлы, идентифицируемые и адресуемые в программах. Файлы разделяются специальными маркерами конца файла EOF.

Около физических концов ленты имеются светоотражающие полосы из фольги или небольшие отверстия, позволяющие накопителю воспринимать сигналы начала и конца ленты. От каждого маркера до конца остается около 3 м пустой ленты, которая служит для заправки ленты на катушку. В конструкции катушки предусмотрено съемное

кольцо защиты от записи: если оно удалено, записать на ленту какие-либо данные невозможно.

Собственно НМЛ состоит из лентопротяжного механизма, блока магнитных головок, электронного контроллера-форматтера, схем интерфейса с ЭВМ и блока питания. Накопители часто объединяются в группы (до 8 штук) с общим групповым контроллером. С пульта на каждом накопителе осуществляется автономное управление, обычно обеспечивающее следующие функции: выбор режима работы (автономный или комплексный), транспорт вперед и назад, перемотку (с повышенной скоростью), задание физического адреса накопителя и др.

Рассмотренные НМЛ, имея большую емкость и высокое быстродействие, обладают значительными габаритами и массой, оказываются дорогими и обычно рассчитаны на эксплуатацию в вычислительных центрах. Для уменьшения габаритов накопителя разработана конструкция привода «катушка-катушка», в которой отсутствуют ведущие ролики, а транспорт ленты обеспечивается вращением катушек. В этой конструкции требуется следящая система, поддерживающая скорость движения ленты относительно головок примерно постоянной независимо от степени заполнения катушек (допустимое отклонение скорости составляет $\pm 5\%$).

Кассетные накопители на магнитной ленте. Дешевые и компактные кассетные НМЛ имеют по сравнению с рассмотренными выше НМЛ значительно меньшую емкость и быстродействие. Лента и миниатюрные катушки находятся в пластмассовой кассете, защищающей ленту от пыли и отпечатков пальцев. В первых устройствах применялись кассеты бытовых магнитофонов, но затем появились специальные цифровые кассеты с более долговечной лентой, хотя габариты кассеты не изменились. Типичные характеристики кассетного НМЛ таковы:

Длина ленты, м	46	Скорость движения, мм/с	244
Ширина, мм	3,8	Время разгона и останова, мс	20
Толщина, мкм	30	Емкость, байт	92К
Плотность записи с фазовой модуляцией, бит/мм	14—28	Скорость обмена данными, байт/с	560

В более совершенных устройствах емкость кассеты доведена до 288К байт.

Иногда в качестве дешевых НМЛ используются бытовые кассетные магнитофоны, которые подключаются к сис-

темной шине микро-ЭВМ с помощью схем параллельного или последовательного интерфейса. При этом возникают две основные проблемы: тракты записи-считывания и магнитные головки рассчитаны на частоты, ограниченные звуковым диапазоном; нестабильность скорости движения ленты может привести к потере синхронизации, если в способе записи не предусмотрены специальные меры. В одном из разработанных способов записи биты данных представляются различными тональными частотами: 3700 Гц соответствуют двоичному нулю и 2400 Гц — двоичной единице. Следовательно, в интерфейсную плату необходимо вводить модулятор для записи и демодулятор для считывания. Чтобы идентифицировать начало и конец битовых элементов, их отмечают тональными частотами соответственно 3700 Гц и 2400 Гц. Собственно информационный бит соответствует середине битового элемента. Такое кодирование позволяет компенсировать изменения скорости движения ленты. Плотность записи составляет всего 3 бит/мм, скорость обмена данными 136 бит/с, а емкость кассеты при длине ленты 170 м — около 500К бит.

Отечественный НМЛ МЛ-45 имеет следующие характеристики:

Длина ленты, м . . .	$41 \pm 1,5$	Емкость, Мбит . . .	2,5
Ширина ленты, мм . . .	12,7	Скорость движения, м/с	$1 \pm 0,1$
Плотность записи, бит/мм	32	Скорость обмена, кбит/с	32
Число дорожек	2	Наработка на отказ, ч	20 000

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Привести графическую схему классификации периферийных устройств современных мини- и микро-ЭВМ.
2. Построить структурную схему шифратора клавиатуры.
3. Разработать принципиальную схему преобразователя 4-битного кода тетрады, представляющего собой 16-ричную цифру, в 7-битный код сегментного индикатора.
4. Чем определяется частота коммутации в индикаторе с временным мультиплексированием? На что влияет разрядность индикатора?
5. Каковы достоинства и недостатки посимвольных и построчных принтеров? Ударных и безударных принтеров?
6. Какие функции может выполнять встроенный микропроцессор в посимвольных и построчных принтерах?
7. Дать сравнительную характеристику способов организации кристаллов ЦМД- и ПЗС-памяти.
8. Чем определяется частота квантования аналоговых сигналов?
9. Разработать алгоритм и построить структурную схему устройства обнаружения и исправления одиночных ошибок в блоке символов.

10. Чем определяется информационная емкость символьного дисплея?

11. Определить емкость диска диаметром 130 мм, имеющего 40 дорожек и 10 секторов по 256 байт.

12. Построить диаграммы тока в магнитной головке для всех способов записи на гибкий диск и магнитную ленту, если информационный байт содержит 10000000, 10101010, 11111111. Подсчитать число переходов потока.

ГЛАВА СЕДЬМАЯ

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МИКРО- И МИНИ-ЭВМ

7-1. НАЗНАЧЕНИЕ И КЛАССИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Все средства ЭВМ принято делить на две основные категории. Под аппаратными средствами понимают разнообразные механические, электромеханические, магнитные и электронные элементы, из которых реализована машина. К программным средствам, называемым также программным обеспечением (ПО), относят все программы, ассоциируемые с машиной. Обычно в ПО включают также компоненты, не имеющие прямого отношения к аппаратным средствам: документацию, описания и руководства, языки программирования с соответствующими трансляторами и т. п.

Промежуточная категория, называемая аппаратно-программными средствами, включает в себя «материализованные» программы, т. е. ПЗУ, в которых хранятся программы и управляющие микропрограммы.

В настоящей главе рассматривается программное обеспечение мини- и микро-ЭВМ, включая системные программы, языки программирования и организацию подпрограмм.

Необходимо отчетливо представлять, что подавляющее большинство современных ЭВМ базируется на принципах, сформулированных еще в 40-х годах Дж. Нейманом. Команды программы и данные хранятся в основной (оперативной) памяти и закодированы в двоичной форме. Команды последовательно считываются из памяти и интерпретируются аппаратными средствами ЭВМ. Следовательно, естественный (внутренний) машинный язык есть язык двоичных единиц и нулей. Программа, представленная в терминах этого языка, называется машинной или объектной программой. При всей простоте машинного языка он совершенно не устраивает пользователей, поэтому сейчас

разработано множество разнообразных внешних языков, на которых ведется программирование и общение с ЭВМ. Соответствующие исходные программы приходится переводить (транслировать) на машинный язык. Как бы ни варьировались и ни усложнялись внешние языки, действиями аппаратных средств управляют только машинные команды.

Современные мини- и микро-ЭВМ представляют собой довольно сложное сочетание аппаратных и программных средств, но большинство пользователей их не знает и даже (как это ни парадоксально) не заинтересовано в знании всех тонкостей (микроархитектуры) машины. Главное для них — использовать машину как инструмент для решения конкретных прикладных проблем. Это достигается посредством разработки и отладки прикладных (проблемных) программ. Ускорить и упростить этот процесс призвано ПО, поставленное вместе с машиной. Оно превращает физические аппаратные средства в виртуальную (кажущуюся) машину, более удобную для пользователей. По существу, сейчас немисливо пользоваться только «голыми» аппаратными средствами, как это было раньше.

ПО мини- и особенно микро-ЭВМ развито слабее ПО больших ЭВМ широкого назначения. Такое положение объясняется рядом причин, среди которых отметим следующие: 1) сравнительно невысокая и со временем уменьшающаяся стоимость аппаратных средств не стимулирует разработки дорогого ПО; часто с машиной поставляется только минимальное ПО, необходимое для реализации основных функций; 2) разнообразие и специфика применений препятствует созданию стандартизованных прикладных программ и ПО широкого назначения; 3) широкий диапазон конфигураций машин приводит к многочисленным версиям ПО; 4) из-за ограниченной системы команд разработка ПО требует значительных усилий и времени, а расходы на нее практически не сокращаются.

Характерной чертой современного ПО является модульность. Для каждого модуля необходима некоторая минимальная конфигурация, характеризующаяся емкостью основной памяти, средствами прерываний и набором периферийных устройств. Чем проще конфигурация системы, например ЭВМ без массовой памяти на дисках и без видеодисплея, тем более трудной и долгой становится разработка и отладка прикладных программ.

Ниже рассматриваются следующие три класса программных средств.

Базовое программное обеспечение. Базовое ПО состоит из набора автономных программ и подпрограмм, предназначенных главным образом для интерфейса с пользователем как средства автоматизации разработки прикладных программ. Автономность означает, что в каждой программе операции ввода-вывода реализованы на машинном языке и для работы им не требуется взаимодействовать с другими программами. Ниже дается краткая характеристика программ базового ПО, а некоторые из них подробнее изложены в § 7-3.

Редактор текста упрощает и ускоряет создание и модификацию исходных программ, представленных в виде последовательностей (строк) символов. Пользователь взаимодействует с редактором посредством удобных содержательных приказов, инициирующих операции введения и удаления строк и символов, а также поиска символьных цепочек.

Язык ассемблера (язык символического кодирования), по существу, эквивалентен машинному языку и явно отражает архитектуру ЭВМ, но все элементы программ представляются в символической форме. Ассемблерные программы оказываются наиболее эффективными, и пользователь может непосредственно управлять вводом-выводом.

В зависимости от конфигурации могут быть реализованы три разновидности ассемблеров: базовый ассемблер, рассчитанный на минимальную конфигурацию, расширен-

Таблица 7-1

Характеристики ассемблеров

Характеристика	Ассемблер		
	базовый	расширенный	макроассемблер
Основная память байт (слов)	4К	8К	8К
Периферийные устройства	Пишущая машинка	Пишущая машинка, НМД (НМЛ)	Пишущая машинка, НМД (НМЛ)
Число определяемых имен	100	450	500
Скорость ассемблиро- вания, операций/мин	1000	3000	3000—4000
Число директив	8—12	10—25	25—40
Макрокоманды	Нет	Нет	Есть
Сообщения об ошиб- ках	Закодиро- ваны	Подробные	Подробные

ный ассемблер и макроассемблер. Краткие их характеристики приведены в табл. 7-1. Как правило, ассемблеры обеспечивают получение объектных программ в абсолютном или переместимом формате.

Программа-загрузчик предназначена для записи объектных программ в основную память (из устройств ввода) и инициирования их выполнения. Абсолютный (двоичный) загрузчик помещает программу в фиксированную область памяти, а перемещающий загрузчик — в произвольную область памяти.

Редактор связей реализует функцию объединения (связи) нескольких независимо ассемблируемых программ в единый загрузочный модуль. Иногда перемещающий загрузчик и редактор связей реализуется в виде одной программы, которая называется связывающим загрузчиком.

Программа отладки (отладчик) представляет собой удобное средство отладки (локализации и удаления ошибок) прикладных программ. Наиболее эффективным оказывается диалоговый (интерактивный) режим отладки.

Подпрограммы управления вводом-выводом используются как удобное средство управления стандартными периферийными устройствами. Эти подпрограммы обеспечивают необходимое форматирование данных, автоматическое преобразование входных данных в форму, соответствующую представлению их в ЭВМ, и контроль ошибок. Часто пользователь может управлять операциями ввода-вывода независимо от физических устройств, а посредством обращения к логическим устройствам, указывая только адрес логического устройства, адрес буфера данных и число передаваемых байт (слов).

Программы-утилиты предназначены для выполнения тех функций, которые не реализованы аппаратно. Их набор в значительной степени определяется конфигурацией машины. Математические утилиты представляют набор подпрограмм часто используемых математических преобразований, что упрощает программирование и расширяет ограниченные вычислительные возможности машины. В микро-ЭВМ таким образом выполняются операции над числами в форматах с плавающей точкой и повышенной точностью. В мини-ЭВМ программы-утилиты осуществляют вычисление логарифмов, некоторых тригонометрических и обратных тригонометрических функций, извлечение квадратного корня, возведение в степень, вычисление гиперболических функций и т. п. Кроме того, в старших моделях имеются подпрограммы статистической обработки, напри-

мер генерирования псевдослучайных чисел, корреляционного анализа, операций над матрицами и т. п.

В составе базового ПО могут быть трансляторы с языков высокого уровня. Они различаются по числу проходов исходных программ, по требуемой минимальной конфигурации и степени стандартизации. Минимальный объем оперативной памяти в зависимости от возможностей языка составляет 8—16К байт. В микро-ЭВМ и мини-ЭВМ младших моделей обычно реализуются интерпретирующие языки «Бейсик» и «Фокал», а в мини-ЭВМ старших моделей — компилирующие языки «Фортран», «Паскаль», язык С, «Ада» и др.

Средства диагностики представлены комплексом программ (тестов), предназначенных для проверки процессора, оперативной памяти и всех периферийных устройств. Обычно имеется тест-программа общей проверки машины, а при обнаружении неправильной работы вызываются специализированные программы для локализации отказавшего аппаратного компонента.

Для управления программами базового ПО предусматривается сравнительно простая программа-монитор.

Сперационные системы. Дать точное и краткое определение операционной системы (ОС) затруднительно, если вообще возможно. В первом приближении можно считать ОС набором программ (системных), которые кроме рассмотренного выше базового программного обеспечения, осуществляют эффективное управление ресурсами ЭВМ. К главным ресурсам машины относят процессор, основную память и периферийные устройства. Основными функциями ОС являются: планирование работы процессора, распределение и защита памяти, управление периферийными устройствами (вводом-выводом), обработка внутренних и внешних прерываний, управление данными и библиотеками программ, загрузка и выполнение прикладных программ, интерфейс с пользователем (пользователями).

Каждый из элементов этого списка по мере расширения конфигурации аппаратных средств становится все более сложным. Например, в мини-ЭВМ старших моделей в управлении памятью и программами появляются оверлейные структуры, свопинг, сегментация памяти, страничная организация памяти.

Поскольку желательно, чтобы ОС занимала как можно меньше оперативной памяти, ее программы хранятся в накопителе на магнитном диске и передаются в оперативную память, как только в этом возникает необходимость. Одна-

ко некоторая часть ОС, управляющая функциями самой операционной системы, постоянно находится в оперативной памяти. Эта часть называется ядром, исполнительной программой, организующей программой или супервизором.

В машинах, оснащенных ОС, прикладным программам доступ разрешается не ко всем ресурсам ЭВМ. Эта ситуация достигается введением двух режимов работы: супервизорного (системного) и пользовательского. Текущий режим определяется специальным полем в слове состояния процессора. В системном режиме разрешается выполнение всех команд, а в пользовательском некоторые привилегированные команды запрещены. К ним относятся все команды ввода-вывода, управления системой прерываний и некоторые другие. Использование их в прикладных программах осуществляется через запрос супервизора. Попытка выполнить привилегированные команды прикладной программой вызывает внутреннее прерывание, анализ причины его и определенные действия, например прекращение исполнения прикладной программы.

Объем и возможности ОС очень сильно зависят от конфигурации аппаратных средств: емкости оперативной памяти, состава и характеристик периферийных устройств. Важную роль играют также быстродействие и система команд процессора. Для небольшой специализированной системы не требуется сложная ОС, но по мере расширения конфигурации и числа возможных применений необходима эффективная и мощная ОС. Наиболее сложные ОС реализованы в мини-ЭВМ старших моделей.

Базовая ОС предназначена для загрузки и инициирования программ под управлением оператора и рассчитана на простые периферийные устройства. Система минимальной конфигурации включает в себя процессор, оперативную память емкостью 4—8К байт, пишущую машинку, перфоленточный считыватель и перфоратор, а также (необязательно) накопители на диске или ленте. ОС имеет следующие модули: модуль генерации системы, обеспечивающий настройку ОС на конкретную конфигурацию аппаратных средств; резидентный монитор, посредством которого пользователь вызывает программы ОС; редактор текста; ассемблер; перемещающий загрузчик; систему управления вводом-выводом, представляющую собой набор подпрограмм управления всеми периферийными устройствами; программу отладки и диагностики; транслятор с одного из простых языков высокого уровня.

Дисковая ОС (иногда ленточная ОС) реализует все функции базовой ОС, возможно с некоторыми расширениями, а также более сложные функции управления периферийными устройствами. Дисковая ОС благодаря более быстрому доступу обеспечивает большую гибкость и эффективность работы ЭВМ. Для реализации дисковой ОС, как правило, достаточна следующая минимальная конфигурация: процессор, оперативная память 8—32К байт, накопитель на магнитных дисках (ленте), пишущая машинка, быстродействующие принтер, перфоленточный считыватель и перфоратор. По сравнению с базовой ОС здесь имеются развитые средства управления файлами, автоматической каталогизации файлов, организации библиотек. Как правило, имеются трансляторы нескольких языков высокого уровня.

Мультипрограммные ОС рассчитаны на параллельное обслуживание нескольких пользователей в пакетном или интерактивном режимах. В таких ОС важное значение приобретают вопросы планирования (диспетчирования) ресурсов ЭВМ и защиты памяти, чтобы программы не могли влиять друг на друга.

ОС реального времени представляют собой мощные мультипрограммные системы, предназначенные для управления сложными производственными процессами. Параллельно со сбором данных, их обработкой и выдачей управляющих воздействий они могут обслуживать нескольких пользователей в режиме с разделением времени. Для таких ОС характерны жесткие временные ограничения на обработку прерываний от внешних источников.

Минимальная конфигурация для реализации ОС реального времени включает в себя процессор (иногда несколько), оперативную память 16—32К байт, таймер реального времени, систему многоуровневых приоритетных прерываний, средства защиты памяти, накопители на магнитных дисках (и лентах), быстродействующие перфоленточные считыватель и перфоратор, построчный принтер и (необязательно) карточный считыватель.

Кроме программных модулей, входящих в состав дисковой ОС, здесь необходимы планировщик реального времени, библиотека переместимых программ, загрузчик реального времени и сложный монитор, выполняющий разнообразные функции.

Прикладное программное обеспечение создается пользователем или для пользователя и определяется требованиями конкретного применения. В некоторых случаях (так

называемые системы под ключ) прикладное ПО поставляется вместе с ЭВМ.

Прикладное ПО отличается наибольшим разнообразием, так как сфера применения мини- и микро-ЭВМ расширяется чрезвычайно быстрыми темпами. Мини-ЭВМ сейчас успешно заменяют средние и даже большие ЭВМ широкого назначения, а также различные специализированные системы. Разработка адекватного ПО становится важнейшей частью проектирования любой системы на базе ЭВМ. Особую роль играют средства диагностики и самоконтроля, которые необходимы практически в любой области применения.

В заключение отметим два направления в развитии ПО. Первое касается аппаратной реализации функций ОС благодаря все более увеличивающейся емкости микросхем ПЗУ (отсюда несколько странный термин «кремниевая ОС»). Во-вторых, уровень ПО мини-ЭВМ приближается к уровню ПО средних и больших ЭВМ. Очевидно, что со временем мини-ЭВМ будут обладать большей частью возможностей современных средних и больших ЭВМ.

7-2. ЯЗЫКИ ОПИСАНИЯ АЛГОРИТМОВ

В зависимости от имеющихся средств автоматизации программирования разработка прикладных программ ведется на исходном языке одного из трех уровней: машинном языке, языке ассемблера или языке высокого уровня. Естественным языком всех ЭВМ является язык двоичных последовательностей (цепочек), с помощью которых представляются команды и данные. Программа, которую интерпретируют аппаратные средства ЭВМ, называется объектной программой (кодом) или программой на машинном языке. Следовательно, исходные (входные) программы на языках ассемблера или на языках высокого уровня необходимо транслировать на машинный язык.

Исходные языки отличаются друг от друга в той степени, в которой они отражают структуру машин и выполняемые ими операции. Машинный язык и ассемблер явно отражают аппаратные средства и обеспечивают максимальные возможности непосредственного управления работой ЭВМ. Оба этих языка низкого уровня естественно оказываются машинно-зависимыми (машинно-ориентированными), а составленные на них программы обладают наименьшей мобильностью.

Языки высокого уровня (алгоритмические языки) почти не отражают структуру ЭВМ и считаются машинно-незави-

симыми. Программы на этих языках характеризуются высокой мобильностью. Наибольшее распространение для мини- и микро-ЭВМ получили такие языки высокого уровня, как «Фортран», PL/M и различные его разновидности, «Бейсик», «Паскаль» и «Ада».

При желании любой алгоритм можно запрограммировать на машинном языке в двоичной или более удобных 8-ричной и 16-ричной формах. Однако этот язык обладает рядом существенных недостатков и на практике почти не применяется за исключением тех ситуаций, когда прикладная программа невелика и нет никаких средств автоматизации программирования (или средств поддержки). Усложнение структуры ЭВМ, расширение системы команд, несколько форматов данных еще более затрудняют практическое применение машинного языка.

Язык ассемблера отличается от машинного языка тем, что операции, операнды и адреса представляются в символической форме, более удобной для пользователя. Ассемблер, по существу эквивалентный машинному языку, позволяет разработать максимально эффективные программы, оцениваемые длиной и временем выполнения. Объектные программы, генерируемые транслятором языка высокого уровня, оказываются менее эффективными.

По мере совершенствования средств автоматизации программирования все более широкое применение будут находить языки высокого уровня благодаря их многочисленным достоинствам. К ним относятся увеличение продуктивности труда программистов, мобильность, лучшая документированность и компактность программ, сокращение времени отладки и некоторые другие.

Программы-трансляторы языков высокого уровня делятся на компиляторы и интерпретаторы. Компилятор обрабатывает исходную программу и порождает полную объектную программу, которая затем выполняется на ЭВМ. Интерпретатор не создает полной объектной программы, а последовательно транслирует и выполняет операторы исходной программы. Подавляющее большинство языков высокого уровня рассчитано на компилятор, а самым распространенным интерпретирующим языком является «Бейсик».

Язык ассемблера. В этом языке элементы программы (коды операций, данные, адреса) представляют в символической форме, отражающей их содержательный смысл (семантику). Программы почти не содержат численных значений, их заменяют символические имена; мнемоники опе-

раций, метки адресов и идентификаторы переменных и констант. Первым символом имени не должна быть цифра, а длина его изменяется от одного до 5—8 символов. Некоторые ассемблеры допускают имена произвольной длины, но распознают только первые 5—8 символов. В именах не допускается пробелов и управляющих символов. Вместо пробела можно использовать некоторый особый символ, например, значок подчеркивания.

Программа — транслятор с языка ассемблера называется ассемблирующей программой или, короче, ассемблером. Если ассемблер реализован на ЭВМ, которая может выполнять полученную объектную программу, его называют естественным ассемблером. Ассемблер, который постоянно находится в памяти ЭВМ, называется резидентным. Для хранения резидентного ассемблера, рассчитанного на минимальную конфигурацию машины, достаточно ПЗУ емкостью 4К—8К байт. Нерезидентный ассемблер хранится во внешней памяти, например в накопителе на гибком диске или даже на перфоленте, а при необходимости загружается в оперативную память.

Ассемблер, реализованный на ЭВМ, которая не может выполнять объектную программу, называется кросс-ассемблером. Кросс-ассемблеры для микро-ЭВМ обычно реализуются на мини-ЭВМ и в сетях ЭВМ с разделением времени. Сформированные кросс-ассемблерами объектные программы выводятся на некоторый носитель и загружаются с него в память ЭВМ, для которой они предназначены (целевой ЭВМ). Такую загрузку можно осуществить также по линии связи.

При разработке языка ассемблера необходимо определить формат строки, мнемонику всех команд, способы задания режимов адресации, числовых и символьных констант. Кроме того, вводятся специальные директивы для самой программы-ассемблера. Была сделана попытка стандартизировать язык ассемблера для микропроцессоров и микро-ЭВМ.

Формат программ. Ассемблерная программа состоит из последовательности символьных строк, имеющих определенный формат. Каждая строка содержит одну команду или оператор. Алфавит символов включает в себя прописные латинские буквы, цифры, символы разделителей и ограничителей, а также символы формата (возврат каретки, перевод строки, перевод формата).

В большинстве ассемблеров современных ЭВМ строки имеют четыре поля: метку, операцию (код операции, код),

операнд (аргумент) и комментарий. Из всех полей в каждой строке только поле кода должно быть непустым, остальные поля могут быть пустыми.

Чтобы показать характерные особенности языка ассемблера, приведем простой фрагмент программы для процессоров СМ. Программа инициализирует массив ARRAY из 10 слов, причем в i -е слово ($1 \leq i \leq 10$) загружается значение $i-1$:

Метка	Код	Операнд	Комментарий
	==1000.		; Начало программы с
			; ячейки 1000
START:	MOV	#20., R1	; Длина массива 20 байт
NEXT:	SUB	#2, R1	; Декремент индекса
	MOV	R1, R0	; Вычисление значения
	ASR	R0	; элемента массива
	MOV	R0, ARRAY(R1)	; Инициализация элемен-
			; та
	BNE	NEXT	; Заикливание
	HALT		; Останов процессора
ARRA	.WORD	0, 0, 0, 0, 0, 0, 0, 0, 0	; Память для массива
	.END	START	; Пусковой адрес про-
			; граммы

Символическому имени в поле метки присваивается значение адреса памяти, по которому хранится отмеченная команда (точнее, ее первый байт или первое слово). Исключение составляет только директива EQU. Метки служат операндами команд передачи управления, освобождая пользователя от манипуляций абсолютными адресами памяти, тем более, что при разработке программы они могут быть неизвестны. Появление метки, по существу, является ее определением, поэтому каждая из меток в программе может быть определена только один раз, но число обращений к меткам не ограничивается.

Метка идентифицируется двумя способами: либо начинается с самой первой позиции строки (а строка без метки начинается минимум с одного пробела), либо заканчивается обязательным двоеточием. Обычно допускается несколько меток в строке, каждая из которых заканчивается двоеточием. Конечно, все такие метки имеют одно и то же численное значение. Самые распространенные ошибки, относящиеся к использованию меток, связаны с многократным определением меток и с неопределенными метками.

Наиболее важное поле кода содержит мнемонику машинной команды или директивы ассемблера. Мнемоника выражает основные функции команды или директивы и со-

стоит из 1—10 символов, например A, ADD (add) — сложить; S, SUB (subtract) — вычесть; ROTL (rotate left) — циклически сдвинуть влево; B, BR (branch) — перейти.

Часто мнемоника специфицирует размер операнда, например MOV или MOVE.W — переслать слово; MOVB или MOVE.B — переслать байт; MOVE.L — переслать длинное слово (32 бита).

По мере расширения системы команд мнемоники становятся более громоздкими, например MOVSB (move string of byte) — переслать байтную цепочку; LOOPNZ (loop if not zero) — повторить цикл, если не ноль.

Несмотря на некоторое неудобство, связанное с использованием английских аббревиатур, запоминать и использовать мнемоники проще, чем численные коды операций.

Мнемоники языка ассемблера каждой ЭВМ фиксированы и входят в множество зарезервированных (ключевых) слов языка. Ассемблер использует мнемоники для адресации внутренней таблицы, хранящей машинные коды операций. В этом кроется главное отличие мнемоник от меток, которые каждый пользователь вводит по своему усмотрению и которые заранее неизвестны. Поэтому ассемблер для каждой программы должен строить динамическую таблицу имен, в которой хранятся имена, определяемые пользователем, и их численные значения.

Поле операнда идентифицирует один или два операнда команды, поэтому оно интерпретируется в соответствии с функцией команды. Операнды определяются именами, численными и символьными константами, а также выражениями. В командах с неявной адресацией поле операнда пустое, например в командах CMA — инвертировать аккумулятор, STC — установить флажок переноса и др.

Имена в поле операнда могут быть зарезервированы словами ассемблера, например идентифицирующими внутренние регистры процессора, данными, определяемыми пользователем с помощью директив присвоения, или метками, находящимися в поле метки других команд.

Символьные константы в большинстве ассемблеров вводятся с помощью заключения их в апострофы или кавычки, причем иногда достаточно только начального апострофа. Численные константы представляются в различных системах счисления, что указывается специальными дескрипторами. Например, в ассемблере микропроцессора КР580 система счисления определяется буквой, завершающей число: H — 16-ричная, O (или Q) — 8-ричная, D — десятичная, B — двоичная. Отсутствие дескриптора означает десятич-

ное число. 16-ричные числа должны начинаться с цифр 0—9. В ассемблере процессоров СМ десятичное число идентифицируется завершающей точкой, а остальные числа считаются 8-ричными.

Чтобы проследивать размещение ассемблируемой программы в памяти, во всех ассемблерах используется внутренняя переменная, называемая счетчиком адресов (ячеек). Она обозначается специальным символом, например звездочкой (*) или точкой (.). Счетчик адресов инициализируется специальной директивой и модифицируется после обработки каждой строки путем прибавления длины команды или данных. При ассемблировании счетчик адресов ведет себя как аппаратный программный счетчик РС. Однако при обработке констант счетчик адресов принимает такие значения, которые РС никогда не примет.

Допустимые выражения в поле операнда зависят от конкретного ассемблера, но, как минимум, в них могут фигурировать константы, текущее значение счетчика адресов, имена и указанные выше термы, соединяемые знаками арифметических и логических операций. Обычно допускаются операции сложения, вычитания, дизъюнкции и конъюнкции. В более совершенных ассемблерах можно использовать операции умножения, деления, сдвига, вычисления остатка от деления, а также применять скобки. Вычисление выражения производится строго слева направо, а старшинство операций, обычное для языков высокого уровня, может не соблюдаться. Например, выражение $A + B * C$ вычисляется как $(A + B) * C$, а не как обычно $A + (B * C)$. Значение выражения из поля операнда вычисляется один раз во время ассемблирования.

В поле комментария содержится краткое пояснение действий команды в контексте конкретной программы. Начало комментария идентифицируется разделителем (чаще всего точка с запятой или наклонная черта), а текст его, содержащий любые печатные символы (включая пробел), полностью игнорируется при ассемблировании, но печатается в листинге программы. Комментарием может быть целая строка, начинающаяся с соответствующего разделителя. Интуитивное желание пользователя экономить память (каждый символ — байт!) не должно приводить к сокращению комментариев в ущерб пониманию программы другими людьми. Комментарии ради удобства даются малыми русскими буквами.

Имеются две разновидности размещения полей в позициях строки. Фиксированный формат рассчитан на то, что

каждое поле занимает определенные позиции строки. Например, при подготовке программ для кросс-ассемблера на перфокартах каждому полю отводятся конкретные колонки. Здесь не требуются специальные разделители полей, но несколько усложняется подготовка программ. В более удобном свободном формате границы полей не фиксируются, но необходимы разделители полей, в качестве которых используются пробел, двоеточие, точка с запятой, запятая и др. Для правильной идентификации полей устанавливается набор правил следующего вида: 1) любое число пробелов эквивалентно одному пробелу; 2) имя, заканчивающееся двоеточием, считается меткой; 3) в первой позиции строки, которая не содержит метки, должен быть пробел; 4) символы между первым и вторым разделителями интерпретируются как мнемоника; 5) если команда не требует операнда, оставшаяся часть строки считается комментарием; 6) когда операнд необходим, символы между вторым и третьим разделителями считаются операндом; аналогично интерпретируются операторы с несколькими операндами, и разделителем операндов обычно служит запятая; 7) комментарий начинается со специального разделителя, например с точки с запятой.

Режимы адресации. Необходимы специальные соглашения об идентификации в поле операнда многочисленных режимов адресации, которые имеются в современных ЭВМ. Например, такие соглашения для процессоров СМ используют условные обозначения: R — регистр, Expr — выражение. Чтобы отличить регистры от ячеек памяти с одинаковыми адресами, используется символ %, например %0 обозначает регистр R0, %1 — регистр R1 и т. д. Более удобные содержательные обозначения R0—R5, SP, PC вводятся в начале программы с помощью операторов прямого присваивания: R0 = % 0, R1 = % 1, ..., R5 = % 5, SP = % 6, PC = % 7.

Директивы ассемблера. Директивы сообщают программе-ассемблеру информацию, необходимую для формирования фиксированных таблиц, определения имен, резервирования памяти для переменных, размещения в памяти программ и подпрограмм и задания формата листинга. Хотя их мнемоники директив появляются в поле кода, директивы не соответствуют машинным командам, что объясняет их другое название «псевдокоманды». Метки и комментарии в директивах необязательны, а содержимое поля операнда зависит от их функций. В языке ассемблера процессоров СМ первым символом директивы должна быть точка,

но это не является обязательным условием для ассемблеров других ЭВМ. Ниже рассматриваются наиболее распространенные директивы.

В директиве начала **ORG** значение выражения из поля операнда определяет адрес памяти, по которому размещается первый байт или слово следующих за директивой команды или данных. Эта директива дает возможность пользователю размещать сегменты программ и данных в конкретных областях памяти. До новой директивы **ORG** команды и данные размещаются в последовательных ячейках памяти. При отсутствии первой директивы **ORG** принимается нулевой начальный адрес.

Директива окончания **END** свидетельствует о достижении физического конца исходной программы. Следовательно, эта директива должна быть в последней строке любой программы. Операнд директивы **END** определяет начальную точку выполнения программы (пусковой адрес) после ее ассемблирования без ошибок.

С помощью директивы **EQU** прямого присваивания имени, находящемуся в поле метки (чтобы отличить имя от обычной метки, после него обычно не ставят двоеточия), присваивается вычисленное значение выражения из поля операнда. Эту же функцию в ассемблере процессоров **СМ** выполняет оператор прямого присваивания со следующим форматом: `<имя> = <выражение>`. Когда слева от знака равенства находится точка, обозначающая счетчик адресов, получается эквивалент директивы **ORG**.

Важно отметить, что директива **EQU** не вызывает запоминания в программе никакого значения. Она ассоциирует имя и число только на время ассемблирования. Когда ассемблер встречается в другом месте программы имя, фигурирующее в строке с директивой **EQU**, он подставляет соответствующее численное значение, как будто встретилось именно оно. Обычно директива **EQU** не допускает переопределения имени, т. е. каждое имя может появиться в поле метки только одной директивой **EQU**. При необходимости такого переопределения используется директива **SET** (установить), имеющая аналогичный формат. Оператор прямого присваивания допускает переопределение имен. Директивы **EQU** применяются для определения часто используемых имен и обычно группируются в начале программы, образуя раздел определений.

Директивы **DB** (другие названия **.BYTE** и **FCB**) и **DW** (другие названия **.WORD** и **FCW**) предназначены для задания констант длиной в байт и слово соответственно.

Операнд этих директив представляет собой список выражений, элементы которого разделяются запятыми. При обработке директив DB и DW ассемблер вычисляет значения каждого из выражений и включает их в программу с последующим запоминанием в памяти. Таким образом, с помощью этих директив определяются константы времени выполнения, которые может считывать процессор. Например,

Метка	Код	Операнд	Комментарий
ARRAY:	.WORD	127., 63., 31., 15., 7., 3., 2., 1.	

	MOV	AR R1	; Загрузка в R1 чис ; ла 127

	MOV	ARRAY+3, R1	; Загрузка в R1 чис ; ла 15

в следующем фрагменте директива .WORD запоминает восемь значений массива, начиная с адреса ARRAY. В таблице имен, которую формирует ассемблер, значением ARRAY будет адрес, по которому хранится число 127.

Значения, формируемые директивами DB и DW, загружаются в память только один раз, поэтому эти директивы используются для инициализации констант, которые программа не должна изменять. Инициализировать переменные с помощью директив DB и DW не следует, так как, если программа выполняется повторно без загрузки, значения переменных будут модифицированы. Для инициализации переменных в программе необходимы специальные команды.

В директиве DC определения символов (другое обозначение .ASCII) поле операнда содержит цепочку символов, заключенную в апострофы. При ассемблировании коды символов помещаются в смежные байты программы.

Директива DS (другое обозначение RMB) резервирования памяти с выражением в поле операнда распределяет память для хранения переменных. Ассемблер вычисляет значение выражения и прибавляет его к счетчику адреса. При загрузке программы эти байты не инициализируются, в частности, не следует полагать, что они содержат нули. Директива .BLKW выполняет аналогичную функцию, но

резервирует слова. Отметим, что зарезервировать память можно с помощью счетчика адреса, например оператор. = +32. обеспечивает резервирование 32 байтов памяти.

Директивы условного ассемблирования дают возможность включать или не включать в объектную программу фрагменты исходной программы. В наиболее простой форме директива условного ассемблирования имеет следующий формат:

Метка	Код	Операнд
[метка:]	IF	<выражение>
Фрагмент исходной программы		
[метка:]	ENDIF	

Ассемблер вычисляет значение выражения из поля операнда. Если оно равно нулю, операторы между директивами IF и ENDIF пропускаются, а когда значение выражения отлично от нуля, фрагмент ассемблируется обычным образом. Ассемблер может иметь целый набор директив условного ассемблирования (разумеется, с различными мнемониками), которые позволяют проверить значение выражения «на нуль», «не нуль», «меньше нуля», «не больше нуля», «больше нуля», «не меньше нуля».

В ассемблерах процессоров, адресующих слова памяти, полезной является директива .EVEN, которая устанавливает счетчик адреса на четное значение, т. е. производит его инкремент в случае нечетного текущего значения. Ее рекомендуется использовать после директив .BYTE и .ASCII.

Макрокоманды. В языке ассемблера многих ЭВМ имеется удобное средство, позволяющее уменьшить длину исходной программы, улучшить ее структуру и сократить время разработки. Этим средством являются макрокоманды. Макрокоманды дают возможность пользователю один раз в начале программы определить (наименовать) последовательность ассемблерных операторов, а затем использовать введенное имя как замену определенной последовательности. Таким образом, макрокоманды как бы расширяют систему команд ЭВМ, добавляя в нее «большие» команды, наиболее удобные для конкретной прикладной программы. С их помощью можно имитировать мнемоники команд других ЭВМ и генерировать для них объектные программы.

Принцип использования макрокоманд ясен из следующего простого примера. Для увеличения содержимого регистра REG на 4 команду INC инкремента на единицу придется повторить четыре раза:

Метка	Код	Операнд	Комментарий
	INC	REG	; Эти команды увеличивают содержимое регистра REG на 4
	INC	REG	
	INC	REG	
	INC	REG	

Когда необходимость увеличения содержимого REG на 4 встречается еще несколько раз, пользователь должен всякий раз копировать приведенные команды. (Конечно, эту же функцию выполняет команда сложения с непосредственным операндом 4, поэтому данные команды следует рассматривать только как пример). Если же один раз назвать эту последовательность, например INC4, то затем имя INC4 можно использовать в поле кода, а простую задачу замены имени INC4 четырьмя командами INC REG может выполнить программа-ассемблер. Ассемблер, в котором допускается использовать макрокоманды, называется макроассемблером.

Определение и вызов макрокоманд подчеркивают их отличие от подпрограмм (см. § 7-5), которые иллюстрирует рис. 7-1. Тело подпрограммы включается в объектную программу один раз, а выполнение ее инициируется специальной командой вызова CALL или JSR. После выполнения подпрограммы другая специальная команда RET или RTS осуществляет возврат из подпрограммы в точку вызова. Подпрограммы со стандартным механизмом вызова и возврата называются замкнутыми подпрограммами. Недостаток их заключается в непроизводительных потерях времени на передачу параметров, вызов и возврат, а достоинство — в экономии программной памяти. Тело макрокоманды при каждом употреблении ее имени в поле кода целиком включается в объектную программу, что объясняет другое название макрокоманд — открытые подпрограммы. Макрокоманды не экономят программную память, но устраняют потери времени процессора на вызов и возврат.

С макрокомандами связаны три различных действия. В начале программы пользователь должен определить все необходимые макрокоманды. Макроопределение имеет следующий формат:

Метка Код Операнд

.MACRO <имя> , <список формальных параметров>

Операторы ассемблера, образующие тело макрокоманды

.ENDM

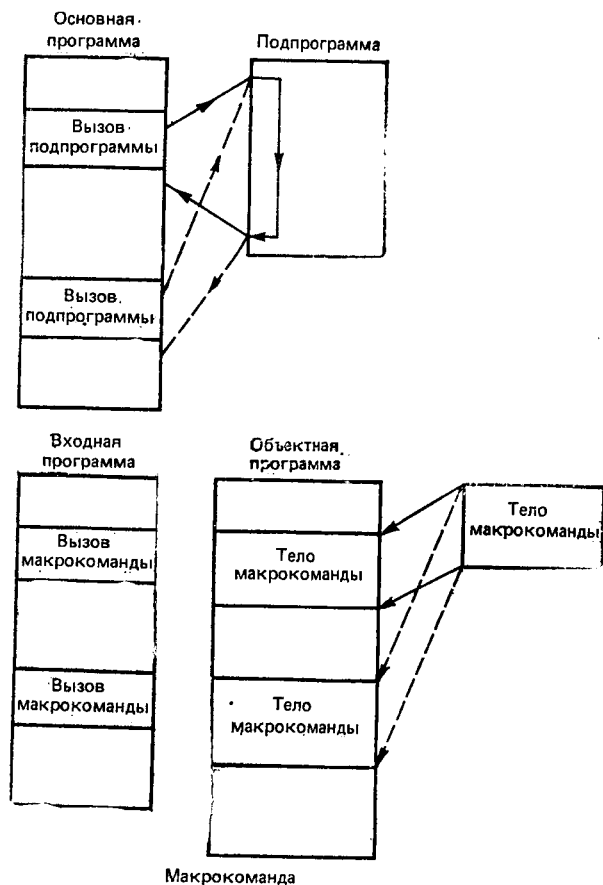


Рис. 7-1. Обращение к подпрограмме (а) и вызов макрокоманды (б)

Приведенная выше макрокоманда INC4 определяется таким образом:

Метка	Код	Операнд
	.MACRO	INC4, REG
	INC	REG
	INC	REG
	INC	REG
	INC	REG
	.ENDM	

Несколько отличающийся формат макроопределений принят в ассемблере микропроцессора КР580:

Метка	Код	Операнд
<имя>	MACRO	<список формальных параметров>
		Операторы ассемблера, образующие тело макрокоманды
	ENDM	

Отметим появление специальных директив MACRO и ENDM, а также список формальных (фиктивных) параметров (аргументов). Список представляет собой последовательность имен с разделителями между ними. В качестве разделителя обычно используется запятая, но иногда допускаются также пробел и точка с запятой. Макроопределения не генерируют команд объектной программы, они предназначены для «внутреннего использования» макроассемблером. Для проверки парности директив MACRO и ENDM в поле операнда директивы ENDM рекомендуется указывать имя макрокоманды.

Вызов макрокоманды (макрывызов) имеет следующий формат:

Метка	Код	Операнд
		<имя> <список фактических параметров>

Таким образом, макрвызов заключается в том, что имя ранее определенной макрокоманды появляется в поле кода и сопровождается списком фактических (действительных) параметров. Каждый макрвызов заставляет ассемблер производить расширение макрокоманды (макрорасшире-

ние), т. е. включать в программу тело макрокоманды, в котором формальные параметры заменяются на фактические. Замена производится текстуально в соответствии с положением параметров в списках макроопределения и макровызова. Например, вызов INC R0 приведет к следующему расширению:

Метка	Код	Операнд
	INC	R0
	INC	R0
	INC	R0
	INC	R0

При использовании макрокоманд некоторые сложности возникают, когда операторы тела макрокоманды имеют метки. Каждая метка должна появиться в программе только один раз (так как это эквивалент адреса памяти), поэтому несколько вызовов макрокоманд с метками могут привести к появлению многократно определенных меток.

Применение вложенных макровыводов часто помогает улучшить структуру программы и всегда сокращает длину исходной программы.

В заключение отметим, что макрокоманды могут быть рекурсивными, т. е. макроопределение содержит вызов определяемой макрокоманды. При использовании рекурсивных макрокоманд необходимо внимательно следить за тем, чтобы расширение макрокоманды в конце концов закончилось. Обычно окончание расширения достигается с помощью директив условного ассемблирования.

Оптимизация ассемблерных программ. При программировании на ассемблере используются несколько приемов, направленных на сокращение длины и времени выполнения программы. Ниже приведены некоторые приемы для микропроцессора КР580.

1. Если сразу после команды вызова следует команда возврата, их можно заменить одной командой перехода к подпрограмме. Например, исходный фрагмент из двух команд CALL SUBR; RET заменяется одной командой JMP SUBR. Предполагается, конечно, что сама подпрограмма SUBR заканчивается командой возврата.

2. Когда подпрограмме передаются фиксированные данные, вместо инициализации и передачи указателя данных можно передать в программе сами данные. Например, фрагмент

Метка	Код	Операнд	Комментарий
	LXI	H, DATA	; Инициализация
	CALL	SUBR	; указателя
			; Вызов подпро-
			; граммы

DATA:	DB	8, 16, 32 64	; Фиксированные
			; данные

допускает следующую замену:

Метка	Код	Операнд	Комментарий
	CALL	SUBR	; Вызов подпро-
			; граммы
DATA:	DB	8, 16, 32, 64	; Фиксированные
			; данные

Здесь подпрограмма SUBR должна сформировать новый адрес возврата после данных и использовать его вместо адреса, включенного в стек командой CALL.

3. Короткими таблицами, элементы которых содержат более одного байта, удобнее и быстрее манипулировать, если длина элемента составляет степень двух. Этот прием увеличивает емкость памяти для хранения таблицы, но сокращает время операций с таблицей, так как при вычислении индекса операция умножения заменяется операцией сдвига.

4. Особенно важно оптимизировать циклы, что сокращает время выполнения программы лучше других приемов. Приемы оптимизации циклов включают удаление из цикла команд с инвариантными результатами, перемещение безусловных переходов вне цикла, сжатие нескольких команд условных переходов в одну, хранение часто используемых переменных и констант в регистрах процессора, а не в ячейках памяти. Пусть, например, в цикле имеется фрагмент, запоминающий в ячейке MAX большее из двух чисел, находящихся в ячейках NUM1 и NUM2. Ниже приведена очевидная последовательность команд (предполагается, что регистры H, L содержат адрес NUM2):

Метка	Код	Операнд	Комментарий
	LDA	NUM1	; Загрузка первого числа
	CMP	M	; Сравнение со вторым
			; числом
	JNC	ONE	; Первое число больше
	MOV	A, M	; Загрузка второго числа
	STA	MAX	; Запоминание максиму-
			; ма
	JMP	TWO	; Переход
ONE:	STA	MAX	; Запоминание максиму-
			; ма
TWO:	***		; Следующая команда

Эту последовательность можно заменить фрагментом:

Метка	Код	Операнд	Комментарий
	LDA	NUM1	; Загрузка первого числа
	CMP	M	; Сравнение со вторым
			; числом
	JNC	ONE	; Первое число больше
	MOV	A, M	; Загрузка второго чис-
			; ла
ONE:	STA	MAX	; Запоминание максиму-
			; ма
	***		; Следующая команда

Для ускорения многократно повторяющегося цикла целесообразно расширить команды, образующие тело цикла. Пусть подпрограммы SUB1 и SUB2 должны выполняться по 10 раз. Стандартное решение имеет следующий вид (регистр C — счетчик):

Метка	Код	Операнд	Комментарий
	MVI	C, 10	; Инициализация счетчи-
			; ка
LOOP:	CALL	SUB 1	; Вызовы подпрограмм
	CALL	SUB 2	
	DCR	C	; Декремент счетчика
	JNZ	LOOP	; Заикливание

В расширенном цикле

Метка	Код	Операнд	Комментарий
	MVI	C, 5	; Инициализация счет-
			; чика
LOOP:	CALL	SUB 1	; Вызовы подпрограмм
	CALL	SUB 2	
	CALL	SUB 1	
	CALL	SUB 2	
	DCR	C	; Декремент счетчика
	JNZ	LOOP	; Заикливание

число выполнений команд DCR и JNZ сокращается вдвое.

7-3. СРЕДСТВА АВТОМАТИЗАЦИИ РАЗРАБОТКИ ПРИКЛАДНЫХ ПРОГРАММ

Для эффективной разработки прикладных программ на языке ассемблера или на языках высокого уровня пользователю, кроме ассемблера или компилятора, потребуется несколько других программ, что иллюстрируется рис. 7-2. Прежде всего необходим редактор текста, позволяющий вводить в машину и модифицировать исходную программу. Результатом последующей трансляции является объектный модуль, который обычно запоминается во внешней памяти. Термин «объектный модуль» подчеркивает тот факт, что программа может состоять из нескольких частей (модулей), допускающих автономную трансляцию. В модулях могут быть внешние обращения (ссылки) к именам, определенным в других модулях. Все внешние обращения оформляются таблицей в самом объектном модуле. Следующая программа — редактор связей, или компоновщик — объединяет объектные модули в единый загрузочный модуль, в котором уже не может быть внешних обращений. Программа-загрузчик размещает загрузочный модуль в памяти и инициирует выполнение программы. Главная функция загрузчика — скорректировать (переместить) адреса в соот-

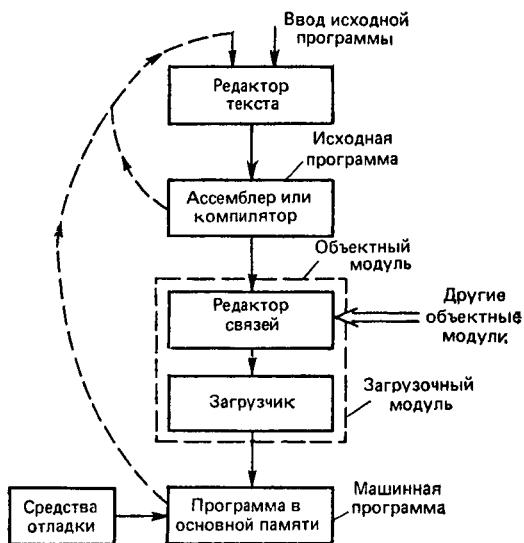


Рис. 7-2. Основные этапы разработки прикладных программ

ветствии с фактическим положением программы в памяти. Информацию для перемещения обеспечивает транслятор. Наконец, для локализации в программе логических ошибок потребуются средства отладки.

Перечисленные выше программы, по существу, помогают эффективно использовать ЭВМ и сокращают сроки разработки прикладных программ. В мини-ЭВМ и системах проектирования микро-ЭВМ они хранятся в накопителях на магнитных дисках и работают под управлением программы-монитора. Для некоторых микро-ЭВМ данные программы поставляются на перфоленте.

РЕДАКТОР ТЕКСТА

Системная программа, предназначенная для создания и модификации исходных программ (текстовых файлов), называется редактором текста. Текстовый файл представляет собой последовательность записей (строк), состоящих из буквенно-цифровых и управляющих символов. В качестве разделителей строк обычно используется комбинация символов возврата каретки CR (BK) и перевода строки LF (PS), а окончание файла идентифицируется символом конца файла EOF.

Собственно редактирование осуществляется пользователем с помощью приказов (команд), вводимых с системной консоли, например электрифицированной пишущей машинки. Во всех редакторах текста имеются приказы манипуляций строками и символами, а более сложные редакторы могут манипулировать целыми файлами, например объединять несколько файлов в один. Размер редактора текста зависит от его возможностей и варьируется в пределах 1К—8К байт.

Редактируемый текст размещается в области оперативной памяти, которая называется буфером текста. Размер буфера текста не фиксируется (кроме естественного максимума) и изменяется в зависимости от реальной длины текста. Для локализации текущей позиции в буфере текста используется переменная, называемая указателем буфера BUFPTR или просто указателем PTR. Указатель напоминает метку на экране символьного дисплея, но в позиционировании его имеется одна особенность. Метка идентифицирует символьную позицию, в которую с клавиатуры или из ЭВМ будет вводиться очередной символ, а указатель буфера адресует позицию между двумя соседними символами. Символ вводится в позицию, непосредственно предшествующую

шую указателю буфера. С помощью специальных приказов допускается позиционирование указателя в любую точку внутри буфера текста, а передвижение его вне границ начала и конца буфера блокируется. В некоторых редакторах текста вводится еще одна переменная (маркер), которая используется для запоминания положения указателя буфера.

Вызов редактора текста из внешней памяти осуществляется приказом EDIT, а готовность его к работе определяется выводом стимулирующего символа, например звездочки. Редактор текста может вводиться также с перфоленты.

Редактор выполняет несколько функций, из которых основными являются модификация ранее созданного файла, а также создание и редактирование нового текстового файла. В обоих случаях приказ EDIT как аргумент содержит имена соответствующих файлов. Рассмотрим типичные приказы редактора, идентифицируемые символьными мнемониками.

С помощью приказа I(insert) осуществляется ввод данных в буфер текста с системной консоли. После распознавания буквы I редактор идентифицирует последующие вводимые символы, как текст, до восприятия символа EOF или изменения режима управляющей клавишей. При нажатии клавиши CR редактор автоматически добавляет к тексту символ LF и переходит к восприятию следующей строки.

Приказ печати T (type) используется для проверки правильности текста, содержащегося в буфере. Для указания объема выводимого текста приказ T имеет префиксный аргумент n, представляющий собой десятичное целое со знаком. Если $n > 0$, на консоли печатается текст от текущего положения указателя буфера до n-го символа возврата каретки (печать n строк вперед). Когда же $n < 0$, печатаются n строк, находящиеся перед текущей строкой до достижения указателя буфера (печать n строк назад). При $n = 0$ печатается текст от начала текущей строки до указателя буфера. По умолчанию (аргумент отсутствует) принимается значение $n = 1$, что вызывает печать текста от указателя буфера до конца текущей строки.

С помощью нескольких приказов осуществляется позиционирование указателя буфера. Приказ B (beginning) передвигает указатель на начало буфера текста и используется для задания начальной точки при подсчете строк текста и поиске определенного фрагмента текста, а также распечатке содержимого всего буфера. Приказ Z перемещает указатель на конец буфера текста и применяется для добав-

ления в буфер нового текста. Приказ L(line) передвигает указатель на задаваемое префиксным аргументом n число строк вперед ($n > 0$) или назад ($n < 0$) относительно текущей строки и помещает его в начало соответствующей строки. Если $n = 0$, указатель буфера передвигается на начало текущей строки. Приказ C(character) имеет такую же семантику, что и приказ L, но его аргумент n определяет число символов, причем по умолчанию принимается значение $n = 1$, а аргумент $n = 0$ не влияет на положение указателя. Приказ C удобно использовать для перемещения указателя в пределах текущей строки.

Еще одну группу образуют приказы удаления текста. Приказ K(kill) стирания строк имеет префиксный аргумент n . Если $n > 0$, производится стирание текста впереди от текущего положения указателя буфера до n -го символа перевода строки, а при $n < 0$ стирается текст, включающий в себя n строк, сзади от текущего положения указателя буфера. Когда $n = 0$, удаляется часть текущей строки от начала ее до указателя буфера. По умолчанию предполагается значение $n = 1$ и удаляется часть текущей строки от указателя до ее конца. Приказ D(delete) удаления символов также имеет префиксный аргумент n . Его семантика аналогична семантике приказа K, но он оперирует символами, а не строками. По умолчанию принимается значение $n = 1$, а приказ OD оказывается пустым.

Приказы поиска предназначены для отыскания в буфере заданных символьных цепочек. Приказ поиска F(find) с последующей цепочкой, максимальная длина которой зависит от возможностей редактора, инициирует поиск первого появления в буфере заданной цепочки. Поиск начинается от текущего положения указателя буфера и заканчивается при достижении конца буфера или при успешном поиске. В последнем случае указатель буфера размещается за последним символом найденной цепочки и выводится стимулирующий символ, запрашивающий ввод следующего приказа. Рекомендуется проверить правильность поиска с помощью приказа печати, поскольку отыскиваемая цепочка может встретиться в нескольких местах текста. Например, при поиске метки MULT поиск заканчивается успешно, если первой встретится цепочка MULT88. Если при достижении конца буфера поиск оказался безуспешным, редактор выводит стандартное сообщение и возвращает указатель на начало буфера.

Приказ S(substitute) отыскивает заданную в нем символьную цепочку и заменяет ее новой цепочкой, также оп-

ределяемой как аргумент приказа. Отыскиваемая и заменяющая цепочки отделяются друг от друга символом-разделителем. Максимальная длина отыскиваемой цепочки ограничена, а длина заменяющей цепочки обычно не ограничивается. Если заменяющая цепочка отсутствует, редактор отыскивает заданную цепочку и удаляет ее. По завершении приказа S указатель буфера находится за последним символом подставленной цепочки.

Последнюю группу образуют приказы ввода и вывода текста. Приказ ввода A(append) используется для ввода текстового файла, например с перфоленты или гибкого диска, в буфер текста. При его выполнении вводимый текст добавляется к тексту, уже находящемуся в буфере. После инициирования этого приказа текст вводится до удовлетворения одного из следующих условий: воспринимается символ конца файла EOF (этот символ не записывается в буфер текста); буфер текста заполнен; в буфер текста введено очередное число строк текста, зависящее от конкретной реализации редактора; воспринимается символ перевода формата FF (он записывается в буфер текста); достигнуто физическое окончание файла.

Приказ записи W(write) применяется для вывода указанного префиксным аргументом n числа строк из буфера текста в выходной файл, например на гибкий диск или перфоленту. Вывод начинается с начала содержимого буфера текста независимо от текущего положения указателя буфера текста, а оставшийся в буфере текст сдвигается к началу его. Аргумент n независимо от знака считается положительным целым.

С помощью приказа E(exit) окончания редактирования и перехода к монитору осуществляется вывод всего содержимого буфера текста и управление передается монитору операционной системы. После выполнения приказа E содержимое буфера текста сбрасывается в нуль.

В некоторых редакторах для удобства работы пользователя допускаются повторяющиеся выполнения (итерации) приказов. Приказ или последовательность приказов заключаются в угловые скобки, первой из которых предшествует число повторений. Максимальная глубина вложения итераций ограничена некоторым значением, превышение которого ведет к выдаче сообщения об ошибке.

Основная функция программы-ассемблера заключается в последовательном просмотре (сканировании) строк исходной программы и преобразовании каждой из них в один оператор объектной программы. Естественное исключение составляют макровыводы и некоторые директивы ассемблера. Необходимо учитывать также, что во всех современных ЭВМ машинные команды имеют различную длину. Полный просмотр исходной программы называется проходом.

По существу, ассемблирование заключается в замене всех символических имен их двоичными эквивалентами. Для этого ассемблер использует внутреннюю таблицу имен, которая содержит символьные цепочки имен, встречающихся в исходной программе, идентификаторы типов имен и численные значения имен. Иногда необходим указатель для дополнительной информации. Имена, которыми манипулирует ассемблер, приведены в табл. 7-2.

Первые два или три элемента имени представляют собой статическую информацию, не изменяющуюся от одной программы к другой, они определяются разработчиком ассемблера, и их можно хранить в ПЗУ. Остальные элементы соответствуют динамической информации, которую создает сам ассемблер в процессе сканирования исходной программы. Эта информация, естественно, может храниться только в оперативной памяти.

Формированию объектной программы за один проход препятствуют обращения (ссылки) вперед, соответствующи-

Таблица 7-2

Имена, обрабатываемые ассемблером

Тип имени	Значение (состав) имени
Мнемоника команды	Код операции, формат и длина команды
Директива	Адрес подпрограммы, осуществляющей ее обработку
Имя регистра	Целое без знака
Метка	Относительный адрес в программе, целое без знака
Имя, определенное пользователем	Целое без знака
Внешнее обращение	Список появлений в программе

щие командам передачи управления, операнд которых, т. е. метка другой команды, определяется в программе позже, например:

```

***
JMP     NEXT
***
NEXT:   RAL
***

```

В этих ситуациях ассемблер не может сразу определить адрес перехода. Тем не менее, при желании можно реализовать однопроходовой ассемблер.

В первом варианте неопределенная метка помещается в таблицу имен и отмечается адрес, по которому находится команда с обращением вперед. После определения метки ассемблер формирует адрес перехода в команде с обращением вперед, как показано на рис. 7-3. Необходимо специально учесть возможность нескольких обращений вперед к одной и той же метке. Команды с такими обращениями организуются в односвязный список, а при определении метки производится прохождение по списку и подстановка правильного адреса перехода.

Во втором варианте ассемблер строит дополнительную таблицу переходов и все команды с обращением вперед осуществляют передачу управления косвенно, через эту таблицу. Однопроходовой ассемблер обеспечивает некоторый выигрыш в скорости ассемблирования, если только вся объектная программа размещается в основной памяти.

Самое простое решение проблемы обращений вперед заключается в сканировании исходной программы два раза:

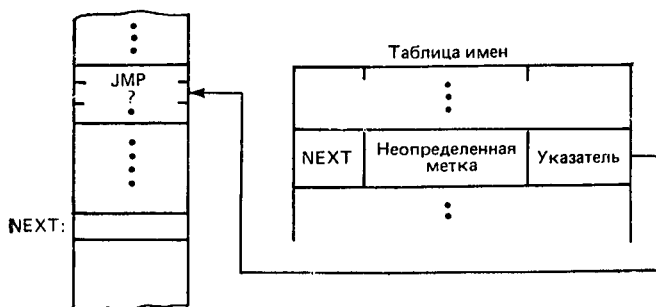


Рис. 7-3. Обработка обращения вперед

первый раз для построения таблицы имен, второй — для фактической трансляции программы и печати листинга. Большинство ассемблеров современных ЭВМ являются двухпроходовыми.

Для хранения исходной программы необходима память значительной емкости. В мини-ЭВМ, а также в микро-ЭВМ с устройством внешней памяти исходная программа оформляется как файл в накопителе на диске или ленте, а для ассемблирования считывается в основную память. Объектная программа помещается во внешнюю память. Если основной памяти для ассемблера, исходной программы и объектной программы недостаточно или внешняя память отсутствует, исходная программа на обоих проходах считывается построчно с перфоленды. Объектная программа на втором проходе выводится на перфоленду.

Целесообразно построить на первом проходе всю таблицу имен, включив в нее метки и определенные пользователем имена. В этом случае перед вторым проходом для производства эффективного поиска ее можно рассортировать, например расположив имена в алфавитном порядке.

Чтобы построить таблицу имен, ассемблер при обработке каждой строки должен правильно модифицировать свою важнейшую внутреннюю переменную — счетчик адреса. Когда строка содержит машинную команду, необходимо учесть ее длину, даже если значения операндов пока неизвестны. Длина команды обычно определяется ее мнемоникой, но иногда приходится проверять режимы адресации. Например, в микропроцессоре КР580 все команды MOV имеют длину один байт, все команды передачи управления (кроме рестарта и возвратов) трехбайтные и т. д.

При обработке директив на первом проходе счетчик адреса может модифицироваться. Например, при обработке директивы DB необходимо произвести инкремент счетчика адреса на число ее операндов. Оно определяется числом запятых в списке поля операнда, причем поля значений операндов могут быть еще неизвестны. С другой стороны, значение операнда директив ORG и DS должно быть определено на первом проходе, так как оно прямо влияет на счетчик адреса.

Некоторые сложности возникают при обработке макрокоманд. Когда на первом проходе встречается директива MACRO, ассемблер переходит в режим макроопределения и помещает имя макрокоманды в таблицу макроопределений. В нее вводится также тело макрокоманды, в котором формальные параметры заменяются специальными иденти-

фикаторами, например $\langle n \rangle$ для n -го параметра. В последующем сканировании программы при обработке поля кода вначале просматривается таблица макроопределений, а затем таблица мнемоник машинных команд и директив. Такой порядок дает возможность пользователю переопределять некоторые машинные команды как макрокоманды. Однако чаще поиск производится в другом порядке: мнемоники команд, директив и имен макрокоманд. Объясняется такой порядок тем, что в программе мнемоники машинных команд встречаются наиболее часто.

Если мнемоника находится в таблице макроопределений, ассемблер переходит в режим макрорасширения. Тело макрокоманды копируется в программу с соответствующей заменой формальных параметров фактическими параметрами макровывода. Когда исходная программа считывается с перфоленты, расширение осуществляется на обоих проходах. Когда же исходная программа хранится во внешней памяти, расширение выполняется только на первом проходе, а второй проход выполняется обычным образом.

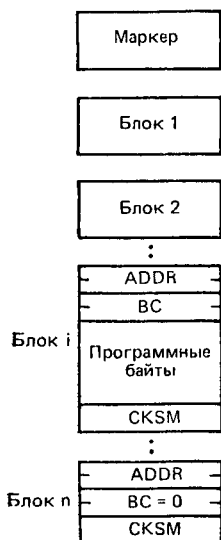
После построения полной таблицы имен второй проход не представляет особых трудностей. При сканировании каждой строки содержимое полей кода и операнда с помощью таблицы имен преобразуется в двоичные эквиваленты и формируется объектная программа, представляющая собой цепочки из единиц и нулей. Отметим, что на втором проходе также необходим счетчик адреса. Двоичное значение директивы END загружается в программный счетчик, и процессор готов к выполнению программы.

На обоих проходах большинство действий ассемблера связано с манипуляциями таблицей имен. Ассемблирование начинается с таблицы имен, содержащей ключевые имена: мнемоники команд и идентификаторы внутренних регистров (иногда последние определяются директивами SET). Когда на первом проходе встречается любое имя, необходимо просмотреть таблицу имен и, если имя в таблице отсутствует, сформировать в ней новую запись. Поиск необходим при определении имени и считывании из таблицы его значения. При определении нового имени приходится просматривать всю таблицу, чтобы зафиксировать многократные определения. Для печати определенных пользователем имен в алфавитном порядке необходимо упорядочивать таблицу имен. Таким образом, важными операциями становятся поиск, введение новых записей и сортировка таблицы имен. Время выполнения этих операций непосредственно влияет на скорость ассемблирования. Одним из эффек-

тивных средств обработки таблиц имен является хеширование. Оно заключается в сравнительно простом преобразовании имени посредством хеш-функции в однозначный индекс таблицы имен.

Ассемблеры классифицируются как абсолютные или перемещающие в зависимости от того, формируют они информацию, позволяющую загружать программу в любую область памяти, или нет. Первый адрес, который занимает программа в памяти, называется адресом загрузки. В абсолютном ассемблере адрес загрузки определяется директивой `ORG` и известен во время ассемблирования. Поэтому при построении адресных констант и команд передачи управления ассемблер использует абсолютные адреса памяти. Следовательно, программа допускает загрузку в одну область памяти или в несколько фиксированных областей.

Сформированная абсолютным ассемблером программа (абсолютный загрузочный модуль) состоит из последовательности записей (блоков), показанной на рис. 7-4. Ей предшествует специальный код (маркер) начала. Поле `ADDR` содержит адрес памяти, по которому загружается первый программный байт блока. Содержимое поля `BC` счетчика байт идентифицирует число программных байт в блоке. Далее находятся собственно программные байты, загружаемые в смежные ячейки памяти. Байт `CKSM` контрольной суммы представляет собой циклическую сумму или



дополнение до нуля циклической суммы всех байт блока, включая `ADDR` и `BC`. Последний блок имеет `BC=0`, а содержимое поля `ADDR` интерпретируется как пусковой адрес программы.

Жесткая фиксация адреса загрузки имеет ряд недостатков и ограничений. При разработке прикладной программы, состоящей из нескольких модулей, нужно знать абсолютные адреса загрузки каждого модуля, а они зависят от размеров модулей, которые неизвестны до окончания разработки. Кроме того, модификация одного модуля приводит к изменению адресов загрузки других модулей. Перемещающий (относительный) ассемблер и тесно связанная с ним про-

Рис. 7-4. Формат выхода абсолютного ассемблера

грамма перемещающего загрузчика дают возможность определять адрес загрузки не во время ассемблирования, а во время загрузки модуля в память. Другими словами, полученный объектный модуль допускает загрузку по любому адресу, т. е. его можно перемещать по всему адресному пространству памяти.

Для реализации перемещающего ассемблера необходимо различать абсолютные и переместимые (перемещаемые) объекты — имена и выражения: значение абсолютного объекта не зависит, а значение переместимого объекта зависит от адреса загрузки. В ассемблер встраивается набор правил, позволяющих ему различать абсолютные и переместимые объекты. Наиболее распространены следующие правила: 1) все численные константы считаются абсолютными; 2) счетчик адреса переместим; 3) имя в поле метки является переместимым и только в директиве EQU определяется типом объекта в поле операнда; 4) выражения, состоящие только из абсолютных имен и констант, считаются абсолютными; 5) разность двух переместимых имен абсолютна; 6) если REL — переместимое имя, а ABS — абсолютная константа, имя или выражение, то выражения REL, REL+ABS, ABS+REL, REL-ABS будут переместимыми.

Чтобы ассемблер мог различать абсолютные и переместимые модули, удобно использовать две различные директивы ORG: директива AORG начинает абсолютный модуль, а директива RORG — переместимый модуль. Встретив директиву RORG, ассемблер строит программу так, как будто она начинается по адресу, равному значению операнда этой директивы. При этом составляется список (таблица перемещения) всех ячеек, содержащих переместимые объекты, так как значения их должны быть скорректированы при загрузке в другую область памяти. Директивы .ASECT и .CSECT языка ассемблера процессоров CM выполняют примерно такие же функции, как и директивы AORG и RORG, но инициализация счетчика адреса производится оператором прямого присваивания. Формат объектного модуля перемещающего ассемблера приведен на рис. 7-5. Каждая запись в таблице перемещения содержит относительное значение переместимого адреса и указатель односвязного списка всех ячеек (операндов команд передачи управления), которые содержат этот адрес. Конец списка идентифицируется нулем.

В процессе загрузки определяется константа, называемая загрузочным смещением (программным смещением), которая прибавляется ко всем переместимым адресам.

ков, выполняющих все более сложные функции.

Наиболее простым является настраивающий загрузчик, предназначенный для загрузки в «пустую» основную память, например после включения машины, более мощного загрузчика с внешнего носителя — перфоленты или диска. Он представляет собой очень короткую программу из одного-двух десятков команд, которые можно ввести с системной консоли при соответствующей инициализации программного счетчика. Настраивающий загрузчик обычно размещается по старшим адресам памяти, и предусматривается аппаратная защита от записи по этим адресам при работе машины. Часто для удобства пользователя он хранится в ПЗУ и вызывается клавишей загрузки (настройки). Некоторые процессоры при включении питания автоматически обращаются к настраивающему загрузчику.

Для загрузки любых программ в произвольные области памяти используется абсолютный загрузчик. На рис. 7-4 был приведен формат выхода абсолютного ассемблера. Абсолютный загрузчик, рассчитанный на данный формат, выделяет необходимую информацию и записывает программные байты по абсолютным адресам памяти. При загрузке осуществляется вычисление контрольной суммы блоков и сравнение ее с содержимым поля CKSM. Расхождение контрольных сумм свидетельствует об ошибках, и операцию загрузки приходится повторять. При восприятии последнего блока с нулевым полем ВС загрузчик помещает в программный счетчик пусковой адрес программы и прекращает свои действия.

Самым гибким по выполняемым функциям является перемещающий загрузчик. Главная функция его заключается в автоматической коррекции переместимых адресов в программе при загрузке ее по фактическому адресу загрузки так, чтобы в новой области памяти программа выполнялась правильно. Загрузчик прибавляет смещение к каждому относительному адресу в таблице перемещения. Затем он «просматривает» список записей в таблице перемещения и записывает перемещенный адрес в каждый элемент списка. Полученный абсолютный загрузочный модуль записывается в память и готов для выполнения.

РЕДАКТОР СВЯЗЕЙ

Функция редактора связей (компоновщика) заключается в образовании одного переместимого загрузочного модуля из нескольких объектных модулей. Необходимость вы-

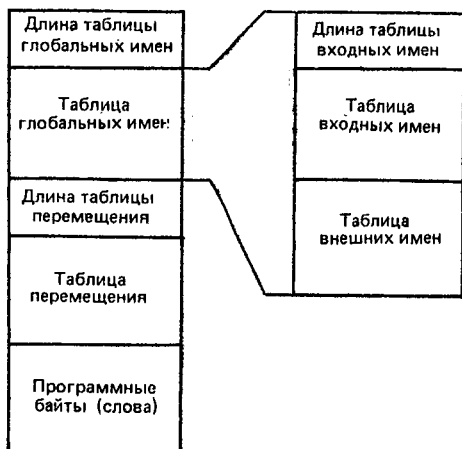


Рис. 7-6. Форматы объектного модуля и таблицы глобальных имен

полнения этой функции объясняется внешними обращениями (ссылками), которые не может обработать транслятор. Под внешним обращением понимается использование как операнда в некотором объектном модуле имени, определенного в другом модуле. Во время трансляции неизвестны ни модуль, содержащий определение имени, ни относительное значение имени в этом модуле. Взаимодействие модулей через абсолютные адреса памяти оказывается совершенно негибким. Например, все подпрограммы, вызываемые в нескольких модулях, потребовалось бы размещать только по абсолютным адресам.

Следовательно, кроме таблицы перемещения, транслятор должен включить в объектный модуль таблицу внешних обращений, или таблицу глобальных имен, что приводит к общему формату объектного модуля, приведенному на рис. 7-6. Таблица содержит описание всех обращений к именам, не определенным в данном модуле. Однако только этой информации для редактора связей недостаточно, необходимо указать еще, где искать определения внешних обращений. Поэтому в таблицу глобальных имен модуля следует ввести информацию о тех именах, к которым возможны обращения из других модулей. Таким образом, таблица глобальных имен содержит имена двух видов: внешние имена, к которым в модуле имеются обращения, но которые в нем не определяются, и входные имена, которые определяются в данном модуле и к которым возможны обращения из других модулей.

Обычно глобальные имена в исходной программе необходимо специально объявлять, чтобы редактор связей выявил ошибки в случае неопределенных имен. Директива EXTERNAL как операнд содержит список внешних имен, а директива ENTRY — список входных имен. В ассемблере процессоров CM единственная директива. GLOBL объявляет имена обоих типов. Имена, определенные в модуле, считаются входными, а остальные — внешними.

Организация таблицы глобальных имен также показана на рис. 7-6. Таблица входных имен содержит все входные имена и их относительные адреса внутри модуля, а таблица внешних имен содержит все внешние имена, с каждым из которых ассоциирован список ячеек, в которых эти имена используются как операнды. Структура таблицы глобальных имен аналогична структуре таблицы перемещения.

Отметим, что внешние имена могут фигурировать в выражениях. Если обозначить EXT — внешнее имя, а ABS — абсолютную константу, имя или выражение, то допускаются следующие формы выражений: EXT, EXT+ABS, ABS+EXT, EXT—ABS.

Редактор связей обрабатывает все объектные модули программы и строит собственные таблицы внешних и входных имен. При этом обнаруживаются ошибки, связанные с неопределенными и многократно определенными внешними именами. Когда редактор связей встречает неопределенное имя, он автоматически обращается к библиотеке стандартных функций и библиотеке пользователя. Все адреса, зависящие от значений внешних имен, соответствующим образом корректируются.

Сформированный редактором связей переместимый объектный модуль загружается в память перемещающим загрузчиком. Иногда функции загрузчика и редактора связей совмещены в одной программе, называемой связывающим загрузчиком. Эта программа обрабатывает внешние обращения, перемещает объектные модули и загружает их в память.

СРЕДСТВА ОТЛАДКИ

Большинство прикладных программ по окончании их разработки содержит логические ошибки, которые выявляются только при выполнении программы. Для их локализации и исправления необходимы специальные отладочные средства. Приходится учитывать, что дать исчерпывающую

проверку сложной программы почти невозможно и даже при обнаружении неправильной работы программы для локализации ошибок потребуется несколько приемов.

Наиболее простой прием заключается в распечатке содержимого всех ячеек памяти, которые использовались в программе (дамп памяти). Дамп представляет собой отображение содержимого памяти после выполнения программы и помогает найти ошибки, которые повлияли на окончательное содержимое памяти. Чтобы обнаружить переходные ошибки, потребуется осуществлять дампы памяти несколько раз. Этот прием довольно громоздок, так как разбираться в длинных цепочках чисел затруднительно.

В программу можно ввести команды печати промежуточных результатов. При удачном выборе места включения команд печати можно найти источник ошибок, но печатать все промежуточные результаты обычно невозможно.

Еще один достаточно надежный, но времязатратный способ заключается в выполнении программы по командам и анализе содержимого регистров процессора и ячеек памяти. В большинстве ЭВМ предусматривается режим пошагового (или пошагового) выполнения программы. Чтобы сократить время отладки, в программу вводятся остановки после выполнения сравнительно крупных сегментов с последующим анализом важных данных. В режиме прослеживания (трассировки) программы, который задается специальным битом Т в слове состояния процессора, машина автоматически печатает команды и данные по мере выполнения программы. Когда бит Т установлен, процессор инициирует программное прерывание после выполнения каждой команды.

Наиболее удобным и мощным средством является диалоговая (интерактивная) программа-отладчик, рассчитанная на взаимодействие ЭВМ с пользователем через консоль или терминал посредством содержательных приказов. Ниже приведены типичные приказы диалогового отладчика;

% n — обратиться к регистру n для просмотра или изменения его содержимого; как реакция на этот приказ, печатается содержимое регистра, после чего пользователь может ввести новые данные;

n/ — обратиться к ячейке памяти с адресом n (этот приказ аналогичен предыдущему);

addr; B — установить контрольную точку по адресу addr; допускается задание до 8 контрольных точек;

; B — сбросить все установленные ранее контрольные точки;

addr; G — начать выполнение программы с адреса addr;

addr; P — продолжить выполнение после останова по адресу addr контрольной точки.

Удобным приемом отладки служат контрольные точки. Пользователь задает список адресов, и когда выполняется команда по одному из адресов, управление возвращается отладчику.

В символических отладчиках сохраняется копия таблицы имен, построенной ассемблером, что дает возможность пользователю определять адреса памяти с помощью символических имен из ассемблерной программы, а не абсолютных чисел.

Для отладки прикладных программ микро-ЭВМ в кросс-системах широко применяются моделирующие (имитирующие) программы, обычно рассчитанные на диалоговый режим и реже на пакетную обработку. Моделирующая программа отражает архитектуру моделируемой (целевой) микро-ЭВМ и алгоритмы выполнения всех ее команд. После загрузки в моделирующую ЭВМ объектной программы целевой микро-ЭВМ начинается выполнение программы так, как будто она действительно выполняется на целевой микро-ЭВМ. При этом у пользователя имеются возможности наблюдать содержимое регистров микро-ЭВМ, производить дампы памяти, проследживать выполнение программы по командам. Кроме того, моделирующая программа накапливает и выводит такую информацию, как время выполнения программы (в тактах синхронизации), листинг выполненных команд с числом их выполнений и др.

Пользователь взаимодействует с моделирующей программой посредством содержательных приказов, вводимых с терминала. Ниже приведены некоторые из допустимых приказов:

START addr (начать) — загрузить в программный счетчик целевой микро-ЭВМ значение addr и начать выполнение программы;

STOP addr (остановить) — прекратить моделирование при достижении в программном счетчике целевой микро-ЭВМ значения addr и вывести состояние программы (число машинных циклов, последние команда и адрес данных, содержимое внутренних регистров микропроцессора);

DUMP addr 1, addr 2, addr 3 (дамп) — напечатать содержимое ячеек памяти от addr 2 до addr 3, когда выполняется команда по адресу addr 1;

PATCH addr, data (загрузить) — загрузить в ячейку памяти с адресом addr данные data;

SET ROM addr 1, addr 2 (определить ПЗУ) — область памяти от addr 1 до addr 2 считать ПЗУ и при попытке записи в нее выдать сообщение об ошибке.

7-4. МОНИТОР

Рассмотренные системные программы предназначены для эффективного использования ЭВМ, а прикладные программы выполняют преобразования данных для пользователя. В большинстве машин для управления самими системными программами имеется программа-монитор, являющаяся управляющим элементом операционной системы и распределяющая программные ресурсы. Монитор осуществляет интерфейс с пользователем в диалоговом режиме, воспринимая и выполняя приказы (команды) с системной консоли — пишущей машинки или видеотерминала. В зависимости от конфигурации аппаратных средств функции монитора варьируются от интерпретации десятка простых приказов, вводимых с консоли, до управления в реальном времени разнообразными периферийными устройствами. В мультипрограммной ЭВМ монитор обслуживает нескольких пользователей и распределяет ресурсы между несколькими программами, которые одновременно находятся в основной памяти. Типичные функции монитора: сканирование клавиатуры консоли, идентификация приказов, назначение периферийных устройств (преобразование логических адресов в физические), управление другими системными программами, поддержание реального времени, управление передачами данных между основной и внешней памятью. Довольно трудно провести четкую границу между монитором и супервизором, но в целом основное назначение супервизора — обрабатывать внутренние (программные) и внешние прерывания.

В зависимости от возможностей машины монитор может быть рассчитан на выполнение одного задания (прикладной программы) или на работу с передним (задним) планом. В первом случае монитор управляет только одной программой, которая выполняется до завершения или до прерывания с консоли. Монитор переднего (заднего) плана управляет двумя независимыми программами, которые одновременно находятся в основной памяти. Одна из них, называемая программой переднего плана, имеет приоритет и

выполняется до передачи управления другой программе — программе заднего плана (фоновой). Последняя выполняется до тех пор, пока не потребуется возобновить выполнение программы переднего плана. Ниже рассматривается простой монитор мини-ЭВМ, рассчитанный на управление одним заданием.

Пользователь вводит с консоли приказы двух типов. Каждый приказ заканчивается возвратом каретки. Приказы первого типа (их можно назвать клавишными приказами) предназначены для управления операционной системой. Они инициируются вводом коротких зарезервированных мнемоник, а также нажатием управляющих клавиш. Ниже даны примеры мнемонических клавишных приказов: ASS (assign) — ассоциирование логических имен с периферийными устройствами; SET — изменение параметров системы; B (base) — задание базового адреса; E (examine) — индикация (вывод) содержимого ячейки памяти; D (deposit) — запоминание значения в ячейке памяти; GE (get) — загрузка в основную память загрузочного модуля; ST (start) — передача управления прикладной программе по заданному адресу; RUN эквивалентен действию приказов GE и ST; R (run) эквивалентен приказу RUN для системной программы; RE (reenter) — повторный запуск (рестарт) загруженной программы; DAT (date) — ввод указанной даты или вывод текущей даты; TIM (time) — ввод указанного времени или вывод текущего времени.

Как правило, в микро-ЭВМ применяются однобуквенные мнемоники аналогичных или несколько отличающихся приказов.

Приказы, вводимые с помощью управляющих клавиш, выполняют следующие функции: прерывание текущей программы и возврат управления монитору; запрещение вывода на консоль; возобновление вывода на консоль; приостановка вывода на консоль до задания следующего приказа; вывод следующей страницы; удаление текущей входной строки; аннулирование командной строки; конец файла.

Приказы второго типа, иногда называемые командными строками, представляют собой запросы к операционной системе на инициализацию входных и выходных файлов для последующих операций считывания, записи или обеих операций. Эти запросы формируются в процессе выполнения программы оператором с консоли либо самой программой, находящейся в основной памяти. Запросы оператора относятся к классу передач файлов из одного места в другое и имеют следующий общий формат:

<устройство>:<выход>/<список переключателей>:=

<устройство> : <вход> / <список переключателей>

Здесь слева направо указываются логическое имя выходного устройства (получателя), имя выходного файла, выходные параметры, логическое имя входного устройства (источника), имя входного файла и входные параметры. Например, после ассемблирования прикладной программы и обнаружения в ней ошибок оператор может запросить вывод на консоль необходимой информации, задавая любые из следующих входных параметров: С — печать листинга; S — печать таблицы имен, определенных пользователем; R — использованные регистры и присвоенные им имена; M — вывод макрокоманд; P — вывод постоянных символических имен (констант); E — печать кодов ошибок.

Программные запросы к монитору включают в себя операции с файлами, передачи данных (ввод-вывод) и служебные функции. Примеры таких запросов: .DELETE — удаление файла; .READ — инициирование операции считывания файла; .WRITE — инициирование операции записи файла; TTY — передача одного символа из буфера консоли в определенный регистр процессора; .TTYOUT — передача одного символа в противоположном направлении; .EXIT — возврат управления монитору; .WAIT — ожидание завершения операции ввода-вывода в указанном канале.

Каждый запрос сопровождается необходимыми параметрами.

Через монитор, как указывалось выше, вызываются также системные программы: макроассемблер, трансляторы с языков высокого уровня, редактор текста и др. Каждая из них имеет однозначную мнемонику.

В соответствии с рассмотренными функциями в составе монитора имеются следующие компоненты: резидентный монитор, монитор клавиатуры, система управления файлами, интерпретатор командных строк и набор подпрограмм управления периферийными устройствами (драйверов).

Резидентный монитор постоянно находится в основной памяти (обычно по старшим адресам) и организует интерфейс с консолью, обрабатывает внутренние прерывания и управляет системной внешней памятью (системным диском), содержащей другие программы операционной системы. В его составе имеются также таблицы, определяющие конфигурацию системы.

Монитор клавиатуры выполняет функции взаимодействия с оператором, главная из которых заключается в интерпретации рассмотренных выше клавишных приказов.

Система управления файлами представляет собой важную часть монитора, поскольку функции всех операционных систем связаны с обработкой файлов. Файл — это физическое представление однородной информации, описывающей набор объектов и имеющей определенное назначение. В большинстве современных ЭВМ файл выступает как основная единица долговременного хранения информации. Он состоит из отдельных записей, содержащих информацию об одном объекте. Пользователь определяет имя файла, которое ассоциируется с действительной идентификацией файла при помощи оглавления, и логическую организацию файла, т. е. формат логических записей. Во внешней памяти хранятся физические файлы, представляющие собой совокупность физических записей, например секторов гибкого диска. В большинстве случаев размеры логических и физических записей не совпадают.

По способу доступа имеются две основные организации файлов. В последовательном файле к физической, а следовательно, и к логической записи можно обратиться, последовательно сканируя файл, т. е. конкретная запись идентифицируется только относительным положением в файле. В индексном файле положение логической записи определяется специальным полем ключа и для доступа к конкретной записи необходимо задать значение ключа.

По виду информации различают программные файлы и файлы данных, а по форме представления информации — текстовые (символьные) и двоичные файлы. Пример текстового файла — исходная программа на языке ассемблера, пример двоичного файла — объектный модуль.

Имя файла обычно состоит из двух полей, разделяемых точкой. В первом из них содержится собственно имя, определяемое пользователем, а второе поле, содержащее до трех символов и называемое расширением, идентифицирует тип информации. Расширение используется для организации передач и форматирования выхода. Примеры расширений: .OBJ — объектная программа, .ASY — программа на ассемблере и т. п.

Для удобства некоторых манипуляций файлами в их именах допускаются специальные символы. Звездочка заменяет любые символы в имени и расширении, например ABC.* идентифицирует все файлы с именем ABC и любым расширением. Вопросительный знак заменяет отдельный

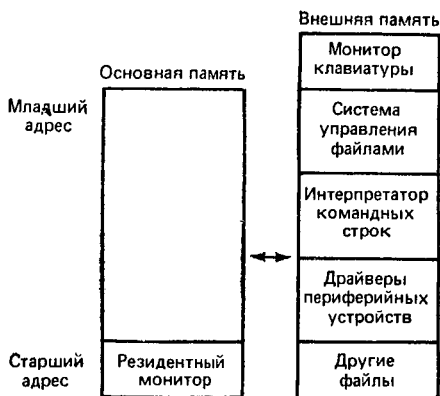


Рис. 7-7. Программы монитора

символ в имени файла, например имя PROG?. OBJ определяет все файлы объектных программ с пятисимвольными именами, начинающимися с четырех символов PROG.

В системе управления файлами имеется оглавление, представляющее собой таблицу (файл) с информацией о текущих файлах. Каждая запись в таблице содержит имя файла, дату его создания или последней модификации, длину и физический адрес.

Приказы манипуляций файлами расшифровывает интерпретатор командных строк. По существу, интерпретатор является частью системы управления файлами, но его могут вызывать прикладные программы и сама система управления файлами.

В резидентном мониторе имеется абсолютный загрузчик, предназначенный для загрузки в основную память остальных программ монитора. Они хранятся во внешней памяти в формате абсолютных загрузочных модулей (рис. 7-7). При включении машины резидентный монитор автоматически загружается в основную память, а затем сам загружает систему управления файлами и монитор клавиатуры по соседним меньшим адресам. Готовность резидентного монитора к работе идентифицируется выводом на консоль стимулирующего символа, например точки.

Приказ с консоли может инициировать загрузку из внешней памяти прикладной программы. Загрузчик помещает ее в оставшуюся область памяти, начинающуюся с младших адресов (область пользователя). При этом про-

грамма может перезаписать монитор клавиатуры и систему управления файлами, но резидентный монитор модифицировать нельзя. В этом случае запросы оператора или программы обрабатывает резидентный монитор. Когда же для обслуживания запроса требуется система управления файлами или монитор клавиатуры, часть программы передается во внешнюю память, а в освободившуюся область основной памяти загружается требуемый компонент монитора.

При разработке прикладной программы пользователь выполняет действия в такой последовательности: 1) ввод текста исходной программы с помощью редактора текста; 2) трансляция программы посредством вызова необходимого транслятора; 3) редактирование связей; 4) загрузка и выполнение модуля; 5) отладка программы путем связи прикладной программы с программой-отладчиком и выполнения полученного загрузочного модуля.

После загрузки любой системной программы приказом R в основную память монитор запрашивает командную строку, выводя на консоль стимулирующий символ, например звездочку (*). Оператор может вводить командную строку с необходимыми параметрами.

7.5. ОРГАНИЗАЦИЯ ПОДПРОГРАММ

Во всех прикладных программах, кроме самых простых, одни и те же операции, но с различными данными должны выполняться несколько раз. Примерами могут служить операции умножения чисел в машинах, не имеющих команд умножения, сортировки массива, ввода символа с терминала и т. п. Каждая из таких операций реализуется последовательностями команд, образующих почти автономные программы. Целесообразно эти «программы в программе» включать в исходную и объектную программы только один раз и в нужных местах вызывать их для выполнения с текущими данными. Такие программы называются подпрограммами или процедурами.

Подпрограмма представляет собой сравнительно автономный программный модуль, который может «приводиться в действие» любое число раз. Широкое использование подпрограмм имеет большое значение в силу следующих причин: сложную программу можно разделить на более простые и управляемые модули (подпрограммы), которые могут параллельно разрабатываться несколькими программистами; каждая подпрограмма допускает автономную отладку, что ускоряет и упрощает отладку всей программы;

подпрограммы экономят программную память; отложенные и часто используемые подпрограммы организуются в библиотеки.

Вызов подпрограмм и возврат из подпрограмм. Иницирование подпрограммы называется вызовом подпрограммы или переходом к подпрограмме. Эту функцию может выполнить команда перехода, целевым адресом которой, загружаемым в программный счетчик, является начальный адрес (т. е. адрес первой команды) подпрограммы, называемый также точкой входа. Однако в отличие от обычного перехода необходим некоторый механизм, обеспечивающий поле завершения подпрограммы продолжения программы с точки зрения вызова. В ЭВМ применяется несколько таких механизмов.

В первых мини-ЭВМ передача управления подпрограмме осуществлялась командой перехода JMS SUBR, где SUBR — метка точки входа. Эта команда выполняет переход (PC←SUBR) и записывает адрес возврата (т. е. адрес команды, следующей за командой JMS SUBR) в первую ячейку подпрограммы. При завершении подпрограммы ее последняя команда косвенного перехода JMPISUBR возвращает управление в точку вызова путем перехода косвенно, через первую ячейку подпрограммы. При этом способе необходима только одна специальная команда JMS и автоматически обеспечивается любая глубина вложения подпрограмм, т. е. вызов одних подпрограмм другими. Однако здесь имеются и недостатки: невозможно прервать и повторно вызвать ту же самую подпрограмму, а кроме того, подпрограммы должны находиться в оперативной памяти.

В современных ЭВМ для организации подпрограмм предусматриваются две специальные команды. Команда вызова CALL или перехода к подпрограмме JSR временно запоминает адрес возврата и загружает в PC начальный адрес подпрограммы. Последняя команда подпрограммы, называемая возвратом из подпрограммы с мнемониками RET или RTS загружает в PC запомненный адрес возврата. Команды вызовов и возвратов могут быть условными, когда выполнение соответствующего действия зависит от удовлетворения некоторого условия (состояния флажка), указанного в коде операции.

Связь вызывающей программы с подпрограммой осуществляется через стек. В микропроцессоре КР580 команда вызова CALL SUBR записывает адрес возврата в стек и загружает в PC начальный адрес SUBR подпрограммы, а

команда возврата RET извлекает из стека адрес возврата и загружает его в PC.

В процессорах CM реализована более сложная организация подпрограмм, с использованием регистра связи reg, который определяется пользователем для запоминания адреса возврата. При выполнении команды перехода к подпрограмме JSR reg, SUBR содержимое регистра связи reg включается в аппаратный стек, в регистре связи запоминается адрес возврата (содержимое PC после выборки из памяти команды JSR), а в PC загружается начальный адрес подпрограммы. Важно подчеркнуть, что эффективный адрес SUBR, определяемый в любом режиме адресации, находится первым и временно запоминается в программно-недоступном регистре процессора.

С подпрограммами связаны некоторые проблемы в использовании внутренних регистров процессора. Необходимо обеспечить, чтобы подпрограмма ни в одном регистре не изменяла содержимого, которое после возврата потребуется вызывающей программе. Следовательно, содержимое регистров, используемых подпрограммой, требуется временно запоминать, а по окончании подпрограммы — восстанавливать. В качестве такого временного хранилища содержимого регистров удобно использовать стек. Служебные действия по запоминанию и восстановлению содержимого регистров можно включить в вызывающую программу или в подпрограмму. Практика программирования показывает, что второй способ предпочтителен: экономится программная память и «регуляризируется» структура подпрограмм.

Для каждой подпрограммы необходимо разработать четкие спецификации, в которых указывается, где находятся данные, обрабатываемые подпрограммой, где будут размещены результаты и какие регистры используются подпрограммой. При программировании на языке ассемблера такие спецификации вместе с коротким описанием функции подпрограммы выносятся в строки комментариев, образующие заголовки (пролог) подпрограммы.

Передача параметров. При организации подпрограмм необходимо каким-либо образом сообщить им о размещении обрабатываемых данных (входные параметры) и формируемых результатах (выходные параметры). Эти процедуры называются передачей параметров. Параметры представляют собой фиктивные переменные в определении подпрограммы, которые связываются с реальными переменными или значениями при каждом вызове подпрограммы. Например, при вызове подпрограммы VECT вычисления

длины вектора необходимо передать ей координаты X и Y, а подпрограмма должна возвратить длину вектора LENGTH. В языках высокого уровня эта подпрограмма определяется в виде VECT (X, Y, LENGTH). Фиктивные переменные в определении подпрограммы называются формальными параметрами, а переменные или значения, заменяющие их при каждом вызове, называются фактическими (действительными) переменными. Например, можно вызвать подпрограмму CALL VECT (N1, N2, RES) с соответствующими фактическими параметрами. Разработано несколько способов передачи параметров, основные из которых рассматриваются ниже.

Наиболее просто передавать параметры в фиксированные ячейки памяти, к которым могут обращаться вызывающая программа и подпрограмма. Вызывающая программа записывает в эти ячейки текущие значения параметров, а подпрограмма считывает их оттуда, обрабатывает и помещает результаты в другие фиксированные ячейки памяти, в которых они становятся доступными вызывающей программе. Передача двух параметров в ячейках PAR1 и PAR2 подпрограмме SUBR в процессорах CM имеет вид:

Метка	Код	Операнд	Комментарий
	MOV	VAL1, PAR1	; Загрузка параметров в
			; ячейки PAR1 и
	MOV	VAL2, PAR2	; PAR2
	JSR	R7, SUBR	; Вызов подпрограммы

Данный способ требует резервирования ячеек памяти, предназначенных только для передачи параметров.

Чтобы упростить программирование и уменьшить объем памяти, можно передавать параметры во внутренних регистрах процессора. Например, в микропроцессоре KP580 перед вызовом подпрограммы умножения множимое загружается в регистр C, а множитель — в аккумулятор A. После вызова подпрограмма выполняет необходимые операции и формирует результат в регистрах H, L. В процессорах CM передача двух параметров в регистры R1 и R2 имеет следующую форму:

Метка	Код	Операнд	Комментарий
	MOV	VAL1, R1	; Загрузка значений
			; параметров в ре-
			; гистры R1 и R2
	MOV	VAL2, R2	; Вызов подпрограм-
	JSR	R7, SUBR	; мы

Недостаток данного способа заключается в естественных ограничениях на число передаваемых параметров. Распределение регистров для параметров обычно приводится в заголовке — комментарии подпрограммы.

Передачу параметров в фиксированной области памяти с начальным (базовым) адресом TBLPR для микропроцессора КР580 иллюстрирует следующий фрагмент:

Метка	Код	Операнд	Комментарий
	LXI	H, TBLPR	; Загрузка адреса об-
	CALL	SUBR	; ласти параметров
	* * *		
SUBR:	MOV	A, M	; Считывание первого
			; параметра
	INX	H	; Инкремент указателя
	MOV	B, M	; Считывание второго
			; параметра
	* * *		
	RET		; Возврат

В заключение остановимся на одном интересном моменте. В простых контроллерах на базе микро-ЭВМ оперативная память может отсутствовать и, следовательно, использовать команды вызова и возврата невозможно. Но для экономии программного ПЗУ все-таки целесообразно по возможности применять подпрограммы. В такой ситуации для перехода к подпрограмме остается только команда безусловного перехода JMP (или BR), а адрес возврата приходится запоминать во внутреннем регистре процессора. Подпрограмма не должна модифицировать этот регистр, а последняя ее команда должна передавать содержимое регистра в программный счетчик. Следующий фрагмент иллюстрирует данный прием для микропроцессора КР580:

Метка	Код	Операнд	Комментарий
	* * *		
	LXI	H, ADDR1	; Первый вызов под-
	JMP	SUBR	; программы
ADDR1:	* * *		
	* * *		
	LXI	H, ADDR2	; Второй вызов под-
	JMP	SUBR	; программы
ADDR2:	* * *		
	* * *		
SUBR:	* * *		; Начало подпрограм-
	* * *		; мы
	PCHL		; Возврат

Специальные разновидности подпрограмм. Для вызывающей программы и подпрограммы характерно отношение подчиненности «ведущий — ведомый»: основная программа вызывает подпрограмму с фиксированной точкой входа; подпрограмма полностью обрабатывает переданные ей параметры и оставляет результаты в определенных ячейках; управление возвращается команде, следующей за командой вызова.

В языках высокого уровня выдерживается такое строгое отношение подчиненности, что после возврата из подпрограммы значения ее локальных переменных не определены.

Обобщением подпрограмм являются так называемые сопрограммы, при организации которых в качестве основной программы попеременно выступают две или более программ, разделяющих общие ресурсы ЭВМ. Каждая сопрограмма считает другую сопрограмму подчиненной (подпрограммой) и имеет свою контекстную среду: стек, регистры, память. Вначале управление ЭВМ осуществляет, например, сопрограмма 1, а в некоторой точке управление передается сопрограмме 2, которая начинает играть роль основной программы. С точки зрения сопрограммы 1 управление передается подчиненной программе — сопрограмме 2, но с точки зрения сопрограммы 2 происходит возврат управления из подчиненной программы — сопрограммы 1. Ситуация изменяется на противоположную, когда сопрограмма 2 передает управление сопрограмме 1. Другими словами, взаимодействие сопрограмм оказывается симметричным и напоминает диалог. Важное отличие сопрограмм от подпрограмм заключается в том, что сопрограммы сохраняют значения своих промежуточных переменных между последовательными передачами управления. Передача управления от одной сопрограммы другой инициируется оператором `RESUME` (возобновить).

Сопрограммы часто применяются для обработки потока данных, включающей в себя ввод, некоторые преобразования и формирование выходов.

Другое применение сопрограмм связано с логической проверкой прикладных программ в системах проектирования. Для этого создается специальная тест-программа и взаимодействие с ней прикладной программы организуется по принципу сопрограмм. Во всех точках, где прикладная программа осуществляет ввод или вывод, управление

передается тест-программе. Она воспринимает и контролирует выходные данные и моделирует входные данные. Особенно удобно реализовать такое взаимодействие при использовании ввода-вывода, отображенного на память.

При программировании сопрограмм на языке ассемблера для каждой из них необходимо временно запоминать адрес возобновления. Наиболее просто эта задача решается в процессорах СМ с помощью команды JSR со специальным режимом адресации JSR R7, @(SP) +. При ее выполнении содержимое вершины стека (т.е. адрес возобновления другой сопрограммы) передается в программно-недоступный регистр TEMP процессора, затем содержимое программного счетчика (т.е. адрес возобновления текущей сопрограммы) выключается в стек и, наконец, содержимое регистра TEMP загружается в программный счетчик. Другими словами, эта команда осуществляет обмен содержимого вершины стека и программного счетчика, что эквивалентно передаче управления от одной сопрограммы другой. Необходимые действия по передаче параметров и временному запоминанию содержимого регистров процессора в сопрограммах выполняются так же, как в подпрограммах. Для правильной работы сопрограмм необходимо запомнить адрес возобновляемой сопрограммы в стеке той сопрограммы, которая начинает выполняться первой.

В микропроцессоре КР580 взаимодействие сопрограмм приходится реализовать с помощью интерфейсных подпрограмм. Ниже приведена такая подпрограмма TRF12, передающая управление от сопрограммы 1 сопрограмме 2. Для возобновления сопрограммы 2 выполняется команда CALL TRF12 с точкой входа интерфейсной подпрограммы. При этом адрес возобновления включается в стек сопрограммы 1. Подпрограмма TRF12 запоминает в этом же стеке содержимое всех внутренних регистров микропроцессора. После этого содержимое SP сопрограммы 1 запоминается в ячейке с адресом STACK1 и в SP загружается указатель стека сопрограммы 2 из ячейки STACK2. Команды POP восстанавливают содержимое всех регистров микропроцессора, относящееся к сопрограмме 2. Заключительная команда RET передает управление сопрограмме 2 по адресу возобновления, ранее запомненному в ее стеке:

Метка	Код	Операнд	Комментарий
TRF42;	PUSH	PSW	; Запоминание в стеке
	PUSH	B	; сопрограммы 1 со-
	PUSH	D	; держимого всех ре-
	PUSH	H	; гистров
	LXI	H, 0	; Запоминание содержи-
	DAD	SP	; мого SP сопрограм-
	SHLD	STACK1	; мы 1
	LHLD	STACK2	; Инициализация SP со-
	SPHL		; программы 2
	POP	H	; Восстановление из сте-
	POP	D	; ка сопрограммы 2 со-
	POP	B	; держимого всех ре-
	POP	PSW	; гистров
	RET		; Передача управления
			; сопрограмме 2

Последующее возобновление сопрограммы 1 осуществляется командой CALL TRF21 с точкой входа аналогичной интерфейсной подпрограммы, в которой изменены операнды только двух команд: SHLD STACK2 и LHLD STACK1.

7-6. РАСПИСАНИЕ ПРОЦЕССОВ В МУЛЬТИПРОГРАММНЫХ ЭВМ

Значительное увеличение емкости и быстродействия памяти, а также быстродействия процессоров привели к реализации в ЭВМ мультипрограммного режима работы. В основной памяти одновременно находится несколько программ, из которых в любой момент времени в однопроцессорной ЭВМ выполняется одна, и предусматриваются средства переключения процессора с одной программы на другую. Основная цель мультипрограммирования заключается в том, чтобы разделить ресурсы системы между несколькими программами (пользователями) и повысить ее пропускную способность. Если, например, одна программа инициирует ввод-вывод и ожидает реакции сравнительно медленнодействующего устройства ввода-вывода, процессор переключается на другую программу. С мультипрограммированием связано значительное число проблем, среди которых отметим планирование выполнения программ, распределение и защиту основной памяти, обеспечение бесконфликтного ввода-вывода, управление обменом данными между основной и массовой памятью и др. Пока большинство этих проблем решается программ-

ными средствами, т.е. с помощью операционной системы, но со временем акцент будет сделан на аппаратную реализацию системных функций.

Рассмотрим простой планировщик задач, реализованный на базе микропроцессора КР 580. На его примере иллюстрируются проблемы, возникающие при разработке программного обеспечения управляющих систем реального времени, в которых широкое применение находят микро-ЭВМ.

В управляющих системах реального времени многие события происходят одновременно и система должна обеспечивать видимую одновременность реакций на эти события, тем более, что на реакции отводится ограниченное время. Однопроцессорная ЭВМ не может физически одновременно выполнять несколько программ, и иллюзию одновременности должны обеспечивать ее программные средства при условии, что процессор обладает достаточной пропускной способностью.

Каждую из функций, реализуемых системой, удобно представить в виде автономной программы или задачи, выполняемой независимо от других задач. Действия, выполняемые в системе по распределению времени для решения задач процессором, называются планированием задач.

Задачи могут находиться в одном из трех состояний: выполнения, когда процессор выполняет задачу; готовности к выполнению при выделении времени процессора; приостановки (ожидания, блокировки), когда задача должна ожидать некоторого внешнего события или когда истек отведенный ей интервал времени процессора.

Когда одна задача переходит в состояние ожидания, процессор переключается на выполнение другой задачи.

Простейшее циклическое планирование предоставляет время процессора каждой задаче по очереди до достижения естественной точки приостановки. Однако в большинстве систем возникает вопрос о приоритетах задач, т.е. выполняемая задача может быть прервана для выполнения другой задачи, которая в силу более жестких временных ограничений имеет больший приоритет. Для учета приоритетов либо все задачи упорядочиваются в соответствии с фиксированными приоритетами, либо используются периодические прерывания от таймера. Во втором варианте при восприятии прерывания таймера начинается выполняться задача с высшим приоритетом, а затем последовательно выполняются остальные задачи

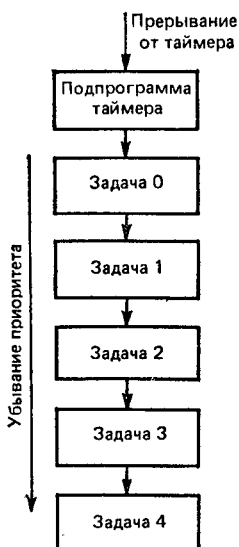


Рис. 7-8. Планирование задач по прерываниям от таймера

(рис. 7-8). Задачей 0 может быть сканирование клавиатуры, задачей 1 — модификация текущего времени и т. д. Каждой задаче должно отводиться время процессора, достаточное для удовлетворения ее временных ограничений. Задача с наименьшим приоритетом всегда должна находиться в состоянии выполнения либо готовности к выполнению.

Период таймера выбирается из общих временных характеристик в системе, например частоты опроса периферийных устройств в задаче с максимальным приоритетом. Если сканирование клавиатуры должно производиться с интервалом 10 мс, частота прерываний таймера составит 100 Гц.

Задачи (их еще называют процессами) можно рассматривать как динамическое выполнение программы. По мере выполнения задача создает данные, которые становятся ее неотъемлемой частью, и манипулирует ими. В процессорах со стековой ориентацией для временного хранения адресов возврата и содержимого внутренних регистров используется стек. Поскольку задачи автономны, у каждой должен быть свой стек, в котором запоминается содержимое регистров в случае прерывания или приостановки задачи. В специальной ячейке запоминается значение ее указателя стека. Таким образом, задача ассоциируется со структурой данных, показанной на рис. 7-9. Основным элементом ее является запомненное значение указателя стека SP, которое адресует сохраненное содержимое регистров и программного счетчика. Продолжение или инициирование выполнения задачи процессором осуществляется путем восстановления содержимого указателя стека, регистров и программного счетчика.

При фиксированном числе задач, что характерно для многих применений микро-ЭВМ, структура данных для планирования задач оказывается простой. В таблице SP5AV запоминается содержимое указателя стека каждой задачи, а в ячейки RUNNR находится номер выполняемой задачи. Для удобства манипуляций адресами номер

задачи представлен 16-битным числом.

Проводимая ниже подпрограмма RESRV используется для запоминания содержимого всех регистров и указателя стека в случае прерывания или приостановки:

Метка	Код	Операнд	Комментарий
RESRV:	XTHL		; Содержимое H, L в стеке
	PUSH	PSW	; ке
	PUSH	B	; Запоминание содержимого регистров про-
	PUSH	D	; цессора
	PUSH	H	; Адрес возврата в стеке
	LHLD	RUNNR	; Вычисление индекса в
	LXI	D, SPSAV	таблице SPSAV
	XCHG		
	DAD	D	
	DAD	D	
	XCHG		; Индекс в регистрах D,
			; E
	LXI	H, 0	; Передача (SP) в (H, L)
	DAD	SP	
	INX	H	; Компенсация адреса
	INX	H	; возврата
	XCHG		; Передача (SP) в (D,
			; E)
	MOV	M, E	; Запоминание (SP) в
	INX	H	; таблице SPSAV
	MOV	M, D	
	RET		; Возврат

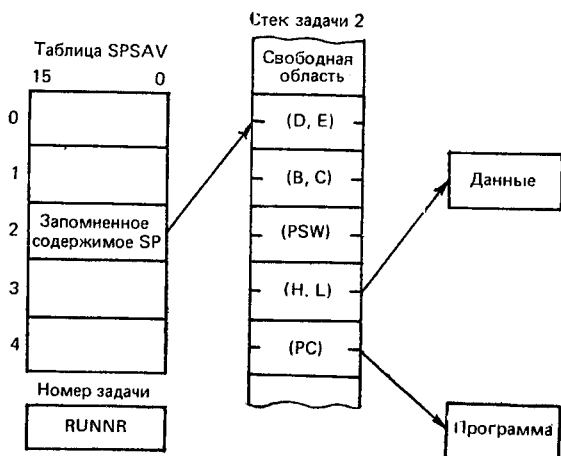


Рис. 7-9. Структура данных задачи (задача 2)

Вначале команда XTHL передает адрес возврата в регистры H, L, а текущее содержимое этих регистров запоминается в стеке. Затем тремя командами PUSH в стеке запоминается содержимое регистров процессора, а последняя команда PUSH H включает в стек адрес возврата. В остальной части программы вычисляется индекс в таблице SPSAV для выполнявшейся задачи, и в соответствующем элементе таблицы запоминается содержимое указателя стека.

Подпрограмма SUSP используется для приостановки одной задачи и инициирования или продолжения задачи со следующим приоритетом. Она вызывается из одной задачи, а в результате производимого ею контекстного переключения управление возвращается по адресу возврата другой задачи. Подпрограмма выполняется с запрещенными прерываниями, чтобы не нарушалось контекстное переключение процессора:

Метка	Код	Операнд	Комментарий
SUSP:	DI		; Запрещение прерываний
	CALL	RESRV	; Сохранение состояния задачи
	LHLD	RUNNR	; Продвижение номера задачи
	INX	H	; задачи
	SHLD	RUNNR	
	LXI	D, SPSAV	; Вычисление индекса в таблице SPSAV
	XCHG		
	DAD	D	
	DAD	D	
	MOV	E, M	; Указатель стека в D,
NEXT:	INX	H	; E
	MOV	D, M	
	XCHG		
	SPHL		; Восстановление указателя стека
	POP	D	; Восстановление содержимого регистров
	POP	B	; процессора
	POP	PSW	
	POP	H	
	EI		; Разрешение прерываний
	RET		; Возврат к следующей задаче

Подпрограмма TIMER обработки прерываний таймера аналогично по структуре подпрограмме SUSP, но иницирует выполнение задачи с максимальным приоритетом (задача 0).

Метка	Код	Операнд	Комментарий
TIMER:	CALL	RESRV	; Сохранение состояния задачи
START:	LXI	H, 0	; Будет выполняться за-
	SHLD	RUNNR	; дача 0
	LXI	H, SPSAV	; Начальный адрес таб-
			; лицы SPSAV
	JMP	NEXT	; Инициирование задачи
			; 0

При запуске системы необходимо инициализировать каждую задачу. Инициализация включает в себя задание начального содержимого указателей стеков в таблице SPSAV, а также регистров процессора и программного счетчика. После инициализации всех задач осуществляется вызов планировщика задач по метке START в подпрограмме TIMER.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что понимается под программным обеспечением современных ЭВМ?
2. Какие функции выполняют программы базового программного обеспечения?
3. Какие достоинства и недостатки присущи языку ассемблера и языкам высокого уровня?
4. Чем отличаются подпрограммы от макрокоманд?
5. Составить макрокоманду сброса всех внутренних программно-доступных регистров микропроцессора КР580.
6. Составить макрокоманду обмена содержимым всех рабочих регистров процессоров СМ посредством круговой передачи.
7. Какие функции выполняет в программе-ассемблере счетчик адресов?
8. Построить синтаксические диаграммы определения целого без знака и символьной константы.
9. Какие функции выполняет указатель буфера в программе-редакторе?
10. Каким образом обрабатываются в макрокомандах локальные имена?
11. Какие преимущества дает использование подпрограмм?
12. В чем заключаются главные особенности сопрограмм?

ПРИМЕНЕНИЕ, ОСНОВНЫЕ ЭТАПЫ ПРОЕКТИРОВАНИЯ И ТЕНДЕНЦИИ РАЗВИТИЯ МИКРО- И МИНИ-ЭВМ

8-1. ОБЛАСТИ ПРИМЕНЕНИЯ МИКРО- И МИНИ-ЭВМ

В настоящее время трудно назвать области и сферы человеческой деятельности, где бы не применялись мини-машины. Перечислим только основные области: научно-технические расчеты; автоматизированные системы управления производством (АСУП); обработка данных и связь; автоматизированные системы управления технологическими процессами (АСУТП); системы автоматизации научных экспериментов (САНЭ); системы автоматизации проектирования (САИР). Естественно, что различные области применения предъявляют к микро- и мини-ЭВМ различные требования. Так, например, в АСУП при обработке данных важными характеристиками являются структура слова и возможности операций с байтами, в научно-технических расчетах важна скорость выполнения операций, производительность системы. При оценке мини-ЭВМ, включаемой в АСУТП, преимущественное значение имеют система прерываний и возможности передачи данных. Характеристики современных микро- и мини-ЭВМ являются результатом компромиссных решений, однако каждый из типов малых машин, несмотря на возможный широкий спектр применения, ориентирован на определенную его область.

Рассмотрим подробнее основные сферы применения мини-ЭВМ.

Научно-технические расчеты. Данная сфера применения, казалось бы, не соответствует идеологии мини-машин. Однако в некоторых случаях мини-ЭВМ успешно конкурируют и с большими ЭВМ. Учитывая потребности чисто вычислительных приложений, разработчики мини-ЭВМ предусмотрели резкое повышение арифметической мощности машин путем включения в состав мини-машин специальных устройств для выполнения операций с высокой скоростью при повышенной точности представления чисел, а также чисел с плавающей запятой.

Под влиянием использования мини-машин для научно-технических расчетов появились многомашинные и многопроцессорные вычислительные системы на базе мини-ЭВМ с типичными для больших ЭВМ организациями параллельной обработки информации.

Автоматизированные системы управления производством, обработка данных и связь. Задачи, решаемые мини-ЭВМ, используемыми в подобных сферах применения, сводятся в основном к следующим: организации эффективных информационно-массовых, оперативной обработке больших массивов символьной информации, оперативной связи, оперативному представлению информации различными иерархическими уровнями управления и в конечном счете к управлению производственным циклом. Например, мини-ЭВМ, включенная в процесс производства дискретных изделий, контролирует состояние оборудования, занятого в производстве, и сам процесс производства изделий. Мини-ЭВМ создает, дополняет и обновляет массивы (банк) данных, отображающие текущее состояние процесса производства. Банки данных, используемые в задачах управления производством, имеют значительно более сложную структуру, чем в задачах управления процессами, поэтому применение мини-ЭВМ в данной области предполагает наличие программных устройств, включающих в себя универсальную систему обработки файлов.

Автоматизированные системы управления технологическими процессами. Применяемые здесь мини-ЭВМ должны удовлетворять тем же требованиям, что и в АСУП, а именно наращиваемости, надежности, доступности информации и средств управления оператору. Основные функции, которые возлагаются на микро- и мини-ЭВМ, работающие в составе АСУТП, — это функции контроля, управления и оптимизации технологических процессов, управления отдельными агрегатами и автоматическими линиями.

Информация, вводимая в мини-ЭВМ, может использоваться в различных контурах управления. Простейшие из них — контуры контроля и сравнения с пороговыми (предельными) значениями. По заложенным программам мини-машина сравнивает текущие значения параметров с уставками и в случае выхода за допустимые пределы вырабатывает сигналы для оператора, причем пороговые значения могут изменяться во времени (программный режим) и корректироваться в ходе процесса. Оператор, используя данные сигналы, может переключать исполнительные механизмы с целью управления ходом технологического процесса. Если сигналы управления непосредственно передаются на исполнительные органы, то говорят о прямом цифровом управлении, т. е. о той концепции управления, которая появилась в начале 60-х годов и предусматривала разделение времени управляющей ЭВМ между множеством входных датчиков и выходных исполнительных органов. Данная концепция означала качественный скачок вперед в области управления с точки зрения простоты и стоимости и позволила разработчикам систем управления систематизировать параметры технологического процесса, управляемого с помощью вычислительной машины. Применение мини-ЭВМ в АСУТП обеспечило требуемый уровень производства, существенное увеличение производительности технологического оборудования, высвободило операторов от ручных операций контроля, вывело операторов из производственных зон с повышенной опасностью, с высокой запыленностью, загазованностью и т. д. В настоящее время отечественные мини-ЭВМ серии СМ ЭВМ и других типов с успехом используются в АСУТП в различных областях народного хозяйства, таких как машиностроение (металлообработка), электронная и радиоэлектронная промышленность, металлургия (управление доменным процессом), химия и нефтехимия (большинство производств современной технологии), горная промышленность (управление технологическими процессами на горно-обогатительных комбинатах), цементная промышленность (управление технологическими процессами) и т. д.

Системы автоматизации научных экспериментов. Основные особенности применения малых ЭВМ в этой сфере определяются необходимостью в экспресс-обработке данных, в подключении лабораторных датчиков, сервисными возможностями устройств ввода-вывода, необходимостью гибкости системы в целом и устройств сопряжения в частности для быстрой перестройки аппаратуры при проведении различных экспериментов. На мини-машины в составе САНЭ возлагаются следующие функции: регистрация и обработка экспериментальных данных в реальном масштабе времени; управление экспериментальной установкой на этапе ее настройки на заданный режим; управление экспериментом в соответствии с алгоритмами принятия решений; передача экспериментальных данных для сложной обработки в ЭВМ высшего иерархического уровня и прием результатов обработки; обеспечение работы экспериментатора в диалоговом режиме с вычислительной машиной; функции контроллера или концентратора в многомашинных иерархических системах автоматизации научного эксперимента (САНЭ).

Сопряжения мини-ЭВМ с экспериментальной аппаратурой реализуется на базе технических средств, отвечающих различным стандартам.

В нашей стране получила распространение международная система КАМАК (Computer application for measurement and control) и отечественная система, выполненная в соответствии с системой стандартов на агрегатные средства электроизмерительной техники (АСЭТ).

На базе мини-ЭВМ серии СМ ЭВМ в нашей стране выпускаются измерительно-вычислительные комплексы (ИВК), которые являются аппаратной и программной основой для построения САНЭ. ИВК осуществляет программную коммутацию, цифровое измерение и выдачу электрических сигналов с датчиков, регистрацию данных, построение графиков, сбор, хранение и обработку результатов измерений, программное управление экспериментом.

Системы автоматизации проектирования. Известно, что проектирование в любой отрасли (машиностроение, кораблестроение, приборостроение, радиоэлектроника и т. д.) требует огромных затрат труда на обработку самой разнообразной графической информации. Широкое внедрение систем автоматизации проектирования на базе мини-ЭВМ во многих областях науки и производства позволяет осуществить значительную экономию трудовых затрат.

Непрерывное усложнение систем, агрегатов, машин и отдельных устройств, с одной стороны, и необходимость учета множества факторов, взаимосвязь и взаимозависимость показателей качества систем — с другой стимулируют развитие систем автоматизированного проектирования.

Если на первых этапах развития САПР решались задачи оптимизации отдельных характеристик элементов и систем, то современный этап характеризуется внедрением комплексных систем автоматизированного проектирования — от уточнения постановки задачи проектирования в диалоговом режиме до формирования управляющей информации (перфолист, магнитных лент), поступающей на вход технологического оборудования.

Необходимо отметить, что на ранних этапах развития САПР машинные результаты проектных расчетов выдавались в цифровой или алфавитно-цифровой форме, после чего они обрабатывались вручную и преобразовывались в графическую форму. Появление устройств ввода и отображения графической информации (дисплей, устройства ввода с чертежей, графопостроители и т. п.) позволило реализовать интерактивную обработку графической информации.

В нашей стране на основе мини-ЭВМ СМ-3 и СМ-4 серийно выпускаются вычислительные комплексы двух типов, предназначенные для САПР: автоматизированное рабочее место конструктора радиоэлектронной аппаратуры (АРМ-Р) и автоматизированное рабочее место конструктора изделий машиностроения (АРМ-М). Программное обеспечение позволяет пользователю АРМ одновременно решать до четырех различных задач, вводить в ЭВМ и просматривать введенные чертежи, рассматривать любые фрагменты увеличенных чертежей, корректировать чертежи, создавать архивы описаний чертежей и обращаться к этим архивам, вводить и редактировать текстовые материалы, получать окончательные результаты проектирования в виде чертежей и листингов текста, подготавливать перфолисты для управления станками с числовым программным управлением, разрабатывать новые пользовательские программы.

Автоматизированные рабочие места находят применение при решении задач, требующих сравнительно простых вычислений: задач типа размещения стандартных элементов на плоскости (расстановка оборудования в цехе, компоновка печатной платы, составление структурной схемы или сетевого графика, раскрой материала и т. п.); задач, решение которых носит операционный характер при нечетко определенном

критерии качества (трассировка печатных плат); задач, связанных с периодическим внесением небольших изменений в сложные чертежи, с периодическим внесением изменений в текстовые материалы; задач, связанных с разработкой и отладкой прикладных программ.

Направление дальнейшего развития АРМ — улучшение эксплуатационных характеристик, таких, как стоимость в расчете на одно рабочее место, расширение набора выполняемых функций, повышение точности, удобства и скорости работы, установка терминалов непосредственно на рабочих местах конструкторов.

Основные требования к микро- и мини-ЭВМ и перспектива их развития. Различные области применения требуют создания ЭВМ с соответствующей структурой аппаратных и программных средств. Так, например, для научно-технических расчетов, систем автоматизации научных исследований и АСУП существенными являются характеристики производительности и объем внутренней и внешней памяти. При применении ЭВМ в АСУТП, АСУП, САПР и САНЭ исключительно важны унификация и гибкость подсистемы ввода-вывода информации. В АСУТП возможен использование только микро- и мини-ЭВМ с многоуровневой системой прерываний. Это требование важно также и при применении мини-ЭВМ в САПР.

Распределение микропроцессорной техники и микро-ЭВМ по различным областям носит несколько иной характер, чем распределение мини-ЭВМ. Это определяется, с одной стороны, более ограниченными функциональными возможностями микро-ЭВМ, а с другой — более низкой стоимостью и возможностью встраивания микро-ЭВМ непосредственно в объекты измерения, управления и контроля. Наибольшее распространение микро-ЭВМ получили в информационно-измерительных и информационно-управляющих комплексах, в системах связи, в авиации, космонавтике и военной технике. В последние годы микропроцессорные функционально-ориентированные системы внедряются в бытовую технику, в медицину, транспорт, учебное оборудование и т. д. Использование микропроцессорных систем непосредственно в вычислительной технике сравнительно узко и определяется микро- и мини-машинами и периферийным оборудованием, используемыми для научно-технических и коммерческих расчетов.

Микро- и мини-ЭВМ получают все большее применение в машиностроении. Современное развитие машиностроения характеризуется все более широким применением средств автоматизации технологических процессов на самых различных уровнях, от автоматизации отдельных элементов рабочего цикла станков и машин до комплексно-автоматизированных производственных участков, цехов и предприятий; при этом централизация программного управления обеспечивается с помощью вычислительных комплексов.

Тенденции развития управления оборудованием — введение комплексных автоматизированных иерархических систем, включающих в себя следующие уровни: проектирования и производственного планирования, управления и оптимизации обработки, календарного планирования продукции. Использование принципов адаптивного управления увеличивает гибкость системы.

Применение устройств числового программного управления в машиностроении и других областях промышленности показало их высокую технико-экономическую эффективность за счет повышения производительности оборудования с числовым программным управлением, сокращения сроков перехода на изготовление деталей новых видов, снижения требований к уровню профессиональной подготовки рабочих-станочников и пр.

На периоды 1981—1985 и 1986—1990 гг. выделен ряд приоритетных областей применения мини-ЭВМ серии СМ ЭВМ с учетом развития средств вычислительной техники и накопленного опыта автоматизации объектов различной сложности и назначения.

Наряду с традиционным применением в АСУТП, САПР, САНЭ микро- и мини-ЭВМ широко используются в следующих областях:

при создании локальных и распределенных сетей ЭВМ различных классов, при подготовке информации и телеобработке данных, при организации и развитии систем коллективного использования средств вычислительной техники;

на транспорте для управления сортировочными станциями на железных дорогах и обработкой грузов в морских, речных и авиапортах; управления регистрацией пассажиров и багажа, системой продажи билетов, дорожным движением; управления, контроля, технической диагностики транспортных средств и т. п.;

в торговле и материально-техническом снабжении для управления складскими комплексами и базами товаров; в информационно-контрольных системах крупных торговых предприятий;

в сельском хозяйстве для управления технологическими процессами крупных животноводческих и птицеводческих комплексов, комбикормового производства; управления микроклиматом теплиц; управления процессами послеуборочной обработки и хранения сельскохозяйственных продуктов; управления водораспределительными сетями; в информационных системах агрохимической службы;

в пищевой промышленности для управления технологическими процессами на молокозаводах, мясокомбинатах, холодильниках, консервных заводах, мельницах и элеваторах, кондитерских фабриках;

в здравоохранении для оперативного врачебного и доврачебного контроля пациентов в клиниках, в диагностических системах, для автоматизированного ведения историй болезни, управления диагностическим и терапевтическим оборудованием, управления технологическими процессами в медицинских учреждениях;

в жилищном, гражданском, промышленном и дорожном строительстве для автоматизации проектирования;

при профессиональном обучении и переподготовке кадров (автоматизированные классы и экзаменационные стенды, информационно-справочные системы, системы программированного обучения, тренажеры);

в разведочной и промысловой геофизике для экспресс-обработки информации на полевых станциях и в региональных центрах;

при охране окружающей среды для автоматизированных систем контроля состояния атмосферы и водоемов.

8-2. ФУНКЦИОНАЛЬНО-СТРУКТУРНЫЙ ПОДХОД К ПРОЕКТИРОВАНИЮ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ СИСТЕМ

Эволюционный синтез систем базируется на диалектической концепции развития систем и функционально-структурном подходе к анализу и созданию систем.

Целью эволюционного синтеза систем является формирование систем, обладающих следующей совокупностью свойств: преемственностью функционально-структурной организации; адекватностью структурной организации системы ее функциональному назначению и условиям эксплуатации; ориентацией на максимальное использование унифицированных элементов; возможностью развития антропогенных систем.

Преемственность функционально-структурной организации достигается учетом предыстории и закономерностей развития систем данного и родственных классов.

Основные этапы синтеза ориентированы на формирование структуры системы с максимальным соответствием конструктивных элементов системы реализуемым функциям. Формируемая структура системы должна представлять собой рациональное сочетание многофункциональных и специализированных элементов, причем предпочтение отдается одно-типным многофункциональным элементам при наиболее полном использовании их функциональных возможностей.

Предпочтительное использование многофункциональных элементов позволяет минимизировать структуру системы и обеспечить высокие показатели качества функционирования в процессе эксплуатации.

Учитывая необходимость сокращения сроков создания сложных систем и необходимость их совершенствования в процессе эксплуатации, при функционально-структурной организации системы следует предусматривать возможность реконфигурации системы и замены отдельных модулей.

Следует отметить, что вопросы структурного синтеза многоуровневых систем до настоящего времени решаются интуитивно и являются прерогативой мышления главного конструктора системы.

Этапы эволюционного синтеза систем. Методология эволюционного синтеза систем предусматривает следующие этапы формирования функционально-структурной организации систем:

1. Выделение основных и дополнительных функций системы.
2. Декомпозицию основных и дополнительных функций с целью формирования дерева функций системы.
3. Выделение набора вещественно-энергетических и информационных операторов, необходимых для формирования структуры системы.
4. Представление функций i -го уровня декомпозиции набором функциональных операторов.
5. Анализ связей между операторами i -го уровня.
6. Построение временных диаграмм активности операторов i -го уровня.
7. Формирование функциональных модулей системы.
8. Эквивалентные преобразования операторных моделей с целью формирования структуры системы.
9. Выбор базовых функциональных структур.
10. Покрытие базовых функциональных структур конструктивными модулями.
11. Оценку технико-экономических показателей.
12. Рассмотрение возможных вариантов формирования структуры системы на других уровнях декомпозиции.
13. Сравнительный анализ вариантов функционально-структурной организации системы.
14. Выбор рациональной структуры системы.

Выделение основных и дополнительных функций системы. Стратегия эволюционного синтеза систем базируется на анализе общих закономерностей развития систем и закономерностей развития систем определенного функционального назначения.

При создании принципиально нового класса систем необходим анализ развития систем родственных классов. На основе анализа перечисленных закономерностей выявляется ряд положений, присущих создаваемой системе. Одним из основных результатов проводимого анализа является определение множества основных $\{BF_i\}$ и дополнительных $\{AF_j\}$ функций.

Множество основных функций определяет класс системы. Изменение множества $\{BF_i\}$ порождает систему нового класса. Множество дополнительных функций определяет сервис системы и такие показатели качества системы, как надежность, живучесть. Изменения множества

{AF} порождают различные варианты систем данного класса, определяемого множеством {BF₁}.

Декомпозиция основных и дополнительных функций — формирование дерева функций системы. На данном этапе осуществляется анализ наборов основных {BF₁} и дополнительных {AF_j} функций системы. На основе анализа множества основных и дополнительных функций {RF_n}, где {RF_n} = {BF₁} ∪ {AF_j}, устанавливается их преемственность по отношению к функциям, характерным для систем рассматриваемого класса, сформировавшимся в процессе эволюции. Такое сопоставление позволяет уточнить состав множества {RF_n}, учитывая различные особенности условий эксплуатации системы и ее конкретного применения.

В результате анализа формируется древовидная модель системы — дерево функций системы. Каждый последующий уровень функций получается из предыдущего в результате декомпозиции функций предыдущего уровня. Функции предыдущего уровня декомпозиции могут рассматриваться как макрофункции MF по отношению к микрофункциям mF последующего уровня. На промежуточном i-м уровне декомпозиции j-я функция является макрофункцией MF по отношению к функциям (i+1)-го уровня и одновременно может рассматриваться как микрофункция mF_jⁱ по отношению к функциям (i-1)-го уровня. Каждый новый, более подробный уровень описания функций, воспроизводимых системой, отличается уменьшением уровня абстракции и повышением детализации.

Степень декомпозиции макрофункций и собственно число уровней древовидной модели определяются функциональными возможностями элементов, используемых при создании системы. Обычно число уровней декомпозиции не превышает 5—7.

Выделение набора вещественно-энергетических и информационных операторов, необходимых для формирования структуры системы. Операторное представление функций и структуры системы целесообразно по следующим соображениям. Оператор представляет собой более общее понятие, чем функция, и позволяет отображать функцию одной переменной в функцию другой переменной.

Понятие оператора может покрывать как элементарные операции — микрофункции (простой оператор), так и совокупность макрофункций (сложный оператор). Над операторами возможны различные преобразования: вложение, декомпозиция и т. д., причем сложный оператор может рассматриваться как множество простых. Таким образом, при синтезе систем оператор отождествляется с направленным воздействием, необходимым для реализации соответствующей макро- или микрофункции.

Полное определение оператора включает в себя: вид выполняемого простого или сложного преобразования; элементы (объекты), подлежащие преобразованию; условия выполнения преобразования.

Будем различать вещественные (V), энергетические (E) и информационные (I) операторы:

V — операторы, описывающие процессы в технологических системах, назначением которых является переработка, преобразование и транспорт вещества;

E — операторы, отражающие реальные процессы в энергетических системах: получение, аккумулятирование, обмен, передачу и преобразование энергии;

I — операторы, отражающие процессы получения, хранения, переработки и передачи информации абстрагированно от конкретных веществных или энергетических носителей и преобразователей информации.

Вводится также совокупность комплексных информационно-энергетических, информационно-вещественных и вещественно-энергетических прямых и инверсных операторов. Очевидно, что в силу закономерности вещественно-энергетической и информационной целостности систем такое абстрактное, удобное для синтеза разделение операторов весьма условно. В реальных системах процессы преобразования вещества, энергии и информации взаимосвязаны и взаимообусловлены.

Представление функций определенного уровня декомпозиции набором функциональных операторов. Для каждого класса антропогенных систем должны быть разработаны методы перехода от типовых функций, характерных для данного класса, к соответствующим функциональным операторам.

В соответствии с закономерностью функциональной полноты элементов системы может быть выделена базовая совокупность операторов для систем определенного класса — функционально полный набор операторов (ФПНО). ФПНО позволяет покрыть множество функций древовидной структуры системы на произвольном уровне декомпозиции. Переход от дерева функций к операторным моделям осуществляется путем декомпозиции внешних и внутренних связей между функциями каждого уровня, выделенными при формировании дерева функций системы.

Анализ связи между операторами i -го уровня декомпозиции. Одним из основных этапов структурного синтеза систем является формирование операторных моделей.

Для полноты анализа и последующего синтеза базовых структур систем определенного функционального назначения наряду с декомпозицией множества $\{RF_n\}$ основных и дополнительных функций необходимо произвести декомпозицию внешних и внутренних связей выделенных функций, образующих множества $\{BF_i\}$ и $\{AF_j\}$.

Так как каждый уровень декомпозиции системы отражает функционально-структурную организацию системы в целом с различной степенью детализации, то декомпозиция системных связей соответствует выделению и анализу в системе следующих организационных уровней: исполнительного, подготовительного, вспомогательного, координационного. Таким образом, построение операторных моделей позволяет описать последовательность вещественно-энергетических и информационных процессов и определить их логическую взаимосвязь. Это позволяет формировать структуру системы на уровне функциональных модулей.

Построение временных диаграмм активности. При формировании структуры системы необходимо учитывать не только совокупность функций, подлежащих реализации, но и установить определенную последовательность активности выделенных функций и условия их выполнения. В системе должна быть предусмотрена реализация определенных функций при различных конкретных ситуациях.

Для реализации конкретных операторов на входах определенных функциональных модулей в соответствующие моменты времени должно быть организовано требуемое вещественное, энергетическое и информационное обеспечение, а ресурсы соответствующего модуля должны быть освобождены к моменту воспроизведения следующего цикла операций.

Таким образом, в результате построения и анализа временных диаграмм активности операторов каждого i -го уровня множества $\{RF_n\}$ реализуемых системой основных и дополнительных функций могут быть разделены на подмножества активных и пассивных элементов для каждого интервала времени функционирования системы.

Временной анализ активности элементов осуществляется с целью определить избыточность древовидной структуры системы.

Формирование функциональных модулей системы. Эквивалентные преобразования операторных моделей с целью формирования функциональных структур. Необходимо различать следующие виды системной организации: функциональную и структурную. Функциональная организация описывается моделью системы на основе функциональных элементов и отражает основные связи между выделенными, в соответствии с принятым уровнем декомпозиции системы, элементами системы соответствующего уровня (функциональными подсистемами).

Структурная организация системы описывается моделью покрытия функциональных модулей виртуальной системы i -го уровня набором конструктивных элементов.

Функциональная структурная организация системы отражает связь и взаимное влияние функциональной и структурной организаций системы, т.е. учитывает особенности строения и взаимодействия элементов в системе.

Функциональный модуль (ФМ) — элемент функционально-структурной организации, несущий информацию о функциональной структуре системы или ее частей.

Конструктивный модуль (КМ) — конструктивно законченный элемент функционально-структурной организации, предназначенный для реализации определенной совокупности функций и используемый для покрытия множества элементов функциональной организации системы.

Таким образом, структурный синтез на рассматриваемом этапе сводится к построению функционально-структурной модели системы на основе функциональных модулей с последующим покрытием этой структуры конструктивными модулями.

Формирование функциональных и конструктивных модулей осуществляется посредством эквивалентных преобразований функциональных операторов на основе эвристических приемов, базирующихся на единстве многофункциональности и специализации: на программном и аппаратном разделении или совмещении реализуемых функций; использовании операторов непрерывной и дискретной настройки; регенерации функций, энергии, материалов; инверсии функций системы (подсистемы); реализации новых функций и макрофункций на основе определенного набора функций (микрофункций); усложнении базовых операторов и функциональных модулей. В результате формируются альтернативные варианты структуры системы на основе функциональных модулей.

Выбор базовых функциональных структур. При проектировании функционально-ориентированных систем, реализующих заданную совокупность функций $\{RF_n\}$, необходимо учитывать взаимосвязь функциональной и структурной организации системы.

Синтез базовых функциональных структур должен осуществляться с учетом развития системы при воспроизведении множества дополнительных (сервисных) функций $\{AF_j\}$.

Наилучшей с позиций функциональной расширяемости является система с минимальной структурой, обладающая свойствами однородности элементов и регулярности связей между функциональными и конструктивными элементами. Добиться желаемых результатов можно за счет повышения удельного веса многофункциональных элементов. Необходимо учитывать, что для каждого уровня развития технологии и определенного этапа эволюции системы существует определенное оптимальное соотношение многофункциональных и специализированных компонентов системы.

Покрытие базовых функциональных структур конструктивными модулями. Результатом последовательных этапов эволюционного синтеза является структура системы на уровне реальных конструктивных моду-

лей. Основные конструктивные модули системы являются серийными, а число вновь разрабатываемых модулей сводится до минимума. Необходимость в разработке новых конструктивных модулей возникает только в случае создания систем нового класса и при очень жестких требованиях к показателям качества системы.

Все этапы эволюционного синтеза направлены на обеспечение высоких качественных показателей системы за счет усиления требований к составным элементам и подсистемам и за счет наиболее полного использования функциональных возможностей типовых элементов.

Таким образом, покрытие базовых функциональных структур конструктивными модулями осуществляется с учетом минимизации структуры проектируемой системы; равномерности загрузки отдельных компонентов системы в процессе эксплуатации. Для реализации этого этапа в процессе синтеза реальной системы рекомендуется составлять таблицы соответствия функциональных и конструктивных модулей — обобщенные функционально-структурные карты. Результатом покрытия функциональной структуры системы конструктивными модулями является реальная структура системы на базе унифицированных модулей.

Функционально-структурная организация системы на других уровнях декомпозиции. Учитывая многоуровневость декомпозиции множества $\{RF_n\}$ основных и дополнительных функций, каждую из функций можем рассматривать как макрофункцию или как микрофункцию в зависимости от уровня декомпозиции.

Декомпозиция макрофункций на сравнительно большое число микрофункций характерна для анализа различных классов сложных систем. Идентичность декомпозиции макрофункций и соответствующая им регулярность (повторяемость) процессов во времени и пространстве позволяет создавать экономичные (однородные) структуры, эффективно воспроизводимые современными прогрессивными технологическими приемами, методами интегральной технологии.

Как уже отмечалось, для сложных антропогенных систем число уровней декомпозиции множества основных и дополнительных функций $\{RF_n\}$ обычно не превышает 5—7.

Осуществляемый в процессе синтеза многоуровневых систем переход от предельной декомпозиции функций (дерево функций систем) к предельному сжатию (базовые функциональные модели) соответствует принципу многофункциональности и физической интеграции функционально-связанных элементов в подсистемах.

Ограничение числа уровней декомпозиции связано с прагматическим подходом к синтезу систем.

Сравнительный анализ вариантов функционально-структурной организации системы. Выбор рациональной структуры. Возможна организация структуры системы в соответствии с первым уровнем декомпозиции. В этом случае система базируется на функциональных модулях высокого уровня, каждый из которых реализует основную или дополнительную функцию системы. При этом наблюдается взаимно-однозначное соответствие

$$FM_i \rightarrow RF_j \in \{RF_n\}.$$

При однозначном соответствии указанным функциональным модулям соответствующих конструктивных модулей задача проектирования системы сводится к комплектованию системы из готовых модулей, организации их внутреннего взаимодействия и сопряжению с внешней средой. Такой подход обычно возможен при создании неоригинальных (типовых) систем. Формирование подобных систем не связано с необходимостью теоретических исследований.

Очевидно, что при создании систем нового класса невозможно использовать модули высокого уровня, соответствующие элементам множеств $\{BF_i\}$ и $\{AF_j\}$. Использование таких модулей бывает неэкономичным ввиду их сравнительно малой загрузки.

Как показывает анализ деревьев функций широкого класса антропогенных систем, воспроизведение макрофункций (функций множества $\{FR_n\}$) связано с реализацией ряда идентичных процессов на уровне макрофункций. Это является стимулом для создания систем с более экономичной структурой, использующих принципы временного совмещения-разделения функций.

Необходимость в пересмотре традиционных или в разработке новых принципов функционально-структурной организации и в создании новых технических средств возникает в следующих случаях: при создании оригинальных систем принципиального нового функционального назначения; при создании систем с экстремальными характеристиками; при разработке новых физических принципов работы элементов и создании новых технологических процессов; при использовании новых принципов управления системами и обработки информации и т. д.

8-3. МИКРОПРОЦЕССОРЫ В СИСТЕМАХ УПРАВЛЕНИЯ¹

Освоение отечественной промышленностью производства больших интегральных схем (БИС), в частности микропроцессорных наборов, служит основной предпосылкой для массового внедрения универсальных вычислительных средств в системы числового программного управления (СЧПУ). Это позволит резко сократить номенклатуру разрабатываемых СЧПУ, повысить эффективность и качество процесса управления, его надежность и живучесть. Комплексный подход к проектированию СЧПУ должен учитывать не только возможности, предоставляемые микропроцессорными наборами, но и требования, предъявляемые, в свою очередь, к ним реальными технологическими процессами. В настоящем параграфе рассматриваются принципы построения СЧПУ на основе микропроцессорных наборов.

Базовая конфигурация СЧПУ. Под базовой конфигурацией будем понимать систему, содержащую минимальные аппаратные и программные средства, необходимые для реализации процесса управления заданным технологическим оборудованием.

Анализ отечественного и зарубежного опыта проектирования СЧПУ металлорежущим оборудованием показывает, что базовая конфигурация (рис. 8-1) включает в себя следующие основные блоки: пульт оператора (ПО), предназначенный для задания всех режимов системы, а также ввода, редактирования программы и индикации требуемой информации (может быть конструктивно выполнен встроенным в СЧПУ или выносным); буферную память программы обработки детали (БППОД), необходимую для хранения программы обработки детали (или ее части); память коррекций (ПК) для хранения констант коррекции длины и радиуса инструмента, смещений, компенсации погрешностей механики объекта и т. д.; память редактирующей информации (ПРИ), где хранится информация, введенная в режиме редактирования для исправления ошибок или изменения программы обработки детали; память программ подготовки данных (ПППД), в которой размещены программа-диспетчер, пакет программ по упаковке и преобразованию информации, подготовке данных (алгоритм расчета эквидистанты, интерполяции и т. п.); процессор (П) — основной вычислительный

¹ Параграф написан авторами совместно с М. С. Куприяновым.

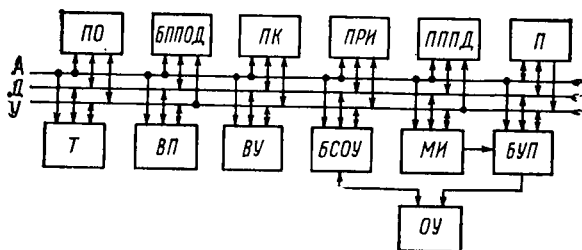


Рис. 8-1. Базовая конфигурация СЧПУ

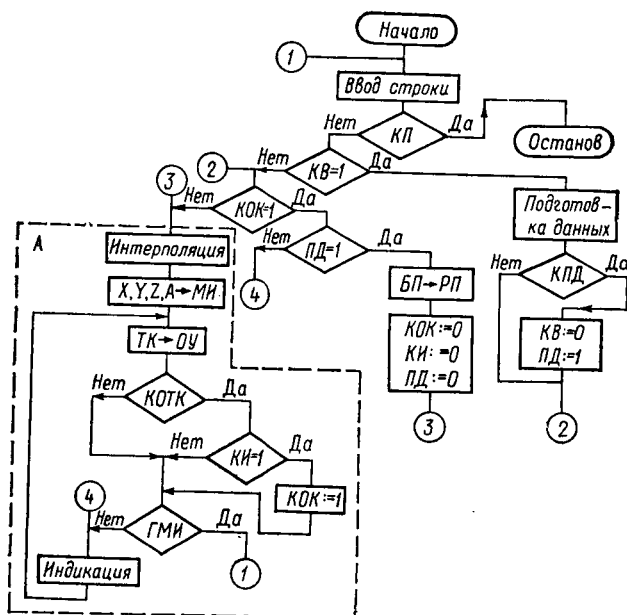


Рис. 8-2. Алгоритм функционирования СЧПУ базовой конфигурации

КП — конец программы; КВ — конец ввода кадра; КОК — конец отработки кадра; ПД — признак подготовки данных; КПД — конец подготовки данных; X, Y, Z, A — приращения по координатам, выдаваемые в соответствующие блоки управления приводом; ТК — технологические команды; КОТК — конец отработки технологических команд; КИ — конец интерполяции; ГМИ — готовность микроинтерполятора; БП — буферная память; РП — рабочая память; ОУ — объект управления

элемент, управляющий также работой всех блоков системы; таймер (Т) для задания временных интервалов системы, работающей в реальном масштабе времени; вспомогательный пульт (ВП), дублирующий некоторые основные функции пульта ПО и расположенный непосредственно у объекта управления (отсутствует, если пульт ПО выносной); внешнее устройство (ВУ), выполняющее функцию выдачи в СЧПУ и хранения библиотеки программ обработки деталей на языке ISO; блок связи с объектом управления (БСОУ), необходимый для выполнения технологических команд; блок управления приводом (БУП), обеспечивающий связь микроинтерполятора или процессора с приводом объекта управления; микроинтерполятор (МИ), преобразующий рассчитанные процессором приращения в выходную частоту (это может быть БУП с кодовым приемом информации); объект управления (ОУ).

Взаимодействие всех блоков системы в основном осуществляется посредством трех шин: адреса А, данных Д, управления У. По обратной связи в блоке БСОУ фиксируется состояние объекта управления (положения конечных выключателей, аварийные ситуации и т. п.).

Функционирование системы зависит от режимов, которые задаются с пультов ПО и ВП и определяются во многом технологическим процессом. Наиболее распространенные режимы: автоматический и кадровый для обработки одного кадра программы, поиск кадра, преднабор (ручной ввод), редактирование и режим ручного управления. Основной и наиболее сложный режим, требующий максимальной производительности процессора, — автоматический.

На рис. 8-2 приведен алгоритм функционирования СЧПУ базовой конфигурации в автоматическом режиме для случая, когда память БППОД рассчитана только на один кадр программы и в качестве ВУ используется фотосчитывающее устройство (ФСУ).

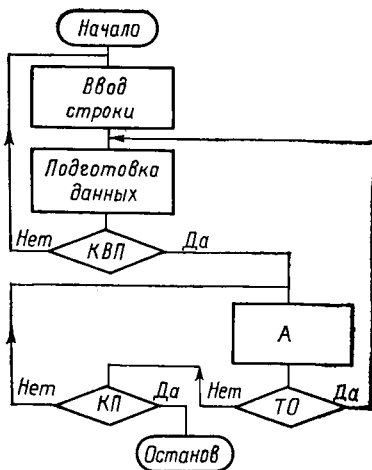
Для анализа алгоритма функционирования введем понятие вычислительного цикла, под которым будем понимать время, определяемое выражением $T_{\text{ц}} = t_1 + t_2$, где t_1 — время, необходимое для выполнения процессором обязательных функций, обеспечивающих обработку детали по заданной траектории и технологических команд; t_2 — время, затрачиваемое на осуществление функций (полностью или частично), не обязательных для данного цикла. Вычислительный цикл определяется с учетом следующих факторов: характеристик приводов подачи, быстрого действия процессора, временных интервалов управляющих дискретных сигналов, выдаваемых на объект управления, диапазона реализуемых скоростей подачи, режимов прерываний и т. д. Из алгоритма, приведенного на рис. 8-2, следует, что вычислительный цикл может определяться выполнением одной из следующих групп операций: фрагмента А; фрагмента А и ввода; фрагмента А и подготовки данных; ввода и подготовки данных. Распределение ресурсов вычислительного цикла зависит от информации в кадре, скорости работы ФСУ, частоты появления сигнала ГМИ (адаптация по скорости).

Для системы, показанной на рис. 8-1, разработка программы-диспетчера — основного элемента программного обеспечения (ПО) — представляет собой трудоемкий процесс, так как взаимодействие блоков зависит от многих условий, анализ которых требует затрат программных средств и памяти. Внутреннее ПО значительно упрощается, если в системе имеется память (в нашем случае память БППОД), рассчитанная на всю программу. Это позволяет в продолжение ввода программы проводить всю подготовку данных (подготовленная программа занимает в 3 раза меньший объем памяти, чем исходная программа на языке ISO), включать ФСУ в работу только при вводе программы (при этом ФСУ работает в непрерывном, а не в старт-стопном режиме), благо-

Рис. 8-3. Алгоритм функционирования СЧПУ с памятью БППОД для всей программы

КВП — конец ввода программы; КП — конец программы

даря чему уменьшаются потребление энергии и выделение тепла, повышается надежность работы системы в целом, обеспечивается в дальнейшем затрате ресурсов вычислительного цикла только на выполнение фрагмента А. Если предусматривается ввод новых коррекций, когда наступает технологический останов (ТО) с подтверждением, то с этого момента требуется некоторый перерасчет данных для оставшегося участка траектории. Алгоритм функционирования СЧПУ с памятью БППОД на всю программу представлен на рис. 8-3.



Мультипроцессорная конфигурация СЧПУ. В случае обслуживания СЧПУ сложного технологического оборудования (например, станка с двумя столами, на одном из которых ведутся подготовительные операции, а на другом непосредственная обработка, или станка с роботом) число параллельно выполняемых процедур увеличивается, что требует от процессора более высокой производительности и наличия в нем развитой системы прерываний. Указанное усложнение требований затрудняет внедрение микропроцессоров в СЧПУ и делает неэффективным использование конфигурации СЧПУ, приведенной на рис. 8-1. Высокий параллелизм осуществляемых СЧПУ функций обуславливает применение мультипроцессорной конфигурации с проблемной ориентацией каждого из процессоров.

На рис. 8-4 представлена двухпроцессорная конфигурация СЧПУ, в которой процессор П1 обеспечивает ввод программы и подготовку данных, а процессор П2 — решение траекторной задачи (см. фрагмент А на рис. 8-2). Пакет прикладных программ процессора П2 расположен в памяти обработки текущего кадра (ПОТК). Взаимодействие процессоров осуществляется посредством устройства АКК сопряжения магистралей адаптер — канал — канал, с помощью которого при нормальном функционировании всей системы процессор П1 передает процессору П2 подготовленную для обработки информацию.

Алгоритмы функционирования двухпроцессорной СЧПУ приведены на рис. 8-5. Момент передачи информации определяется признаком ПД, который устанавливается процессором П1, а сбрасывается процессором П2. Функционирование процессора П1 представлено для случая, когда память БППОД рассчитана на один кадр программы. Если же память БППОД рассчитана более чем на один кадр, процессор П1 может выполнять ряд других функций (контроля, диагностики и т. п.). Мультипроцессорная организация системы позволяет, кроме использования процессоров с упрощенным внутренним МО и структурой в сравнении с аналогичными данными процессоров однопроцессорной системы, повысить также надежность функционирования СЧПУ. Это достигается благодаря тому, что при отказе одного из процессоров его функции может

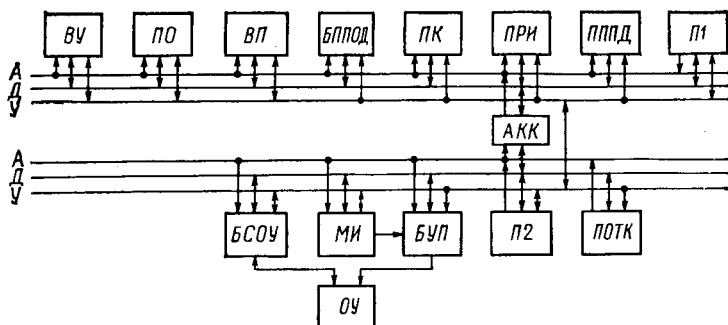


Рис. 8-4. Двухпроцессорная конфигурация СЧПУ

осуществлять другой процессор, и система будет работать с пониженным, но допустимым уровнем качества.

Число процессоров в системе можно увеличить при дальнейшей декомпозиции функций. Так, например, функции процессора П1 можно распределить между двумя процессорами, один из которых будет выполнять ввод информации, ее упаковку и предварительную обработку, а второй — подготовку данных.

Проблемная ориентация процессоров для СЧПУ. Проблемная ориентация процессора определяется специализацией его аппаратных и программных средств и производится для более эффективной реализации функций в системе. Как следует из приведенных алгоритмов, основными функциями, реализуемыми в системе, являются ввод информации

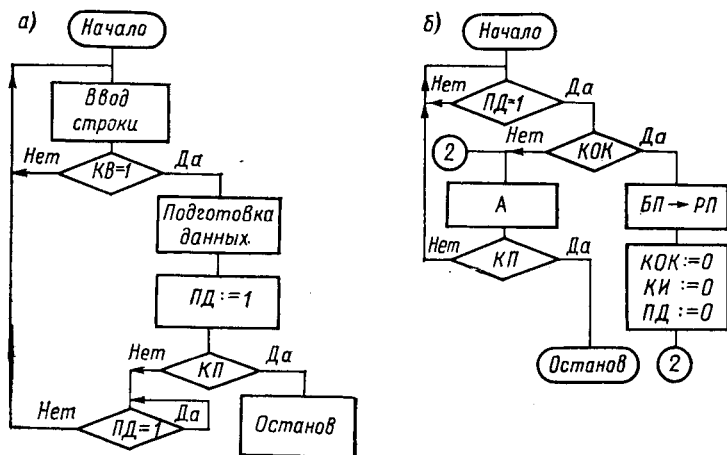


Рис. 8-5. Алгоритмы функционирования двухпроцессорной системы: а — алгоритм работы процессора П1; б — алгоритм работы процессора П2

в память БППОД и ее упаковка; подготовка данных; интерполяция; реализация технологических команд; индикация. Рассмотрим важнейшие пути специализации аппаратных и программных средств, ориентируясь на перечисленные функции.

Информация вводится побайтно, что определяет байтовую организацию процессора. Поскольку процедура ввода не изменяется в зависимости от ориентации СЧПУ на конкретное оборудование, то ее целесообразно реализовать на микропрограммном уровне. Это требует, во-первых, микропрограммного доступа к ячейкам БППОД, во-вторых, гибкого маскирования разрядов байта, в-третьих, достаточно большой микропрограммной памяти. Желательно введение микроинструкций, отражающих характер алгоритма преобразования, например перестановки тетрад в байте, установки и сброса флажков и т. п.

Для подготовки данных используются операнды длиной в 3 байта. Следовательно, обработка может выполняться последовательно-параллельно (побайтно) или параллельно (в зависимости от требований по быстродействию). По этой причине структура процессора должна обеспечивать наращивание разрядности. Для упрощения программирования и отладки алгоритмов по подготовке данных и интерполяции целесообразно названные алгоритмы реализовать на программном уровне, что также гарантирует преемственность разработок и инвариантность пакета прикладных программ элементной базы.

При использовании в системе блока БУП без кодового приема информации для управления приводом требуется цифровой интегратор, разрядность которого определяется разрядностью рассчитанного приращения. В свою очередь, эта разрядность зависит от длительности вычислительного цикла, а также от максимальной скорости обработки данных в системе.

Технологические команды выполняются в соответствии с реализацией логических алгоритмов посредством функционирования релейно-контактных блоков технологического оборудования. Замена ненадежных и занимающих большой объем релейно-контактных блоков табличными или таблично-алгоритмическими процессорами, на основе интегральной памяти с логическим характером обработки информации значительно повышает регулярность СЧПУ в целом, ее надежность и эффективность.

Индикация результатов производится путем вывода информации, необходимой оператору, на отображающие элементы (возможно также преобразование информации из одной системы счисления в другую). Для выполнения подобных функций, не зависящих от времени и других функций, целесообразно иметь в системе специализированные процессоры (в большинстве случаев таблично-алгоритмические).

Организовать вычислительные средства для СЧПУ с удовлетворением перечисленных требований позволяет подход, основанный на наиболее полном использовании функциональных возможностей памяти, а также многофункциональных модулей с регулярной структурой для построения алгоритмических структур. Реализация модулей в виде БИС дает основание говорить о микропроцессорных структурах, что определяет прежде всего конкретную техническую реализацию вычислительной структуры. Примером такого рода структур является семейство БИС К589. Микропроцессорные регулярные структуры позволяют не только компоновать процессор нужной структуры и архитектуры и тем самым проблемно ориентировать его, но и использовать отдельные модули для реализации других блоков (например, блоков МИ и БУП).

Указанный подход дает возможность в конечном счете организовать СЧПУ на двух типах модулей: памяти и микропроцессорной БИС.

В заключение следует отметить также, что построение СЧПУ на основе микропроцессорных регулярных структур соответствует наблюдающейся тенденции к модульной конструкции системы, что упрощает изменение ее функций и показателей, размещение относительно объекта управления, профилактику и ремонт, улучшает теплоотвод.

Структура программного обеспечения и автоматизация его проектирования в системах программного управления. Массовое производство микропроцессоров и микропроцессорных наборов изменило принципы организации аппаратурных средств систем программного управления (ПУ), что позволило расширить список реализуемых функций. Внедрение универсальных вычислительных средств обуславливает в первую очередь разработку развитого программного обеспечения для эффективной эксплуатации систем ПУ.

При управлении технологическим оборудованием в системе ПУ реализуются следующие функции: загрузка с предварительной упаковкой программы обработки детали; редактирование и хранение программы обработки деталей; подготовка данных для вывода на технологическое оборудование; вывод определенных порций информации на технологическое оборудование по мере обработки; анализ состояния технологического оборудования и формирование соответствующих признаков; контроль и учет состояния органов управления на пультах оператора, наладчика, технолога-программиста; управление логикой станка; учет погрешностей кинематики станка и износа инструмента; связь с ЭВМ высшего ранга (если необходимо).

Программное обеспечение современной системы ПУ, реализующей перечисленные функции, должно отвечать следующим требованиям: обеспечивать проблемную ориентацию аппаратурных средств системы; отвечать принципам совместимости и преемственности в семействе систем ПУ на основе МП; обеспечивать повышенную надежность системы; структура ПО должна отвечать модульному принципу, упрощающему модернизацию системы.

Приведенные перечни функций и требований могут уточняться в зависимости от назначения системы, однако они определяют типовой вариант структурной схемы ПО (рис. 8-6).

Служебные подпрограммы (ПП) включают в себя загрузчик, редактор и библиотечарь. Загрузчик выполняет ввод кадра программы (или всей программы) в буферную (или общую оперативную) память, осуществляя одновременно сжатие программы. Редактор служит для редактирования введенной программы. При этом должны обеспечиваться стирание, замена, вставка отдельных слов или кадров. Библиотечарь осуществляет подключение стандартных ПП в процессе функционирования системы.

Подпрограммы организации вычислительного процесса, входящие в операционную систему, должны обеспечивать эффективное функционирование системы ПУ различной конфигурации в заданном режиме работы. Эти подпрограммы должны обеспечивать работу системы как в однопрограммном, так и в многопрограммном режиме (при управлении сложным технологическим комплексом, оснащенным, например, роботами). Подпрограмма начального пуска предназначена для перевода системы из исходного состояния в заданный с пульта оператором режим работы. Она осуществляет сброс необходимых регистров МП, ввод констант, начальный контроль блоков системы и передачу управления диспетчеру. Диспетчер определяет стратегию работы системы, контролируя работу устройства прерывания, а затем передает управление супервизору, который определяет порядок выполнения подпрограмм в зависимости от режима работы, установленного диспетчером.

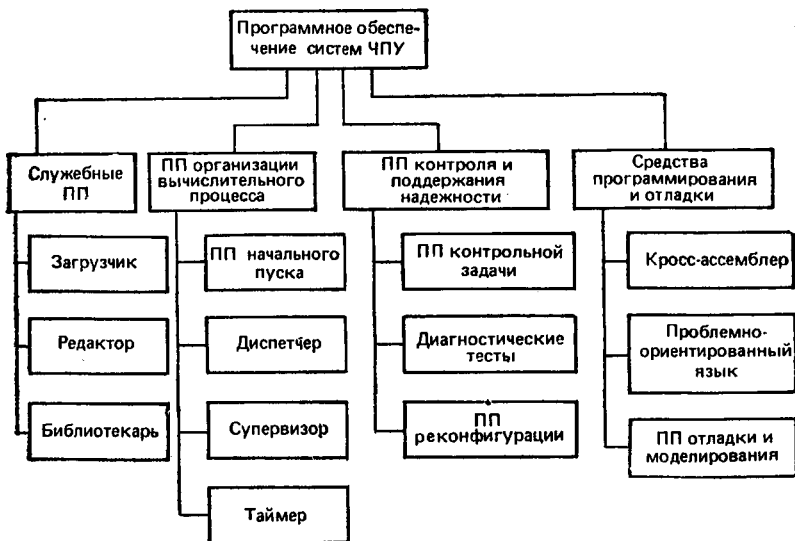


Рис. 8-6. Структурная схема ПО систем ЧПУ

Супервизор организует диалог с человеком через пульт; в него входят также ПП, обеспечивающие вывод информации на внешние устройства. Таймер осуществляет контроль счетчиков времени, организуя реализацию технологических команд, определяет периодичность включения ПП и т. п.

Подпрограммы контроля и поддержания надежности включают в себя ПП контрольной задачи, обеспечивающую проверку взаимодействия системы и оборудования; диагностические тесты, осуществляющие проверку блоков системы и выявление неисправностей; ПП реконфигурации, обеспечивающие подключение резервных блоков системы и необходимых программ при отказах, учет наиболее часто встречающихся отказов, индикацию изменения конфигурации системы, обновление программы и др.

Средства программирования и отладки включают в себя кросс-ассемблер, проблемно-ориентированный язык и ПП отладки и моделирования. Кросс-ассемблер используется для создания элементов ПО; он предпочтителен по сравнению с другими средствами программирования, так как позволяет создавать эффективные программы, а его транслятор не занимает памяти непосредственно в системе. Проблемно-ориентированный язык необходим для написания программ управления технологическим оборудованием и позволяет организовать эффективный ввод программ непосредственно с пульта, минуя процесс изготовления перфоленты. Подпрограммы отладки и моделирования также целесообразно использовать все средства диалога, представленного в этой машине. Эти подпрограммы предназначены для отладки ПО и моделирования системы с целью получения статистических данных, необходимых для проектирования, а также для проверки функционирования системы.

Прикладные программы представляют собой наборы отдельных программ, реализующие стандартные вычисления (преобразование из одной системы координат в другую, интерполяция, расчет эквидистанты и т.п.) или функции сбора и обработки информации. Они определяют проблемную ориентацию ПО и могут создаваться самим пользователем.

Для упрощения элементов внутреннего программного обеспечения распределение буферной памяти (памяти, содержащей один кадр программы и информацию, необходимую для его отработки) на зоны целесообразно сделать фиксированным, ориентируясь на максимальный кадр программы. Буферная память распределяется следующим образом. В первой зоне располагается кадр программы управления в упакованном виде. За каждой ячейкой этой зоны жестко закреплена строка (или их совокупность) кадра программы. Сжатие осуществляется за счет исключения признаковых символов строк кадра, функции которых в данном случае выполняют адреса ячеек зоны.

Во второй и третьей зонах формируются признаки, возникающие при вводе и подготовке данных и необходимые для организации вычислительного процесса, а также признаки, возникающие в системе в процессе управления (положения различных органов, наличие обратных связей и т.п.) и необходимые для оперативной связи системы с технологическим оборудованием.

В четвертой зоне формируется «образ» пульта, с которым супервизор работает между моментами опроса состояния органов пульта. Частота обращений супервизора к пульту определяется таймером и выбирается такой, чтобы реакция системы ПУ происходила без заметной для глаза человека задержки. Уменьшение частоты обращений супервизора к пульту разгружает МП и увеличивает помехозащищенность информационного тракта в случае, если пульт удален от основной стойки.

Пятая зона содержит информацию, подлежащую выводу на индикацию пульта, а шестая зона — информацию о пройденной инструмент траектории (длина запоминающего участка может быть различной), необходимую для возврата инструмента.

В зоне рабочих ячеек (седьмая зона) хранятся промежуточные результаты, базовые адреса и т.п. В зоне служебной информации (восьмая зона) запоминаются слова состояний программ при прерываниях.

В девятой зоне хранятся константы, счетчики циклов, предельные значения контролируемых параметров и т.п. В десятой зоне располагается информация о числе работающих МП и других блоков, о несправностях, возникающих в системе, о приоритетах.

8-4. СИНТЕЗ БАЗОВЫХ СТРУКТУР МНОГОФУНКЦИОНАЛЬНОГО КОНТРОЛЛЕРА ВЗУ

Назначение контроллера. Для общения с микро-ЭВМ требуются периферийные устройства, соответствующие характеристикам систем нового типа. Большое внимание при этом уделяется разработке внешних запоминающих устройств (ВЗУ). Особое место ВЗУ в комплексе аппаратных средств информационно-управляющих систем обусловлено следующими причинами: ВЗУ являются комбинированными устройствами ввода-вывода; обладают быстройдействием, превышающим быстродействие перфорационных носителей информации; обладают повышенной плотностью записи информации на носитель; могут использоваться многократно.

В системе микро-ЭВМ широко применяются накопители на магнитной ленте, магнитных дисках, цилиндрических магнитных доменах, а также различные типы кассетных накопителей.

К средствам управления внешними запоминающими устройствами относятся каналы и устройства управления. Электронику нескольких однотипных устройств заменяет общее устройство управления или контроллер.

Классификация контроллеров ВЗУ. Контроллер реализует совокупность функций, общих для конкретной группы ВЗУ, аналогично тому, как в канале реализуются функции, общие для периферийных устройств вычислительной системы. Использование контроллеров ВЗУ позволяет освободить центральный процессор (ЦП) от функций управления ВЗУ; сократить затраты времени на обработку прерываний, связанных с обслуживанием запросов процессов, протекающих в системе; обеспечить параллельную работу большого числа ВЗУ и унифицировать программирование ввода-вывода.

Контроллер занимает промежуточное положение между ЦП и ВЗУ. Система управления ВЗУ и микро-ЭВМ представляет собой трехуровневую систему. Верхним, командным уровнем (I) является ЦП, который инициирует работу коллектора (уровень II). Контроллер, в свою очередь, управляет всеми действиями группы ВЗУ (уровень III).

Связь между уровнями осуществляется с помощью стандартных интерфейсов, характеризующихся набором управляющих, адресных и информационных шин, алгоритмами обмена информацией, временными и техническими параметрами накопителей и ЦП.

Контроллер ВЗУ имеет программное и аппаратное обеспечение. Контроллер вырабатывает необходимые интерфейсные сигналы для ВЗУ и ЦП, анализирует поступающие от них сигналы и инициирует соответствующие действия.

Операции контроллера можно разделить на две группы: основные (базовые) операции, которые определяются назначением контроллера; дополнительные операции (операции расширенного списка), предназначенные для разгрузки ЦП.

Исходя из функций контроллера и его технической реализации, можно классифицировать контроллеры по признакам, указанным в табл. 8-1. Такому обилию признаков отвечает многообразие контроллеров ВЗУ. Таблица иллюстрирует основные принципы их построения, является основой для выбора типа контроллера в соответствии с техническим заданием.

Постановка задачи синтеза многофункционального контроллера. Совокупность аппаратных и программных средств контроллера предназначена для связи между центральным процессором и ВЗУ. Различия в функционировании, определяемые, с одной стороны, алгоритмами управления внешними накопителями, а также форматами записи информации, а с другой — различными требованиями центрального процессора, привели к появлению большого числа различных специализированных контроллеров. Эти контроллеры ориентированы на управление определенным типом ВЗУ и их сопряжение с микропроцессорной системой.

Учитывая большое число общих функций управления различными ВЗУ, сходство алгоритмов управления ими, аналогичность временных характеристик накопителей и принципов функционирования ВЗУ, ставят задачу унификации процесса управления и создания контроллера, адаптивного к управлению различными типами ВЗУ. Определим многофункциональный контроллер как устройство, способное управлять различными внешними устройствами без существенного изменения его

Таблица 8-1

Классификационные признаки контролеров

Способ реализации аппаратуры контроллера	МИС и СИС; МПК; заказные БИС
Вид управления ВЗУ	Групповое; управление одиночными ВЗУ
Тип подключаемых ВЗУ	Однотипные; разнотипные
Степень автономности	Встроенные; автономные; комбинированные
Организация управления	Программное; микропрограммное; жесткое
Организация интерфейса с ТП	Промежуточный интерфейс; программируемый; жесткий
Способ опроса интерфейса	Программный; аппаратный; смешанный
Способ передачи данных	Программная; передача с прерыванием; с прямым доступом
Способ передачи служебной информации	Программная; передача в режиме ПДП
Способ формирования блока	С буферизацией данных; без буферизации
Обработка информации	Распределенная; централизованная

структуры. Адаптивность к различным ВЗУ обычно достигается изменением управляющей программы.

В настоящее время широкое распространение получают микропроцессорные контроллеры. Применение микропроцессоров уменьшает сроки разработки и стоимость изделия за счет уменьшения числа микросхем и увеличивает функциональные возможности. Разработаны сложные БИС для управления различными типами ВЗУ. Эти контроллеры совместимы с микропроцессорной шиной и обеспечивают подключение устройств определенного типа к микро-ЭВМ. Вследствие жесткого управления они ориентированы на определенные скорости передачи данных, способы записи и форматы.

На основе анализа функционирования современных ВЗУ, сопряжения с ЦП и функциональной нагрузки контроллера можно предъявить многофункциональному контроллеру ВЗУ совокупность требований и предложить методы обеспечения этих требований (табл. 8-2).

Таблица 8-2

Требования к контроллерам и средства реализации их

Адаптивность: к виду интерфейса к методу обмена к скоростям передачи данных	Программируемый интерфейс Адаптеры сопряжения и промежуточный интерфейс Внешняя передача синхросигналов; программируемая схема тактирования
Управление различными типами ВЗУ	Программный анализ сигналов состояния накопителя и выработка управляющих сигналов; многофункциональные схемы кодирования информации
Распределенная обработка данных	Буфера для промежуточного хранения данных; память для программ обработки; собственные шины для подключения внешнего оборудования
Расширение функциональных возможностей	Проблемно-ориентированная система команд
Простота программирования	Проблемно-ориентированная система команд; микропрограммируемость контроллера
Низкая стоимость	Многофункциональное использование аппаратных средств; высокая серийность
Минимальное число типов интегральных схем	Реализация контроллера на базе БИС и серийных микропроцессорных комплектов

Выделим основные аспекты многофункциональности контроллера ВЗУ и способы их реализации в многофункциональном контроллере.

1. Реализация управления различными типами накопителей осуществляется за счет общего набора модулей управления.

2. Учет особенностей алгоритмов управления различными типами накопителей осуществляется посредством программной реализации алгоритмов управления.

3. Учет особенностей ВЗУ реализуется в результате применения сменных блоков контроллера, а также введения избыточности в аппаратные и программные средства.

Используемый в работе функционально-структурный подход основывается на первичности функционального назначения системы по отношению к ее реализации и предполагает ряд взаимосвязанных этапов, основными из которых являются: выделение основных и дополнительных функций и их декомпозиция (построение дерева функций); построение операторных моделей каждого уровня декомпозиции системы (позволяет проанализировать совокупность внешних и внутренних связей в системе); построение временных диаграмм активности (позволяет провести анализ временных соотношений между функциональными модулями различных уровней системной организации); эквивалентные преобразования на уровне функциональных модулей (реализация принципа временного совмещения и разделения функций с целью формирования многофункциональных и специализированных модулей); покрытие функциональной структуры системы конструктивными модулями (формирование ряда базовых структур); оценка произведенного покрытия (выбор и оценка вариантов структурной реализации).

Синтез базовых структур многофункционального контроллера ВЗУ микро-ЭВМ. Ранее было отмечено деление операций контроллера на основные и дополнительные. Исходя из многоуровневой организации системы, в которой функционирует контроллер, разделим набор функций на следующие группы: функции сопряжения с ЦП; функции обработки информации в контроллере; функции сопряжения с ВЗУ.

Построим дерево функций многофункционального контроллера ВЗУ микро-ЭВМ.

I уровень F^0 — управление ВЗУ.

II уровень

$$F^0: \begin{cases} F^1_1 & \text{— организация сопряжения с ЦП,} \\ F^2 & \text{— промежуточная обработка информации,} \\ F^3 & \text{— организация сопряжения с ВЗУ.} \end{cases}$$

III уровень

$$F^1: \begin{cases} F^1_1 & \text{— обмен параллельной информацией,} \\ F^1_2 & \text{— формирование и хранение слова состояния канала (ССК),} \\ F^1_3 & \text{— управление обменом;} \end{cases}$$

$$F^2: \begin{cases} F^2_1 & \text{— хранение параллельной информации,} \\ F^2_2 & \text{— обработка принимаемой информации;} \end{cases}$$

$$F^3: \begin{cases} F^3_1 & \text{— управление приводом,} \\ F^3_2 & \text{— обработка последовательной информации.} \end{cases}$$

IV уровень

$$\begin{aligned}
 F_1^1: & \begin{cases} F_{1,1}^1 & \text{— прием параллельной информации из ЦП,} \\ F_{1,2}^1 & \text{— передача параллельной информации в ЦП.} \\ F_{1,3}^1 & \text{— хранение (кратковременное) передаваемой информации;} \end{cases} \\
 F_2^1: & \begin{cases} F_{2,1}^1 & \text{— прием слова состояния канала (ССК),} \\ F_{2,2}^1 & \text{— передача ССК;} \end{cases} \\
 F_3^1: & \begin{cases} F_{3,1}^1 & \text{— анализ поступающих сигналов,} \\ F_{3,2}^1 & \text{— выдача управляющих сигналов;} \end{cases} \\
 F_1^2: & \begin{cases} F_{1,1}^2 & \text{— прием передаваемых данных,} \\ F_{1,2}^2 & \text{— хранение передаваемых данных,} \\ F_{1,3}^2 & \text{— прием служебной информации,} \\ F_{1,4}^2 & \text{— хранение служебной информации;} \end{cases} \\
 F_2^2: & \begin{cases} F_{2,1}^2 & \text{— анализ слова состояния ВЗУ,} \\ F_{2,2}^2 & \text{— формирование управляющего слова ВЗУ,} \\ F_{2,3}^2 & \text{— анализ информации, передаваемой с ВЗУ;} \end{cases} \\
 F_1^3: & \begin{cases} F_{1,1}^3 & \text{— передача управляющего слова на ВЗУ,} \\ F_{1,2}^3 & \text{— прием слова состояния ВЗУ;} \end{cases} \\
 F_2^3: & \begin{cases} F_{2,1}^3 & \text{— кодирование информации,} \\ F_{2,2}^3 & \text{— декодирование информации,} \\ F_{2,3}^3 & \text{— формирование циклического кода контроля (ЦКК),} \\ F_{2,4}^3 & \text{— опознавание маркеров,} \\ F_{2,5}^3 & \text{— параллельно-последовательное преобразование информации.} \end{cases}
 \end{aligned}$$

Выше приведено множество функций, реализуемых многофункциональным контроллером (МФК), и их декомпозиция до IV уровня. Каждый из уровней декомпозиции представлен множеством функций $\{F_N^i\}$, которые могут быть разделены на подмножества: $\{P_j^i\}$ — обработки, $\{M_k^i\}$ — хранения, $\{C_n^i\}$ — управления, $\{T_m^i\}$ — обмена. Выделенные подмножества функций Р, М, С, Т-типов для каждого i-го уровня декомпозиции удовлетворяют следующим соотношениям:

$$\{F_N^i\} = \{P_j^i\} \cup \{M_k^i\} \cup \{C_n^i\} \cup \{T_m^i\},$$

$$N = j + k + n + m,$$

где i — номер уровня декомпозиции; j, k, n, m — размерности выделенных подмножеств.

Имеет смысл провести более детальный анализ функции МФК по управлению ВЗУ и интерфейсу с ЦП. С учетом приведенных данных

декомпозиция наиболее существенных функций на V и VI уровнях выглядит следующим образом:

V уровень

- $$F_{2.1}^3 : \begin{cases} F_{2.1.1}^3 & \text{— параллельно-последовательные преобразования,} \\ F_{2.1.2}^3 & \text{— формирование смешанного потока данных и информации,} \\ F_{2.1.3}^3 & \text{— исключение битового сдвига,} \\ F_{2.1.4}^3 & \text{— исключение синхробитов при записи маркеров;} \end{cases}$$
- $$F_{2.2}^3 : \begin{cases} F_{2.2.1}^3 & \text{— выделение такта воспроизведения,} \\ F_{2.2.2}^3 & \text{— выделение последовательных данных,} \\ F_{2.2.3}^3 & \text{— выделение последовательных синхрои́мпульсов;} \end{cases}$$
- $$F_{2.3}^3 : \begin{cases} F_{2.3.1}^3 & \text{— сдвиг данных,} \\ F_{2.3.2}^3 & \text{— сложение по модулю 2 отдельных разрядов контрольного кода с входным битом,} \\ F_{2.3.3}^3 & \text{— проверка конечного кода на 0;} \end{cases}$$
- $$F_{2.4}^3 : \begin{cases} F_{2.4.1}^3 & \text{— последовательно-параллельные преобразования синхросигнала и данных,} \\ F_{2.4.2}^3 & \text{— анализ кода синхробитов,} \\ F_{2.4.3}^3 & \text{— анализ кода данных,} \\ F_{2.4.4}^3 & \text{— установка признака наличия маркера.} \end{cases}$$

VI уровень

- $$F_{2.1.1}^3 : \begin{cases} F_{2.1.1.1}^3 & \text{— прием слова данных,} \\ F_{2.1.1.2}^3 & \text{— сдвиг вправо,} \\ F_{2.1.1.3}^3 & \text{— счет битов,} \\ F_{2.1.1.4}^3 & \text{— выдача запроса данных;} \end{cases}$$
- $$F_{2.1.2}^3 : \begin{cases} F_{2.1.2.1}^3 & \text{— проверка разрешения записи,} \\ F_{2.1.2.2}^3 & \text{— анализ битовой информации,} \\ F_{2.1.2.3}^3 & \text{— проверка такта,} \\ F_{2.1.2.4}^3 & \text{— выдача импульсов записи;} \end{cases}$$
- $$F_{2.1.3}^3 : \begin{cases} F_{2.1.3.1}^3 & \text{— проверка комбинации битов;} \\ F_{2.1.3.2}^3 & \text{— проверка возможности сдвига бита,} \\ F_{2.1.3.3}^3 & \text{— сдвиг импульса;} \end{cases}$$
- $$F_{2.1.4}^3 : \begin{cases} F_{2.1.4.1}^3 & \text{— проверка записи маркера,} \\ F_{2.1.4.2}^3 & \text{— определение типа маркера,} \\ F_{2.1.4.3}^3 & \text{— исключение импульса синхросигнала.} \end{cases}$$

Характеристики контроллера ВЗУ и интерфейса

Сопряжение контроллера с ЦП				Функциональные характеристики ВЗУ				Сопряжение контроллера с ВЗУ			
Тип интерфейса		Метод обмена кодами с ЦП		Обмен информацией		Контроль информации		Управление приводом ВЗУ			Управление передачей данных
Число линий передачи	Назначение линий передачи	Последовательность управляющих сигналов	Набор управляющих сигналов	Быстродействие ВЗУ	Формат записи на носитель	Исправление ошибок	Замена дефектных дорожек	Временные характеристики позиционирования	Анализ сигналов состояния привода	Назначение управляющих выводов	

Основные характеристики контроллера ВЗУ и интерфейса представлены в табл. 8-3.

Взяв за основу полученное дерево функций (древовидную модель) контроллера, можем приступить к формированию операторных моделей.

На рис. 8-7 приведена обобщенная операторная модель многофункционального контроллера с учетом проведенной декомпозиции базовых функций и использованием функциональных операторов Р, С, М, Т.

На операторной модели с учетом выполненной декомпозиции выделены функциональные модули высокого уровня: $\Phi M_{\text{сопр1}}$ — модуль сопряжения контроллера с ЦП; $\Phi M_{\text{обр}}$ — модуль обработки последовательных данных, $\Phi M_{\text{сопр2}}$ — модуль сопряжения с ВЗУ. Построение операторной модели сопряжено с учетом следующих характеристик: передачи (T_i), обработки (P_i) и хранения (M_i) данных: D_z — данных записи, D_n — данных воспроизведения, D_k — данных команд, $D_{c.k}$ — служебной информации. Кроме того, должны быть отражены: X_i ($i = 1 \div 3$) — логические условия протекания информационных процессов, Y_i ($i = 1 \div 3$) — микрокоманды, микрооперации, а также управляющие сигналы интерфейса (Y_n) и сигналы управления накопителей ($Y_{\text{нак}}$).

Операторная модель может быть использована для формирования структуры системы на основе функциональных модулей (ФМ). С целью синтеза базовых функциональных структур строятся временные диаграммы активности на уровне базовой операторной модели (рис. 8-8).

Учитывая проведенный анализ функций и то обстоятельство, что контроллер функционирует в трехуровневой иерархической системе, в функциональной структуре контроллера можно выделить следующие блоки: блок сопряжения с ЦП (БС); байтовый процессор (БП) с блоком буферной памяти (ББП); блок сопряжения с ВЗУ битового процессора (БГП) и контроллера прямого доступа (КПД).

Временные диаграммы активности отражают возможность временного согласования основных функциональных блоков контроллера с учетом внутрисистемных процессов.

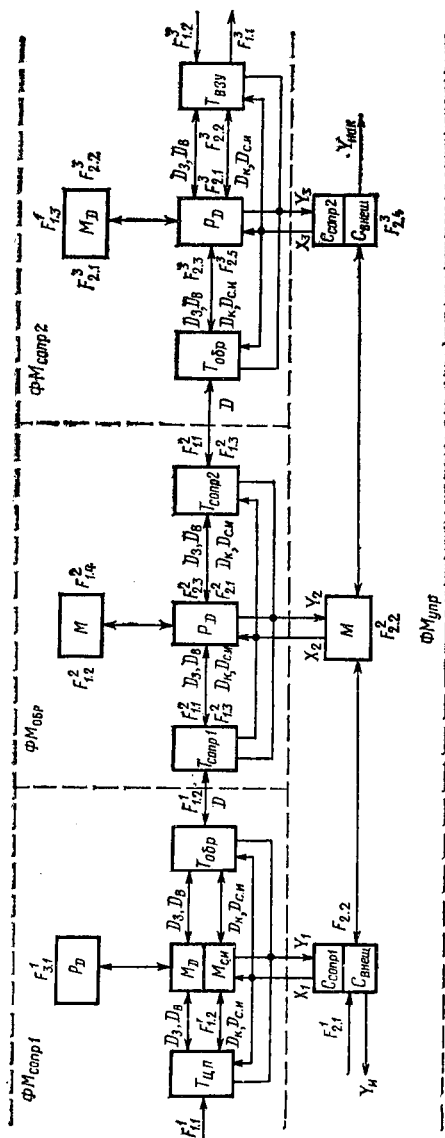


Рис. 8-7. Операторная модель многофункционального контроллера.

Рис. 8-8. Временные диаграммы активности базового набора функциональных модулей

При подключении ВЗУ к вычислительной системе контроллер, как показал анализ функций, реализует две группы операций: операции над данными и операции по управлению приводом.

В контроллере выполняются следующие операции над данными: синхронная передача последовательных данных между ВЗУ и контроллером; обработка параллельных данных; передача слов данных между контроллером и ЦП.

Формирование и сравнительный анализ базовых структур многофункционального контроллера. Анализ операторных моделей указывает на различные временные соотношения при обработке данных. Одной из основных задач контроллера является временное согласование этих процессоров. Каждый блок обобщенной структуры характеризуется временем выполнения процесса: байтовый процессор (БП) — временем обработки слова данных ($T_{обр}$), битовый процессор (БтП) — временем формирования слова ($T_{сл}$), блок сопряжения (БС) — временем обмена словом ($T_{обм}$).

В зависимости от временных соотношений между процессами могут быть предложены различные варианты базовых структур с учетом проведенного ранее анализа.

Первая из предлагаемых структур (рис. 8-9, а) соответствует случаю, когда устройство управления (УУ) обладает достаточным быстродействием для обработки и передачи слова данных без промежуточно-го хранения:

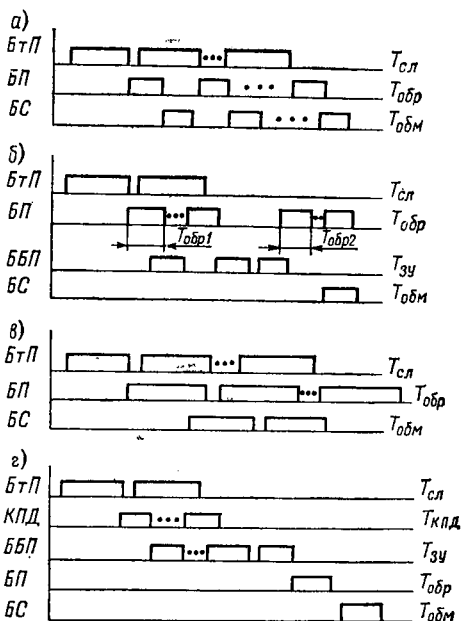
$$T_{сл} \geq T_{обр} + T_{обм}.$$

Если УУ не успевает управлять обработкой и передачей данных в период формирования одного слова, т. е.

$$T_{сл} < T_{обр} + T_{обм},$$

то возможны два варианта структур. В первом случае (рис. 8-9, б) в структуру включается буфер блока данных (ББД). Этот вариант целесообразно применять, если над данными необходимо выполнить в большом объеме операции по обработке. Если время обработки и время передачи в ЦП меньше времени формирования слова, т. е.

$$T_{обр} < T_{сл}, \quad T_{обм} < T_{сл},$$



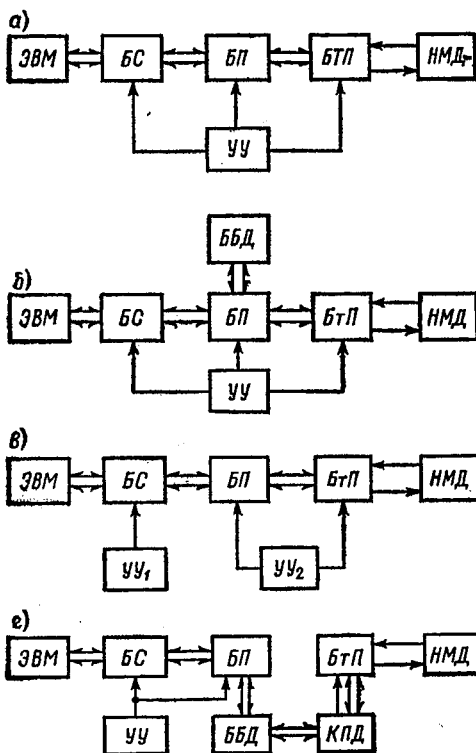


Рис. 8-9. Базовые структуры многофункциональных контроллеров

увеличить быстродействие можно путем включения устройства управления интерфейсом (рис. 8-9, а). В таком случае все три уровня системы работают параллельно.

Четвертая из предлагаемых структур (рис. 8-9, г) соответствует случаю, когда УУ не успевает управлять передачей слов данных в ЦП или буфер: $T_{\text{обр}} \gg T_{\text{сл}}$.

Для обеспечения требуемой пропускной способности необходимо включить контроллер прямого доступа. В этом варианте увеличиваются аппаратные затраты, а время передачи блока данных сокращается.

В предлагаемых структурах не учитываются функции контроллера по управлению механикой накопителя и выработке управляющих сигналов связи с ЦП. Для реали-

зации функций управления необходимо анализировать поступающие в контроллер управляющие сигналы процессорного интерфейса и сигналы состояния накопителя.

В зависимости от того, где анализируются и обрабатываются эти данные, возможны два варианта структур. В первом случае эти сигналы обрабатываются на командном уровне, во втором — анализируются и обрабатываются на микрокомандном уровне в УУ, что позволяет существенно увеличить быстродействие системы. Выбор той или иной структуры зависит от требований по быстродействию и аппаратным затратам, от элементной базы и от функциональной нагрузки контроллера (рис. 8-9).

Рассмотренные базовые структуры отражают разнообразие возможных вариантов построения контроллеров. Применение микропроцессорных комплектов БИС в качестве элементной базы позволяет сократить сроки разработки и расширить функциональные возможности контроллера. Основной задачей проектирования при этом является разработка битового процессора и соответствующего программного обеспечения.

Богатые функциональные возможности микропроцессорных комплектов БИС позволяют передать контроллеру ряд дополнительных функций и таким образом существенно разгрузить ЦП.

8-5. МИКРО-ЭВМ В СИСТЕМАХ СВЯЗИ

Одной из перспективных областей применения МП и микро-ЭВМ являются системы связи, в которых микро-ЭВМ выполняет функцию обработки при приеме и передаче сообщений и управления периферийными устройствами, например телетайпами.

Однако структура серийно выпускаемых микро-ЭВМ, их быстродействие, система команд, разрядность и другие параметры не всегда соответствуют характеру решаемых задач в системах связи, что приводит к необходимости исследовать принципы разработки структур проблемно-ориентированных однопроцессорных и мультимикропроцессорных систем на основе микропроцессорных комплектов БИС.

Система связи представляет собой комплекс аппаратных и программных средств, предназначенных для установления, ведения и отбоя связи между абонентом-источником и абонентом — приемником сообщений.

Анализ алгоритмов, реализуемых в системе связи, позволил выделить три основные задачи: прием и обработку кадра сообщения по приемному каналу; формирование и выдачу кадра по запросу каналу; обслуживание периферийных устройств (телетайпы, пульт управления оператора и др.). Причем возможны режимы работы системы связи, когда эти три задачи необходимо решать одновременно.

Основными функциями при решении вышеперечисленных задач являются следующие: фазирование по циклам, идентификация признаков служебной информации, кодирование-декодирование информации, обмен информацией с периферийными устройствами.

Автоматическое установление связи возможно при условии, если передаваемая информация представлена в формализованном виде. Каждое сообщение имеет информационную и служебную часть (рис. 8-10, а). Служебная часть (заголовок), содержащая специальные признаки, обеспечивает автоматическое установление телефонной или телеграфной связи. Структура заголовка определяет состав, расположение и взаимосвязь элементов, с помощью которых обеспечивается автоматическая передача, контроль, обработка сообщений, и содержит все сведения, необходимые для передачи сообщения от отправителя к



Рис. 8-10. Структура сообщения (а) и состав кадра (б)

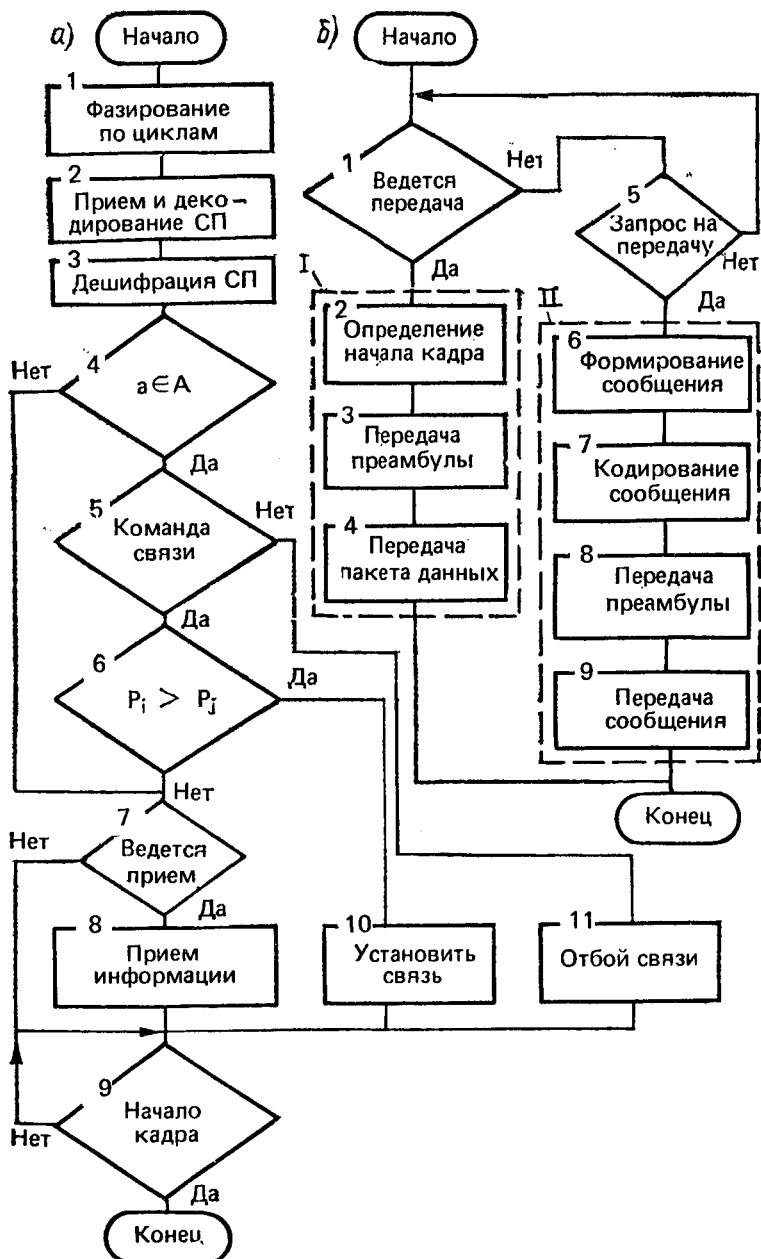


Рис. 8-11. Алгоритм обработки кадра сообщения в приемном канале (а) и в передающем (б)

получателю. Все признаки в заголовке объединены в функциональные группы, имеющие общее смысловое значение. Первая группа определяет адреса получателей, причем для передачи циркулярных сообщений введены обобщенные адреса для групп абонентов. Вторая группа содержит признаки, обеспечивающие приоритетность и выбор типа связи, например: установить телеграфную связь с симплекс-приемом, отбой телефонной связи и т. п. Третья группа определяет параметры связи, а признаки четвертой группы повышают достоверность приема заголовка.

Информация по каналу связи поступает в микропроцессорную систему (МПС) в виде последовательности кадров, имеющих постоянный формат и содержащих поле фазового пуска, поле служебных признаков с адресами абонентов — приемников сообщений, приоритетами, типом связи, а также информационное поле, распределенное по временным каналам (рис. 8-10, б).

Один из вариантов алгоритма обработки кадра сообщения в приемном канале представлен на рис. 8-11, а, в передающем канале — на рис. 8-11, б, где блок I соответствует передаче телеграфного сообщения, а блок II — передаче запросного сообщения.

Если адрес «а» в служебном поле принадлежит множеству адресов А абонента—получателя сообщения, то необходимо продолжить дешифрацию признаков данного поля. Связь, указанная в служебном поле, устанавливается при условии, что ее приоритет P_i выше, чем приоритет текущей связи P_j . Прием телеграфной информации состоит в поиске выделенного временного канала и в приеме определенного числа символов в память МПС из данного канала для дальнейшего вывода на телетайп.

Для передачи сообщения необходимо сформировать запрос, содержащий следующие сведения: адрес абонента, передающего запрос, адрес абонента—получателя сообщения, тип канала предполагаемой связи, приоритет сообщения. При этом запросное сообщение передается с использованием помехоустойчивого кода. По запросу абонента выделяется временной интервал (канал) длительностью τ_j с периодом передачи $T_{\text{прд}}$, в течение которого должны передаваться только символы от данного абонента. Получив разрешение и номер канала (данная информация поступает по приемному каналу и содержится в служебном поле кадра), необходимо синхронизировать передачу таким образом, чтобы передаваемые сигналы попадали точно в отведенный интервал времени. Для этого по приемному каналу передается специальная комбинация сигналов, являющаяся цикловой комбинацией для всех абонентов-источников. Так как период $T_{\text{прд}}$ больше чем период повторения временных интервалов в приемном канале, то передача информации от абонента может произойти в любой момент обработки кадра приемного канала. Информация, подлежащая передаче, хранится в памяти или накапливается в течение времени $T_{\text{прд}}$, а затем передается в виде пакета за время τ_j . При этом сначала выбирается преамбула (вводная часть), содержащая последовательности для восстановления несущей и тактовой синхронизации символов и кодовое слово для фазирования по циклам.

Анализ реализации основных функций системы связи позволяет выделить ряд особенностей, которые необходимо учитывать при построении МПС. МПС должна обрабатывать непрерывный поток кадров, имеющих постоянную структуру в реальном режиме времени. Обслуживание приемного и передающего каналов связи требует мультипрограммного режима работы МПС.

На быстродействие МПС оказывает влияние несоответствие разрядности процессора и обрабатываемых данных при выполнении некоторых функций. Например, поиск синхросигнала требует выполнение логических операций над кодами разрядностью 20—30 бит, что приводит к последовательно-параллельной обработке их на 8- или 16-битных МП. Увеличение же разрядности МП при использовании секционных процессорных элементов типа К584, К589 нецелесообразно, так как другие функции используют операнды значительно меньшей разрядности (6—8 бит).

Широкий диапазон требований к системе связи по таким характеристикам, как скорость передачи информации, тип используемых помехоустойчивых кодов, число каналов связи, требует разработки семейства МПС, удовлетворяющих заданным требованиям.

Структурная схема МПС для абонентского пункта системы связи представлена на рис. 8-12. Входной и выходной регистры выполняют функции последовательно-параллельного и параллельно-последовательного преобразования информации, принимаемой и передаваемой соответственно в канал связи. Память данных используется для временного хранения служебного поля кадра принимаемой и передаваемой информации. МПС работает в мультипрограммном режиме и реализует все функции программным путем, время ввода и вывода кодов с регистров ограничено периодом поступления битов кадра.

Рассмотрим подробно реализацию двух типичных функций: фазирования по циклам и кодирования-декодирования информации. Подпрограмма установления и контроля синхронизма реализует этапы вхождения в синхронизм и последующий его контроль. Условия вхождения в синхронизм могут быть различными, например правильный прием фазирующей комбинации в k последовательно принятых кадрах. Срыв синхронизации фиксируется в случае неприятия определенной комбинации сигналов фазового пуска (ФП) в m последовательно принятых кадрах. Сложность синхронизации определяется этими условиями. Однако наибольшие затраты времени происходят на приеме ФП, т. е. при определении, является ли входная комбинация совпадающей с эталон-

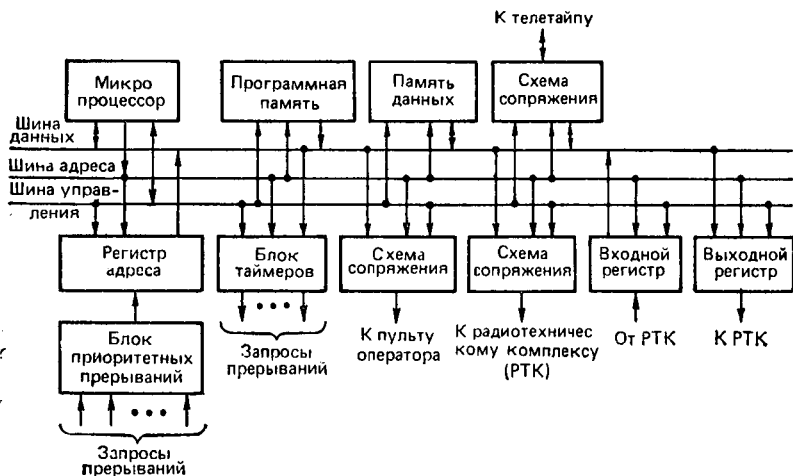


Рис. 8-12. Структурная схема МПС абонентского пункта системы связи

ным ФП. Например, комбинация ФП имеет длину 1 бит и условие правильного приема — наличие не более g ошибок при сравнении с эталоном. Дополнительной трудностью при реализации такой подпрограммы является несовпадение длины ФП и разрядности микро-ЭВМ, а также последовательная побитная обработка кода. После определения первой фазирующей комбинации посредством побитного просмотра поступающей из РТК информации и сравнения с эталоном ФП запускается таймер для отработки времени Δt , равного длительности кадра. После срабатывания таймера во входном регистре находится предполагаемый код ФП следующего кадра. Такое решение позволяет не просматривать на наличие ФП все поле кадра, так как существует вероятность ложного обнаружения комбинации фазового пуска, т. е. потери синхронизации приема кадров.

В целях повышения достоверности приема и передачи кадров заголовков каждого из них, как правило, передается в помехоустойчивом коде. Процедура декодирования служебного поля, например, при использовании кода БЧХ сводится к делению одного многочлена на другой. Остаток от деления при кодировании является проверочным многочленом, а при декодировании — синдромом ошибок. Равенство синдрома ошибки нулю соответствует отсутствию ошибок. Операция деления сводится к сдвигам и суммированию по mod 2 делимого и делителя. МП выполняет операции декодирования заголовка принимаемых кадров и кодирования запросного сообщения. Таким образом, реализация функций синхронизации, кодирования, декодирования не требует выполнения команд умножения, деления (доля команд сложения и вычитания тоже мала, менее 8 %), а сводится к таким командам, как логические, команды передачи, управления. После декодирования служебного поля выполняется его дешифрация, целью которой является определение адреса абонента, кода несущей вызывного канала, типа команды, вида канала, приоритета сообщения и других признаков.

В табл. 8-4 приведен процентный состав команд для выполнения функций абонентского пункта, а в табл. 8-5 — сравнительные характеристики программных модулей данных функций для МП КР580 и К589.

Для МП К589 время микрокомандного цикла составляет 300 нс, формат микрокоманды (4 байта) содержит помимо обязательных по-

Таблица 8-4

Состав команд для реализации функций абонентского пункта

Функция	Доля операций, %				
	пересылки данных	арифмети- ческих	логических и сдвига	управления	вывода-вывода
Кодирование	59	8	17	12	4
Декодирование	36	14	18	25	7
Синхронизация	24	10	24	29	13
Идентификация признаков за- головка	32	8	25	25	10
Разуплотнение	24	12	24	22	18
Прием и обработка кодовой комбинации с ТТУ	32	10	20	18	20
Передача пакета на РТК	36	10	18	20	16

дей поле маски и управляющих сигналов. На основании полученных результатов при проектировании МПС на основе двух вариантов МП можно сделать следующие выводы: системы команд МП КР580 и К589 являются избыточными для реализации задач системы связи; средняя скорость при реализации МПС на основе рассмотренных микропроцессоров составляет 1,2К и 9,6К бод для МП КР580 и МП К589 соответственно; объем программной памяти составляет 1600 байт для МПС на основе КР580 и около 3500 байт для МПС на основе К589.

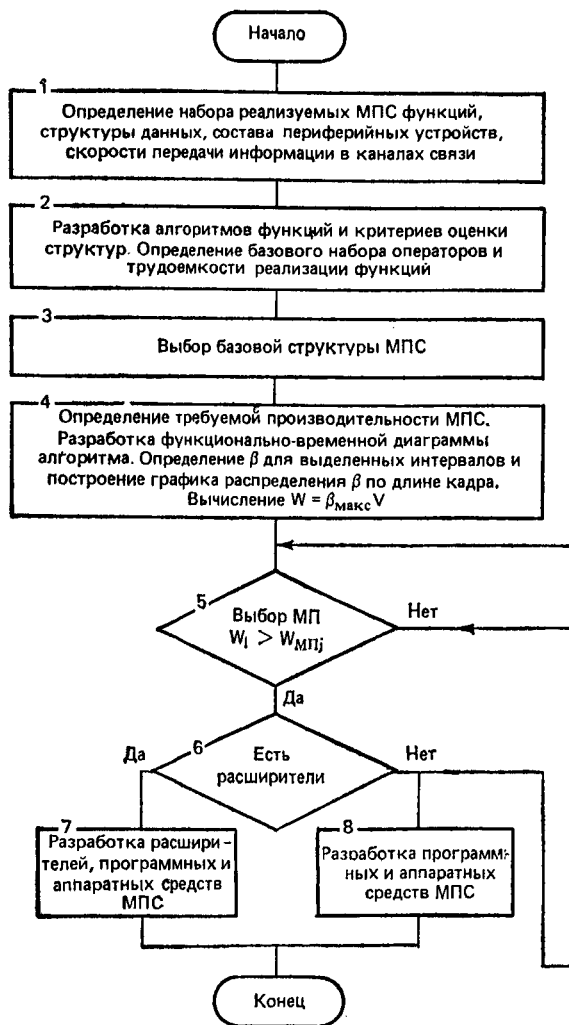


Рис. 8-13. Этапы проектирования МПС системы связи

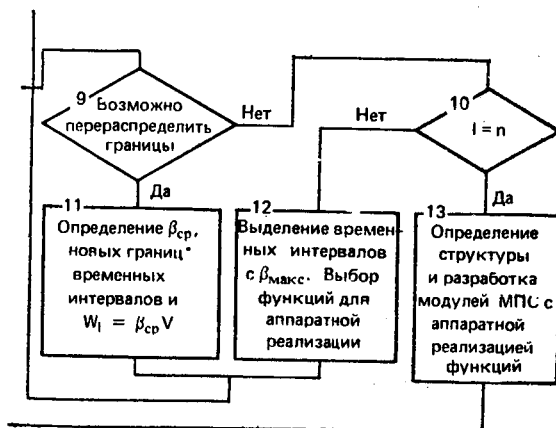
Таблица 8-5

Характеристики программных модулей

Функция	Тип МП	Число команд (микрокоманд)	Объем памяти, байт	Номенклатура команд (микрокоманд)	Время выполнения, мкс
Кодирование запроса	KP580	53	96	15	115*
Декодирование заголовка	K589	57	228	16	12,3*
	KP580	28	53	14	80*
Синхронизация	K589	30	120	14	10,5*
	KP580	120	229	22	650
Идентификация признаков заголовка и определение режима работы	K589	118	472	19	63,2
	KP580	182	333	17	440
	K589	170	680	18	38,4
Разуплотнение	KP580	14	24	16	108
	K589	17	68	16	12
Прием и обработка кодовой комбинации с ТТУ	KP580	30	54	18	132
	K589	25	100	17	14
Передача файла данных на РТК	KP580	41	59	12	972
	K589	40	160	12	102

Примечание: звездочкой отмечено время обработки одного бита.

Анализ однопроцессорного варианта МПС показал удовлетворительное быстродействие системы, но поскольку имеет место тенденция



увеличения скоростей передачи данных в каналах связи, то встает вопрос повышения производительности МПС. Одним из путей достижения этой цели является использование специализированных расширите-

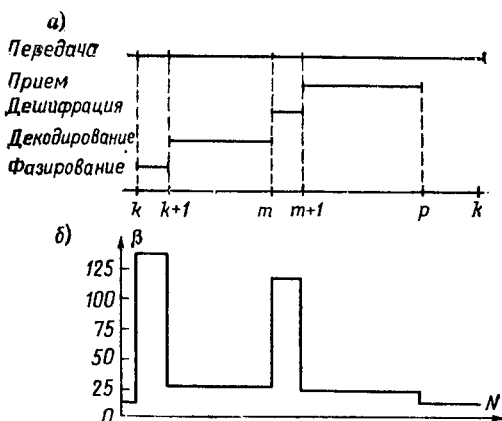


Рис. 8-14. Функционально-временная диаграмма распределения этапов при обработке кадра (а) и распределения β (б)

лей, выполняющих наиболее трудоемкие функции или проектирование мультимикропроцессорных систем.

Методика проектирования МПС для системы связи. Зависимость скорости передачи V , обеспечиваемой МПС в канале связи, от производительности МПС W можно определить следующим образом:

$$V = W/\beta,$$

где β — удельная сложность алгоритма обработки.

Под удельной сложностью алгоритма обработки будем понимать число команд, выполняемых за время приема одного бита информации:

$$\beta = Q_{\text{макс}}/N,$$

где $Q_{\text{макс}}$ — максимальная трудоемкость алгоритма, определяемая числом команд; N — число бит, принятое в интервале времени, отведенном для реализации алгоритма обработки.

Удельная сложность алгоритма обработки на отдельных интервалах кадра изменяется в широких пределах, так как это связано с выполнением неодинакового числа функций различной сложности. Реальная производительность микропроцессора при заданной скорости передачи вычисляется с учетом $\beta = \max(\beta_1, \beta_2, \dots, \beta_\alpha)$, где α — число временных интервалов. Уменьшить β возможно за счет перераспределения границ интервалов в рамках кадра. При этом β стремится к $\beta_{\text{ср}}$, определяемому формулой

$$\beta_{\text{ср}} = \frac{\sum_{i=1}^{\alpha} Q_i}{\sum_{i=1}^{\alpha} N_i},$$

где Q_i — трудоемкость алгоритма обработки на i -м временном интервале; N_i — число бит в i -м временном интервале.

Генерация вариантов структуры МПС происходит в следующей последовательности: кадр разбивается на временные интервалы; для каждого интервала определяются реализуемые функции; вычисляются значения β ; вычисляется $V_{\text{мин}}$, которое и определяет скорость передачи информации в канале.

На рис. 8-13 приведены основные этапы предлагаемой методики проектирования МПС. В качестве базовой выбрана МПС, в которой все функции реализуются программно и в состав которой входят: микропроцессорный модуль, входное (выходное) устройство для последовательно-параллельных преобразований, память программ и данных. Основой данной методики является функционально-структурный подход. Для этого определяется набор функций, реализуемых МПС, трудоемкость их реализации, вычисляется необходимая производительность МПС, разрабатывается функционально-временная диаграмма (рис. 8-14, а), на которой показаны зоны выполнения функций, приведенных к временным интервалам кадра. Каждый временной интервал кадра характеризуется определенным числом бит. Выделенным участкам соответствуют прием поля фазового пуска $0-k$, служебного поля $(k+1)-m$, обработка которого сводится к декодированию и дешифрации, и прием символов информационного поля из выделенного временного канала $(m+1)-p$. Функции передачи и приема сводятся к выдаче слова из памяти данных в выходной регистр и приему из входного регистра в память, причем запрос на передачу может возникнуть в любой момент времени. Для выделенных интервалов определяется β , строится график распределения β по длине кадра (рис. 8-14, б) и вычисляется требуемая производительность МПС. Выбор МП производится путем сравнения вычисленной производительности с производительностью j -го МП. Иногда число рассматриваемых МП ограничивается на системном этапе.

Сложность алгоритма обработки, высокие скорости передачи и приема требуют поиска решений для увеличения производительности вычислительных средств, например путем аппаратной реализации наиболее трудоемких функций. На рис. 8-14, б $\beta_{\max}$ имеет место на интервале $k-(k+1)$, на котором выполняются две функции: фазирование по циклам и передача слова из памяти в выходное устройство, причем трудоемкость реализации функции фазирования по циклам составляет 130 команд, а передачи — 13 команд. Особенностью фазового пуска является необходимость его реализации за время приема одного бита информации, и поэтому данную функцию необходимо выполнять аппаратно. Затем определяется новое значение β для данного временного интервала и производительность МПС по $\beta_{\max}$ на интервале $m-(m+1)$. Если МП не соответствует требованию производительности, то необходимо перераспределение зон выполнения функций, определяющих значение $\beta_{\max}$.

Таким образом, происходит переход от программной реализации функций к программно-аппаратной и при $l=n$, где n — набор функций, к построению МПС с аппаратной реализацией всех функций.

8.6. ОСНОВНЫЕ ЭТАПЫ ПРОЕКТИРОВАНИЯ АППАРАТНЫХ И ПРОГРАММНЫХ СРЕДСТВ

С появлением МП изменилась не только элементная база, но и подход к проектированию средств ВТ. Анализ опыта разработок систем на базе МПК позволил выделить следующие основные особенности проектирования МПС.

1. Микропроцессоры представляют собой алгоритмически универсальные модули, функциональная ориентация которых определяется посредством разработки соответствующего программного обеспечения. При этом интерфейс данных модулей организуется по достаточно разработанным рекомендациям и сроки разработки аппаратной части системы значительно сокращаются. Использование МП является скачком от схем малой и средней степени интеграции с «жесткой» структурой в область

программирования, к структурным схемам уровня процессор — память — коммутатор.

2. Функциональная ориентация системы требует больших затрат времени и средств на создание прикладных и системных программ для каждой конкретной области. Это усугубляется тем, что микропроцессорные системы начинают широко использоваться в различных областях, где ранее применение средств ВТ было невозможным или экономически нецелесообразным. Практика разработки МПС показывает, что доля экономических затрат на разработку программ по отношению к затратам на разработку аппаратуры неуклонно увеличивается. Например, стоимость разработки программного обеспечения в среднем составляет 60—80 % общей стоимости разработок МПС.

3. Появилась тенденция, направленная на реализацию функций программного обеспечения аппаратными средствами. Примером этого является разработка БИС, ориентированных на управление периферийными устройствами, такими, как дисплей, печатающие устройства, БИС обработки прерываний, а также разработка БИС для математических вычислений, имеющих специальный набор команд. Данные БИС позволяют расширить области применения МПС и повысить их производительность. Так, фирма Intel разработала микропроцессорный модуль 8087, ориентированный на быстрое выполнение арифметических операций с плавающей запятой над числами до 64 разрядов и вычисление тригонометрических функций. Отличительной чертой вспомогательных БИС является их программируемость, а также то, что их сложность соответствует сложности МП или даже выше ее. Например, МП КР580 содержит 5000 транзисторов, контроллер управления режимом прямого доступа к памяти содержит 4000 транзисторов, а БИС для подключения дисплеев с растровым сканированием к МП КР580—15 000 транзисторов. Это способствовало развитию проектирования систем с расширителями.

4. Использование МП при проектировании систем изменяет процесс их отладки с перенесением центра тяжести на отладку средств программного обеспечения. Это требует таких основных программных средств, как транслятор с языка ассемблера или языка высокого уровня, программ редактирования, моделирования, загрузки.

5. С появлением микропроцессорных наборов модулей изменились критерии проектирования систем, важнейшим из которых являются: критерий модульности, заключающийся в организации МПС с минимальным возможным числом типов различных компонентов; относительная регулярность структуры связей между модулями при минимальном числе связей, что снижает требования к числу выводов модуля БИС; требование «поддержки» систем, тестирование и диагностика которых становятся все более трудными и важными.

Этапы проектирования аппаратных и программных средств микропроцессорных систем (рис. 8-15). Одним из важнейших этапов является системный анализ, цель которого состоит в определении требований к проектируемой системе: определить набор функций, которые должны выполняться с помощью системы; необходимую производительность; границы стоимости на этапах разработки, изготовления, обслуживания системы; уточнить критические функции в соответствии с критериями разрабатываемой системы; проанализировать состав периферийных устройств и режимы их работы; установить характер связей отдельных функций (подзадач) системы, например последовательные, параллельно-последовательные, связь по прерыванию; определить структуру и «происхождение» данных, а также требования к выходным данным и необходимости их хранения.

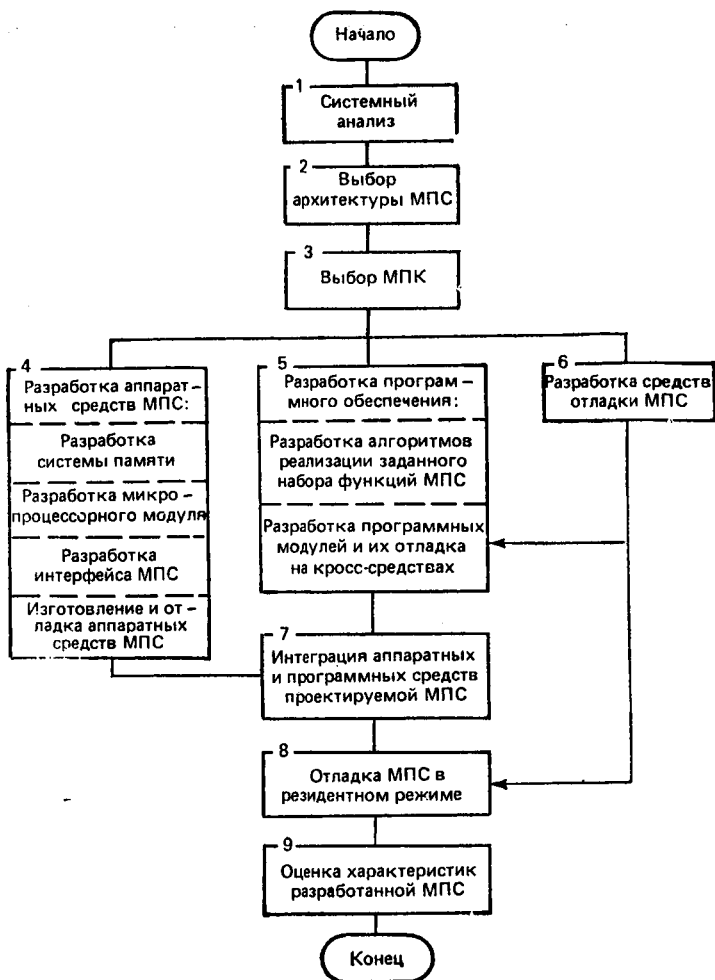


Рис. 8-15. Этапы проектирования аппаратных и программных средств МПС

На втором этапе согласно анализу определяются требования к архитектуре системы и предварительно решается задача реализации выделенных функций программным и аппаратным способом. Трудность решения данного вопроса часто состоит в том, что имеется множество новых применений МП, которые требуют исследования до принятия решения.

Кроме того, часто приходится проводить синтез модулей для реализации функций аппаратным путем из-за отсутствия в микропроцессорных комплектах соответствующих БИС.

Следующий этап проектирования — выбор МПК, который необходимо осуществлять, исходя из трех основных моментов.

1. С точки зрения разработки программного обеспечения необходимо анализировать такие характеристики микропроцессора, как набор команд и способы адресации, разрядность, число используемых регистров общего назначения, тип стековой памяти, способы обработки прерываний и т. п. Для секционных МП с микропрограммным уровнем необходимо, если требуется, выбрать систему команд, разработать микропрограммы их реализации, что связано с определенными трудностями, так как разработчику микропрограмм необходимо выбрать формат микрокоманд, хорошо изучить механизмы формирования адресов следующих микрокоманд, тракты передачи кодов и возможные совмещения при передачах, учитывать задержки.

2. Исходя из системного проектирования, необходимо анализировать полноту комплекта, который должен содержать, помимо микропроцессора, БИС ПЗУ и ОЗУ, модули для организации интерфейса с периферийными устройствами, контроллеры для обработки прерываний, управления прямым доступом к памяти, шинные формователи, буферные регистры, генератор тактов, системный контроллер. В табл. 8-6 приведено влияние различных критериев выбора МПК на разработку и производство МПС.

3. С точки зрения разработки и отладки программного обеспечения необходимо наличие таких программ, как транслятор с символического языка на двоичный (объектный) код, редактирующая программа, программа моделирования, без которых время и стоимость разработки МПС значительно увеличиваются. Обычно подобные данные в програм-

Таблица 8-6

**Влияние критериев выбора МПК
на разработку и производство МПС**

Критерий	Характеристики МПК			
	Быстро-действие	Универ-сальность	Стоимость разработки	Стоимость производства
Разрядность	1	1	2	1
Набор команд, вид адресации	2	2	2	3
Архитектура МП	2	2	2	3
Микропрограммируемость	1	1	1	3
Время выполнения команды	1	1	—	2
Вид прерывания	2	1	2	3
Гибкость интерфейсов	2	2	2	3
Выбор ЗУ	—	—	3	2
Документация	—	3	1	3
Полнота серии и микро-схем	3	3	1	1
Требования к потребителю	—	2	3	2

Примечание: 1 — большое влияние, 2 — среднее, 3 — слабое; — не влияет.

ме разрабатываются для других ЭВМ и называются кросс-средствами для разработки программного обеспечения проектируемой МПС. Кроме того, для отладки программ на проектируемой ЭВМ в резидентном режиме необходимо иметь монитор, который позволит управлять с терминала пуском и остановом МПС, осуществлять загрузку памяти, например, с перфоленты, печатать содержимое памяти и регистров микропроцессора, редактировать содержимое ячеек памяти и регистров, выполнять некоторые другие функции. При этом функции монитора могут использоваться не только при отладке, но и при эксплуатации системы для оперативного «вмешательства» человека, например, в процесс управления (смена параметров процесса, индикация каких-либо переменных), иначе говоря, для интерактивной работы человека и управляющей системы.

Необходимость фазы резидентной отладки обусловлена следующими причинами: трудно моделировать зависящие от времени и асинхронные события на кросс-средствах; скорости периферийных устройств в ЭВМ общего назначения фиксированны и известны, а в системах реального времени интенсивности входных потоков часто неизвестны, моменты их поступления определяются управляемым процессом и т. д.; в системах реального времени, как правило, имеет место сложное взаимодействие программ при использовании режимов мультипрограммирования для обработки параллельных процессов или прерываний; из-за повышенных требований к надежности систем недопустима коррекция ошибки во время их работы.

Рассмотрим подробнее этап разработки программного обеспечения МПС, который выполняется параллельно этапу разработки и отладки аппаратных средств МПС. После завершения этих этапов осуществляется интеграция программных и аппаратных средств и отладка МПС в резидентном режиме.

Главной задачей проектирования МПС является создание прикладных и системных программ. Проектирование же аппаратных средств сводится к соединениям БИС в соответствии с рекомендуемыми конфигурациями, расчетом электрических нагрузок и временных соотношений. В настоящее время отсутствуют четкие количественные критерии и характеристики оптимальности программ. Естественная традиционная оценка их по длине и, следовательно, емкости памяти может отступить на второй план по сравнению с удобством эксплуатации систем, гибкостью при расширении ее функциональных возможностей и т. п. Кроме того, удельная емкость полупроводниковой памяти постоянно уменьшается. На практике одним из важных факторов, учитываемых при разработке программных средств, является производительность системы. В общем, пока создание прикладных программ остается «более искусством, чем наукой», и здесь большую роль играют опыт и здравый смысл, которым очень трудно научить.

Разработка прикладных программ МП-систем включает в себя следующие крупные этапы, характерные для разработки любых программных средств: точную постановку решаемой проблемы и определение функциональных спецификаций; разделение проблемы на части и алгоритмизацию каждой из них — так называемое проектирование (или даже конструирование) программы; собственно программирование (кодирование); проверку и отладку программы.

Остановимся на каждом из приведенных этапов.

Разработка функциональных спецификаций. Успех проектирования в значительной мере определяется тем, насколько хорошо разработчик понял и сформулировал решаемую проблему и трансформировал ее в рабочие характеристики системы. Общие спецификации обычно в себя

включают: четкое описание проблемы, решаемой системой; список аппаратных средств и внешних сигналов; описание связей программных модулей; полное описание системы с упором на интерфейс с ВУ; инструкцию для пользователей с описанием входных и выходных данных, реакций на особые случаи и т. п.

Практика показывает, что примерно 70 % ошибок в программах появляется из-за недостаточно полных и четких спецификаций. Такие логические ошибки трудно локализовать и исправлять. Оставшаяся треть приходится на ошибки кодирования, исправлять которые обычно проще. Разработка адекватных спецификаций на самом раннем этапе проектирования экономит много времени и средств. Здесь целесообразно использовать графические представления решаемых задач в виде блок-схемы алгоритмов, диаграмм и таблиц. Наконец, после оформления функциональных спецификаций рекомендуется строго их придерживаться.

Разделение проблемы на части и алгоритмизация. Большинство применений МП-систем связано с решением довольно сложных проблем, поэтому целесообразно разделить общую проблему на более простые и управляемые части. Программная реализация каждой из частей называется блоком или модулем. Сложные блоки разделяются на субблоки до такого уровня, чтобы разработка алгоритма работы каждого субблока стала достаточно простой. Такой пример называется проектированием сверху вниз или нисходящим проектированием.

Основные блоки выделяются из функциональных спецификаций и содержат управляющий блок (основную программу), блоки интерфейса с внешними устройствами (ВУ), блоки реакций на прерывания и блоки разнобразного преобразования данных. Особенное внимание уделяется организации ввода-вывода (ВВ).

Большое значение имеет правильная организация структуры данных. Следует задать и описать форматы входа и выхода, промежуточных и окончательных результатов, возможные варианты упаковки данных, выбрать способы размещения данных в памяти. Целесообразно каким-либо образом «регуляризовать» данные в виде таблиц, массивов, списков и т. п. Рациональная организация данных поможет сократить длину программы и уменьшить время ее выполнения.

После выделения функциональных блоков разрабатываются алгоритмы их работы с ориентацией на выбранный МП. Логiku алгоритмов удобно представлять графическими блок-схемами такого уровня, когда отдельные их элементы соответствуют нескольким машинным командам. Общие рекомендации по разработке алгоритмов включают в себя подробное рассмотрение назначения блока; анализ форматов и возможностей регуляризации входных данных; предварительную обработку данных типа масштабирования, упаковки и т. п.; алгоритмизацию преобразований данных с максимальным использованием имеющихся подпрограмм; определение форматов выходных данных и анализ связей данного блока с другими блоками.

При разработке алгоритмов значительное внимание приходится уделять временным ограничениям, характерным для большинства прикладных программ. Часто производительность системы можно существенно повысить за счет увеличения объема памяти (примером может служить табличная реализация функций).

Современная методика структурного программирования рекомендует составлять программы из небольшого числа стандартных программных структур (последовательная структура, обобщенный цикл и дихотомическое решение). Главной особенностью этих структур является наличие одной точки входа и одной точки выхода, что упрощает объ-

единение структур в сложную программу и облегчает последующие проверку и отладку. На практике эта особенность сводится к требованию минимального числа команд безусловного перехода.

Кодирование. Преобразование блок-схемы программы в операторы некоторого языка, воспринимаемого ЭВМ, традиционно называется программированием. Однако такое преобразование правильно спроектированной программы оказывается не слишком сложным, поэтому данный этап стали называть кодированием. Результатом кодирования является входная программа, или входной код. В зависимости от возможностей дальнейшего преобразования входных программ кодирование может вестись на машинном языке, языке ассемблера или языке высокого уровня. Очевидно, в течение некоторого времени основным языком программирования МП-систем будет язык ассемблера.

Следует тщательно спланировать программу и обратить серьезное внимание на ее документирование. Обычно программа начинается с заголовка-комментария, содержащего основные характеристики программы: имя (название), дату составления, версию, требования к памяти, собственно функции программы и особенности ВВ. После заголовка следуют директивы определения непосредственных данных, назначения адресов портов ВВ, директивы резервирования памяти, таблицы переходов и подпрограммы ВВ. После них размещаются основная программа, представляющая собой последовательность инициализаций и вызовов подпрограмм, затем подпрограммы в порядке уровней вложения и таблицы данных.

Для наглядного представления размещения элементов программы можно построить карту памяти с выделением следующих областей: специальной памяти, определяемой особенностями МП; рабочей памяти стека, подпрограмм ВВ, буферных областей; общих областей для передачи параметров; памяти портов ВВ при использовании ВВ, отображенного на память; резидентной памяти для системных программ. Целесообразно назначать области памяти в пределах страниц, что упрощает дешифрацию адресов и их преобразование.

Проверка и отладка. Тесно связанные друг с другом проверка и отладка программы заключаются в локализации и удалении из программы ошибок. Проверка начинается с тщательного просмотра программы с последующим переходом к отладке подпрограмм низшего уровня, которые не вызывают других подпрограмм. Следует убедиться, что кодирование соответствует логике блок-схемы, что данные правильны, а циклы верно инициализируются и заканчиваются. Необходимо разработать конкретные примеры (тесты) для каждого блока. В тестах предусматриваются наихудшие случаи и предельные значения параметров. Закончив отладку простейших подпрограмм, можно переходить к подпрограммам следующего уровня и так до основной программы. Завершается отладка функциональным тестом, проверяющим работу программы в условиях эксплуатации. Иногда такой тест включается в прикладную программу как средство самоконтроля.

Кросс-средства создания прикладных программ. Кросс-средства представляют собой комплекс программ, позволяющих отладить прикладную программу МПС на универсальной ЭВМ до проектирования или параллельно с проектированием аппаратных средств. В их состав обычно входят программа кросс-ассемблера и моделирующая программа. Кросс-ассемблер осуществляет перевод программы в объектный код. При этом осуществляется синтаксическая и семантическая проверка исходной программы, распечатываются листинги исходной и объектной программ. Кросс-ассемблер представляет собой сложную программу. Например, программа кросс-ассемблера на ЕС-1033 для МП КР580 со-

держит около 1700 операторов языка ПЛ/1, а объем памяти ЭВМ для хранения кросс-ассемблера — 70К байт.

Программа моделирования предназначена для отладки объектных программ проектируемой МПС. В моделирующей программе все регистры МП представляются ячейками памяти, и программа имитирует выполнение каждой команды прикладной программы с соответствующим преобразованием этих ячеек. С помощью команд, вводимых с терминала при интерактивной отладке или с перфоносителя при пакетной отладке, пользователь может выполнять следующие действия: вывод на терминал полной трассы выполняемых команд; распечатку времени реализации отдельных участков или всей программы; останов в контрольных точках программы с выводом диагностической информации; вывод на печать содержимого регистров микропроцессора, памяти слова состояния программы и т. п.

Так, разработанная для МП КР580 программа моделирует один такт работы микропроцессорной системы на ЕС-1033 за 10 мкс, требуемый объем памяти составляет примерно 77К байт, а объем программы моделирования — примерно 800 операторов.

После завершения моделирования объектной программы необходимо вывести объектный код на перфоленгу для последующей его загрузки и отладки в резидентном режиме в МПС.

8-7. ТЕНДЕНЦИИ В РАЗВИТИИ АРХИТЕКТУРЫ МИКРО- И МИНИ-ЭВМ И РАСПРЕДЕЛЕННЫХ СИСТЕМ НА ИХ ОСНОВЕ

В настоящем параграфе обзорно рассмотрены некоторые аспекты развития мини- и микро-ЭВМ. Содержание его не претендует на исчерпывающую полноту, поскольку развитие цифровой вычислительной техники идет стремительными темпами.

Однокристальные микропроцессоры. В конце 70-х годов появились 16-битные микропроцессоры нового поколения, функциональные возможности которых приближаются к возможностям процессоров современных мини-ЭВМ. Архитектура, система команд, адресное пространство и внешние сигналы микропроцессоров удовлетворяют таким требованиям, как работа в режиме мультипрограммирования, организация мультипроцессорных систем, простота реализации языков высокого уровня, позиционно-независимых, реентрантных и динамически перемещаемых программ. Кроме того, расширены форматы данных: биты, байты, слова (16 бит), длинные слова (32 бита), иногда четверенные (64 бита), цепочки байт и слов, упакованные и неупакованные двоично-десятичные числа. Расширена система команд, в частности введены команды умножения и деления. Хотя число базовых команд возрастает незначительно и составляет примерно 100, увеличение числа режимов адресации примерно до 20 привело к тому, что число кодов операций превышает 1000. Расширение функций микропроцессоров и стремление сохранить корпус с 40—48 (редко с 64) контактами привело к широкому использованию мультиплексирования адресов и данных, а также многофункциональных линий.

Однокристальные микро-ЭВМ. Увеличение плотности упаковки позволило разместить на одном кристалле все компоненты ЭВМ: процессор, память, средства ввода-вывода; последние, разумеется, будут обладать ограниченными возможностями. Поэтому однокристальные микро-ЭВМ предназначены для использования в качестве встраиваемых элементов различных изделий.

На рис. 8-16 показана структурная схема микро-ЭВМ 8748 фирмы Intel. В состав этой 8-битной ЭВМ входят: процессор с аккумуляторной

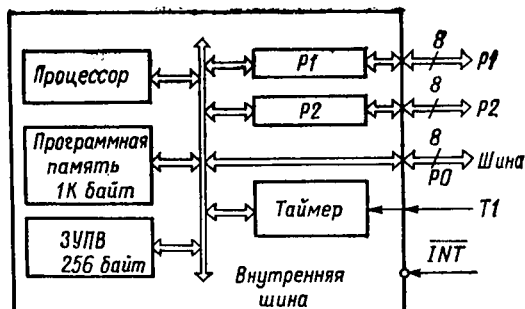


Рис. 8-16. Структура однокристальной микро-ЭВМ

архитектурой и простой системой команд, программная память в виде перепрограммируемого ПЗУ емкостью 1К байт, оперативная память емкостью 256 байт, 8-битный таймер, два программно-управляемых порта ввода-вывода P1 и P2 и двунаправленная шина P0.

Программная и оперативная память логически и физически разделены, в частности отсутствуют команды записи в программную память. При подключении внешних микросхем допускается адресовать программную память до 4К байт и оперативную память 256 байт. Имеется одноуровневая система векторных прерываний, которая воспринимает запросы прерываний от внутреннего таймера и внешнего источника.

Система команд насчитывает 96 команд, большинство которых (75 %) однобайтные, а остальные двухбайтные. Около половины команд выполняется за один цикл (2,5 мкс), а другие — за два цикла. К последним относятся команды с непосредственными операндами и все команды ввода-вывода. Функционально команды делятся на семь групп: аккумуляторные, передачи данных, передачи управления, ввода-вывода, управления микро-ЭВМ (например, разрешения и запрещения прерываний), таймером и флажками.

В разработке микро-ЭВМ использовано несколько интересных новинок, некоторые из которых рассматриваются ниже. Микро-ЭВМ имеет резервный режим работы с пониженным потреблением энергии. Специальная схема позволяет снять питание со всех компонентов, кроме оперативной памяти. В этом режиме БИС потребляет всего 10—15 % номинальной энергии.

Внутренняя схема одношаговой работы дает возможность выполнять программу по командам. После завершения каждой команды микро-ЭВМ останавливается и 12-битный адрес следующей команды выводится на 8 линий шины и 4 младших линии порта P2.

С помощью внутреннего 8-битного таймера обеспечивается подсчет числа внешних событий (сигналов) и генерирование точных временных интервалов без загрузки процессора. Входными сигналами таймера служат либо переходы от низкого уровня к высокому на входе T1, либо внутренние сигналы синхронизации с частотой 12,5 кГц. Сигнал переполнения счетчика устанавливает специальный флажок, состояние которого проверяется командой условного перехода. Этот же сигнал может вызывать прерывание процессора, если оно разрешено специальной командой.

Процессор с объектной ориентацией. Микросхема $\mu\text{APX}432$, разработанная фирмой Intel в 1981 г., приближается по своим возможностям

тям и быстродействию к процессорам больших ЭВМ и архитектурно значительно отличается от прежних процессоров. Микросистема состоит из трех сверхбольших ИС, содержащих более 200 000 транзисторов. Процессор данных образован двумя микросхемами: дешифратором команд iAPX-43201 и операционным устройством iAPX-43202. Они взаимодействуют по двунаправленной шине микрокоманд и реализуют выборку, дешифрацию и выполнение команд программы с быстродействием 200К операций/с. Интерфейсный процессор iAPX-43203 обеспечивает взаимодействие с другими процессорами системы и управляет пакетной шиной, связывающей процессор данных и память. За одну передачу по пакетной шине в режиме «запрос—ответ» передается от 1 до 16 байт данных.

Процессор iAPX-432 имеет обширную систему команд (операторов), которая обрабатывает 8 аппаратно-распознаваемых типов данных: 8-битный символ С, 16-битное короткое целое КЦ (со знаком), 16-битное короткое порядковое КП (без знака), 32-битное целое Ц (со знаком), 32-битное порядковое П (без знака), 32-битное короткое действительное КД (знак, 9-битная характеристика и 22-битная мантисса), 64-битное действительное Д (знак, 11-битная характеристика и 52-битная мантисса) и 80-битное временное действительное ВД (знак, 15-битная характеристика и 64-битная мантисса). Операторы, реализуемые процессором, приведены в табл. 8-7.

Приведенные выше типы данных называются примитивными, так как из них образуются более сложные структуры данных: массивы и записи. Массивы состоят из нескольких элементов одного и того же типа, т. е. допускаются массивы целых, символов и т. д. Записи имеют несколько полей, которые могут представлять данные различных типов, например запись может состоять из символа, целого и действительного чисел. В процессоре нет аппаратных команд, рассчитанных на структурированные типы данных, но предусмотрены механизмы их простой обработки. Обращения к каждому примитивному типу данных реализуются с помощью специальных режимов адресации, которые упрощают выбор отдельных элементов из массивов и записей.

Команды определяют операторы и операнды, причем в некоторых операторах требуется до трех операндов. Команды размещаются в аппаратно-распознаваемых командных сегментах памяти. Для процессора данных командные сегменты представляют собой смежные цепочки бит, называемых потоком команд. Команды имеют переменную длину и размещаются в потоке команд произвольно, начиная с любого бита. Про-

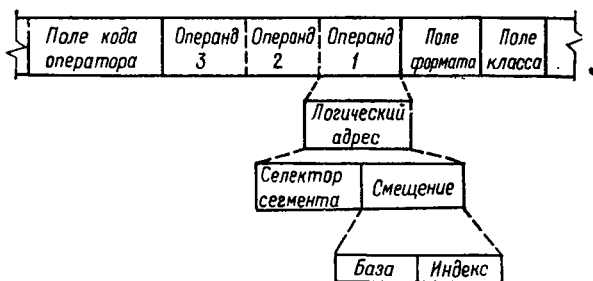


Рис. 8-17. Общий формат команды

Операторы и типы данных процессора iAPX-432

Оператор	Типы данных							
	С	КП	П	КЦ	Ц	КД	Д	ВД
Переслать	×	×	×	×	×	×	×	×
Сохранить	×	×	×	×	×	×	×	×
Сбросить	×	×	×	×	×	×	×	×
Загрузить единицы	×	×	×	×	×			
Логическое И	×	×	×	—	—	—	—	—
Логическое ИЛИ	×	×	×	—	—	—	—	—
Логическое «исключающее ИЛИ»	×	×	×	—	—	—	—	—
Инверсия	×	×	×	—	—	—	—	—
Сложить	×	×	×	×	×	*	*	×
Вычесть	×	×	×	×	×	*	*	×
Умножить		×	×	×	×	*	*	×
Разделить		×	×	×	×	*	*	×
Вычислить остаток		×	×	×	×			×
Инкремент	×	×	×	×	×	—	—	—
Декремент	×	×	×	×	×	—	—	—
Изменить знак	—	—	—	×	×	×	×	×
Найти абсолютное значение	—	—	—			×	×	×
Извлечь квадратный корень								×
Выделить бит		×	×	—	—	—	—	—
Вставить бит		×	×	—	—	—	—	—
Проверить старший бит		×	×	—	—	—	—	—
Проверить на равенство	×	×	×	×	×	×	×	×
» » неравенство	×	×	×	×	×	×	×	×
» » равенство нулю	×	×	×	×	×	×	×	×
» » неравенство нулю	×	×	×	×	×			
» » больше	×	×	×	×	×	×	×	×
» » больше или равно	×	×	×	×	×	×	×	×
» » плюс	—	—	—	×	×	×	×	×
» » минус	—	—	—	×	×	×	×	×
Преобразовать в символ	×	—	×					
» в короткое порядковое		×	—		×			×
» в порядковое в короткое целое				—	×			

Оператор	Типы данных							
	С	КП	П	КЦ	Ц	КД	Д	ВД
Преобразовать в символ								
» в целое			×	×	—			×
» в короткое действительное						—		×
» в действительное							—	×
» во временное действительное		×	×	×	×	×	×	—

Примечание. В таблице приняты следующие обозначения: × — оператор есть для данного типа данных; * — оператор есть для данного типа данных и операций, где один из операндов — временное действительное; — оператора нет, он бесполезен даже при его наличии; пробел — оператора нет.

цессор выполняет считывание из командного сегмента блоками по 32 бита.

Общий формат команды показан на рис. 8-17. Смещение в команде имеет два подполя: базу и индекс. Содержимое их трактуется либо прямо (подполе содержит само значение), либо косвенно (подполе содержит указатель значения). Прямое обращение в обоих подполях эквивалентно однокомпонентному смещению, которое используется для обращения к неструктурированным данным (скалярам). Косвенное обращение допускается объединять с прямым обращением для управления доступом к структурированным данным. Четыре комбинации прямого и косвенного обращения называются режимами адресации: базовая прямая и индексная прямая (для обращения к скалярам); базовая косвенная и индексная прямая (для обращения к записям); базовая прямая и индексная косвенная (для обращения к статическим массивам); базовая косвенная и индексная косвенная (для обращения к динамическим массивам).

Поле селектора сегмента можно также определять косвенно, что упрощает создание больших многосегментных массивов.

Процессор данных автоматически масштабирует индекс, умножая его на 1, 2, 4, 8 или 16 в зависимости от того, занимает ли адресуемый операнд байт, двойной байт, слово, двойное слово или расширенное слово.

Система команд процессора полностью симметрична: имеются все необходимые операторы для каждого типа данных, и все четыре режима адресации доступны для всех операндов команды. Такая симметрия очень важна при реализации компиляторов, так как отсутствие симметрии приводит к многочисленным особым ситуациям, которые должны обрабатываться программно. Например, каждый из следующих операторов кодируется всего одной командой:

$A = B * C$ — трехадресное умножение, все величины являются скалярами;

$A[I] = B[J] / C[K]$ — трехадресное деление, все величины представляют собой элементы статических массивов;

$R.Q = A[I] * C$ — трехадресное умножение смешанного типа: элементу R записи P присваивается значение произведения I -го элемента статического массива A и скаляра C .

Процессор, рассчитанный на сложные вычисления, не имеет системы прерываний. Предполагается, что все внешние прерывания, например от клавиатуры, обрабатываются менее мощными процессорами. При нажатии клавиши такой процессор записывает символ в буфер. Когда воспринимается символ возврата каретки (как указатель конца строки текста), он также запоминается в буфере. Перед выполнением команды возврата из прерывания в процессор $iAPX-432$ посылается сообщение и он обрабатывает всю строку текста.

Процессор рассчитан на сегментную организацию памяти огромной емкости: $16M$ сегментов по $64K$ байт в каждом. Длина адреса составляет 40 бит, а адресное пространство — более 1 триллиона байт.

Аппаратные средства, операционная система и язык системного программирования опираются на концепцию «объекта» — структуру данных переменной длины, представляющую более высокую ступень по сравнению с данными, которые распознаются и обрабатываются обычными процессорами.

Для определения смысла некоторого набора бит в памяти ЭВМ необходимо знать, какой тип информации он представляет. Например, 16-битный набор может быть числом со знаком, если его тип — целое, или двумя символами, если его тип — символ. Целые числа, символы и действительные числа считаются простыми объектами, которые называются примитивами. Примитивы имеют три важных свойства: они являются структурами данных, представляющими некоторым образом организованную информацию; с ними ассоциируется набор базовых операций, которые обрабатывают непосредственно примитивы, например сложение целых или умножение действительных чисел; примитивы можно адресовать как неделимые сущности, т. е. обращения к их частям невозможны (бессмысленны).

Более сложные структуры данных называются объектами. Имеются различные типы объектов. В развитии ЭВМ можно проследить эволюцию типов объектов. Первые машины имели простейшие аппаратно-реализованные операции, например «переслать байт» или «сложить целые», которые манипулировали такими аппаратно-распознаваемыми типами данных, как байты и целые. Но первые ЭВМ не могли обрабатывать числа с плавающей запятой, и операции над ними выполнялись программно.

По мере совершенствования аппаратных средств и расширения функциональных возможностей ЭВМ появились аппаратные реализации более сложных операций, например арифметические операции над числами с плавающей запятой. Это увеличивало быстродействие машины и ускоряло программирование.

В процессоре $iAPX-432$ сделан еще один шаг вперед: функции управления системой (планирование процессоров, управление памятью и взаимодействие процессов) реализованы аппаратно в отличие от программной реализации в большинстве современных ЭВМ. Следовательно, процессор должен оперировать структурами данных, ассоциируемых с упомянутыми выше функциями. Главными из таких структур данных, называемых объектами, являются объект-процесс, объект-процессор и объект — порт связи.

Система использует метку для контроля типа объекта до использования его в своих действиях. Объекты во многих отношениях напоминают примитивы, поскольку объект представляет собой структуру данных, содержащую организованную определенным образом информацию;

с объектом ассоциируется набор базовых операций, которые непосредственно манипулируют структурой данных (контроль допустимости операций реализован аппаратно); к объекту можно обращаться (адресовать объект) как к неделимой сущности; объект имеет метку, идентифицирующую его тип.

Таким образом, объект представляет собой структуру данных произвольной сложности, которая обязательно должна находиться в одной (неразрывной) области памяти. Объект может занимать один сегмент памяти, несколько сегментов или часть сегмента. Понятие объекта относится к логической структуре, показывающей, каким образом используется память.

Для понимания программы необходимо рассмотреть программную структуру, представляющую собой цепочку связанных объектов. Первым элементом этой цепочки является процессор, который выбирает и выполняет команды, хотя физический процессор, разумеется, не является объектом. Однако с ним ассоциируется объект-процессор, который представляет собой структуру данных в памяти, содержащей такую информацию о процессоре, как его состояние (работает или остановлен), диагностическую информацию и требующиеся ему обращения к другим объектам. Объект-процессор размещается в любой области памяти, и его при необходимости можно перемещать.

Следующим элементом в программной цепочке является процесс, представляющий собой запрограммированные действия, которые можно запланировать для автономного выполнения их процессором. Объект-процесс представляет собой структуру данных в памяти, содержащую информацию о конкретном процессе: каким образом необходимо планировать процесс, каково состояние процесса и др. Объект-процессор содержит объект-обращение для текущего объекта-процесса, выполняемого процессором. Объектно-ориентированные механизмы адресации и защиты не разрешают процессору доступ к объекту, если процессор не имеет объекта-обращения для этого объекта. Объект-обращение от объекта-процессора к объекту-процессу не является постоянным, а изменяется, когда происходит переключение процессора с одного процесса на другой.

Очередным элементом в программной цепочке служит процедура — набор команд для выполнения определенной операции, например вычисления синуса угла. При вызове процедуры ей передаются параметры, она выполняет операцию и возвращает результат в вызывающую процедуру. В системе может быть несколько процессов, выполняющих одну и ту же операцию, и потребуется разделять процедуру между процессами. Система реализует разделение, выделяя каждому процессу копию процедуры, называемую контекстом. Контексты не обязательно дублируют всю информацию, например команды могут использовать несколько процессов.

Объект-контекст представляет собой аппаратно-распознаваемую структуру данных, содержащую следующую информацию о конкретной копии процедуры: указатель команд для конкретного контекста; указатель стека для стека контекста, информацию о возврате в вызывающий контекст, обращения ко всем объектам, к которым контексту разрешен доступ.

Полный список объектов, к которым процедуре разрешены обращения, называется ее средой доступа. Когда одна процедура вызывает другую, среда доступа изменяется, так как вызванная процедура имеет другой контекст.

Последний элемент в программной цепочке связан с выборкой и исполнением команд, т. е. с программной обработкой данных. Следо-

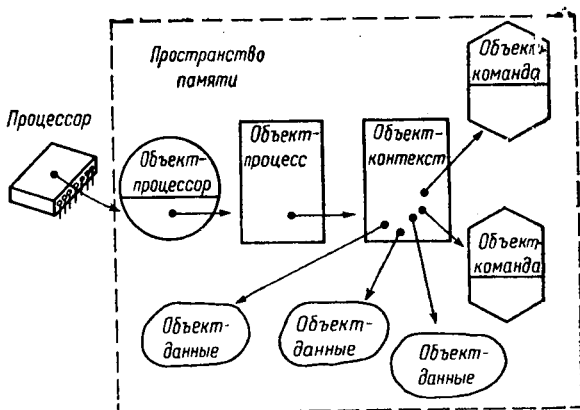


Рис. 8-18. Структура программы системы iAPX-432

вательно, контекст должен обеспечивать доступ к объектам, содержащим команды и данные. Объект-команда содержит только команды и представляет собой единственный тип объектов, который процессор использует как источник команд, подлежащих выборке и выполнению. Объект-данные содержит такие элементы, как целые, символы, действительные числа, таблицы или комбинации примитивных типов данных. Благодаря такому разделению команд и данных процессор не может ошибочно выбрать и выполнить объекты, отличающиеся от объектов-команд.

Полная структура программы показана на рис. 8-18. Отметим, что объект-контекст содержит обращение ко всем объектам-командам и объектам-данным, которые будет обрабатывать данная копия процедуры. Приведенная структура не является статической, а динамически изменяется по мере того, как программа выполняет различные процессы, процесс создает различные контексты, а контекст создает и уничтожает объекты-данные.

Системным языком программирования является язык «Ада», который, как полагают, объединяет простоту языка «Паскаль» и возможности, необходимые для многофункциональных программных систем. Язык «Ада» непосредственно реализует объектно-ориентированное программирование и, по существу, является ассемблером системы iAPX-432.

Вспомогательные схемы. За последнее время появилось очень много БИС, предназначенных для разгрузки центрального процессора от узкоспециализированных операций. По-видимому, тенденция разработки специализированных БИС сохранится и в будущем. Наиболее широкую группу составляют схемы управления операциями ввода-вывода и периферийными устройствами. В нее входят такие БИС, как контроллеры накопителей на гибком диске, контроллеры символьных и графических дисплеев, устройства шифрования данных, анализаторы и синтезаторы речи, аналоговые микропроцессоры и др. Каждая БИС по своей сложности и архитектуре приближается к специализированной ЭВМ и имеет на кристалле тысячи и десятки тысяч транзисторов.

Логическим развитием этих БИС являются специализированные процессоры, предназначенные для увеличения производительности системы и расширения системы команд центрального процессора. Наиболее

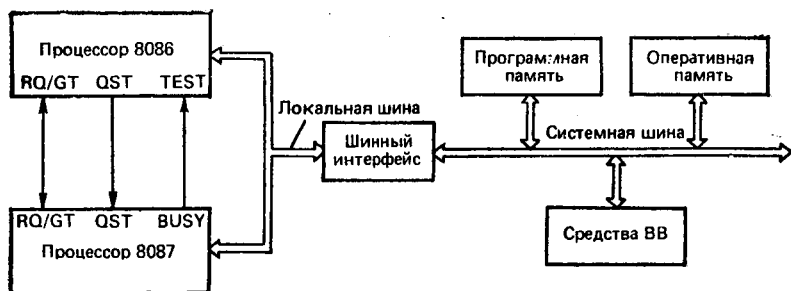


Рис. 8-19. Организация математического сопроцессора

примечательны в этом отношении процессоры ввода-вывода и так называемые математические процессоры, ориентированные на сложные операции в формате с плавающей запятой. Типичным примером математических процессоров служит микросхема 8087 фирмы Intel с плотностью упаковки 65 000 транзисторов. По сравнению с программной реализацией операций в формате с плавающей запятой эта БИС увеличивает производительность примерно в 100 раз.

Программная модель процессора содержит стек из восьми 80-битных регистров данных, допускающий также прямую адресацию всех регистров, регистры управления и состояния, а также регистры адресов команд и данных. Регистры данных хранят числа с 15-битной характеристикой и 64-битной мантиссой, что обеспечивает диапазон представимых чисел $\pm 10^{\pm 4900}$. Система команд охватывает все арифметические операции, извлечение квадратного корня, тригонометрические функции и др.

Процессор 8087 спроектирован для работы с процессором 8086 в качестве сопроцессора, расширяющего вычислительные возможности центрального процессора. Организация сопроцессоров является частным случаем мультипрограммирования, когда оба процессора выполняют команды одной программы. Оба процессора работают синхронно, и каждый выполняет свои команды, встречающиеся в программе (рис. 8-19).

Процессор 8086 осуществляет сегментацию памяти, вычисляет адреса операндов, а процессор 8087 выполняет сложные математические операции, которые иначе пришлось бы реализовать программно. Когда процессор 8086 встречает команду ESC, идентифицирующую операцию для процессора 8087, он вычисляет адрес операнда и выдает его на разделенную локальную шину. Процессор 8087 фиксирует адрес, считывает операнд и начинает выполнять операцию. Он управляет шиной только при необходимости загрузки и запоминания данных.

Управляющая линия запроса/подтверждения RQ/GT используется процессором 8087 для передачи управления локальной шине при обмене данными. Линия QST состояния очереди используется для синхронизации выборки и дешифрации команд. Сигнал занятости BUSY идентифицирует состояние процессора 8087, и процессор 8086 с помощью команды ожидания WAIT проверяет, когда процессор 8087 готов выполнить следующую команду.

Программирование комбинации двух процессоров аналогично программированию одного процессора 8086. С точки зрения пользователя добавляется 68 новых команд и 7 типов данных.

Развитие мини-ЭВМ. По своим возможностям новые мини-ЭВМ быстро приближаются к средним и большим ЭВМ середины и конца 70-х годов, а сфера их применения становится очень широкой. Ниже кратко рассматриваются только некоторые направления в развитии аппаратных и программных средств мини-ЭВМ.

Процессор. Как правило, длина машинного слова новых мини-ЭВМ составляет 32 бита. Такое слово обеспечивает реализацию многочисленных типов данных с различными форматами. Увеличивается до 32—64 число регистров общего назначения, что увеличивает производительность процессора. Благодаря широкому применению СИС и БИС время командного цикла доведено до 200—1000 нс. В разработках устройств управления преобладает принцип микропрограммного управления с конвейеризацией микрокоманд. Иногда вводится внутренних буфер команд, обеспечивающий опережающую выборку команд из программной памяти с совмещением выборки и выполнения команд.

Система команд становится все более обширной, в частности она осуществляет операции с плавающей запятой, табличные преобразования, десятичную арифметику и др.

Память. Емкость и быстродействие основной памяти мини-ЭВМ заметно увеличиваются благодаря применению компактной, быстродействующей и экономичной полупроводниковой памяти. Для компенсации энергозависимости полупроводниковой памяти во многих мини-ЭВМ предусматривается энергонезависимая ферритовая память небольшой емкости.

Однако быстродействие памяти большой емкости пока несколько ниже быстродействия процессора. Для уменьшения влияния этой диспропорции в некоторых мини-ЭВМ применяется кэш-память. Это быстродействующая биполярная память сравнительно небольшой емкости 2К—8К байт, которая находится между процессором и основной памятью. Работа кэш-памяти опирается на принцип локальности программ: значительная часть обращений к памяти при выполнении программы приходится на небольшие ее области. В кэш-памяти копируется содержимое этих областей, и в операциях считывания, составляющих до 80 % обращений к памяти, она заменяет основную память.

В мини-ЭВМ широко применяются средства преобразований (отображения) сравнительно коротких логических адресов памяти, формируемых процессором, в данные физические адреса, содержащие до 32 бит. Такие физические адреса обеспечивают емкость основной памяти более 4 млрд. байт.

Средства ввода-вывода. Совершенствование средств ввода-вывода идет по пути введения каналов ввода-вывода, позволяющих совместить операции ввода-вывода с действиями центрального процессора. Расширяется набор периферийных устройств, улучшаются их технические характеристики, создаются более удобные условия для работы пользователей. Особенно следует подчеркнуть широкое применение символьных и графических дисплеев. Благодаря встроенным микропроцессорам и микро-ЭВМ периферийные устройства выполняют многие функции автономно от центрального процессора, приобретая черты «искусственного интеллекта».

Программное обеспечение. В разработке новых мини-ЭВМ наблюдается устойчивая тенденция сохранения программной совместимости «вверх» с предыдущими моделями, особенно получившими широкое распространение. Объясняется это стремлением максимально использовать те огромные средства, которые были вложены в разработку программного обеспечения ЭВМ.

Программное обеспечение становится все более мощным, создаются совершенные операционные системы, например система UNIX. Мини-

ЭВМ оснащаются трансляторами со многих языков высокого уровня, в том числе языков С, «Паскаль» и «Ада». Как уже отмечалось в § 7-1, заметна тенденция к аппаратной и аппаратно-программной реализации функций операционных систем.

Распределенные системы на базе мини- и микро-ЭВМ. Встраивание микро-ЭВМ в цифровые системы обеспечивает такие преимущества программного управления, как гибкость, надежность, удобство эксплуатации и дешевизна. Сейчас производятся системы, выпуск которых ранее оказывался неэкономичным. Естественной эволюцией систем на базе мини- и микро-ЭВМ стала распределенная обработка информации («распределенный интеллект»), базирующаяся на мультипроцессорных и мультимашинных системах. Эти два класса систем имеют некоторые принципиальные различия, поэтому ниже они рассматриваются отдельно.

Мультипроцессорные системы. Мультипроцессорные системы характеризуются тем, что имеют один входной поток заданий (рабочую нагрузку). Единая операционная система распределяет аппаратные ресурсы, а физические расстояния между элементами системы невелики. Такие системы применяются в ситуациях, где определяющим фактором является надежность, которая обеспечивается либо введением значительной избыточности аппаратуры, либо средствами быстрой реконфигурации системы при отказе.

Различают два типа мультипроцессорных систем: слабосвязанные и сильносвязанные. В слабосвязанных системах каждый процессор имеет локальные программную память, средства ввода-вывода и память данных (оперативную память). Получающуюся структуру называют процессорным элементом. Для межпроцессорной связи (взаимодействия) предусматриваются резидентные средства, с помощью которых можно косвенно использовать другие элементы системы. Каждый процессорный элемент выполняет сравнительно автономные функции, например опрос датчиков и калибровку. Взаимодействия с другими процессорными элементами оказываются нечастыми и обычно реализуются с помощью блоковых передач либо непосредственно в нужный элемент, либо через общую память данных (рис. 8-20).

Когда данные передаются непосредственно в другой процессорный элемент, их необходимо записать в оперативную память элемента-приемника. Для компенсации уменьшения производительности и в некотором смысле изолирования источников и приемников применяется буферная память типа FIFO. Она представляет собой двухпортовую оперативную память, допускающую асинхронную запись и считывание данных.

Если для взаимодействия используется разделенная общая память, каждому процессорному элементу выделяется область общей памяти, адрес которой известен остальным процессорам системы (рис. 8-21). Такая область памяти, обычно называемая почтовым ящиком, становится как бы частью процессорного элемента. Процессор опрашивает свой почтовый ящик по некоторому периодическому алгоритму, либо организуется общее системное прерывание для инициирования опроса процессорами их почтовых ящиков.

Независимо от способа взаимодействия процессоров важное значение имеет организация системной шины. Простой вариант с общей шиной, к которой обращаются все процессорные элементы, имеет серьезные ограничения, если важно обеспечить немедленный доступ к шине. В системе появляются дополнительные аппаратные средства, которые управляют доступом к шине и называются арбитром шины. Арбитр анализирует запросы шин от всех процессорных элементов и в соответствии с их приоритетами разрешает доступ к шине. В более дорогом

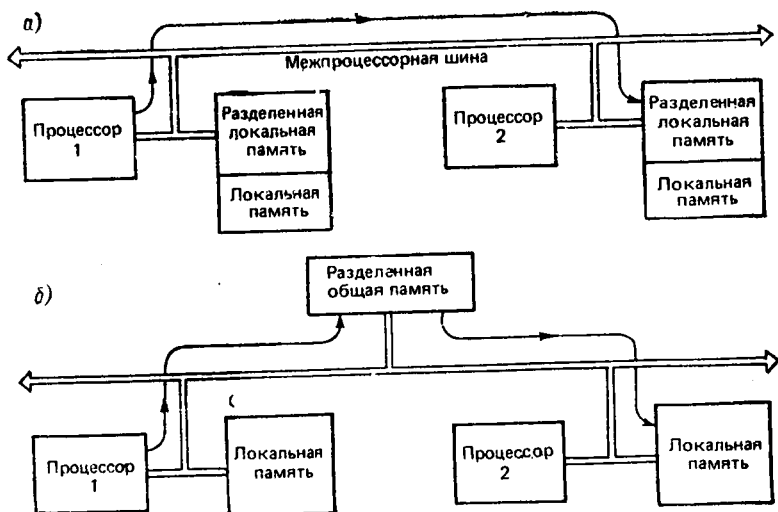


Рис. 8-20. Межпроцессорная связь: а — с помощью прямых передач; б — с помощью общей памяти

варианте вводится несколько трактов связи и взаимодействия становятся менее зависимыми от других действий в системе. Если допустима сравнительно невысокая скорость передачи данных, применяются линии последовательной связи.

Для сильносвязанных мультипроцессорных систем характерны близость элементов, параллельная структура шины и разделенные ресурсы. Процессоры могут иметь общую программную память, память данных или средства ввода-вывода. Каждый процессор обращается к общей шине для выполнения циклов памяти или ввода-вывода. Это предъявляет жесткие требования к быстродействию шины и ограничивает число процессоров в системе. На рис. 8-22 показаны известные зависимости производительности системы T_c от числа процессоров N . Величина T_π соответствует производительности одного процессора, а коэффициент использования шины b представляет собой часть доступных циклов ши-

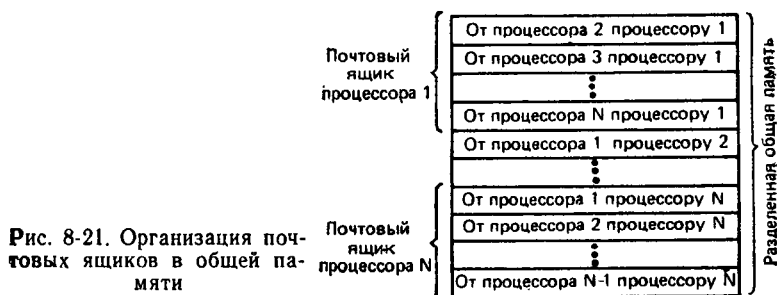


Рис. 8-21. Организация почтовых ящиков в общей памяти

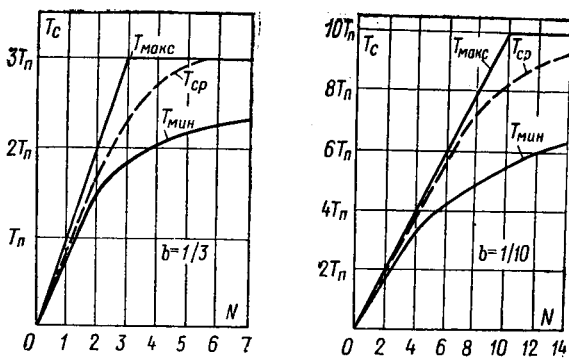


Рис. 8-22. Зависимость общей производительности системы от числа процессоров

ны, необходимых одному процессору. Его численное значение зависит от циклов команд процессоров и времени обращения памяти.

Различают две структуры сильносвязанных систем: симметричную и асимметричную. В симметричной структуре все процессоры выполняют одну и ту же программу с различными данными. Когда в систему поступает запрос на работу, управляющий элемент направляет его в любой свободный процессор. При отказе одного процессора нагрузка перераспределяется между другими процессорами. В асимметричной структуре каждый процессор выполняет различные задачи, а результаты используются другим процессором. Например, процессор ввода направляет данные в процессор с плавающей запятой, а тот передает результаты обработки в связной процессор. Таким образом, асимметричная структура характеризуется предопределенностью передач данных.

Независимо от способа реализации системы (сильно- или слабосвязанная) необходимо установить иерархию между процессорными элементами. Элемент может быть ведущим (М-элемент) или ведомым (S-элемент). М-элемент самостоятельно инициирует выполнение задачи и передачи данных. Каждый М-элемент (если их несколько) программируется с расчетом на его автономность в терминах данного алгоритма. Действия S-элементов зависят от приказов М-элемента о выполнении какой-либо задачи. Передачи результатов также могут осуществляться под управлением М-элемента.

М/S-системы довольно распространены, так как межпроцессорные связи проще реализовать, когда один М-элемент управляет несколькими S-элементами. М-элемент в М/S-системе иногда называют процессором распределения ресурсов, так как S-элементы начинают действия по приказам М-элемента.

Восьмибитные микропроцессоры с фиксированной системой команд и длиной слова практически не имеют средств организации мультипроцессорных систем (схем последовательной связи и сигналов запроса шины). Поэтому для реализации мультипроцессорной системы необходимы значительные дополнительные аппаратные средства. Рассмотрим некоторые вопросы объединения в сильносвязанную систему микропроцессоров КР580.

Число объединяемых в систему микропроцессоров, обеспечивающее существенный прирост производительности, зависит в основном от чис-

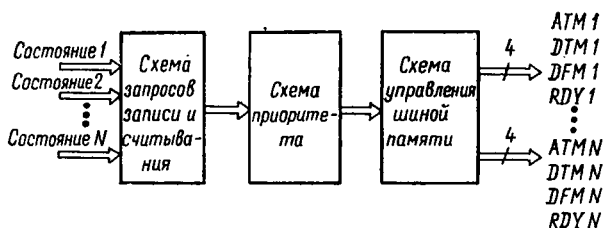


Рис. 8-23. Структура контроллера памяти

ла периодов синхронизации в одном машинном цикле. В микропроцессоре КР580 каждая команда требует от 1 до 5 машинных циклов продолжительностью 3—5 периодов синхронизации. В первом машинном цикле из памяти всегда считывается код операции, а в зависимости от функции команды могут потребоваться дополнительные обращения к памяти. Статистический анализ большого числа программ показал, что одно обращение к памяти приходится на 3,5 периода синхронизации. Поэтому, если время обращения к памяти не более одного периода синхронизации, она может обслужить параллельно три микропроцессора. Однако неизбежно несколько меньшее увеличение производительности, чем трехкратное, так как при одновременном обращении к памяти нескольких микропроцессоров разрешается обращение одному из них, а остальные должны ожидать.

Контроллер памяти, структура которого приведена на рис. 8-23, синхронизирует передачи данных между памятью и микропроцессорами. При наличии единственной шины памяти обращения микропроцессоров к памяти должны подчиняться определенным правилам. Для каждого микропроцессора необходимы четыре управляющих сигнала, формируемых, когда микропроцессор выдает адрес на шину адреса (мнемоника сигнала ATM — адрес в память), считывает данные с шины данных (мнемоника сигнала DFM — данные из памяти), выдает данные на шину данных (мнемоника сигнала DTM — данные в память) или когда необходимо приостановить обращение к шине, если память удовлетворяет запрос другого микропроцессора (мнемоника сигнала RDY — готовность).

Указанные управляющие сигналы формируются из сигналов состояний микропроцессоров (SYNC, MEMR, WO и др.) с учетом приоритетов микропроцессоров. Наиболее просто реализуются фиксированные приоритеты. Отметим, что контроллер памяти должен иметь достаточно высокое быстродействие.

Мультимашинные системы обрабатывают несколько входных потоков, в них нет единой операционной системы, а межмашинное взаимодействие осуществляется в основном на уровне данных. Однако в сложных ситуациях данные могут дополняться приказами на выполнение определенных действий. Одна из основных функций системы заключается в распределении различных задач для достижения максимальной производительности. В идеальном случае каждая ЭВМ системы выполняет конкретную задачу и работает относительно независимо от других. Передачи данных между ЭВМ невелики по сравнению с общим потоком данных в системе. К настоящему времени разработано множество структур мультимашинных систем, основные из которых рассматриваются ниже.

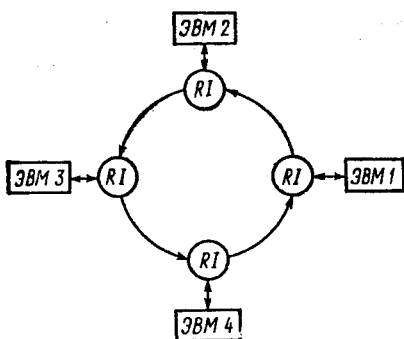


Рис. 8-24. Кольцевая структура

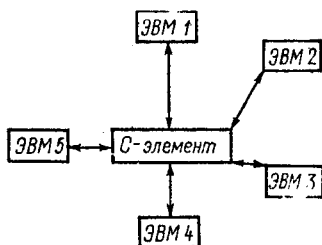


Рис. 8-25. Звездная структура

Кольцевая структура. Система с кольцевой структурой опирается на быстродействующую однонаправленную линию (канал) связи, которая образует замкнутый тракт — кольцо или петлю. ЭВМ и общие периферийные устройства подключаются к линии с помощью аппаратных устройств, называемых кольцевым интерфейсом RI (рис. 8-24).

Взаимодействие между ЭВМ (узлами) системы осуществляется в виде сообщений, передаваемых по каналу. Сформированное узлом-передатчиком сообщение передается по каналу до тех пор, пока оно не достигнет адресованного узла-приемника или не возвратится в узел-передатчик. Сообщения из канала удаляет либо узел-передатчик, либо узел-приемник. В первом случае узел-передатчик может сравнить исходное сообщение с сообщением, вернувшимся по каналу, и проконтролировать наличие ошибок. Узел-приемник сигнализирует о получении сообщения установкой определенной биты в этом сообщении. Во втором случае контроль ошибок выполняется узлом-приемником. Отметим, что во втором варианте нагрузка на канал связи уменьшается.

Сообщения передаются с разделением во времени в виде адресуемых блоков, называемых пакетами или кадрами. Локальный кольцевой интерфейс формирует кадр, вводит адрес узла-приемника и передает кадр в канал связи. Интерфейсы остальных узлов принимают кадр, проверяют адрес приемника и либо сразу ретранслируют кадр в канал (адреса приемника и интерфейса не совпали), либо удаляют кадр из канала и направляют сообщение подключенной ЭВМ (кадр достиг приемника).

Кольцевая структура имеет следующие положительные качества: благодаря единственному каналу связи не возникает проблем маршрутизации сообщений; физическое размещение узла-приемника не играет роли, и упрощается передача сообщений для всех узлов; передачи по каналу осуществляются в цифровой форме с высокой скоростью, поэтому модемы и преобразователи данных не нужны.

Основной недостаток кольцевой структуры — требование очень высокой надежности канала, так как отказ канала приводит к выходу из строя фактически всей системы. Для увеличения надежности системы применяется тот или иной способ дублирования канала связи.

Звездная структура. В данной структуре одна ЭВМ образует центр (С-элемент), выполняющий в системе функции управления и соединен-

ный отдельными линиями связи с другими ЭВМ (рис. 8-25). Когда ЭВМ1 необходимо передать сообщение в ЭВМ2, она посылает в С-элемент сообщение — запрос на передачу. С-элемент устанавливает тракт связи с ЭВМ2 при получении от последней сообщения о готовности к приему. Если в процессе передачи данных между ЭВМ1 и ЭВМ2 в ЭВМ3 возникает необходимость передачи сообщения в ЭВМ2, эта передача задерживается до освобождения тракта связи между ЭВМ1 и ЭВМ2. Однако обычно допускают передачи между ЭВМ3 и ЭВМ4 параллельно с передачами между ЭВМ1 и ЭВМ2.

С-элемент наряду с коммутацией сообщений может выполнять функции обработки данных, но более распространены системы, где задачи С-элемента ограничены коммутацией сообщений и диагностическими операциями.

Критическим компонентом системы является С-элемент, выполняющий сложные функции параллельного управления большим числом трактов связи. Если отказ одной линии связи приводит к невозможности использования одной ЭВМ, то отказ центрального коммутатора приводит к более тяжелым последствиям. Быстродействие коммутатора определяется временем обработки сообщений и частотой поступления сообщений. Время же обработки сообщений в некоторой степени зависит от числа ЭВМ в системе и от длины сообщений.

Иерархическая структура. В иерархической (древовидной) структуре, приведенной на рис. 8-26, из практических соображений возможности ЭВМ увеличиваются при движении снизу вверх. ЭВМ нижнего уровня специализируются на конкретных прикладных задачах и часто реализуются как микро-ЭВМ или мини-ЭВМ младших моделей. Их функциями могут быть сбор, сжатие и несложная обработка данных в реальном времени. Информация периодически запоминается для целей диагностики или контроля. ЭВМ уровня 1 выполняют более сложные и разнообразные функции. Как минимум, они осуществляют передачу подготовленных программ или приказов в ЭВМ уровня 0, а также принимают от них информацию, производят ее оперативный анализ и запоминают результаты в устройствах массовой памяти (на дисках). Кроме того, они обеспечивают пакетную обработку, реализуют интерпретирующие языки высокого уровня и т. п. На верхнем уровне применяются ЭВМ широкого назначения (часто мини-ЭВМ старших моделей), которые управляют всей системой. Обычно они же обслуживают разделенные базы данных.

В иерархической структуре учитываются два аспекта надежности. Отказ ЭВМ верхнего уровня приводит к потере управляемости всей системой, а неисправность любой линии связи влечет за собой отключение от системы всех ЭВМ на уровнях, находящихся ниже неисправной линии. Проблема надежности решается с помощью введения избыточности. Следует отметить, что наличие нескольких уровней увеличивает время реакции системы.

Многосвязные структуры. Мультимашинные системы с данной структурой содержат ряд ЭВМ, соединенных несколькими линиями связи (рис. 8-27). В предельном случае полностью связанной структуры каждая ЭВМ имеет связи со всеми остальными ЭВМ системы. При этом сложность связей быстро увеличивается с ростом числа ЭВМ, но в конкретных системах необязательно наличие всех связей, что приводит к частично связанной структуре. Отказ одной линии связи имеет ограниченные последствия, но расширение системы затруднено.

В больших распределенных сетях мини-ЭВМ используются специальные связные процессоры (часто также на базе мини-ЭВМ), которые образуют подсеть связи и обслуживают все машины системы (рис. 8-28).

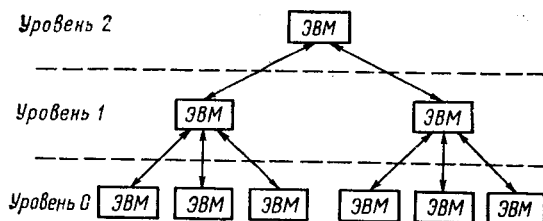


Рис. 8-26. Иерархическая структура

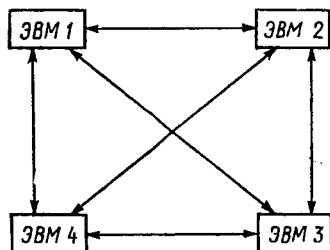


Рис. 8-27. Многосвязная структура

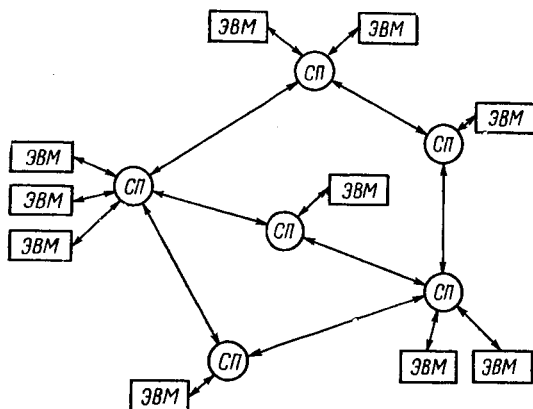


Рис. 8-28. Многосвязная структура со связными процессорами (СП)

Для передачи данных мини-ЭВМ обращается к своему связному процессору (СП), который передает сообщение по подсети связи.

Многосвязные системы классифицируются по тому, используется ли в них коммутация сообщений или пакетов. В первых из них сообщение от машины-источника передается машине-приемнику по фиксированному тракту связи в подсети. В каждом СП этого тракта сообщение вна-

чале запоминается (на диске), а затем передается в следующий СП при доступности соответствующего канала связи. Этому способу присущи такие недостатки, как большая задержка сообщений в условиях интенсивных передач, неэффективное использование ресурсов и др.

В системах с пакетной коммутацией сообщение до передачи разделяется на пакеты (кадры) длиной около 1000 бит. Пакеты можно хранить в оперативной памяти и передавать получателю по различным трактам связи. Другими словами, в системе реализуется адаптивная маршрутизация: тракт связи между двумя любыми узлами системы не фиксируется заранее, а выбирается исходя из текущих условий во время передачи. Благодаря такой организации в системе проще решаются проблемы перегрузки каналов и неисправных компонентов.

Для передачи сообщения машина-источник вводит в него адрес приемника и посылает сообщение в свой СП. Последний динамически выбирает наилучший маршрут, контролирует ошибки и уведомляет источник о приеме сообщения. Затем СП разделяет сообщение на пакеты, причем каждый пакет содержит идентифицирующую информацию. Каждый пакет передается в подсети независимо друг от друга в СП машины-приемника, где пакеты объединяются в исходное сообщение, которое и поступает в машину-приемник. Связные процессоры и подключенные к ним мини-ЭВМ размещаются в физической близости и соединяются параллельной шиной данных.

В заключение отметим, что техника распределенных систем получает все большее распространение и постоянно совершенствуется. Некоторые авторы считают, что 80-е годы будут десятилетием распределенных систем обработки данных.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как обеспечивается проблемная ориентация при проектировании ЭВМ различного назначения?
2. Выделите основные этапы проектирования аппаратных и программных средств. Какие задачи решает разработчик на каждом из этапов?
3. Какие используются подходы к построению распределенных систем?
4. Как осуществляется обмен информацией между компонентами распределенной системы?
5. Перечислите основные тенденции в развитии архитектуры микро- и мини-ЭВМ.

СПИСОК ЛИТЕРАТУРЫ

1. Алексенко А. Г. Основы микроэлектроники. — М.: Сов. радио, 1977. — 408 с., ил.
2. Балашов Е. П., Петров Г. А. Микропроцессоры — новая элементная база радиоэлектроники. — Изв. вузов СССР. Радиоэлектроника, 1978, № 11, с. 16—28.
3. Балашов Е. П., Пузанков Д. В. Микропроцессоры и микропроцессорные системы: Учебн. пособие для вузов/Под ред. В. Б. Смолова. — М.: Радио и связь, 1981. — 328 с., ил.

4. Безбожодов Ю. М. Индивидуальная отладка программ. — М.: Наука, 1982. — 190 с., ил.
5. Вирт Н. Систематическое программирование. Введение: Пер. с англ. — М.: Мир, 1977. — 188 с., ил.
6. Гарет П. Аналоговые устройства для микропроцессоров и мини-ЭВМ: Пер. с англ. — М.: Мир, 1981. — 268 с., ил.
7. Грогано П. Программирование на языке «Паскаль»: Пер. с англ. — М.: Мир, 1982. — 382 с., ил.
8. Каган Б. М., Сташин В. В. Микропроцессоры в цифровых системах. — М.: Энергия, 1979. — 192 с., ил.
9. Кросс-система разработки программного обеспечения микро-ЭВМ/ И. В. Герасимов, Г. А. Петров, В. С. Скворцов и др.—Приборы и системы управления, 1982, № 7, с. 10—13.
10. Куценко А. В., Полосьянц Б. А., Ступин Ю. В. Мини-ЭВМ в экспериментальной физике. — М.: Атомиздат, 1975. — 288 с., ил.
11. Липят А. Архитектура малых вычислительных систем: Пер. с англ./Под ред. К. В. Песелева. — М.: Мир, 1981. — 187 с., ил.
12. Малиновский Б. Н., Яковлев Ю. С., Маковенко Е. Т. Анализ и синтез мини-ЭВМ.—Киев: Наукова Думка, 1976.—174 с., ил.
13. Малые ЭВМ и их применение /Ю. А. Дедов, М. А. Островский, К. В. Песелев и др.; Под общ. ред. Б. Н. Наумова. — М.: Статистика, 1980.—231 с., ил.
14. Микропроцессорные комплекты повышенного быстродействия/ А. И. Березенко, Л. Н. Корягин, А. Р. Назарьян. — М.: Радио и связь, 1981. — 168 с., ил.
15. Многофункциональные регулярные вычислительные структуры/ Е. П. Балашов, В. Б. Смолов, Г. А. Петров и др.—М.: Сов. радио, 1978.—299 с., ил.
16. Палагин А. В., Иванов В. А., Кургаев А. Ф. Мини-ЭВМ. Принципы построения и проектирования. — Киев: Наукова Думка, 1975.— 199 с., ил.
17. Прагишвили И. В. Микропроцессоры и микро-ЭВМ.—М. Энергия, 1979.—232 с., ил.
18. Применение микропроцессорных средств для абонентских пунктов систем связи/Е. П. Балашов, С. Д. Бороненко, Г. А. Петров и др.—Электронное моделирование, 1982, № 3, с. 84—87.
19. Секен К., Томсет М. Приборы с переносом заряда: Пер. с англ.— М.: Мир, 1978.—327 с., ил.
20. Смолов В. Б., Пузанков Д. В., Петров Г. А. Применение современной микропроцессорной техники в проблемно-ориентированных системах. — Электронное моделирование, 1981, № 4, с. 45—51.
21. Соботка З., Стары Я. Микропроцессорные системы: Пер. с чешск. — М.: Энергоиздат, 1981.—495 с., ил.
22. Соучек Б. Микропроцессоры и микро-ЭВМ: Пер. с англ./Под ред. А. И. Петренко. — М.: Сов радио, 1976.—520 с., ил.
23. Соучек Б. Мини-ЭВМ в системах обработки информации: Пер. с англ. /Под ред. А. И. Петренко. — М.: Сов. радио, 1976.—520 с., ил.
24. Хилбурн Дж., Джулич П. Микро-ЭВМ и микропроцессоры: Пер. с англ./Под ред. С. Д. Пашкеева. — М.: Мир, 1979.—463 с., ил.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

- Адаптер параллельный 147
 - последовательный 135
- Адресация абсолютная 81
 - автодекрементная 33
 - автоинкрементная 33
 - индексная 34
 - косвенная 32
 - непосредственная 32
 - относительная 34
 - прямая 32
 - регистровая 32
 - регистровая косвенная 32
- Аналого-цифровой преобразователь 120, 227
- Аккумулятор 47
- Арифметико-логическое устройство 35, 47
- Архитектура ЭВМ 39
- Ассемблер 280
- Байт 33
- Бод 229
- Буферное ЗУ 108
- Буфер текста 276
- Буферный регистр 47
- Ввод и вывод информации с отображением на память 42, 122
- Вложение подпрограмм 302
- Время обращения к памяти 93
- Выбор кристалла 94
- Генератор тактовых импульсов 40
- Готовность устройства 50
- Графический дисплей 236
- Двоично-десятичное сложение 48
- Диск гибкий 240
 - металлический 240
 - , формат 244
- Дисковая операционная система 258
- Дополнительный код 17
- Динамическая память 103
- Директивы ассемблера 282
- Дисплей алфавитно-цифровой 231
 - графический 236
- Дорожка диска 239
- Загрузчик абсолютный 287
 - перемещаемый 287
- Запоминающие устройства
 - — буферные 108
 - — динамические 103
 - — оперативные 252
 - — статические 103
 - — стековые 111
 - — постоянные 96
- Захват микропроцессора 51
- Запись в память 95
- Индикаторы 218
 - газоразрядные 218
 - жидкокристаллические 218
 - светодиодные 218
- Инкремент 27
- Интерфейс 114
 - периферийных устройств 116
 - программируемый 135, 147
- Интерфейсные схемы 124
- Кассетные накопители на магнитной ленте 250
- Коды обмена информацией 115
- Код операции 21
- Кодирование команд 22
- Кодирование микрокоманд 88
- Команды ввода и вывода информации 24
 - обработки данных 26
 - передачи данных 23, 69
 - передачи управления 29
- Контроллер обработки прерываний 128, 185

- Контроллер прямого доступа к памяти 204
- Кросс-ассемблер 327, 353
- Макрокоманда 268
- Машинный цикл 55
- Микрокоманда 73
- Микропрограммное устройство управления 72
- Микропроцессор однокристалльный 43, 52
 - секционный 71
- Мнемоника команды 262
- Многорежимный буферный регистр 125
- Модем 228
- Монитор 292
- Мультипрограммные ЭВМ 304
- Объектный код 259
- Операционная система 256
- Память динамическая 103
 - дисковая 238
 - перепрограммируемая 101
 - постоянная 96
 - статическая 103
 - стековая 45, 111
- Передача данных 162
 - асинхронная 167
 - программно-управляемая 166
 - с прерыванием программы 170
 - с прямым доступом к памяти 200
 - синхронная 166
- Периферийные устройства 214
- Подпрограмма 297
 - , возврат из подпрограммы 298
 - , вызов 298
 - , передача параметров в подпрограмму 299
- Признак результата операции 27, 47
 - знака 48
 - переноса 48
 - нулевого результата 48
 - четности 48
- Прикладное программное обеспечение 258
- Принтер 219
- Приоритет 176
- Программная модель микропроцессора 64
- Проектирование микропроцессорных систем 348
- Программатор 100
- Программное обеспечение ЭВМ 252
- Программный счетчик 44
- Процессор 40, 43
- Псевдокоманда 265
- Регенерация динамической памяти 104
- Регистр команд 45
 - общего назначения 46
 - признаков 47
- Редактор связей 255, 287
 - текста 276
- Режим обмена последовательными кодами 135
 - асинхронный 135
 - синхронный 136
- Сектор дорожки диска 244
- Синхронизация микрокоманд 91
- Система памяти 92
- Системный контроллер 134
- Скорость передачи информации 229
- Способы записи информации в накопитель на магнитном диске 242
 - на магнитной ленте 247
- Средства автоматизации программ 275
 - отладки программ 289
- Стековая память 45, 111
- Устройства ввода и вывода информации 214
- Указатель стека 45
- Управляющие сигналы в ЭВМ 57
- Устройство управления 84
 - микропрограммное 85
 - специализированное 85
- Файл 295
 - двоичный 295
 - текстовый 295
- Формат данных 16
 - команды 20
 - программы 261
 - сектора накопителя на магнитном диске 244
- Цикл машинный 55
- Шина адреса 40, 49
 - данных 40
- Шинный формирователь 134
- Язык ассемблера 254, 260

ОГЛАВЛЕНИЕ

Предисловие	3
ГЛАВА ПЕРВАЯ. МНОГОФУНКЦИОНАЛЬНОСТЬ И СПЕЦИАЛИЗАЦИЯ В ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКЕ	5
1-1. Принцип многофункционального использования аппаратных и программных средств	—
1-2. Эволюция средств вычислительной техники	6
1-3. Проблемно-ориентированные вычислительные и управляющие системы	11
ГЛАВА ВТОРАЯ. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ И СИСТЕМА КОМАНД	16
2-1. Форматы данных	—
2-2. Структура и типы команд	19
2-3. Команды передачи данных	23
2-4. Команды обработки данных	26
2-5. Команды передачи управления и дополнительные команды	29
2-6. Режимы адресации информации	31
ГЛАВА ТРЕТЬЯ. АРХИТЕКТУРА МИКРО- И МИНИ-ЭВМ	36
3-1. Особенности структурной организации систем обработки информации на основе БИС	—
3-2. Архитектура ЭВМ	39
3-3. Однокристалльные МП	43
3-4. Однокристалльный 8-битный МП	52
3-5. Однокристалльный 16-битный МП	62
3-6. Секционные МП	71
3-7. Процессоры мини-ЭВМ СМ-3 и СМ-4	75
3-8. Устройство управления	84
3-9. Основные характеристики систем полупроводниковой памяти	92
3-10. Постоянные запоминающие устройства	96
3-11. Запоминающие устройства с произвольной выборкой	102
3-12. Буферные ЗУПВ	108
3-13. Стековая память	111
ГЛАВА ЧЕТВЕРТАЯ. ОРГАНИЗАЦИЯ ИНТЕРФЕЙСА В МИКРО- И МИНИ-ЭВМ	114
4-1. Основные понятия и требования к интерфейсу	—

4-2. Интерфейс с изолированной и общей системой шин	116
4-3. Компоненты для организации интерфейса	124
4-4. Программируемый адаптер последовательного интерфейса	135
4-5. Программируемый адаптер параллельного интерфейса	147
4-6. Интервальный таймер	155
ГЛАВА ПЯТАЯ. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА	162
5-1. Режимы и структура команд ввода-вывода	—
5-2. Программный ввод-вывод	166
5-3. Ввод-вывод в режиме прерываний	170
5-4. Организация блоков обработки прерываний	185
5-5. Ввод-вывод с прямым доступом к памяти	200
ГЛАВА ШЕСТАЯ. ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА	214
6-1. Классификация периферийных устройств	—
6-2. Устройства для связи с пользователем	216
6-3. Устройства массовой памяти	223
6-4. Устройства связи с объектами управления и передачи данных	227
6-5. Символьные и графические дисплеи	231
6-6. Накопители на магнитных дисках	239
6-7. Накопители на магнитной ленте	246
ГЛАВА СЕДЬМАЯ. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МИКРО- И МИНИ-ЭВМ	252
7-1. Назначение и классификация программного обеспечения	—
7-2. Языки описания алгоритмов	259
7-3. Средства автоматизации разработки прикладных программ	275
Редактор текста	276
Ассемблер	280
Загрузчик	286
Редактор связей	287
Средства отладки	289
7-4. Монитор	292
7-5. Организация подпрограмм	297
7-6. Расписание процессов в мультипрограммных ЭВМ	304
ГЛАВА ВОСЬМАЯ. ПРИМЕНЕНИЕ, ОСНОВНЫЕ ЭТАПЫ ПРОЕКТИРОВАНИЯ И ТЕНДЕНЦИИ РАЗВИТИЯ МИКРО- И МИНИ-ЭВМ	310
8-1. Области применения микро- и мини-ЭВМ	—
8-2. Функционально-структурный подход к проектированию проблемно-ориентированных систем	314
8-3. Микропроцессоры в системах управления	320
8-4. Синтез базовых структур многофункционального контроллера ВЗУ	328
8-5. Микро-ЭВМ в системах связи	339
8-6. Основные этапы проектирования аппаратных и программных средств	347
8-7. Тенденции в развитии архитектуры микро- и мини-ЭВМ и распределенных систем на их основе	354
Список литературы	371
Предметный указатель	373