

**БИБЛИОТЕЧКА
ПРОГРАММИСТА**

Н. П. БРУСЕНЦОВ

Миникомпьютеры



БИБЛИОТЕЧКА ПРОГРАММИСТА

Н. П. БРУСЕНЦОВ

МИНИКОМПЬЮТЕРЫ

Под редакцией
Л. Н. КОРОЛЕВА



МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1979

МИНИКОМПЬЮТЕРЫ. Брусенцов Н. П.
Главная редакция физико-математической литературы
изд-ва «Наука», М., 1979.

Миникомпьютер — это тот вариант цифровой машины, в котором она стала, наконец, действенным орудием производства. Книга представляет собой попытку раскрыть сущность миникомпьютеров, показать, каким образом достигнута необыкновенная практичность, обусловившая лавинообразное увеличение их выпуска и быстрое проникновение во все сферы жизни. Главное внимание уделено особенностям архитектуры миникомпьютеров, таким как экономная адресация и управление ходом программы в условиях короткого машинного слова, микрооперационное преобразование данных, оперативное взаимодействие с периферией. В качестве образцов подробно описана архитектура важнейших миникомпьютерных семейств.

Книга рассчитана как на разработчиков, так и на потребителей миникомпьютерной техники.

ОГЛАВЛЕНИЕ

| | |
|---|------------|
| Предисловие редактора . . . | 5 |
| Предисловие автора . . . | 6 |
| Глава 1. Общие сведения о миникомпьютерах | 7 |
| § 1. Начальное понятие о миникомпьютере | 8 |
| § 2. Возникновение и развитие промышленности мини- компьютеров | 14 |
| § 3. Истоки эффективности миникомпьютеров | 22 |
| § 4. Применения миникомпьютеров | 25 |
| Глава 2. Структура и состав миникомпьютерной системы | 34 |
| § 1. Аппаратура миникомпьютерной системы | 36 |
| § 2. Программное оснащение миникомпьютерной системы | 54 |
| Глава 3. Особенности архитектуры миникомпьютеров . . . | 62 |
| § 1. Понятие об архитектуре цифровой машины | 62 |
| § 2. Средства описания архитектуры | 64 |
| § 3. Факторы, определяющие характер архитектуры мини- компьютеров | 74 |
| § 4. Особенности архитектуры миникомпьютеров | 75 |
| § 5. Микропрограммирование | 77 |
| Глава 4. Адресация и управление ходом программы . . . | 84 |
| § 1. Способы и механизмы адресации | 85 |
| § 2. Управление ходом программы | 97 |
| § 3. Команды структурированного программирования | 105 |
| Глава 5. Управление периферией . | 109 |
| § 1. Способы ввода/вывода | 110 |
| § 2. Системы прерывания | 118 |
| Глава 6. Одноаккумуляторные миникомпьютеры серии DEC PDP-8 | 129 |
| § 1. Структура процессора и типы команд | 131 |
| § 2. Адресные команды и система адресации | 132 |
| § 3. Команды-микропрограммы | 135 |
| § 4. Команды ввода/вывода | 142 |
| § 5. Система прерывания и прямой доступ к памяти | 144 |
| § 6. Добавочное и периферийное оборудование | 149 |
| § 7. Программное оснащение PDP-8 | 152 |

| | |
|--|------------|
| Глава 7. Двухаккумуляторные миникомпьютеры Hewlett-Packard | 156 |
| § 1. Регистры процессора и форматы команд | 157 |
| § 2. Адресные команды | 160 |
| § 3. Регистровые команды | 165 |
| § 4. Команды ввода/вывода и система прерывания | 171 |
| § 5. Периферийное оборудование и программное оснащение миникомпьютеров Hewlett-Packard | 177 |
| Глава 8. Четырехаккумуляторные миникомпьютеры DG NOVA | 184 |
| § 1. Виды команд и регистры процессора | 185 |
| § 2. Система адресации и адресные команды | 186 |
| § 3. Команда-микропрограмма | 192 |
| § 4. Команды ввода/вывода и управление прерыванием | 199 |
| § 5. Периферийное оборудование миникомпьютеров NOVA | 206 |
| § 6. Программное оснащение миникомпьютеров NOVA | 210 |
| Глава 9. Многорегистровые миникомпьютеры DEC PDP-11 | 212 |
| § 1. Форматы команд и данных | 213 |
| § 2. Способы адресации | 217 |
| § 3. Команды преобразования-тестирования данных | 230 |
| § 4. Средства управления ходом программы | 234 |
| § 5. Управление периферией и система прерывания | 243 |
| § 6. Периферийное оборудование миникомпьютеров PDP-11 | 250 |
| § 7. Программное оснащение семейства миникомпьютеров PDP-11 | 252 |
| Послесловие | 256 |
| Приложение. Общие правила записи программы на языке ассемблера | 259 |
| Литература | 265 |
| Предметный указатель | 270 |

ПРЕДИСЛОВИЕ РЕДАКТОРА

Миникомпьютеры получают все большее распространение в качестве эффективного средства автоматизации всевозможных производственных процессов, управления машинами и системами. Они нашли широкое применение также в системах обработки и передачи информации наряду с большими вычислительными машинами, выполняя такие функции, как управление устройствами ввода/вывода и линиями связи, сопряжение главных процессоров с обслуживаемыми ими объектами и т. п.

Предлагаемая читателю книга Н. П. Брусенцова — разработчика весьма известной в свое время малой цифровой машины «Сетунь», которую справедливо считают прообразом миникомпьютеров, — раскрывает на примере переосмысленного автором зарубежного опыта сущность и практические возможности миникомпьютерной техники. В ней изложены принципы архитектуры миникомпьютеров и достаточно подробно описана архитектура наиболее распространенных минимашин, разработанных и выпускаемых ведущими американскими фирмами.

В нашей стране в настоящее время миникомпьютерная техника широко развивается, проектируются новые типы мини- и малых машин, создаются миникомпьютерные системы, с большой эффективностью используемые во многих областях науки и техники. Выпускаются или готовятся к выпуску книги, содержащие материалы, необходимые для освоения этой новой техники и ее использования. В этой связи книга Н. П. Брусенцова, анализирующая состояние миникомпьютерной техники за рубежом, будет очень полезной как для проектировщиков отечественных минимашин и систем, так и для специалистов по их использованию.

Л. Н. Королев

ПРЕДИСЛОВИЕ АВТОРА

Эта книга призвана способствовать более широкой пропаганде в нашей стране миникомпьютеров — простых и эффективных машин, представляющих сегодня наиболее передовую по значимости и темпам роста отрасль цифровой техники. Миникомпьютеры положили начало массовым применениям цифровых машин для автоматизации производства, научных экспериментов, обработки текстов, цифровой связи и многих других видов практической деятельности. Но нередко встречается досадное непонимание существа и назначения этих замечательных машин — их трактуют как «миниатюрные ЭВМ», загружают несвойственной им сугубо вычислительной работой, не обеспечивают периферийным оборудованием для оперативного взаимодействия машины с ее окружением. Автор надсется, что данная книга поможет в преодолении этих недостатков.

Книга основана на зарубежном опыте разработок и применений миникомпьютеров. Личный опыт автора получил отражение главным образом в подходе к предмету, в оценке фактов и тенденций развития. Первые пять глав посвящены анализу миникомпьютеров в общем, последние четыре — конкретным архитектурам наиболее популярных миникомпьютерных семейств: DEC PDP-8, HP-2100, DG NOVA, DEC PDP-11.

Автор выражает благодарность Л. Н. Королеву — инициатору и титульному редактору книги, Х. Рамилю Альваресу и Е. Н. Филинову, высказавшим полезные замечания.

Н. П. Брусенцов

ОБЩИЕ СВЕДЕНИЯ О МИНИКОМПЬЮТЕРАХ

Миникомпьютеры представляют собой одно из самых значительных явлений цифровой техники, которое кратко можно охарактеризовать как поворот от обособленного развития этой техники к широкому и массовому внедрению ее во все сферы жизни. Именно в облике миникомпьютера цифровая машина стала высокоэффективным орудием производства, важнейшим инструментом современной научно-технической революции. Комплексная автоматизация производства, научных исследований и систем обслуживания, оказавшаяся непосильной для больших электронных вычислительных машин (ЭВМ), успешно и с поразительным экономическим эффектом осуществляется на базе миникомпьютеров. В противоположность традиционным ЭВМ, основным назначением которых остаются академические вычисления и обработка административно-хозяйственных данных, миникомпьютеры уже за первые десять лет своего существования проникли всюду и освоили сотни самых различных специальностей, от управления станками и проведения программируемых экспериментов до автоматизированного редактирования книг и наблюдения за состоянием больных. В любом деле использование миникомпьютеров, как правило, позволяет существенно уменьшить затраты труда и обеспечить высокое и стабильное качество выполняемой работы.

Миникомпьютеры сравнительно дешевы и быстро окупаются, поэтому спрос на них растет, как снежный ком. Промышленность миникомпьютеров функционирует как система с положительной обратной связью: быстрое увеличение выпуска и острая конкуренция

подхлестывают автоматизацию производства (осуществляемую с использованием самих же миникомпьютеров), вследствие автоматизации производства происходит снижение цен, открывающее миникомпьютерам новые, еще более широкие применения, а расширение сферы применений стимулирует дальнейшее увеличение выпуска и модернизацию производства — этот цикл повторяется снова и снова. На протяжении полутора десятилетий, прошедших со времени появления первых миникомпьютеров, происходит устойчивое снижение цен приблизительно на 20% в год (т. е. каждые 4 года миникомпьютеры дешевеют более чем вдвое) и каждые 2 года число имеющихся в мире миникомпьютеров более чем удваивается. Так, к концу 1970 г. было около 30 тыс. миникомпьютеров, к концу 1972 г. — более 60 тыс., к концу 1974 г. — около 150 тыс., а в 1976 г. их число превысило 300 тыс. Минипроцессор, стоивший в середине 60-х годов более 10 тыс. долл., десять лет спустя продавался в модернизированном исполнении за 1—2 тыс. долл. Новый мощный импульс возник с появлением в 1972 г. микропроцессора, соединившего в себе достоинства миникомпьютеров с преимуществами больших интегральных схем: выпуск микропроцессоров исчисляется уже миллионами экземпляров в год, причем цены достигли уровня 10—20 долл. Этим открыта возможность поистине неограниченных применений цифровой техники и не только в производстве или в науке, но и даже, скажем, в бытовых приборах или в игрушках.

§ 1. Начальное понятие о миникомпьютере

Слово *миникомпьютер* появилось в 1968 г., а обозначаемая им разновидность цифровой машины возникла на 3—4 года раньше. Несмотря на то, что наука о цифровых машинах (computer science) в то время уже существовала, новая машина родилась без ее ведома, так сказать, непредвиденно. Сущность миникомпьютера раскрывается по мере осознания его практического значения, и процесс этот все еще не завершен.

Авторы статей и книг о миникомпьютерах обычно не пытаются прямо определить миникомпьютер, а предоставляют читателю самому составить представле-

ние о предмете на основе излагаемых ими сведений. Наша книга в этом смысле не будет исключением — она целиком посвящена характеристике того, что, по мнению автора, представляет собой миникомпьютер, чем достигнута его исключительная универсальность и эффективность.

Хотя основной принцип миникомпьютера — просто-та, реализовать этот принцип в таком многоплановом деле, как цифровая техника и ее применения, совсем не просто. Миникомпьютеры — это особый цифровой мир с оригинальной идеологией и методикой, с новыми возможностями и проблемами.

Конечно, не все разделяют нашу точку зрения. Известно, например, что миникомпьютеры иногда определяют просто как цифровые машины (точнее, цифровые системы) с ценой ниже 50 тыс. или ниже 25 тыс. долл. Иные, полагая будто существо дела в физических размерах машины, утверждают, что миникомпьютер — это миниатюрная ЭВМ или, сокращенно, мини-ЭВМ. Но нетрудно заметить, что как первому, так и второму определению удовлетворяют, например, электронные карманные калькуляторы, которые никто к миникомпьютерам не относит.

Надо сказать, что *мини* в слове «миникомпьютер» значит не миниатюрный, а *минимальный*, т. е. в каком-то отношении предельно ограниченный, урезанный. Замечательно, что в миникомпьютере урезаны как раз его вычислительные атрибуты — арифметическое оборудование и точность (длина) чисел, которыми он манипулирует. Арифметический арсенал большинства миникомпьютеров исчерпывается командами сложения и вычитания с фиксированной запятой (на некоторых моделях даже нет команды вычитания), причем эти команды выполняются над короткими двоичными числами, содержащими 16 или даже 12 двоичных разрядов (*битов*), что эквивалентно приблизительно 3—4 десятичным разрядам.

Таким образом, в отличие от калькуляторов и традиционной большой ЭВМ, миникомпьютер — это по существу не вычислительная машина или по крайней мере это вычислитель (компьютер) в минимальной степени. Вместе с тем способность выполнять хранимую в памяти программу, осуществлять предписанный алгоритм свойственна миникомпьютеру не в меньшей

мере, чем большим автоматическим вычислительным машинам. Правда, миникомпьютерные программы построены из более мелких (элементарных) операций, но именно этим обеспечивается возможность тонко программировать алгоритмы самого разного характера. Кроме того, чем мельче операции, тем более компактным может быть набор команд универсальной машины.

По сравнению с миникомпьютером «универсальная» ЭВМ представляется специализированной машиной для выполнения вычислений, недаром ей дали прозвище *numbercruncher* — перемалыватель чисел. Главной характеристикой такой машины является арифметическое быстродействие, а наиболее дорогой ее частью — арифметическое устройство, причем прочие части, в сущности, призваны загружать это устройство работой. В применениях невычислительного характера данная машина, естественно, не может быть эффективной, потому что ее дорогое арифметическое оборудование будет работать на холостом ходу. Другими словами, она оказывается экономически неприемлемой для этих применений. Замечательно, что подлинная универсальность (широкая применимость) миникомпьютера достигнута не путем введения в традиционную ЭВМ каких-либо недостающих ей устройств, а, наоборот, путем изъятия из нее того, в чем заключается ее вычислительная специализация.

Неверно, разумеется, представлять дело так, будто кто-то, взяв автоматическую вычислительную машину, урезал в ней длину представления чисел, выбросил сложное арифметическое оборудование и таким образом создал миникомпьютер. В действительности было не так. Требовалось разработать контроллер для управления ядерным реактором. Разработчики вопреки традиции нашли целесообразным не создавать его в виде специального устройства «с запаянной логикой», а запрограммировать на цифровой машине с хранимой программой. Чтобы это решение могло быть экономически и технически приемлемым, пришлось создать новый тип цифровой машины, во многих отношениях отличный от традиционного (о чем речь пойдет в следующих главах). При этом решающее значение имело все же «урезывание и выбрасывание»: сокращение арифметического оборудования и длины

машинного слова (т. е. длины ячеек запоминающего устройства и регистров), что позволило резко снизить цену, уменьшить физические размеры и мощность электропитания машины, упростить обслуживание, повысить эксплуатационную надежность.

Следовательно, самым первым отличием миникомпьютера, составившим основу его экономической и технической широкой приемлемости, является короткое машинное слово и отсутствие необязательного для большинства применений сложного и дорогого арифметического оборудования. У машины, положившей начало племени миникомпьютеров, длина слова была 12 битов, т. е. в 3—5 раз меньше, чем у больших вычислительных машин. В дальнейшем в связи с введением 8-битного байта миникомпьютеры стали 16-битными (двухбайтными). Затем появились выполненные в виде больших интегральных схем 4-битные и 8-битные машины и, наконец, даже однокбитные. Чем короче слово, тем для более простых и соответственно более массовых применений доступна машина.

Заметим, что бедность арифметического оборудования не означает невозможности вычислительных применений миникомпьютера. Арифметические операции и математические функции программируются на языке команд миникомпьютера так же эффективно, как и любые другие. При этом требуемая точность обеспечивается использованием для представления числа нескольких машинных слов. В случаях, когда необходима значительная вычислительная мощность, миникомпьютер доукомплектовывают специальным арифметическим процессором.

Ценными свойствами миникомпьютера являются *наращиваемость, адаптивность и открытость* для сопряжения с другим оборудованием. В противоположность вычислительным системам общего назначения, для которых характерна жесткость и замкнутость конфигурации («если система перестала удовлетворять возросшие вычислительные потребности — приобретай более мощную модель»), минисистемы допускают наращивание мощности и варьирование конфигурации аппаратуры в самых широких пределах. Можно начать с базовой конфигурации (процессор, 4 тыс. слов памяти, телетайп) и по мере роста

потребностей доукомплектовывать ее блоками памяти вплоть до 32 тыс. или даже до 128 тыс. слов, дополнительными процессорами, памятью на магнитных дисках и магнитных лентах, всевозможными устройствами ввода/вывода, сопряжения с линиями связи, с приборами и машинами. При этом, как правило, не требуется вносить изменения в уже используемые устройства или переделывать созданные программы. Варьируемость конфигурации позволяет настраивать минисистему на конкретное применение, приспособливать ее к конкретным требованиям.

Открытость миникомпьютера для сопряжения с оборудованием пользователя означает, что помимо подключения к процессору произвольного набора минипериферийных устройств пользователю предоставляется возможность сравнительно просто модифицировать или встроить разработанную заново логику управления внешними объектами, приспособив миникомпьютер для работы в качестве элемента той или иной системы.

Еще одно замечательное свойство миникомпьютера — его *активность*, способность непосредственно взаимодействовать с внешним миром, следить за протекающими в нем процессами и управлять ими. Выполняемая миникомпьютером программа имеет возможность тестировать состояние обслуживаемого объекта и манипулировать им так же просто, как это обычно делается с внутренними регистрами. Кроме того, миникомпьютер располагает совершенной системой прерывания, обеспечивающей быстрое реагирование на нерегулярно поступающие запросы, а также способностью высокоскоростной передачи данных между внешними устройствами и главной памятью. Развитость указанных функций позволяет миникомпьютеру производить сбор, обработку и выдачу данных во взаимодействии с процессами реального времени. (Поэтому миникомпьютеры называют машинами реального времени.) В отличие от обычной вычислительной машины, для которой данные готовит человек, миникомпьютер реального времени сам извлекает их из управляемого процесса, а в системе с замкнутой цепью сам же реализует в виде управляющих воздействий результаты их обработки.

Активность свойственна миникомпьютерам не только в системах управления — миникомпьютерные

системы программирования обычно являются диалоговыми, разговорными: в процессе редактирования текста, отладки и выполнения программы машина направляет действия программиста, указывает обнаруживаемые ею ошибки, запрашивает значения параметров и т. п. Все это существенно облегчает работу.

Наконец, нельзя не назвать такие качества мини-компьютера, как портативность, терпимость к условиям эксплуатации, высокая надежность. По оформлению и условиям использования миникомпьютеры сродни скорее лабораторным приборам, а не цифровым машинам в привычном представлении. Они не требуют кондиционированного климата, не боятся пыли, работают в лабораториях и цехах. Подобно приборам, миникомпьютеры выпускаются в настольном исполнении и в виде блоков, рассчитанных на вмонтирование в приборную стойку вместе с внешними устройствами и включенными в систему приборами.

Рассмотренные особенности миникомпьютера не только обусловили его высокую экономическую эффективность и невиданно широкую применимость, но коренным образом изменили характер использования цифровой техники. Доминикомпьютерные системы непременно строились по принципу сконцентрированной (централизованной) обработки. Господствовал «закон Гроша», согласно которому стоимость компьютера пропорциональна корню квадратному из его мощности. Для миникомпьютеров этот закон оказался недействительным. Они предоставили более дешевую мощность, чем большие цифровые машины, а микропроцессорная мощность оказалась дешевле суперпроцессорной. В связи с этим возник новый принцип построения систем — принцип *распределенной (рассредоточенной)* обработки. Согласно этому принципу процессорная мощность цифровой системы не концентрируется в одном месте, а рассредотачивается так, чтобы была вблизи ее потребителей. Данное превращение структуры системы напоминает изменение характера электропривода, произошедшее в 30-х годах, когда один большой электромотор с ременными передачами к станкам был заменен индивидуальными для каждого станка электродвигателями.

§ 2. Возникновение и развитие промышленности миникомпьютеров

Первым миникомпьютером явилась созданная в 1963 г. американской фирмой Digital Equipment Corp. (DEC) 12-битная машина PDP-5 (Programed Data Processor-5 — программируемый обработчик данных-5). Именно ее употребили в качестве контроллера для управления ядерным реактором. Машина, характеризующаяся временем выполнения одноадресной команды 18 мкс, поставлялась с памятью 4096 слов и телетайпом за 30 тыс. долл. По-видимому, воплощенный в ней новый подход сразу получил понимание и правильную оценку: менее чем за 2 года было поставлено сто PDP-5, а с апреля 1965 г. начался выпуск ее усовершенствованной версии — PDP-8 с временем выполнения одноадресной команды 3 мкс и ценой 18 тыс. долл. Выпуск миникомпьютеров серии PDP-8 (ей посвящена гл. 6 в нашей книге) продолжается до сих пор, причем число этих машин, выпущенных фирмой DEC, превысило в 1977 г. 50 тыс.

Следует заметить, что PDP-5 не была первой машиной с коротким словом. С 1960 г. Control Data Corp. выпускала 12-битную CDC-160 стоимостью 60 тыс. долл., в 1961 г. появилась 15-битная микропрограммная TRW-130. Однако ни в этих, ни в ряде других короткословных машин, в том числе появившихся после создания PDP-5 и PDP-8, идея миникомпьютера не получила достаточно полного воплощения. Их производители не смогли воспринять нового подхода и не сумели реализовать связанных с ним преимуществ. Только PDP-5 явилась совершенным прототипом компьютера-контроллера и вместе с последовавшей за ней PDP-8 убедительно продемонстрировала исключительные возможности машины этого типа как высокоэффективного орудия рационализации и автоматизации труда, повышения его производительности и качества.

Впрочем, PDP-5 возникла не на голом месте, и вовсе не случайность, что ее создали инженеры DEC. По признанию президента DEC К. Олсона, отправным пунктом того направления цифровой техники, на котором важнейшей вехой явилась PDP-8, была созданная в начале 50-х годов в Массачусетском техноло-

гическом институте цифровая машина Whirlwind («Вихрь»).

Эта машина, разработка которой началась в 1944 г., явилась первой машиной реального времени и, по-видимому, первой машиной с коротким (16-битным) словом. Кроме того, с ней связан ряд технических новшеств, существенно повлиявших на дальнейшее развитие цифровой техники, в частности, в ней впервые использована память, выполненная на ферритовых сердечниках.

Выдающаяся роль «Вихря» по достоинству оценена — после многих лет эксплуатации (до 1969 г.) он занял почетное место в национальном музее техники США (Smithsonian Institution вблизи Вашингтона) в ряду таких известных первых электронных цифровых машин, как ENIAC, SEAC, Univac 1. По физическим размерам «Вихрь», состоявший из стоек высотой 3 м, нисколько не походил на миникомпьютер, однако по существу в нем было уже многое из того, чем миникомпьютер принципиально отличается от машин чисто вычислительного назначения, и прежде всего его активный характер, способность к взаимодействию с людьми и с окружающей средой.

К. Олсон считает, что важнейшим достоинством «Вихря», заимствованным от него миникомпьютерами, является доступность. Разработчики «Вихря» стремились к тому, чтобы машину было легко понять и использовать. Вполне естественно, что DEC, основанная в 1957 г. группой инженеров, вышедших из Массачусетского технологического института, во главе с Олсоном, восприняла и продолжила эту похвальную традицию.

В выступлениях сотрудников DEC неизменно подчеркивается, как важный фактор успеха, легкость понимания и простота использования миникомпьютера: «Система должна быть не только умной, но и легкой для понимания», «Машины широкого применения должны быть простыми для использования» и т. п. Все это, конечно, прописные истины, но как раз они послужили главной причиной беспрецедентного успеха.

По-видимому, ко времени появления PDP-8 необходимость этого типа машины более чем назрела: новое направление сразу привлекло внимание как пользователей, так и производителей и стало бурно

развиваться. В течение 1966—1968 гг. в производство миникомпьютеров включились десятки фирм, многие из которых были специально образованы как «минимейкеры» — производители мини. Из уже имевшихся фирм преимущества миникомпьютеров первыми оценили, естественно, производители измерительных приборов и научного оборудования. В 1966 г. приборостроительная компания Hewlett-Packard начала выпуск 16-битного миникомпьютера HP 2116A, за которым последовал ряд все более совершенных моделей, поставивших эту фирму в число ведущих в миникомпьютерной промышленности. К производству миникомпьютеров подключились также фирмы Varian Assoc., Texas Instruments Inc., Lockheed Electronics Co. и др. Из вновь образованных фирм наибольшую известность как производители миникомпьютеров приобрели: Interdata Inc. (основана в 1966 г.), Computer Automation Inc. и General Automation Inc. (обе созданы в 1967 г.) и, наконец, Data General Corp., основанная в 1968 г.

Computer Automation специализируется на поставке машин-полуфабрикатов для комплектования установок и систем, выпускаемых другими фирмами. Ее машины отличаются доступностью цен: первая 8-битная модель стоила менее 7 тыс. долл., а 16-битная — менее 12 тыс. долл. В 1971 г. 8-битная Alpha 8 с 4 тыс. слов памяти стоила 2,8 тыс. долл., а в «голом» исполнении (Naked Mini) при поставке партией на менее 200 экз. — 1,7 тыс. долл.

General Automation в 1968 г. выпустила SPC-8 — первую 8-битную машину с ценой менее 5 тыс. долл. с 4 тыс. слов памяти. Она поставляла также 16-битные миникомпьютеры общего назначения SPC-16.

Texas Instrument в 1968 г. выпустила 16-битную TI 980, затем TI 960. Interdata производит многорегистровые 16-битные миникомпьютеры и различного назначения минисистемы. Data General всемирно известна семействами четырехаккумуляторных миникомпьютеров NOVA и ECLIPS.

В конце 60-х годов существовало около 70 фирм — производителей миникомпьютеров, а число выпускаемых ими моделей, по-видимому, было более ста. Спрос на мини быстро возрастал, и начать дело можно было при наличии 2—3 инженеров, программиста и биз-

несмена. Поэтому фирмы по производству мини-компьютеров возникали и росли, как грибы, появлялись все новые модели машин, высокими темпами совершенствовалась технология, неуклонно снижались цены. Быстрому развитию способствовала непродолжительность цикла разработки миникомпьютеров: в то время как на создание большой цифровой машины уходит обычно 3—5 лет, разработку миникомпьютера удается выполнить за 6—8 месяцев.

Примером того, как из года в год происходили модернизация и удешевление миникомпьютеров, может служить серия PDP-8, в которой менее чем за 10 лет, с 1965 по 1974 гг., появились 8 конструктивно-технологических разновидностей одной и той же машины, от исходного образца, выполненного на транзисторных модулях с проволочными междумодульными соединениями, до размещенного на одной печатной плате процессора PDP-8/A, поставляемого за 1/8 исходной (1965 г.) цены. Основным источником удешевления машины явилась автоматизация производства, в частности, тестирования и испытаний всех деталей, узлов и машины в целом. Осуществленная на базе миникомпьютерных систем автоматизация не только свела к минимуму затраты труда, но и обеспечила безупречное качество продукции. В 1971 г., введя очередную усовершенствованную версию — PDP-8/E, DEC создала для ее производства линию, рассчитанную на выпуск 1 тыс. машин в месяц. Заметим, что общая численность персонала фирмы составляла в то время около 6 тыс. человек, причем помимо PDP-8, выпускавшейся в нескольких модификациях, параллельно производились по меньшей мере 3 модели PDP-11, большая система обработки данных DECsystem-10, десятки наименований периферийных устройств и ряд других изделий.

Первые миникомпьютеры использовались главным образом в качестве запрограммированных для конкретного применения контроллеров в измерительных и управляющих системах. При этом не требовалось сколько-нибудь сложного программного оснащения (операционных систем, развитых средств автоматизации программирования) — реализующая требуемые функции прикладная программа создавалась на языке ассемблера или непосредственно в коде машины

и в процессе эксплуатации ее, как правило, не приходилось перепрограммировать. Соответственно данному способу применения аппаратура первых минисистем также была предельно скромной — процессор, 4 тыс. слов памяти, телетайп и несложное оборудование для взаимодействия с исследуемыми или управляемыми объектами — аналого-цифровые и цифро-аналоговые преобразователи, мультиплексор, датчик времени.

Вскоре, однако, практика показала, что скромные и выносливые мини могут много больше, чем простое выполнение функций контроллера. Им стали поручать обработку накапливаемых в процессе эксперимента результатов измерений, подготовку данных о ходе производства, в котором они участвуют, выполнение инженерных расчетов. В связи с начавшимся таким образом расширением сферы применений мини стали обрастать периферийным оборудованием и более совершенным программным оснащением, что создавало еще более широкие возможности их использования в новых областях.

Дорогие, громоздкие и сложные в обслуживании периферийные устройства больших цифровых машин не подходили для использования в минисистемах, поэтому с возникновением потребности оснащения миникомпьютеров периферийным оборудованием развернулись работы по созданию менее производительных, но дешевых, простых и малогабаритных устройств, появились специализирующиеся на этом деле фирмы, образовалась новая отрасль промышленности — производители минипериферийных устройств.

Появление сравнительно дешевых и легко обслуживаемых периферийных устройств открыло широкие возможности создания миникомпьютерных систем обработки данных (специализированных и общего назначения), послужило технической базой и стимулом для ускоренного развития программного оснащения. Миникомпьютеры вторглись в сферу господства больших цифровых машин, успешно конкурируя с ними в административно-хозяйственных, научно-технических и иных применениях. Первостепенная роль принадлежит миникомпьютерам в современных системах связи, в сетях распределенной обработки данных, в автома-

тизации сферы обслуживания, торговли, банков, издательского дела и т. п. Очень полезными оказались миникомпьютеры в образовании.

Структура промышленности миникомпьютеров существенно отличается от структуры промышленности больших цифровых машин, которая представлена немногими, но крупными фирмами — поставщиками готовых систем, и менее крупными, но более многочисленными поставщиками периферийного оборудования. Производители миникомпьютеров поставляют не только и даже не столько готовые системы, как отдельные процессоры с памятью и необходимым периферийным оборудованием для построения систем пользователями или фирмами — строителями систем. Периферийные устройства и память поставляют также фирмы, специализирующиеся на производстве устройств того или иного типа. Строители систем укомплектовывают заказанную систему подходящей аппаратурой и создают прикладные программы. Системные программы, как правило, поставляются самими производителями миникомпьютеров. Такое разделение труда позволяет сочетать высокий технический уровень и преимущества крупносерийного производства цифрового оборудования с компетентностью строителей систем в приложениях, для которых они создают эти системы.

Другая характерная для промышленности миникомпьютеров форма сотрудничества связана с существованием фирм, выпускающих оборудование, в которое миникомпьютеры входят как компоненты. Эти так называемые OEM (Original Equipment Manufacturers) фирмы приобретают миникомпьютеры в незавершенном конструктивном оформлении, в виде своего рода «полуфабрикатов» и обычно крупными партиями, а следовательно, по сниженным ценам. В той или иной мере на сектор OEM работают практически все производители миникомпьютеров, причем, например, Computer Automation работает только на этот сектор.

Важной частью промышленности миникомпьютеров является производство полупроводниковых приборов. Существует даже тенденция рассматривать миникомпьютеры как результат развития технологии интегральных полупроводниковых приборов («интегральных схем»). Однако первые миникомпьютеры

были построены не на интегральных схемах, и правильнее считать, что как раз миникомпьютеры, создали основу для успешного применения в цифровой технике больших интегральных схем, продемонстрировав практическую важность и выработав образцы простейшего типа цифровой машины, которую оказалось возможным реализовать в виде выполненного на миниатюрной полупроводниковой пластинке *микропроцессора*. В больших цифровых машинах условия применения приборов высокой степени интеграции значительно менее благоприятны, и поэтому миникомпьютеры располагают существенным технологическим (а следовательно, и экономическим) преимуществом.

С появлением в начале 70-х годов микропроцессора нижний уровень цен на цифровые машины опустился вскоре до десятков долларов и сфера экономически доступных применений цифровой техники вновь многократно расширилась. С одной стороны, развернулось массовое производство 4-битных и 8-битных микропроцессоров, нашедших обширные применения, для которых миникомпьютеры в обычном исполнении были неприемлемы по цене или физическим характеристикам. С другой стороны, осуществление на микропроцессорной основе 12-битных и 16-битных миникомпьютеров привело к дальнейшему их удешевлению, вследствие чего возникли такие новые массовые применения, как, например, «персональный компьютер» или one-on-one (отдельный компьютер для каждого студента).

Следует заметить, что низкая цена микропроцессора имеет существенное значение главным образом при использовании его в качестве контроллера, т. е. без периферийных устройств, а в системах с периферийными устройствами стоимость процессора составляет малую часть общей стоимости. Поэтому наряду с представленным микропроцессорами развитием в направлении все более маломощных машин, с другой стороны, совершается неуклонное возрастание мощности процессоров, используемых в старших моделях минисемейств. Появившиеся в середине 70-х годов так называемые «супермини» обладают мощностью средних моделей семейства IBM S/370, в то время как стоят в 3—5 раз дешевле.

Темпы наращивания производства в промышленности миникомпьютеров в 2—3 раза выше, чем в промышленности, выпускающей большие вычислительные машины, причем преимущественное развитие в последние годы невычислительных применений цифровой техники, таких как обработка текстов (word processing), цифровая связь, информационно-поисковые системы, а также все более широкое внедрение в практику концепции распределенного процессирования служат гарантией того, что ускоренный рост минисектора в будущем сохранится.

Определенное представление о темпах роста промышленности миникомпьютеров можно составить на основании следующих данных о фирме — зачинателе этой промышленности — Digital Equipment Corp. За пять лет, с 1972 по 1977 гг., годовой доход DEC увеличился более чем в 6 раз, составив в 1977 финансовом году 1,06 млрд. долл.; число выпущенных машин, составлявшее к 1972 г. немногим более 12 тыс., в конце 1977 г. вплотную приблизилось к 100 тыс.; в 1977 г. корпорация обладала 25 заводами в нескольких странах и капиталом более 2 млрд. долл. (Во время основания в 1957 г. персонал DEC состоял из 3 человек, а капитал составлял 60 тыс. долл.) Заметим, что все это происходит в условиях хронического застоя в экономике капиталистических стран, в то время, когда такие авторитетные производители больших цифровых машин, как General Electric, RCA и Херох, вынуждены были отказаться от дальнейшего участия в бизнесе компьютеров.

До середины 70-х годов промышленность миникомпьютеров практически не затрагивала интересов производителей больших вычислительных машин и, можно сказать, последние не придавали миникомпьютерам серьезного значения. Наряду с большими машинами они выпускали малые, как правило, специализированные машины («псевдомини»), например, IBM System/3, Burroughs B1700, Control Data System 17, но по существу эти машины к миникомпьютерам не имели отношения, за исключением только серий 12-битных и 16-битных машин Honeywell, начатых в 1969 г. моделями H112 и H316.

Однако после того, как миникомпьютеры проникли на рынок систем обработки данных и стали весьма

успешно конкурировать с большими машинами в безраздельно принадлежавших этим машинам областях, отношение ведущих производителей больших машин к «минибизнесу» изменилось. В ноябре 1976 г. корпорация IBM анонсировала начало производства миникомпьютеров Series/1. В течение 1977 г. было поставлено около 3 тыс. машин Series/1 и IBM планирует значительно увеличить их выпуск, надеясь к 1980 г. занять до 15% рынка миникомпьютеров.

Включение в производство миникомпьютеров крупнейших поставщиков цифровой техники, несомненно, вызовет дальнейшее убыстрение роста промышленности миникомпьютеров.

§ 3. Истоки эффективности миникомпьютеров

Эффективность можно определить как отношение полезного эффекта к затратам, ценой которых этот эффект произведен. Когда говорят, что миникомпьютеры эффективны, то имеют в виду значительное увеличение производительности труда, которое они способны обеспечить при сравнительно небольших капиталовложениях.

Но почему миникомпьютеры эффективны? Каким образом и за счет чего достигнута их способность быстрой и обильной отдачи? Почему значительно более мощные, выполняющие более сложные операции большие вычислительные машины обладают этой способностью в гораздо меньшей мере?

На данные вопросы нельзя ответить, указав какую-нибудь одну причину вроде того, что секрет успеха — низкая цена. Нет такого секрета, а есть множество взаимосвязанных факторов и решений, рациональным сочетанием которых достигается успех.

Отправным пунктом является реализуемое в миникомпьютерах принципиально отличное от традиционного понимание сущности и назначения цифровой машины. В традиционном представлении цифровая машина — это математический инструмент, в огромной степени увеличивший способность человека вычислять характеристики и моделировать механизмы исследуемых явлений. Очевидным недостатком такого подхода является то, что машина замкнута на человека: человек предоставляет ей подлежащие обработке данные

и сам использует результаты. Отсутствие непосредственного контакта машины с исследуемым объектом не только замедляет работу, но и является причиной крайней ненадежности: как правило, гарантировать достоверность результатов невозможно, потому что достаточно малейшего изъяна в методе вычислений, или в его программной реализации, или просто случайной ошибки в исходных данных и решение, получаемое при помощи высокоточных машинных операций, оказывается совершенно неверным (хорошо еще, если это удастся установить). Но и добившись решения, удовлетворяющего условиям поставленной задачи, нередко обнаруживают, что задача поставлена неверно, недостаточно точно соответствует действительности.

Указанные и многие другие трудности использования цифровой машины как математического инструмента не существуют или существенно ослаблены в случае, когда машина используется как инструмент *кибернетический*, т. е. как автомат, систематически корректирующий свою деятельность при помощи *обратной связи* от обслуживаемых им объектов (которыми, в частности, могут быть и люди). Мини-компьютер является воплощением именно такого кибернетического подхода к цифровой машине, и в этом заложена возможность высокой эффективности, которую создатели миникомпьютера неплохо реализовали, хотя и не руководствовались общей теорией вопроса, а просто стремились создать практичную машину. Нетрудно понять, что машине данного типа не нужна большая точность представления чисел и высокопроизводительная арифметика, потому что она не занимается длинными вычислениями, а продвигается малыми шагами, постоянно оценивая расхождения и внося поправки, но зато ей крайне необходима гибкая и быстродействующая система взаимодействия с внешним миром и надежность функционирования, без которой немыслима долговременная работа в автоматически действующей системе.

Конечно, кибернетическое использование цифровой машины не исключает и не заменяет традиционного математического использования, так же как экспериментальное исследование действительности не отрицает необходимости теоретических моделей — прогресс происходит путем сочетания и взаимодействия того и

другого. До появления миникомпьютеров машин-экспериментаторов практически не было, и хотя их численность увеличивается теперь очень быстро, по-видимому, оптимальное соотношение «теоретического и практического» секторов цифровой техники далеко еще не достигнуто.

В экономическом отношении кибернетическая машина по сравнению с математической обладает очевидными преимуществами более простого, скорого, широкого и менее рискованного применения. Кроме того, способность машины самостоятельно добывать необходимые для работы данные и реализовывать полученные результаты, т. е. функционировать совершенно автоматически, без участия человека, позволяет свести к минимуму расходы по ее обслуживанию и потребность в соответствующем персонале.

Другими важными составляющими эффективности миникомпьютеров являются рациональность и гибкость организации как самих миникомпьютерных систем, так и их производства. Характер производства миникомпьютеров в значительной степени обусловлен пониманием того, что для различных применений нужны не различные машины, а различные комбинации одних и тех же машин. Реализация этого принципа позволяет выиграть дважды. Во-первых, производство изделий небольшой номенклатуры проще, его можно лучше оснастить и автоматизировать, а освоение, использование и обслуживание систем, построенных из однотипных компонент, существенно облегчается и удешевляется. Во-вторых, осуществление конкретной системы в виде комбинации стандартно сопрягаемых компонент позволяет варьировать ее состав, выбирая наиболее подходящую для данного применения конфигурацию, наращивая или модифицируя систему по мере изменения предъявляемых к ней требований.

Приспособляемость, наращиваемость и модифицируемость миникомпьютерных систем предоставляет пользователю возможность приобретать только необходимое ему оборудование и доукомплектовывать или перестраивать свою систему, если в этом возникает потребность. Вместе с низкими (вследствие массового производства) ценами комплектующих устройств данная особенность миникомпьютерных систем представ-

ляет собой весомый фактор увеличения их эффективности. В практике традиционных цифровых машин действие этого фактора ограничено: конкретному применению так называемой «универсальной» машины, как правило, сопутствует недоиспользование ее ресурсов, а специализированная для данного применения машина оказывается обычно более дорогой вследствие мелкосерийности производства. В миникомпьютерах преимущества специализации и массового производства совмещены.

Высокая эффективность миникомпьютеров имеет и другие источники: простую машину удастся тщательно спроектировать, умеренность требований к быстродействию упрощает проблему охлаждения и допускает применение схем с высокой степенью интеграции. Благоприятным является также преимущественное развитие в последние годы наиболее соответствующих природе миникомпьютеров невычислительных применений и распределенной обработки данных. Наконец, очень важным фактором является простота миникомпьютерных систем — в простой системе «накладные расходы» всегда меньше, чем в сложной, и работать с ней легче.

В отличие от больших цифровых машин, для которых главным показателем является «сырая» вычислительная мощность (число операций в секунду), миникомпьютеры оптимизируются на достижение реальной эффективности в широком спектре применений. Производители миникомпьютеров исходят из того, что пользователям нужна не «сырая» мощность, а орудие, с помощью которого они смогут решить свои практические проблемы за приемлемую цену. Другими словами, главным показателем при оценке миникомпьютера (как и всякого орудия производства) является реальная эффективность, и ее увеличение естественно становится предметом особой заботы.

§ 4. Применения миникомпьютеров

Говорят, что проще перечислить те немногие области, в которых миникомпьютеры еще не нашли использования, чем дать сколько-нибудь полный перечень применений этих поистине универсальных машин. Добавим, что сложным может оказаться поиск

незатронутых миникомпьютерами областей, поскольку таковых практически не осталось.

Совершенно не соответствующим духу миникомпьютеров представляется употребление их в качестве вычислителей, т. е. в роли специально предназначенных для вычислений традиционных ЭВМ. Однако, вопреки своей, казалось бы, непригодности для вычислительных применений, миникомпьютеры все шире используются и как вычислительные машины, вытесняя «приспособленные» ЭВМ малой и средней мощности, обладающие меньшей величиной отношения производительность/цена. В ряде случаев миникомпьютеры-вычислители успешно конкурируют даже с большими ЭВМ. Например, в вузах используемые для обслуживания студентов большие системы разделения времени заменяют несколькими минисистемами, обеспечивающими требуемый уровень обслуживания за существенно меньшую цену и без риска поломки системы в целом.

Учитывая потребности вычислительных применений, производители миникомпьютеров предусмотрели возможность резкого повышения арифметической мощности машин путем подключения необязательных в типовом комплекте блоков для выполнения операций с высокой скоростью при повышенной точности представления чисел, а также с плавающей запятой. Программное оснащение миникомпьютеров пополнилось эффективными компиляторами, средствами редактирования и отладки программ, вполне комфортабельными операционными системами пакетной обработки и мультитерминального разделения времени.

В большой степени под влиянием вычислительных применений возникли особо мощные минисистемы («супермини»), по производительности сравнимые со средними моделями IBM 360/370 при значительно меньшей стоимости. «Супермини» успешно конкурируют с традиционными вычислительными системами общего назначения как в деловых, так и в научных применениях. Широкое распространение получили также малые минисистемы административно-хозяйственного и учебного назначения. А в качестве вызова самым мощным вычислительным машинам создаются мультиминикомпьютерные комплексы, характеризующиеся огромной производительностью и исключитель-

ной живучестью: выход из строя отдельных работающих в комплексе машин приводит к соответственному снижению его производительности, но не к прекращению функционирования. Например, комплекс Tandem-16 (Non-Stop «Miniplex»), включающий до 16 миникомпьютеров, связанных высокоскоростными магистралями, обеспечивает готовность 0,999999 (один час простоя в 100 лет). В ряде случаев путем подобного комплексирования возможно решение вычислительных проблем, непосильных даже самым мощным современным машинам, однако организовать эффективное использование комплекса в общем случае чрезвычайно сложно.

Ввиду эффективности миникомпьютерных систем обработки данных и возможности создания миникомпьютерных комплексов высокой производительности нередко говорят о конце эры больших ЭВМ, не обеспечивающих дешевой вычислительной мощности. Вместе с тем создаются все более мощные процессоры и не видно, чтобы потребность в них могла полностью исчезнуть. Применения, требующие высокой концентрации процессорной мощности, существуют и будут существовать. Отказываться в таких применениях от больших машин так же нелепо, как и насаждать чрезмерную централизацию обработки в тех случаях, когда она может производиться рассредоточено и автономно. Как мини, так и «макси» хороши на своих местах, и надо стремиться к разумному их употреблению и сочетанию.

Первостепенным по масштабам и по эффективности является применение миникомпьютеров для автоматизации управления процессами и машинами, т. е. именно то применение, для которого миникомпьютеры были созданы. В сфере производства это оптимизация непрерывных технологических процессов (химия, нефтепереработка, металлургия), управление автоматическими линиями, конвейерами, числовое управление станками, контроль качества деталей, тестирование и регулировка узлов, автоматическое изготовление изделий микроэлектроники и печатных плат, регулирование систем производства и распределения электроэнергии, в частности ядерных реакторов, управление складами, упаковкой и транспортировкой изделий, сбор и обработка данных о ходе производства и т. д.

В сфере науки и образования это автоматизированное проведение экспериментов, работа в испытательных и измерительных комплексах и приборах (спектроскопия, хроматография, кристаллография, кардиология), сбор, упорядочение и обработка результатов наблюдений, моделирование, программированное обучение, информационно-поисковые системы, машинное проектирование, редактирование текстовых материалов, научно-технические вычисления. В сфере обслуживания это управление транспортом, связью, торговлей, банками, гостиницами, лечебными учреждениями, системами водо-, газо- и электроснабжения, резервирование мест, наблюдение за состоянием госпитализированных больных, контроль параметров окружающей среды, сбор и обработка данных административно-хозяйственного характера и многое другое.

Кроме того, миникомпьютеры широко используются в системах обработки данных в сочетании с большими вычислительными машинами, в частности, в подсистемах сбора, подготовки и ввода данных в большие машины, в сетях цифровой связи, а также в качестве органов управления и диагностики. Например, весьма эффективную форму организации подготовки и ввода данных представляют созданные на основе миникомпьютеров станции клавишного ввода на диск или на магнитную ленту (key-to-disk, key-to-tape). В составе такой станции имеется один или несколько (иногда десятки) терминалов типа телетайпа или электронно-лучевого дисплея с клавиатурой, с которых осуществляют ввод в память миникомпьютера, сверку, редактирование и запись в установленных форматах данных и программ. Отредактированные и выверенные записи миникомпьютер переписывает на подключенный к нему магнитный диск или на магнитную ленту. Затем пакет дисков или бобину ленты переносят на большую машину для считывания. Таким образом достигается увеличение скорости и повышается надежность ввода, предоставляются удобства редактирования и сверки, а по сравнению с использованием перфокарт имеет место значительное удешевление ввода.

Другим примером являются станции теледоступа, в которых миникомпьютер, взаимодействующий с боль-

шой цифровой машиной по линии связи, оборудован устройствами ввода/вывода и обеспечивает возможность подготовки и ввода работ в систему, а также вывод полученных результатов. Существуют специализированные станции теледоступа, в которых мини-компьютеры выполняют сложные и ответственные операции. Примером может служить система безбумажных денежных расчетов (sales order entry system), позволяющая производить оплату покупок путем взаимодействия с сетью цифровой связи, включающей установленные в банках цифровые машины. Находящийся в магазине миникомпьютер, получив код клиента, связывается с машиной соответствующего банка и, соблюдая необходимые предосторожности против несанкционированного доступа, снимает со счета затребованную сумму. Миникомпьютеры выполняют функции управляющих и коммутационных элементов в сетях цифровой связи — связных процессоров (front-ends) при больших машинах, концентраторах данных и коммутаторов сообщений.

Миникомпьютерные системы для измерений и управления в реальном масштабе времени имеют в своем составе специальное оборудование сопряжения с исследуемыми или управляемыми объектами и датчик реального времени (real-time clock). Наличие датчика времени позволяет программировать последовательности отсчетов измеряемых величин и выдачи управляющих воздействий с необходимыми временными интервалами. Измерение физических параметров (температуры, давления, плотности, ускорения и т. п.) производится посредством датчиков, сопоставляющих измеряемым величинам соответственно изменяющиеся электрические сигналы, которые после усиления и масштабирования преобразуются с помощью аналого-цифрового преобразователя в последовательность чисел, представляющих в некотором масштабе значения измеряемой величины в те моменты времени, когда были произведены отсчеты.

Быстродействующий миникомпьютер способен воспринять и зафиксировать в памяти десятки и сотни тысяч чисел в секунду. Однако практически, как правило, требуется значительно меньшая частота отсчетов, поэтому миникомпьютер способен регистрировать одновременно значения десятков, сотен и даже

тысяч различных параметров. Для этого выходы соответствующих датчиков подключают к входу аналого-цифрового преобразователя посредством мультиплексора, который предоставляет возможность воспользоваться преобразователем поочередно каждому из датчиков на время, необходимое для осуществления отсчета. Последовательность обегания датчиков и частоты отсчетов определяются программой. Наряду с отсчетом значений аналоговых сигналов имеется возможность опроса устройств, выдающих данные в цифровой форме, — ключевых и клавишных наборов кода, счетчиков, измерительных приборов с цифровым выходом, а также однобитных датчиков типа «да/нет», например, контактов, которые оказываются замкнутыми или разомкнутыми в зависимости от положения или состояния связанных с ними механизмов.

Измерительные и управляющие минисистемы предполагают также средствами для обратной передачи цифровых и аналоговых сигналов от процессора к контролируемым объектам. В сочетании с возможностью воспринимать и перерабатывать информацию, отражающую состояние данных объектов, это позволяет осуществить управление ими по принципу обратной связи. Как минимум процессор производит сравнение получаемых от датчиков значений с хранящимися в памяти эталонами и посредством цифро-аналогового преобразователя воздействует на соответствующие объекты корректирующими сигналами, величины которых пропорциональны рассогласованию, т. е. разности между показанием датчика и эталоном. Выданное процессором значение корректирующего сигнала действует, пока не произойдет новый отсчет корректируемого параметра и не будет вычислено новое рассогласование. С помощью мультиплексоров на входе аналого-цифрового и на выходе цифро-аналогового преобразователей один миникомпьютер легко обеспечивает регулирование по нескольким десяткам параметров, одновременно осуществляя выдачу управляющих импульсов в предусмотренной программой временной последовательности, запоминание необходимых данных о ходе процесса, сигнализацию о выходе контролируемых параметров за пределы установленного для них диапазона значений и т. п.

Минисистема, работающая в режиме сбора и регистрации данных, производит опрос датчиков и запоминание отсчитанных значений, возможно, с проверкой на невыход за установленные пределы, но не выдает корректирующих сигналов по цепи обратной связи. В измерительных и испытательных системах воздействие процессора на исследуемый объект употребляется обычно не с целью регулирования, а для автоматического управления последовательностью операций и задания условий, при которых требуется выполнить измерения или испытания. Например, процессор в программированной последовательности включает и выключает генераторы возбуждающих импульсов, устанавливает значения их параметров, управляет приборами, обеспечивающими требуемый температурный режим, режим электропитания, уровень помех, амплитуду и частоту тряски и т. п. Одним словом, миникомпьютер выполняет все, что приходится делать в процессе проведения эксперимента человеку-экспериментатору, от поддержания условий в которых должны осуществляться измерения, до упорядочения и обработки полученных результатов. Перепоручение этой кропотливой и утомительной работы машине, так же как передача машине функций человека-оператора в системе управления, позволяет во много раз увеличить эффективность труда, повысить достоверность получаемых результатов, сократить сроки и небывало расширить масштабы исследований. Применение миникомпьютеров в качестве программируемых контроллеров и регистраторов не ограничивается сферой производства и научных исследований — аналогичные функции миникомпьютеры выполняют во множестве других областей, таких как навигация, управление оружием, охрана окружающей среды, медицинское обслуживание и т. д.

Коренные изменения претерпевает с внедрением миникомпьютеров важная и обширная сфера текстовой информации. На технической основе миникомпьютеров возникла особая отрасль промышленности по выпуску систем обработки текстов (word processing), совершенно преобразующих традиционную технику составления и редактирования текстов, от написания писем до подготовки к изданию книг. В простейшем варианте это однотерминальная минисистема с

магнитными кассетами, позволяющая с помощью клавиатуры и электронно-лучевого дисплея редактировать введенный в память машины текст и печатать его копии. Мощные системы включают 16 и более терминалов, обслуживаемых в режиме разделения времени, дисковую память, запоминающую десятки и сотни миллионов знаков, оборудование для работы по линиям связи и сопряжение с фотонаборными устройствами для автоматического набора текстов. В сочетании с возможностью передачи текстовой информации по линиям связи компьютеризованные системы редактирования приобретают особую гибкость и эффективность.

Например, газетные издательства, используя системы этого рода, вплотную приблизились к безбумажному выпуску газеты: тексты сообщений корреспондентов, поступающие по линиям связи, запоминаются на магнитных дисках, редакторы «правят» и компануют материал в памяти машины с помощью дисплейных терминалов, подготовленный таким образом номер газеты выдается на фотонаборное устройство.

Приведенные примеры не могут дать представления о всем многообразии применений миникомпьютеров, однако из них нетрудно усмотреть важную особенность, свойственную подавляющему большинству применений миникомпьютеров. Она заключается в том, что применение миникомпьютера непременно оказывается высокоэффективным, резко повышает производительность труда, открывает качественно новые возможности прогрессивного развития.

Первоисточником этого замечательного качества является удачно реализованное в миникомпьютерах понимание цифровой машины как универсального инструмента — орудия, позволяющего автоматизировать не только вычисления, но и многие другие виды труда. Вместе с тем, необходимое условие высокой эффективности миникомпьютеров составляет принцип нацеленности на конкретное применение. В противоположность вычислительной машине общего назначения, употребляемой одновременно для выполнения многих разнохарактерных работ, миникомпьютер при всей его универсальности используется в каждом конкретном применении, как правило, для одной четко ограни-

ченной цели. Если целей несколько, то используют несколько миникомпьютеров. Это вовсе не значит, что миникомпьютеры не пригодны для многоцелевого употребления — существуют высокоразвитые многофункциональные системы, построенные на базе одного-двух микропроцессоров. Но залогом простоты и эффективности как отдельной минисистемы, так и использования миникомпьютеров в качестве элементов систем, является одноцелевое применение.

ГЛАВА 2

СТРУКТУРА И СОСТАВ МИНИКОМПЬЮТЕРНОЙ СИСТЕМЫ

Определение миникомпьютера как обладающей теми или иными особенностями цифровой машины является весьма расплывчатым в том, что нет скольнибудь четкой очерченности состава цифровой машины. Название «цифровая машина» употребляют применительно как к простейшему контроллеру, так и к комплексу, включающему несколько процессоров, множество контроллеров и различное периферийное оборудование. Трудно провести линию раздела между машиной и ее программным оснащением, без которого многие современные цифровые машины не могут выполнить ни одной команды, хотя программы по установленным представлениям не входят в состав машин. Оставив эти трудные вопросы терминологам, будем впредь в тех случаях, когда существен состав рассматриваемого оборудования, пользоваться термином *миникомпьютерная система*, обозначая им всю совокупность аппаратуры, программ и прочих средств, вовлекаемых в конкретное применение миникомпьютерной техники.

По составу, сложности и мощности диапазон миникомпьютерных систем так же широк, как и диапазон применений миникомпьютеров. В нижнем конце этого диапазона находятся программируемые контроллеры, выполняемые на базе простейших процессоров с минимумом памяти и внешних связей, в верхнем его конце — многофункциональные системы обработки данных, работающие в режимах реального времени, разделения времени и пакетной обработки, мультипроцессорные комплексы высокой производительности, сети

цифровых машин. Но несмотря на столь сильное различие в масштабах, составе и характере функционирования, миникомпьютерные системы (впрочем, как и цифровые системы вообще) в известном смысле обладают единой структурой, строятся из принципиально одних и тех же единиц аппаратуры (процессоры, запоминающие устройства, средства связи с внешним миром), управляются одним и тем же методом хранимой программы. Таким образом, можно говорить об общей структуре миникомпьютерной системы, включив в нее характерные элементы и связи, присущие системам этого типа в их наиболее развитых и полных версиях, чтобы другие, менее полные варианты минисистем получались из общей схемы удалением тех или иных ее частей.

Как во всякой цифровой системе, в составе миникомпьютерной системы различаются прежде всего две категории средств: аппаратура (hardware) и программное оснащение (software). Соотношение функций, реализуемых в системе аппаратурой, и программируемых функций может быть самым различным. В принципе все программируемые функции можно «встроить» в аппаратуру, отказавшись от возможности программировать работу системы. С другой стороны, можно обходиться крайним минимумом простейшей аппаратуры, реализуя все необходимые функции путем программирования.

Рациональный выбор соотношения аппаратуры и программ является одним из важнейших и сложнейших вопросов проектирования систем. Перепоручение функций аппаратуре позволяет обеспечить более высокую производительность, но сопряжено со значительным удорожанием и специализацией системы. Преимущественно программная реализация дешевле и позволяет оперативно настраивать систему на различные применения, однако в отношении быстродействия программированное исполнение существенно уступает аппаратуре.

В миникомпьютерных системах разделение функций между аппаратурой и программами носит подвижный, плавающий характер: имеется возможность работать с минимумом аппаратуры, реализующей базовый набор операций, на основе которого программируются все необходимые в системе функции, но имеется также

возможность добавления в состав системы устройств, выполняющих те или иные функции со значительно большей скоростью, чем соответствующие программы. Кроме того, возможен промежуточный вариант — *встроенные программы (firmware)*, т. е. программы, хранимые в постоянной (только для чтения) памяти, обладающей обычно в 3—5 раз большим быстродействием, чем основная память машины.

В последние годы в связи с созданием высокоскоростных запоминающих устройств, допускающих как чтение, так и запись, в миникомпьютерах находит все большее применение микропрограммная реализация критичных в отношении времени выполнения функций: исполнительные механизмы процессора управляются микропрограммами, вызываемыми в специальную *управляющую память*. Увеличение быстродействия достигается при этом как за счет того, что управляющая память характеризуется меньшим временем обращения, так и за счет большей длины хранимых в ней слов, составляющей обычно 24—48 битов.

§ 1. Аппаратура миникомпьютерной системы

Миникомпьютерная система (за исключением, пожалуй, случая, когда она построена по принципу рассредоточенного процессирования) разделяется на две части: *центральную или внутреннюю и периферийную или внешнюю*. Внутренняя часть системы, включающая *центральный процессор* или комплекс процессоров и *главную (первичную, внутреннюю) память*, интерпретирует выполняемые программы и поступающие извне сигналы, осуществляет преобразование данных, управляет работой системы в целом. Внешняя часть включает устройства периферийной (вторичной) памяти и устройства, сопрягающие систему с внешним миром, осуществляющие ввод данных и программ, представленных на перфокартах, перфолентах или на магнитных лентах, прием информации, поступающей по линиям связи, регистрацию показаний приборов и датчиков в измерительных системах, восприятие сигналов, а также производящие вывод информации из системы в том или ином виде (печатание или выдача на экраны текста, графика, передача сообщений в линии связи, различного рода сигнализация и управляющие воздей-

ствия на исполнительные механизмы связанных с системой машин). С точки зрения внутренней части системы основная функция периферии заключается в том, чтобы вводить в систему необходимые для работы данные и выводить результаты их переработки, поэтому периферийное оборудование часто называют оборудованием *ввода/вывода*. Однако устройства периферийной памяти, хотя отчасти и выполняют функцию ввода/вывода, характеризуются другой важной функцией — они, как правило, составляют основную память системы, превышающую по объему внутреннюю (главную) память в сотни и тысячи раз.

Центральный процессор включает три основные функциональные единицы: (центральное) устройство управления, устройство преобразования данных, называемое обычно арифметико-логическим устройством, и аппаратуру управления периферией. Выборку команд выполняемой программы и обрабатываемых данных процессор производит из главной памяти и из регистров подключенных к нему периферийных устройств. Кроме того, процессор располагает собственной памятью в виде отдельных регистров того или иного назначения, а нередко также в виде набора регистров, организованного как стек или как массив. По сравнению с ячейками главной памяти регистры характеризуются значительно большим быстродействием и компактностью адреса, поскольку их число обычно не превышает 16.

Минипроцессоры существенно отличаются от процессоров больших цифровых машин, предназначенных главным образом для выполнения вычислений. Типичный минипроцессор — это не столько вычислитель, сколько распорядитель, координирующий работу всех звеньев системы, в том числе, может быть, имеющихся в ее составе специальных арифметических процессоров, анализирующий возникающие в ходе работы ситуации и обеспечивающий предусмотренное программой в каждом конкретном случае реагирование. Соответственно, основными параметрами минипроцессора являются не точность представления чисел (практически все современные минипроцессоры обладают 16-битным словом, совершенно недостаточным для вычислительной машины) и не скорость выполнения арифметических операций, а быстрота реакции на поступающие

извне запросы, переключения с программы на программу, восприятия, тестирования и пересылки данных. Обычные для современных минипроцессоров длительности этих операций выражаются микросекундами, а в особо быстродействующих моделях — десятками долями микросекунды.

Арифметические способности центрального процессора минисистемы, как правило, более чем скромны — в наборе его команд имеется обычно только команда сложения (иногда сложения и вычитания) двоичных чисел с фиксированной запятой, представленных каждое одним машинным словом. Поэтому арифметические действия реализуются подпрограммами, причем в виду того, что машинное слово не обеспечивает удовлетворительной точности представления чисел, программируются операции с удвоенной и с утроенной длиной операндов как с фиксированной, так и с плавающей запятой. Такая арифметика, естественно, не является быстрой: минипроцессор, выполняющий команду сложения однословных операндов за две-три микросекунды, затрачивает на выполнение программированной операции типа умножения-деления с операндами удвоенной длины миллисекунды. Значительная часть времени при программированном выполнении операций расходуется на выборку команд из главной памяти, поэтому использование более быстрого постоянного запоминающего устройства или микропрограмм, вызываемых в быстродействующую управляющую память, позволяет сократить указанное время в 5—10 раз, с миллисекунд до сотен микросекунд. Применение специальных арифметических процессоров, включаемых в систему на правах периферийных устройств, обеспечивает большее убыстрение — до десятков микросекунд с учетом пересылки операндов и записи результата в главную память, составляющих существенную часть суммарного времени выполнения операции с помощью дополнительного процессора.

В противоположность дорогостоящим и громоздким процессорам больших вычислительных машин, которых изготовители миникомпьютеров шутливо называют «бегемотами», минипроцессоры дешевы и компактны. Даже в минисистемах, бедно оснащенных периферийным оборудованием, стоимость центрального процессора обычно не превышает 10% стоимости всей

входящей в систему аппаратуры, а в системах с расширенной главной памятью и развитой периферийной частью стоимость процессора становится просто несущественной в общей сумме затрат: минипроцессор стоит сотни (выполненный в виде большой интегральной схемы — десятки) долларов, а цена системы редко бывает ниже 10 тыс. долл.

По физическим размерам минипроцессор также занимает в общем объеме оборудования системы скромное положение: даже в исполнении на элементах малой интеграции он представляет собой либо 3—5 малых (размером менее странички из школьной тетради) плат с печатным монтажом, либо одну плату соответственно большего размера, а в условиях большой интеграции весь процессор, называемый в этом случае микропроцессором, выполняется на одной пластинке полупроводника («чипе») размером приблизительно 5×5 мм. Обычно минипроцессор вместе с главной памятью, источником электропитания и аппаратурой управления некоторыми периферийными устройствами конструктивно оформляется, подобно радиоизмерительным лабораторным приборам, в виде ящика, на лицевой панели которого, выполняющей функции пульта управления, размещены ключи для набора кодов и световые индикаторы состояния регистров. Как правило, изготовители предоставляют два варианта конструктивного оформления — настольный и для размещения в стандартной 19-дюймовой монтажной стойке.

Конструктивная завершенность указанных вариантов оформления миникомпьютера не является препятствием для модификации или расширения состава используемого оборудования. Во-первых, в самом ящике предусматриваются свободные места для установки дополнительных плат, благодаря чему возможно наращивание в известных пределах емкости главной памяти, включение специализированных процессоров, расширение возможностей управления периферийными устройствами. Во-вторых, к заключенной в ящике внутренней части системы при помощи кабелей подключаются периферийные устройства с их контроллерами, а также расширители, обеспечивающие, например, дальнейшее наращивание главной памяти или сочленение процессора с другими процессорами.

Приспособленность минипроцессора к варьированию в широких пределах конфигурации системы является его важнейшим качеством. Этим качеством в наиболее полной мере обладают процессоры, организованные по принципу *единой магистрали* (Unibus, Monobus). Принцип единой магистрали выражается в том, что для передачи данных между любыми компонентами системы пользуются одной многопроводной линией (магистралью), к которой подключены все работающие в системе устройства. В составе магистрали имеются провода адреса, провода данных, а также специальные провода для управления и синхронизации. Количества проводов (шин) адреса и данных равны числам битов, составляющих соответственно адрес и слово данных. Для передачи слова некоторому устройству код адреса этого устройства выдают на шины адреса, код слова — на шины данных и возбуждают управляющий провод приема данных. Для получения слова выдают код адреса устройства на шины адреса и возбуждают провод, вызывающий передачу, на что указанное устройство отвечает выдачей слова на шины данных. Замечательно, что требуемое устройство определяется не по месту его нахождения, а по адресу, на который оно откликается. Благодаря этому в каждое из гнезд, предназначенных для подключения к магистрали устройств, можно включать любое используемое в системе устройство, чем и обеспечивается легкая варьированность конфигурации системы.

Рассмотренные выше варианты законченного оформления минипроцессоров предназначены для конечных пользователей, приобретающих миникомпьютерную технику с целью непосредственного использования в тех или иных применениях. Но кроме того, минипроцессоры выпускаются в незавершенном оформлении, в виде своего рода полуфабрикатов для OEM (Original Equipment Manufacturers) — производителей оборудования, в котором минипроцессоры используются как комплектующие изделия. Примерами такого оборудования являются автоматизированные измерительные системы, масс-спектрометры, анализаторы треков, системы редактирования текстов, машины с цифровым управлением, станции ввода данных и т. п. Вариант для OEM, как правило, отличается более скромным оформлением, упрощенной лицевой панелью, отсутст-

вием источника питания, а нередко процессор поставляют буквально «голым» (Naked Mini) — в виде платы с разъемами для включения в систему. Поставки для OEM осуществляются крупными партиями (обычно не менее 50 — 100 экземпляров) и со значительной уценкой.

Эксплуатационные параметры у минипроцессоров намного лучше, чем у процессоров больших вычислительных машин: в то время как последние требуют особых условий эксплуатации — поддержания в узких пределах температуры окружающей среды, нормальной влажности, минипроцессоры способны работать в сравнительно трудных климатических условиях. В обычном исполнении они работоспособны в диапазоне температур от 0 до 50—55°C, а в специальном для тяжелых условий (ruggedized) — от -55 до +95°C, но данное расширение диапазона сопряжено с увеличением цены в 3—5 раз. Соответствующая стойкость обеспечивается в отношении влажности, загрязненности воздуха, механических воздействий, колебаний напряжения электросети. Наряду с этим минипроцессоры практически не требуют технического обслуживания — среднее время между неисправностями составляет для них десятки тысяч часов, т. е. годы непрерывной эксплуатации.

Главная память миникомпьютера — это та память, из которой процессор при выполнении программы считывает команды и данные, а также записывает в нее полученные результаты. В структурном отношении память представляет собой пронумерованную совокупность ячеек. Номера ячеек — целые числа, начинающиеся, как правило, с нуля, — принято называть адресами. Совокупность всех адресов образует *адресное пространство*.

Обычно ячейка обеспечивает запоминание *машинного слова*, т. е. одной команды или порции данных, потребляемых процессором в качестве одного операнда. Но в некоторых миникомпьютерах размер ячейки памяти равен *байту* (8 битам) и для запоминания 16-битного машинного слова используется пара ячеек. При этом считывать и записывать можно как байты, так и целые слова, однако запись или считывание слова возможны только по четному адресу, с которым в таком случае связывается пара ячеек, соответствующая данному адресу и следующему за ним нечетному

адресу. Четный байт является младшей половиной слова, нечетный — старшей. Возможность манипулировать байтами важна в связи с вводом/выводом, обработкой текстов и в ряде других применений, поэтому адресация байтов в главной памяти оправдана, несмотря на то, что адресуемая при данной длине адреса ее емкость уменьшается в два раза.

Основными технико-экономическими характеристиками главной памяти являются: емкость, быстродействие, надежность, цена.

Емкость памяти измеряется числом байтов или слов (с указанием длины слова в байтах или в битах), которые могут храниться в данной памяти одновременно. Главная память миникомпьютера может в зависимости от его применения обладать емкостью от нескольких сот байтов до сотен тысяч и даже миллионов байтов. Так называемая «базовая конфигурация» минисистемы обычно включает память емкостью 4KW ($K = 1024$ — «двоичная тысяча», W — Word, слово), что при длине слова 16 битов, т. е. 2 байта, эквивалентно 8KB (8×1024 байтов, 8 килобайтов). Максимальная емкость адресуемой процессором памяти в минисистемах низкой стоимости составляет, как правило, 32KW или 64KB, в минисистемах средней стоимости (соответственно мощности) — 128KW или 256KB, а в так называемых «супермини» — 1MW или 2MB ($M = 1024K$). Нарращивание памяти производится блоками по 4KW, 8KW, 16KW.

Быстродействие главной памяти характеризуется длительностью *цикла* запоминающего устройства, т. е. минимальным временем между обращениями. Обратная величина называется максимальной частотой обращений. Наиболее быстродействующие современные запоминающие устройства, выполняемые на биполярных полупроводниковых элементах, обладают длительностью цикла 30—40 нс, что эквивалентно 25—30 млн. обращений в секунду. Однако такие устройства в качестве главной памяти миникомпьютеров не используются, потому что они непомерно дороги, потребляют значительную мощность и требуют интенсивного охлаждения. Миникомпьютерные запоминающие устройства выполняются на ферритовых сердечниках и на полупроводниковых элементах типа МОП (металл — оксид — полупроводник). Типичными значениями дли-

тельности цикла этих устройств являются: на ферритах — 0,8—1 мкс, на МОП-элементах — 0,4—0,6 мкс. Существенным преимуществом ферритовой памяти является неразрушаемость ее содержимого при выключении электропитания. Полупроводниковая память «стирается» при нарушении электропитания и для обеспечения надлежащей ее устойчивости приходится принимать специальные меры — подстраховывать источник питания аккумуляторной батареей, дублировать программы и данные в периферийной памяти, вводить коды, обнаруживающие и исправляющие ошибки. Несмотря на это, МОП-память, начиная с середины 70-х годов, когда она стала дешевле ферритовой, завоевывает все большее предпочтение. Ее преимущества — технологичность и успешно осваиваемая возможность дальнейшего совершенствования технологии: на протяжении 70-х годов число битов, размещаемых в одном модуле (на одном чипе), увеличилось с нескольких сот до 4К, а в отдельных случаях — до 16К. При этом стоимость, энергопотребление и габариты модуля практически не увеличиваются, т. е. в соответствующее число раз снижается цена бита и удельная мощность питания, повышается плотность «упаковки» информации, надежность. Постоянно улучшается также, хотя и не в такой степени, быстродействие МОП-памяти. Стоимость бита в 4К-чипе составляет приблизительно 0,1 цента, а в готовом запоминающем устройстве, выполненном на таких чипах, — около 1 цента.

Длительность цикла главной памяти является по существу параметром, характеризующим быстродействие миникомпьютера в целом: время выполнения безадресной (не связанной с выборкой операндов из главной памяти) команды, как правило, равно или немногим больше длительности цикла памяти, выполнение однооперандной команды занимает обычно два (с косвенной адресацией три) цикла памяти и т. д. Грубо можно считать, что время выполнения команды равно суммарной длительности используемых циклов памяти. Чтобы увеличить быстродействие миникомпьютера, надо сокращать цикл главной памяти. Сокращение цикла путем использования физически более быстрых устройств, применяемых в больших цифровых машинах, в миникомпьютерах нецелесообразно по

приведенным выше соображениям. Обычно стремятся убыстрить память без значительного ее удорожания и ухудшения эксплуатационных характеристик. К употребляемым для этого мерам относятся: хранение часто работающих программ в обладающем более коротким циклом постоянном запоминающем устройстве, микропрограммирование с использованием сравнительно небольшой быстродействующей управляющей памяти, высокоскоростное запоминающее устройство небольшой емкости в качестве буфера (cache) между процессором и главной памятью, перемежающееся (interleaving) срабатывание блоков запоминающего устройства при обращениях по последовательным адресам, расщепление цикла на фазы чтения и записи, позволяющее в течение одного (несколько удлиненного) цикла осуществить считывание операнда и запись результата операции.

Другим фактором, существенно влияющим на быстродействие минисистемы, является эффективность управления периферией. Этот вопрос рассмотрен в гл. 5.

Периферийное оборудование миникомпьютерной системы в функциональном отношении разделяется на четыре группы: средства ввода-вывода, устройства периферийной памяти, аппаратура для работы с линиями связи, специальное оборудование для сопряжения миникомпьютера с приборами и машинами. В зависимости от конкретного применения минисистема может включать или не включать те или иные компоненты периферийного оборудования, а также пополняться ими по мере необходимости. Минимальная система общего назначения обычно обходится одним периферийным устройством — телетайпом, обеспечивающим возможность ввода с клавиатуры и с перфоленты и вывода путем перфорации и печати.

На начальном этапе телетайп являлся основным и практически единственным видом периферийного оборудования, широко использовавшегося для миникомпьютеров. Периферийные устройства больших цифровых машин оказались неподходящими для применения в минисистемах: их высокая производительность, как правило, не находила достаточно полного использования, а цены, превосходящие нередко в несколько раз цену самого миникомпьютера, были совершенно

неприемлемы. В дальнейшем были созданы различные виды специально минипериферийного оборудования, обладающего требуемой производительностью, приемлемого по ценам, малогабаритного, надежного и легко обслуживаемого. Возникла и интенсивно развивается новая отрасль промышленности по выпуску минипериферийных устройств, в которой наряду с производителями миникомпьютеров заняты десятки фирм, специализирующихся исключительно на производстве минипериферийного оборудования.

Устройствами ввода/вывода можно считать любые периферийные устройства в том смысле, что все они по отношению ко внутренней части системы выполняют функции ввода или вывода информации. Но в более узком смысле это устройства, осуществляющие ввод информации, подготовленной человеком, и вывод в форме, удобной для использования по усмотрению человека. На больших цифровых машинах традиционными устройствами этого рода являются считыватели и перфораторы перфокарт, а также печатающие устройства (принтеры). На миникомпьютерах перфокарты используются редко (по-видимому, из-за дороговизны и громоздкости карточных перфораторов), а их функцию выполняет *перфолента*.

Работать с перфолентой можно уже при наличии телетайпа. Телетайпы сравнительно дешевы (1,5—2 тыс. долл.) и производят как считывание, так и перфорацию ленты, однако скорость 10 зн./с., с которой работает телетайп, недостаточна, особенно при вводе. Поэтому системы, работающие с перфолентой, доукомплектовываются высокопроизводительными перфолентными устройствами: фотоэлектрическим считывателем (200—500 зн./с. при цене около тысячи долларов) и ленточным перфоратором (50—80 зн./с., 1—2 тыс. долл.). Нередко считыватель и перфоратор совмещены в одном комбинированном устройстве. Более быстрое перфолентное оборудование применительно к миникомпьютерам не имеет смысла — оно значительно дороже и сложнее в обслуживании, а сколько-нибудь заметного увеличения производительности системы, как правило, не обеспечивает.

Повышение эффективности ввода/вывода в минисистемах происходит не путем увеличения скорости перфолентных устройств, а путем замены перфоленты

более совершенным носителем — магнитной лентой, обеспечивающей в десятки-сотни раз бóльшую плотность записи информации и соответственно скорость ввода/вывода, а также возможность стирания и многократного использования для новой записи. В качестве средства ввода/вывода магнитная лента используется в виде *кассет*, заимствованных из кассетных магнитофонов. Типичная миникассета обладает емкостью 100—200КВ, а время ее считывания/записи составляет около 10 с. Устройство кассетного ввода/вывода стоит несколько сот долларов, а сама кассета — не более 10 долл. В небольших системах кассеты используются как периферийная память.

Другое значительное усовершенствование техники ввода/вывода в минисистемах связано с введением электронно-лучевых *дисплеев*. В сочетании с клавиатурой дисплей представляет собой эффективное средство общения человека с машиной. В этой роли дисплейные терминалы, благодаря их бесшумности, быстродействию и удобству редактирования текста, успешно конкурируют с телетайпом. Минисистемы комплектуются недорогими (1—2 тыс. долл.) дисплейными терминалами, обеспечивающими отображение обычно до 24 строк по 60—80 знаков в строке.

Наиболее существенным недостатком экранных дисплеев является то, что они не обеспечивают возможности получения печатной копии, которую телетайп предоставляет. Поэтому, помимо дисплейного терминала, требуется *печатающее устройство*, возможно, одно на несколько терминалов. В противоположность используемым с большими цифровыми машинами громоздким колесным принтерам стоимостью в десятки тысяч долларов, печатающие устройства минисистем малогабаритны, сравнительно дешевы и производят меньше шума, обладая вполне достаточной скоростью печати. Типичным является матричный (т. е. изображающий алфавитно-цифровые знаки точками матрицы, например, 5×7) принтер стоимостью 2—4 тыс. долл., печатающий до 200 строк в минуту, до 120—132 знаков в строке. Имеются также недорогие строчные и побуквенные принтеры, обеспечивающие высококачественную (не точечную) печать.

Периферийная память, в отличие от главной, не используется процессором непосредственно, а представ-

ляет собой хранилище, из которого производится вызов («подкачка») в главную память требующихся по ходу дела программ и данных. От периферийной памяти требуется не быстрота доступа к произвольно адресуемому слову, а достаточно большая емкость за приемлемую цену. Главная память обеспечивает скорость, периферийная — емкость, и их сочетанием достигаются удовлетворительные характеристики памяти в целом при умеренных затратах.

Основным оборудованием периферийной памяти в минисистемах, как и в системах с большими машинами, являются *магнитные диски*. Имеется широкий выбор устройств памяти на дисках, разработанных и выпускаемых специально для использования в мини-компьютерных системах. Эти устройства дешевле, компактнее и проще в обслуживании, чем соответствующие устройства для больших машин, но обладают, как правило, меньшей емкостью.

Основными разновидностями устройств памяти на магнитных дисках, применяемых в минисистемах, являются следующие.

Постоянные (несъемные) диски с неперемещаемыми головками записи/считывания (отдельная головка на каждую дорожку) характеризуются наибольшим быстродействием и наиболее высокой ценой в расчете на байт или бит. Среднее время доступа составляет обычно 8—17 мс, емкость одного дисководов 64КВ — 2МВ, цена 2—4 цента/байт (т. е. 0,25—0,5 цент/бит), включая контроллер, стоимость которого составляет, как правило, не менее половины стоимости дисковода. Стоимость памяти на дисках данного вида только в 2—4 раза ниже стоимости МОП-памяти в 4К-чипах, а с освоением 16К-чипов и эта разница исчезнет — диски с неперемещаемыми головками в ближайшем будущем вымрут, так же как широко распространенные в недавнем прошлом магнитные барабаны. На смену им приходят уже выпускаемые отдельными фирмами запоминающие устройства на полупроводниковых элементах с зарядовой связью и магнитные устройства с перемещаемыми доменами.

Съемные (сменные, пакетные) диски с перемещаемыми головками записи/считывания являются самым дешевым и обеспечивающим наибольшую емкость видом дисковой памяти. В крупных минисистемах

используются устройства этого вида с емкостью пакета дисков до 200МВ при среднем времени доступа 30—70 мс и стоимости $(1-2) \times 10^{-3}$ цент/байт, но типичные для миникомпьютерных применений пакетные диски характеризуются более скромными параметрами: емкость 10—100МВ, среднее время доступа 30—150 мс, стоимость байта памяти — сотые-десятые доли цента.

Диски-картриджи особенно популярны на миникомпьютерах, потому что наиболее просты в обращении и обслуживании, а следовательно, надежны в эксплуатации. Диск-картридж — это диск в футляре, предотвращающем повреждение и попадание на рабочие поверхности пыли, которая для диска весьма опасна. Диск не только хранится в картридже, но и устанавливается в нем на дисковод для работы. Типичные значения емкости картриджа: 1,2—7,5МВ, среднее время доступа такое же, как у пакетных дисков, стоимость байта — десятые доли цента. Широко распространены комбинированные дисководы, содержащие 1—3 несъемных диска с перемещаемыми головками и диск-картридж.

Гибкий (floppy) диск является популярным устройством периферийной памяти в малых минисистемах. Его главное достоинство — малые затраты на приобретение: дисковод стоит 400—700 долл., а собственно диск (пластиковый круг в предохранительном конверт-контейнере) — 5—7 долл. Как правило, емкость гибкого диска не превышает 250КВ, а среднее время доступа составляет приблизительно полсекунды, стоимость байта — десятые доли цента. Существуют многочисленные модификации «стандартного» гибкого диска, введенного в начале 70-х годов корпорацией IBM, — Mini-Floppy, Micro-Floppy и т. п.

Наряду с магнитными дисками в качестве периферийной памяти миникомпьютеров используется также магнитная лента. Устройства памяти на магнитной ленте принципиально отличаются от устройств дисковой памяти, которые представляют собой память с произвольным доступом, в то время как на магнитной ленте доступ последовательный: поиск требующейся записи осуществляется просмотром записей в порядке их расположения вдоль ленты. Стандартная бобина вмещает до 750 м ленты, обладающей ем-

костью до 20МВ (при повышенной плотности записи до 50МВ), и время доступа при этом может достигать нескольких минут. Такая память в случае сколь угодно полного использования ее емкости хороша либо для обработки записей в порядке их расположения на ленте, либо как архив для хранения не участвующей в работе информации, например дубликатов программ, рабочие экземпляры которых находятся на дисках.

Однако на миникомпьютерах применяется, кроме того, форматная, адресуемая, подобно дискам, блоками магнитная лента с бобинами небольшой емкости (например, 250КВ) и средним временем доступа не более 10 с. Применяются также форматные кассеты, обладающие аналогичными параметрами, и ленточные картриджи емкостью до 2МВ при среднем времени доступа в несколько десятков секунд. Стоимость байта на форматной ленте составляет десятые доли цента, на кассетах — сотые, на картриджах — тысячные. Все эти устройства выступают в роли заменителей дешевого диска, и по мере распространения и усовершенствования гибких дисков практическое значение их все более уменьшается. Впрочем, преимущества гибкого диска не так велики, чтобы он мог в скором времени полностью вытеснить ленту, а в перспективе и его положение непрочное: полупроводниковые элементы памяти с зарядовой связью и устройства с перемещаемыми доменами, несомненно, заполнят этот сектор периферии миникомпьютеров, как только их цены понизятся до приемлемого уровня.

Связное оборудование минисистемы обеспечивает возможность приема и передачи представленной в цифровой форме информации по линиям связи. Передача по линиям связи отличается от передачи «внутри системы», т. е. между процессором, главной памятью и периферийными устройствами, по меньшей мере в двух отношениях: во-первых, в противоположность многопроводным внутренним магистралям, передающим все биты слова или байта впараллель, линии связи являются однопроводными и передача по ним должна производиться последовательно (побитно), во-вторых, используемые чаще всего телефонные линии не рассчитаны на передачу сигналов, применяемых для представления информации при внутренней передаче,

поэтому на входе линии надо модулировать этими сигналами передаваемые по линии колебания звуковой частоты, а на выходе производить обратное преобразование (демодуляцию). Кроме того, с передачей по линиям связи связано много других проблем, таких как синхронизация приемника и передатчика, автоматический набор номера абонента, обнаружение и исправление ошибок, которые вследствие характерного для линий связи высокого уровня помех возникают весьма часто, мультиплексирование или концентрация сообщений с целью более полной загрузки линии и т. д.

Основными единицами связного оборудования миникомпьютеров являются *связные адаптеры* и *модемы* (модуляторы/демодуляторы). Адаптер производит преобразование выдаваемых ему процессором байтов или слов, представленных параллельным кодом, в последовательности битов для передачи по линии с заданной скоростью, а также обратное преобразование принимаемых посредством модема из линии побитных сообщений в передаваемые процессору в параллельной форме слова или байты. Модем включается между адаптером и линией для осуществления модуляции при передаче и демодуляции при приеме.

Связные адаптеры соответственно реализуемому ими виду связи бывают *асинхронные* (старт-стопные) и *синхронные*. Асинхронные передают каждый байт в виде короткой серии битов, начинающейся стартовым и заканчивающейся стоповым сигналом, промежутки времени между сериями могут быть произвольной длительности. Синхронная передача осуществляется длинными сериями, содержащими десятки-сотни битов, причем в начале серии помещаются сигналы синхронизации, позволяющие приемнику настроиться на частоту и фазу передатчика.

Асинхронная связь менее быстрая, но обходится более простой и дешевой аппаратурой и поэтому в минисистемах используется чаще (за исключением случаев применения миникомпьютеров в сетях цифровой связи в качестве связных процессоров и терминальных станций). Обычно скорость асинхронной передачи не превышает 1200 бит/с. на коммутируемых (с набором номера) телефонных линиях и 2400 бит/с. на выделенных (некоммутируемых) телефонных линиях, соответ-

ствуя следующим стандартным значениям: 75, 110, 150, 300, 600, 1200, 1800, 2400 бит/с. Программируемые адаптеры обеспечивают возможность выбора из предусмотренного в них набора скоростей. В наиболее простом и дешевом варианте адаптер рассчитан на работу с одной линией, но существуют также двухлинейные и многолинейные адаптеры (*мультиплексоры*).

Синхронная связь характеризуется скоростью до 4800 бит/с. на коммутируемых и до 9600 бит/с. на некоммутируемых телефонных линиях. На кабельных широкополосных линиях, используемых в качестве магистралей в больших системах цифровой связи, скорость синхронной передачи составляет обычно 50 000 бит/с., но существуют и более быстродействующие линии, на которых скорость передачи составляет сотни тысяч битов, а иногда и более миллиона битов в секунду.

Модемы различаются в зависимости от используемых способов модуляции и допустимых скоростей передачи. С повышением скорости передачи быстро увеличивается частота ошибок и соответственно количество сообщений, которые приходится передавать повторно. Поэтому в каждом конкретном случае существует некоторая оптимальная скорость передачи, обеспечивающая наибольшую при данных условиях пропускную способность. Современные модемы автоматически оптимизируют свои параметры, приравливаясь к состоянию линии. Весьма удобны модемы, подключаемые к линии посредством акустической связи (*acoustic coupler*), состоящей в том, что телефонная трубка, через которую производится прием и передача, помещается в соответствующее гнездо модема.

Для связи с местными терминалами, расположенными в радиусе нескольких сот метров, существуют так называемые нулевые или *base-band* модемы, которые не производят модуляции и демодуляции, а формируют импульсы тока обычно с амплитудой 20 ма, передаваемые при помощи пары скрученных проводов.

Помимо адаптеров и модемов, связанное оборудование миникомпьютеров включает ряд других устройств: устройства для сопряжения с телеграфными линиями,

автоматического набора номера абонента, арифметические блоки для обнаружения и исправления ошибок. Центральный процессор взаимодействует с этими устройствами и со связными адаптерами таким же образом, как и с устройствами ввода/вывода или периферийной памяти, запрашивая информацию об их состоянии, выдавая команды и данные для передачи по линиям, осуществляя ввод принятых сообщений.

Стоимость связных адаптеров и модемов составляет обычно сотни долларов в расчете на одну линию.

Аппаратура сопряжения миникомпьютеров с приборами и машинами, называемая также оборудованием для сбора данных (data-acquisition equipment), осуществляет восприятие дискретных и аналоговых сигналов в контролируемых системой точках, шкалирование и преобразование аналоговых величин в цифровую форму при вводе и обратное цифро-аналоговое преобразование при выводе, доставку выдаваемых процессором управляющих сигналов в пункты их назначения, измерение промежутков времени между моментами наступления регистрируемых событий, выдерживание заданной частоты отсчетов и т. п. Функциональные компоненты этого оборудования (аналого-цифровые и цифро-аналоговые преобразователи, усилители, мультиплексоры, счетчики, датчики времени) поставляются как в виде отдельных устройств, так и комплексами (подсистемами) лабораторного или промышленного назначения. Такие подсистемы обладают гибкой модульной структурой, позволяющей варьировать в широких пределах их состав и параметры, получать необходимую в том или ином конкретном применении конфигурацию.

Преобразователи *аналог-цифра* и *цифра-аналог*, используемые в миникомпьютерных системах, характерны умеренностью точностных и скоростных параметров: точность, как правило, не превосходит 10—12 битов, частота отсчетов (в преобразователе цифра-аналог — частота смены входных значений) составляет десятки кГц. Для большинства применений этого достаточно, а в случаях, требующих большего, привлекается специальная, более дорогая аппаратура.

Преобразователь *аналог-цифра* используется обычно с *мультиплексором*, посредством которого к его

входу поочередно подключаются обслуживаемые аналоговые каналы, число которых в лабораторных системах составляет от 4 до 32, а в промышленных контрольно-измерительных комплексах — сотни и даже тысячи при скорости обегания 100—200 каналов в секунду. Программно переключаемые на входе преобразователя масштабирующие цепи позволяют устанавливать для каждого канала одно из предусмотренных в данной системе максимальных значений шкалы, которые в отдельных системах составляют от сотых долей вольта до десятков вольт.

Преобразователь цифра-аналог генерирует аналоговые сигналы, задаваемые на его входе таблицами чисел. Эти сигналы используются для управления механизмами или процессами, а кроме того, для визуального наблюдения за работой системы при помощи осциллографа.

Программируемые датчики реального времени (real-time clocks) имеют в своей основе генераторы сигналов фиксированной частоты (1, 10, 100 кГц) и программно управляемые и опрашиваемые счетчики импульсов, на входы которых могут подаваться сигналы этих генераторов, а также сигналы внешних источников. Для измерения временного промежутка между двумя событиями установленный в нулевое состояние счетчик подключается к генератору импульсов с наступлением первого события и отключается с наступлением второго. Для отсчета промежутка заданной длительности на счетчик устанавливают соответствующее значение и осуществляют счет импульсов в режиме убавления — достижение нуля сигнализирует конец отсчитываемого промежутка.

Подсистемы цифрового управления обеспечивают процессору доступ к сотням-тысячам адресуемых точек (битов), состояние которых может быть скопировано в регистры или в ячейки главной памяти, тестируется с помощью команд условного перехода, а в точках, выполняющих функцию вывода, установлено соответственно предписанным процессором значениям. Физически эти точки представляют собой выходы триггеров, контакты реле и другие индикаторы состояний, а также входы усилителей, вентилей, электромагнитных катушек и прочих воспринимающих элементов.

§ 2. Программное оснащение миникомпьютерной системы

Так же как в системах с большими цифровыми машинами, в миникомпьютерных системах программное оснащение подразделяют на *системное* и *прикладное*, хотя в условиях характерной для миникомпьютеров настроенности системы на конкретное применение такое разделение не всегда оправдано.

Системными называют программы, не связанные с определенным применением, а используемые во всех или во многих применениях цифровой машины. Важнейшей функцией системных программ является организация работы аппаратуры и программного оснащения системы в соответствии с заданным режимом, указаниями оператора и информацией, поступающей извне в виде сигналов от обслуживаемых объектов или директив, сопровождающих вводимые программы и данные. Можно сказать, что в данной функции программы представляют собой средство автоматизированного управления работой системы. Другая важная функция системных программ — механизация процесса создания программ, обеспечение возможности пользоваться цифровой машиной для уменьшения трудоемкости программирования.

Комплекс программ, выполняющих первую из указанных функций, принято называть *операционной системой*, а программы, призванные облегчить труд программиста, вместе с используемыми для этой же цели языками и методиками разработки программ составляют *систему автоматизации программирования*.

Операционные системы миникомпьютеров подчинены тем же принципам простоты и настраиваемости на применение, которыми обусловлены характерные черты миникомпьютерной аппаратуры. Сложные многофункциональные операционные системы типа используемых на больших вычислительных машинах нехарактерны для миникомпьютеров: в таких системах велики накладные расходы, а миникомпьютерам с их скудными ресурсами необходима бережливость. За исключением мощных минисистем общего назначения, составляющих конкуренцию традиционным системам обработки данных, на миникомпьютерах предпочти-

тельные специализированные для конкретных применений операционные системы.

Минимальные конфигурации аппаратуры обычно работают с простейшими мониторами реального времени — программами, отзывающимися на прерывания включением в работу соответствующих обслуживающих подпрограмм, которые, как и сам монитор, постоянно находятся в главной памяти. Система этого рода способна обеспечить, например, регистрацию данных и управление лабораторным оборудованием, числовое управление станком или программируемые в режиме диалога на языке basic несложные вычисления. Для реализации такой системы достаточно процессора с 4KW памяти и телетайпом.

С этой («базовой») конфигурацией аппаратуры работают также, используя язык символьного ассемблера и перфоленту. Система обеспечивает возможность подготовки исходной программы, перевода в машинный код, отладки и выполнения, однако все управляющие функции осуществляет оператор. Несмотря на то, что перевод программы в машинный код, редактирование исходного текста и отладка осуществляются в данной системе с помощью машины, для получения в ней даже небольшой рабочей программы человек должен выполнить длинный ряд процедур, требующих напряженного внимания и занимающих немало времени. Для подготовки системы к работе необходимо произвести ввод начального (bootstrap) загрузчика, с его помощью ввести с перфоленты двоичный загрузчик, обеспечивающий ввод последующих программ; ввести редактор текста и подготовить с его помощью текст исходной программы на языке ассемблера, вывести отредактированную программу на перфоленту; ввести ассемблер, произвести ассемблирование рабочей программы (обычно 2 или 3 прохода), отпечатать ее листинг и отперфорировать рабочую ленту; ввести отладчик, провести проверку работоспособности программы и соответствия реализованного алгоритма тому, что требовалось получить, исправить обнаруженные ошибки и, если необходимо, повторить ассемблирование с получением откорректированных листинга и перфоленты.

Такой способ использования машины, типичный для миникомпьютеров 60-х годов, по мере удешевле-

ния и соответственно увеличения емкости главной памяти, благодаря чему появилась возможность применения операционных систем, на протяжении 70-х годов практически вышел из употребления. Несмотря на то, что конфигурация с 4KW (и даже меньше) памяти во многих конкретных приложениях целесообразна и при наличии подходящего монитора может быть весьма удобной в работе, реализовать на ее основе эффективную разработку программ не удастся. Поэтому программное оснащение для конфигураций этого уровня разрабатывают обычно на более мощной системе, располагающей достаточными ресурсами и оснащенной надлежащими средствами автоматизации программирования.

Минимум главной памяти, при котором может работать удобная для разработки программ операционная система без периферийной памяти (точнее, с памятью на перфоленте или на магнитных кассетах) — Stand-alone Operating System (SOS), составляет, как правило, не менее 12KW. Такая система подобна системам пакетной обработки, широко распространенным на традиционных вычислительных машинах, — управляемая директивами оператора она обеспечивает удобства редактирования, ассемблирования, отладки, загрузки и выполнения программ, автоматизирует процессы ввода и вывода.

Другой вид операционной системы — Real-Time Operating System (RTOS) — в простейшем варианте представляет собой упоминавшийся выше монитор реального времени, который может работать при наличии 4KW главной памяти, телетайпа и датчика времени, не обеспечивая, однако, условий для автоматизированной разработки программ. Более развитые и работающие с большей памятью версии RTOS совмещают обработку в режиме реального времени с предоставлением в качестве работы заднего (background) плана, т. е. наименьшего приоритета, возможности осуществлять разработку программ.

Полноценные операционные системы вводятся при наличии периферийной памяти на магнитных дисках, которая, с одной стороны, предоставляет необходимое для их функционирования место, а с другой стороны, является одним из важнейших объектов, обслуживаемых операционной системой. Дисковая операционная

система (DOS) должна, в частности, обеспечить эффективное взаимодействие памяти на дисках и главной памяти, осуществляя упорядоченное размещение на дисках программ и данных, удобный к ним доступ и автоматическую подкачку в главную память требующихся по ходу обработки их сегментов.

По назначению различают три рода операционных систем: пакетной обработки (batch), реального времени (real-time) и разделения времени (time-sharing). Система пакетной обработки призвана обеспечить по возможности наиболее полную загрузку оборудования и, следовательно, максимум его пропускной способности. Система реального времени должна реагировать на поступающие запросы с установленной в зависимости от их приоритета быстротой. Система разделения времени обязана разделять время процессора, память и другие ресурсы между обслуживаемыми совместно пользователями так, чтобы потребности каждого удовлетворялись в течение установленного срока обслуживания, т. е. чтобы время ожидания не превышало допустимого значения.

Типичные для миникомпьютеров дисковые операционные системы — Real-time Disk Operating Systems (RTDOS, RDOS) — строятся на основе мониторов реального времени с возможностью обслуживания одного или нескольких терминалов пользователей и/или выполнения пакетной обработки на заднем плане. Обычно минимальная версия такой системы может работать при 8KW главной памяти и 128KB памяти на дисках. Система состоит из модулей, комбинацией и настройкой которых достигается подгонка ее к данной конфигурации аппаратуры, называемая генерацией ОС. Генерация осуществляется автоматически специальной программой генерации ОС. При изменении состава аппаратуры систему регенерируют. Рассмотренные ранее бездисковые операционные системы SOS и RTOS являются обычно совместимыми с системой RDOS одного и того же семейства миникомпьютеров в том смысле, что программа, разработанная в любой из них, выполнима в других при наличии требующихся ресурсов.

Наряду с RDOS существуют DOS пакетного типа, используемые преимущественно на маломощных конфигурациях, а также системы разделения времени,

работающие на старших и средних моделях, обслуживая 20—30, а в отдельных случаях до 64 терминалов. Небольшие системы с 4—8 терминалами для учебного программирования на языке basic реализуются без дисков.

Особое положение занимают многофункциональные операционные системы, которыми оснащаются наиболее мощные миникомпьютеры — «супермини». Эти системы сочетают режим реального времени с обслуживанием многих терминалов и с пакетной обработкой.

Системы программирования миникомпьютеров позволяют разрабатывать программы как на машинно-ориентированном языке (языке ассемблера), так и на машинно-независимых языках (языках высокого уровня), в качестве которых особенно популярны basic и фортран.

Язык ассемблера, отражающий архитектуру машины «один к одному», обеспечивает возможность создания наиболее компактных и быстродействующих программ, однако программирование на нем по сравнению с программированием на языках высокого уровня намного труднее и дороже. Считают, что в среднем одна строка программы на языке высокого уровня эквивалентна пяти строкам на языке ассемблера и, следовательно, программирование на последнем является в пять раз более трудоемким и дорогим. Тем не менее программирование на языке ассемблера в миникомпьютерных системах широко применяется, потому что при свойственной этим системам ограниченности ресурсов высокая эффективность программ весьма важна. Как правило, на языке ассемблера производится разработка всех системных программ, а также прикладных программ, которые или рассчитаны на массовое применение, или должны удовлетворять таким требованиям в отношении занимаемого объема памяти и времени выполнения, которые не удастся соблюсти при использовании языков высокого уровня.

Снижение трудоемкости и удешевление программирования на языке ассемблера достигаются путем усовершенствования ассемблеров, в частности, с использованием макрогенерации, применением структурированного программирования, созданием действенных средств редактирования и отладки программ. Важным

резервом в этом деле является рационализация архитектуры миникомпьютеров, повышение эффективности машинного языка.

Ассемблерами оснащены все минисистемы, располагающие хотя бы 4KW памяти и телетайпом, однако удовлетворительные условия для разработки ассемблируемых программ обеспечиваются только, когда ассемблер, редактор, загрузчик и программы отладки работают под управлением операционной системы и имеется резерв памяти, избавляющий от обременительных операций с перфолентой. Как правило, существует несколько версий ассемблера, рассчитанных на разные уровни ресурсов и предоставляющих разные возможности — использование макросов, условное ассемблирование, расширенная диагностика ошибок и т. п. Имеются также так называемые *кросс-ассемблеры*, позволяющие ассемблировать программы для данной машины на машинах другой архитектуры. Обычно кросс-ассемблер написан на фортране и в принципе может работать на любой машине, оснащенной фортран-компилятором и располагающей достаточными ресурсами. Выгодно работать с кросс-ассемблером на машине, обладающей комфортабельной операционной системой и достаточно быстрым принтером — трудоемкость и затрачиваемое на разработку программы время могут быть существенно уменьшены.

Одним из важнейших требований, предъявляемых в миникомпьютерных системах к языку высокого уровня, является легкость освоения и применения его пользователем-непрограммистом. Именно поэтому исключительным предпочтением пользуются basic и фортран. Как правило, выполняемые на миникомпьютерах программы пользователей носят расчетный характер, несложны и невелики по объему, так что возможности basic, а тем более фортрана оказываются вполне удовлетворительными.

Basic замечателен не только крайней простотой, но в еще большей степени тем, что отладка и выполнение написанных на нем программ производятся в режиме диалога — программа способна «разговаривать» с пользователем, выдавая на терминал предусмотренные в ней для конкретных ситуаций фразы. Например, она может задавать вопросы, на которые следует ответить «да/нет», или запрашивать значения

параметров, требующиеся для продолжения вычислений, или указывать пользователю на то, что им совершено то или иное ошибочное действие и т. д. Работать в таком активном (его называют также «интерактивным» от английского *interactive* — основанный на взаимодействии) режиме существенно легче, чем с обычными «немыми» программами: машина показывает, что и как следует делать, помогает оперативно исправлять ошибки. В сущности, это типичный для миникомпьютера режим реагирования на воздействия извне, корректирующего работу их источника по принципу обратной связи.

Работа в *basic*-системе значительно облегчается также тем, что в ней исходная, представленная в виде последовательности предложений языка *basic*-программа не перерабатывается компилятором в пригодную для непосредственного выполнения на машине рабочую программу (как это делается с программами, написанными на фортране), а выполняется предложение за предложением путем их *интерпретации* на языке машинных команд. Пользователь, таким образом, имеет дело с единственным представлением программы, а именно исходным ее представлением на языке *basic*. Выполнение программы в режиме интерпретации занимает больше времени, чем выполнение скомпилированной программы, причем, так как во время выполнения в памяти машины помимо самой программы должен находиться интерпретатор, то ее объем более ограничен. Однако доступность системы для пользователя-непрограммиста важнее, чем максимальная отдача, работа на пределе возможностей. Если же возникает необходимость более эффективного использования машинных ресурсов, то можно программировать на фортране или, в крайнем случае, на языке ассемблера. В некоторых системах наряду с *basic*-интерпретатором имеется *basic*-компилятор, что позволяет сочетать удобство разработки программы в режиме интерпретации с возможностью последующей компиляции машинно-эффективной рабочей программы.

Доступность языка *basic* обусловила появление его расширенных (в частности, структурированных) версий и построенных на его основе специализированных систем (например, для программирования автоматизированных экспериментов или для деловых приме-

ний — business-basic). К сожалению, в этом деле нет еще стандартов и, следовательно, отсутствует совместимость даже одинаковых по назначению разработок, выполненных разными фирмами. Впрочем, такой же разноречивостью имеет место и с расширениями фортрана.

Минимальные версии языков basic и фортран нередко имеются даже на системах с 4KW памяти, но их практические возможности крайне ограничены. Удовлетворительные реализации требуют как минимум 8KW или наличия памяти на дисках. Basic-интерпретатор, как правило, имеется в двух вариантах: для одного пользователя и для группы пользователей, обслуживаемых в режиме разделения времени.

Помимо basic и фортрана, на некоторых мини-компьютерах используются алгол-60, RPG, APL, focal, а также менее распространенные специализированные языки dibol, uniapt, comtex и др. На особо мощных моделях имеется кобол.

ГЛАВА 3

ОСОБЕННОСТИ АРХИТЕКТУРЫ МИНИКОМПЬЮТЕРОВ

Затруднения, возникающие при попытке ответить на вопрос: «Что такое миникомпьютер?», в значительной степени обусловлены тем, что миникомпьютеры отличаются от традиционных машин во многих отношениях. Именно поэтому говорят: «Трудно объяснить, что такое миникомпьютер, но стоит его увидеть, как сразу поймешь, что это он».

Конечно, внешность, конструктивное оформление, соответствуя общему замыслу, отражают в той или иной степени характер машины, но существенное отличие миникомпьютеров от прочих цифровых машин может заключаться только в том, что составляет сущность цифровой машины, ее душу. Эту принципиальную сторону цифровой машины обозначали и обозначают различными терминами: «логика машины», «логическая (функциональная, алгоритмическая, математическая) структура», «организация» и т. п., но в последние годы все большим предпочтением пользуется термин *архитектура цифровой машины*.

§ 1. Понятие об архитектуре цифровой машины

Понятие архитектуры включает все то, что машина предоставляет программисту, программирующему на уровне машинных команд. Другими словами, архитектура — это то, что отражается в языке машины: структура памяти, механизмы адресации, функциональная схема процессора, организация его собственной памяти (регистров), операции, форматы команд, средства управления периферийными устройствами, система пре-

рывания и т. п. К архитектуре не относятся особенности физической (технической) реализации машины и такие характеристики, как быстродействие, невидимое программисту совмещение операций во времени, автоматическое буферирование на стыке процессора с главной памятью и прочие усовершенствования, не отражающиеся в языке машины.

Архитектура машин одинакова, если произвольная программа в машинном коде, выполняемая на одной из этих машин, выполняема и на любой другой из них, причем результаты ее работы на всех машинах совпадают. На практике тождественность архитектур понимают менее строго. Например, машины, составляющие семейство единой архитектуры, могут различаться не только в отношении производительности или каких-либо других технических параметров, но также по объему адресуемой памяти, числу регистров процессора, составу набора команд и т. п. При этом говорят, что в семействе имеет место *совместимость снизу вверх*, если произвольная программа в машинном коде, выполняемая на одной из машин семейства, выполняема также на всех входящих в это семейство машинах более богатой архитектуры.

Оптимизация архитектуры миникомпьютеров является важнейшим фактором повышения их эффективности. На этом пути уже немало сделано: высокая эффективность миникомпьютеров достигнута главным образом благодаря их более рациональной по сравнению с традиционными цифровыми машинами архитектуре. Во всяком случае миникомпьютерная архитектура явилась той основой, на которой стало возможным радикальное удешевление процессоров, невиданно расширившее практические применения цифровой техники. Миникомпьютеры убедительно показали, что архитектура цифровых машин может развиваться не только по пути эскалации вычислительной мощности.

Возможности дальнейшей оптимизации миникомпьютерной архитектуры далеко не исчерпаны. Однако теперь, когда стоимость процессоров сведена к минимуму и составляет незначительную часть стоимости системы в целом, оптимизация должна быть нацелена на сокращение тех ее частей, которые «делают погоду», и в первую очередь на снижение стоимости программного оснащения. Другими словами, главной

целью разработчиков архитектуры теперь должна быть не минимизация стоимости аппаратуры, а создание легко и экономно оснащаемой архитектуры. На архитектуру машины следует смотреть не только и даже не столько с точки зрения того, во что обойдется ее инженерная реализация, как с точки зрения оптимизации стоимости цифровой системы в целом, включая программное оснащение.

Архитектура машины является основой, на которой воздвигается все многоярусное здание программной надстройки, и ясно, что данная основа в значительной мере определяет как методы и технику возведения этого здания, так и его существенные черты — компактность, надежность, легкость освоения и использования, расширяемость, наращиваемость, приспособляемость к применениям.

§ 2. Средства описания архитектуры

Существуют и практически используются различные средства описания архитектуры цифровых машин — от обычного (естественного) языка до специально разрабатываемых для этой цели формализованных языков. Важным преимуществом последних является компактность и недвусмысленность выражения понятий и отношений, которыми характеризуется архитектура машины. Однако эти преимущества, как и аналогичные преимущества языка математики, достигнуты путем введения сложной системы знаков и правил, для овладения которой требуется немало труда и времени.

В данной книге описание архитектуры миникомпьютеров осуществлено в компромиссном варианте, представляющем собой сочетание естественного языка с элементами и конструкциями, заимствованными из языков программирования или построенными по имеющимся в этих языках образцам. Настоящий параграф посвящен рассмотрению применяемых в дальнейшем средств описания архитектуры и разъяснению смысла, в котором они употребляются.

В миникомпьютере, как и во всякой цифровой машине, данные и управляющая информация представлены в виде *слов* — последовательностей цифровых (т. е. принимающих дискретные значения) элементов. Самыми простыми и широко применяемыми являются

двоичные (двузначные) элементы — *биты*. Принимаемые битом значения обозначают обычно цифрами 0 и 1.

Значением двоичного слова является последовательность значений его битов. Например, двоичное слово длины 2 принимает четыре значения: 00, 01, 10, 11. Поскольку однобитное слово принимает два значения, а при добавлении каждого нового бита число принимаемых словом значений удваивается, то двоичное слово длины n принимает 2^n значений.

Возможны различные *интерпретации* (истолкования) значений слова. Например, слово, поступающее в устройство управления машины в качестве команды, истолковывается как предписание выполнить те или иные операции, а будучи посланным в арифметическое устройство, то же слово интерпретируется как число — над ним производят арифметические действия. Интерпретация слова заключается в тех действиях, которые это слово вызывает как носитель управляющей информации, или в тех преобразованиях, которым подвергается его значение при обработке в качестве данного. Поэтому описание архитектуры цифровой машины, в сущности, является описанием реализованных в данной машине интерпретаций слов.

Будем представлять слово как *одномерный массив* элементов (в частности, битов), пронумерованных последовательными целыми числами. Нумерацию можно производить в порядке как возрастания, так и убывания номеров. Например, слово A длины 16 может быть описано как $A(0:15)$, $A(1:16)$, $A(15:0)$, $A(16:1)$, $A(0:-15)$, $A(-8:7)$ и т.д. Описание слов обычно выполняется в виде следующих предложений:

binary array $A(0:15)$, $B(1:8)$ — двоичный массив
 $A(0:15)$, $B(1:8)$,
ternary array $X(-13:13)$ — троичный массив
 $X(-13:13)$.

После того как слово описано, на него можно ссылаться, называя его имя без индекса, если речь идет о слове в целом, и с индексом в скобках, если надо обозначить часть слова. Например, если слова описаны в виде $A(0:7)$, $B(0:11)$, то в выражении $A + B(0:5) + B(11)$ A означает слово A в целом, $B(0:5)$ — левую половину слова B , $B(11)$ — одиннадцатый элемент слова B .

В миникомпьютерах двоичное слово интерпретируется чаще всего как упорядоченная совокупность независимых друг от друга битов (*булевский вектор*) и как целое число без знака. Интерпретация слова A длины n как целого числа без знака заключается в том, что его рассматривают как двоичное число в натуральном коде. Например, если слово описано в виде $A(0:n)$, то его числовое значение определяется выражением

$$A(0) \cdot 2^n + A(1) \cdot 2^{n-1} + \dots + A(i) \cdot 2^{n-i} + \dots \\ \dots + A(n-1) \cdot 2 + A(n),$$

где $A(i)$ — значение i -го бита, т. е. 0 или 1.

Числовая интерпретация автоматически включается при выполнении над словами арифметических операций. При этом числа-константы можно записывать в любой системе счисления, избранной, например, по соображениям удобства. Основание системы, в которой представлено число, указывается справа внизу, например, число тринадцать можно записать в различных системах так: 1101_2 , 111_3 , 15_8 , 13 (десятичная запись без указателя основания).

Всякая часть слова при числовой интерпретации рассматривается как новое слово, представляющее число в натуральном коде. Например, если слово $A(0:7)$ обладает значением 01100100_2 , т. е. 100, то значение его левой половины $A(0:3)$ будет 0110_2 , т. е. 6, а значение его правой половины $A(4:7)$ будет 0100_2 , т. е. 4.

Рассмотренная натуральная числовая интерпретация подразумевается всюду, где слово выступает как число непосредственно. Другие числовые интерпретации вводятся посредством специальных функций. Например, интерпретация двоичного слова как числа со знаком, представленного в *дополнительном* коде, задается функцией sgnvl (signed value), которая определена так:

$$\text{sgnvl}(A(0:n)) = \begin{cases} A, & \text{если } A(0) = 0, \\ -(\bar{A} + 1), & \text{если } A(0) = 1, \end{cases}$$

где \bar{A} — побитная инверсия слова A , т. е. результат замены в нем всех нулей единицами и всех единиц нулями.

Булевские операции инверсии («НЕ»), конъюнкции («И») и дизъюнкции («ИЛИ») определены над двоичными словами как побитные, т. е. выполняемые независимо над каждым битом или над каждой парой битов, соответствующих друг другу в словах-операндах. Ради простоты длина слов-операндов предполагается одинаковой. Примеры:

$$A = 0011, B = 0101, \bar{A} = 1100, A \wedge B = 0001, \\ A \vee B = 0111.$$

Арифметические операции определены над словами произвольной длины, причем двоичные слова нормально интерпретируются как представления целых чисел без знака, а при необходимости истолковать слово как представление числа в дополнительном коде применяется функция sgnvl . Выполнение арифметических операций сложения, вычитания и умножения (операция деления не используется) над двоичными словами производится следующим образом: слова-операнды заменяются соответствующими целыми числами, над этими числами выполняются заданные действия и полученное в результате число представляется в двоичном дополнительном коде. Если регистр, запоминаящий результат, не обладает достаточной длиной, старшие биты результата утрачиваются. Например, при сложении 8-битного слова 11101011_2 , интерпретируемого как число без знака 235, с 5-битным словом 11010_2 , т. е. 26, получим 261 или в двоичном представлении 100000101_2 , однако при запоминании этого результата в 8-битном регистре старшая цифра будет утеряна и получится 00000101_2 , т. е. 5 — число, сравнимое с 261 по модулю 2^8 . Такие случаи искажения результата вследствие ограниченной длины регистра называются *переполнением* регистра.

Переполнение является существенной особенностью машинной арифметики. Употребляя при описании архитектуры знаки арифметических операций, мы не будем сопровождать их пометками, напоминающими о возможности переполнения, поэтому следует всегда иметь в виду, что арифметические операции выполняются над регистрами ограниченной длины и что результат, получаемый в n -битном регистре, является, вообще говоря, наименьшим по модулю 2^n вычетом истинного результата.

Регистры процессора описываются как одномерные массивы битов. Примеры: $PC(0:15)$ — программный счетчик (Program Counter), содержащий 16 битов, пронумерованных слева направо, начиная с нуля, $Ac(8:1)$ — аккумулятор, содержащий 8 битов, пронумерованных справа налево, начиная с единицы.

Прием значения в регистр изображается при помощи знака *присваивания* $:=$, причем имя регистра пишется слева от этого знака. Примеры:

- $PC := 0$ — в регистр PC принять нуль,
- $Ac := PC$ — в аккумулятор принять копию текущего значения PC ,
- $PC := PC + 1$ текущее значение PC увеличить на единицу,
- $Ac := Ac + PC$ — к содержимому аккумулятора прибавить текущее значение PC .

Существенно, что прием в регистр значения, содержащегося в другом регистре, производится путем копирования, т. е. без разрушения содержимого регистра-отправителя. Вообще, изменить хранимое регистром значение можно только присваиванием этому регистру нового значения. Специальным случаем является обмен значениями, когда регистр A принимает значение, хранимое регистром B , а регистр B одновременно принимает значение, хранимое регистром A . Эту процедуру можно, воспользовавшись для временного запоминания регистром C , описать так:

$C := A; A := B; B := C.$

Мы введем для обозначения ее специальный знак $:= :$. Описанный только что обмен выражается при помощи этого знака в виде $A := : B.$

Регистры могут быть разной длины, а кроме того, случается присваивать регистру в целом значение, хранимое частью регистра. Поэтому необходимо определить, как присваивать регистру значение, представленное словом, длина которого отлична от длины регистра. Естественно исходить при этом из интерпретации слова как представления целого числа, а в случае, когда короткому регистру присваивается длинное значение, принять, что утрачиваются левые биты, т. е. происходит переполнение регистра. Таким образом, присваивание при несовпадении длин производится по-

средством выравнивания по правым концам слов. Например, пусть регистры $A(1:8)$ и $B(1:4)$ обладают значениями: $A = 11000110_2$, $B = 1010_2$. Выполнив $A := B$, имеем $A = 00001010_2$. Выполнив $A := \text{sgnvl}(B)$, имеем $B = 1010_2$, $A = 11111010_2$. С другой стороны, выполнив $B := A$ или $B := \text{sgnvl}(A)$, имеем в обоих случаях $A = 11000110_2$, $B = 0110_2$.

Наряду с обозначением регистров именами и обозначением части регистра его именем с индексной парой, указывающей отнесенные к данной части биты, мы используем также *конкатенацию* (сочленение) регистров и их частей. Например, при наличии регистров $Lk(1:1)$ и $Ac(1:16)$ можно сочленить их в 17-битный регистр $LkAc$, в котором самым левым будет единственный бит регистра Lk , а затем следуют в порядке возрастания номеров 16 битов регистра Ac . Возможно и другое сочленение — $AcLk$, в котором бит Lk пристроен к регистру Ac справа в качестве хвостового. Аналогично можно сочленять в различном порядке части регистра. Например, конкатенация $Ac(9:16)Ac(1:8)$ представляет собой модификацию слова $Ac(1:16)$, в которой переставлены его левая и правая половины. Присваивание $Ac := Ac(9:16)Ac(1:8)$ равносильно обмену $Ac(1:8) := Ac(9:16)$.

Конкатенация имен рассматривается как имя. В частности, имя-конкатенацию можно употреблять как в правой, так и в левой частях записи присваивания. Это позволяет просто и наглядно изображать сдвиги в регистрах. Например, циклический сдвиг на одно место влево в кольце, образованном из регистров Lk и Ac , можно выразить в виде присваивания $LkAc := AcLk$, а аналогичный сдвиг вправо — в виде присваивания $AcLk := LkAc$. Простой (логический) сдвиг в регистре $Ac(1:n)$ на k мест вправо, $k \leq n$, выражает пара присваиваний: $Ac(k+1:n)Ac(1:k) := Ac$; $Ac(1:k) := 0$; а сдвиг в этом же регистре на k мест влево — соответственно пара: $Ac(1:k) := 0$; $Ac := Ac(k+1:n)Ac(1:k)$.

Главную память миникомпьютера в простейшем случае можно описать как одномерный массив двоичных слов фиксированной длины $m(0:N)$, где N — максимальное значение адреса. Однако возможны и практически используются более сложные структуры главной памяти, описываемые, например, как

многомерные массивы. Например, память, состоящую из 8 блоков, в каждом из которых содержится 4096 слов по 16 битов в слове, можно рассматривать как трехмерный массив $m(0:7, 0:4095, 0:15)$. В принципе такая память допускает адресацию с точностью до бита, но практически обычно ограничиваются адресацией слов и/или байтов. Естественно условиться, что обозначения укрупненных единиц памяти могут не содержать детального описания их структуры. Так, применительно к описанной только что памяти слово $m(3, 2504, 0:15)$ можно обозначить сокращенно $m(3, 2504)$, а блок $m(5, 0:4095, 0:15)$ обозначить просто $m(5)$.

Запоминание данных и выполнение операций над отдельными словами и элементами слов — это необходимые, но не исчерпывающие атрибуты автоматической цифровой машины. Не менее важна ее способность выполнять операции в предписанной программой последовательности и в зависимости от складывающихся в процессе работы условий. Для описания архитектуры машин в этом отношении удобно использовать средства управления, применяемые в языках программирования, а именно:

обычное соглашение о том, что *предложения* программы выполняются в порядке их следования, пока не встретилось предложение, предписывающее отклониться от этого порядка;

предложение **go to** (перейти к ...), предписывающее перейти к выполнению указанного в нем посредством метки предложения;

предложение **if ... then ... else ...** (если..., то..., иначе...). предписывающее произвести одно или другое действие в зависимости от того, выполнено или не выполнено заданное в предложении условие;

скобки **begin** (начало) и **end** (конец), превращающие заключенную между ними последовательность предложений в единое сложное предложение;

возможность обозначать последовательность предложений именем и употреблять это имя вместо текста данной последовательности всюду, где она должна быть включена (простейшая форма макроса или подпрограммы без параметров).

Рассмотрим перечисленные средства более подробно.

Выполнение предложений программы в порядке их следования означает, что действия, предписываемые предложениями программы, выполняются в том порядке, в котором расположены в тексте программы соответствующие предложения. Например, последовательность действий при выполнении фрагмента программы, состоящего из следующих трех предложений:

$$m(A) := Ac; \quad Ac := m(B); \quad Ac := Ac - m(A);$$

будет такой: содержимое аккумулятора Ac копируется в ячейку главной памяти с адресом A ; в аккумулятор принимается копия содержимого ячейки с адресом B ; из нового содержимого аккумулятора вычитается содержимое ячейки с адресом A . Нетрудно убедиться, что выполнение данных предложений в иной последовательности, скажем, сначала 2-е, затем 3-е и за ним 1-е, придает фрагменту совершенно иной смысл. Следовательно, последовательность выполнения программы важна, и, конечно, естественнее всего писать предложения в той последовательности, в которой они должны быть выполнены. Заметим, что в качестве *разделителя* предложений («знака препинания») используется точка с запятой.

В программах, написанных на языке ассемблера, фрагменты которых будут встречаться в дальнейшем, применяется другой способ разделения предложений — каждое предложение пишется с новой строки, и поэтому разделительных знаков не требуется.

Предложение **go to** позволяет программировать произвольную последовательность выполнения предложений программы. Его называют предложением *безусловного перехода*, потому что предписанное им действие есть несвязанный с какими-либо условиями переход к выполнению предложения, на которое указывает **go to**. Предложение, на которое производится переход, снабжают меткой в виде буквы (или сочетания букв и, может быть, цифр) с двоеточием в качестве разделителя. Предложение **go to** содержит копию этой метки как указатель точки, в которую назначен переход. Следующий пример иллюстрирует технику применения **go to**:

$$\begin{array}{l} \quad \quad \quad Ac := 0; \\ L: \quad \quad Ac := Ac + 1; \text{ go to } L; \end{array}$$

В этом примере первое предложение $Ac := 0$ присваивает аккумулятору значение нуль (очищает аккумулятор). Второе предложение, помеченное меткой L , увеличивает содержимое аккумулятора на единицу. Третье предложение **go to** L производит переход по метке L , т.е. в данном случае возврат на второе предложение. В результате возникает бесконечно повторяющийся цикл, состоящий из второго и третьего предложений. С каждым выполнением цикла в аккумулятор добавляется единица — аккумулятор работает как счетчик повторений цикла.

Предложение **go to** может задавать переходы не только назад (как в рассмотренном примере), но и вперед, в виде скачка через следующие за ним предложения. Необходимо, однако, чтобы меткой, которая содержится в **go to**, было помечено одно и только одно из предложений программы.

Помеченное предложение вместе с его меткой пишут с новой строки, помещая метку перед левым краем текста самих предложений, чтобы метку было легко отыскать (см. приведенный только что пример).

Предложение **if... then... else...** является средством ветвления программы: в зависимости от того, выполнено или не выполнено содержащееся в нем условие, выбирается одна из двух предусмотренных в программе ветвей.

Пример:

```
if  $Ac < 0$  then  $Ac := Ac + 1$  else  $Ac := Ac - 1$ ;
```

«Если содержащееся в аккумуляторе число отрицательно, то прибавить к нему единицу, иначе (т.е. если это число не отрицательно) вычесть из него единицу».

Употребляется также сокращенный вариант данного предложения — **if... then...** (если..., то...), соответствующий случаю, когда вторая ветвь, описываемая после слова **else**, оказывается пустой, не содержащей никаких действий. Например:

```
if  $Ac < 0$  then  $Ac := Ac + 1$ ;
```

«Если содержимое аккумулятора отрицательно, то прибавить к нему единицу (а если неотрицательно, то ничего делать не надо)».

Рассмотренный ранее бесконечный (неуправляемый) цикл легко сделать управляемым, используя в нем

предложение **if then**, например, так:

$Ac := m(A);$
L: $Ac := Ac + 1; \text{ if } Ac < 0 \text{ then go to L};$

Цикл будет выполнен ровно N раз, если в ячейке $m(A)$ содержится отрицательное число $-N$. После N -го прибавления единицы содержимое аккумулятора станет равным нулю, условие $Ac < 0$ при этом не выполняется и переход на метку **L** произведен не будет. Будут выполняться дальнейшие предложения программы в порядке их следования.

Предложения **if then else** и **if then** можно гнездовать (употреблять в составе других предложений этого типа), образуя более сложные ветвления. Например, трехальтернативное ветвление можно получить, используя **if then** в составе **if then else**:

if $Ac < 0$ **then** $Ac := Ac + 1$ **else if** $Ac > 0$
then $Ac := Ac - 1;$

«Если содержимое аккумулятора отрицательно, то прибавить к нему единицу, иначе, если положительно, то вычесть единицу, а если не отрицательно и не положительно (т. е. если равно нулю), то ничего не делать».

Скобки **begin** и **end** служат для обрамления сложных предложений, а также как средство устранения неоднозначности смысла некоторых конструкций.

if $Ac(0) \neq 0$ **then begin** $Ac := \overline{Ac}; Ac := Ac + 1$ **end;**

В этом примере два предложения, заключенные в скобки **begin** и **end**, выполняются в случае, если удовлетворено условие $Ac(0) \neq 0$. При отсутствии скобок только первое из этих предложений окажется связанным с условием, а второе будет выполняться всегда, независимо от условия. Другими словами, при отсутствии скобок только первое предложение войдет в состав предложения **if then**.

Еще пример:

if $Ac \neq 0$ **then begin if** $Ac < 0$ **then** $Ac := Ac + 1$
else $Ac := Ac - 1$ **end;**

В этом предложении использованием скобок достигнута однозначность его понимания. Убрав скобки, невозможно будет определить, что выражает оставшаяся запись: то ли **if then else**, в котором содер-

жится **if then**, то ли **if then**, в котором содержится **if then else**.

Употребление имени, которым обозначена некоторая последовательность предложений, вместо самой последовательности допускается ради компактности описания. Это аналогично тому, как для упрощения записи алгебраического выражения вводят буквенные обозначения повторяющихся подвыражений. При описании архитектуры типичными примерами использования этого приема являются имена машинных операций, выражающихся последовательностями более элементарных операций над регистрами и отдельными битами регистров.

§ 3. Факторы, определяющие характер архитектуры миникомпьютеров

Архитектура миникомпьютеров сформировалась и развивается под прямым влиянием запросов практики и, можно сказать, вопреки теоретической науке, рассматривающей развитие цифровой техники как поступательное увеличение сложности машин и наращивание их вычислительной мощности. В прогнозах развития цифровой техники, сделанных в конце 50-х — начале 60-х годов, предрекались супермашины огромной вычислительной мощности и вычислительные системы с сотнями процессоров. Не было и намека на то, что может появиться нечто подобное миникомпьютеру.

Усечение длины слова и арифметического оборудования обеспечило экономическую приемлемость миникомпьютеров и не противоречило техническим условиям их применений. Однако появились внутренние проблемы, такие как недостаточность короткого слова-команды для представления необходимой управляющей информации (обозначения операции, адреса операнда, указателей индекс-регистров), недостаточность короткого слова-адреса для эффективной адресации большой памяти, недостаточность в отдельных случаях точности представления числовых данных и т.п. Развитие миникомпьютерной архитектуры в значительной мере происходило и происходит путем решения этих проблем.

Другой фактор, сильно влияющий на архитектуру миникомпьютеров, — это специфичность методики их

использования. В отличие от традиционных машин, которые функционируют как переработчики данных, вводимых людьми, миникомпьютеры рассчитаны на «вживание» в системы и должны уметь оперативно реагировать на возникающие в этих системах ситуации, проверяя выполнение тех или иных условий и выдавая соответствующие команды исполнительным органам. Поэтому архитектура миникомпьютеров является *периферийно-ориентированной* — управлению периферией (впрочем, как и управлению вообще) в ней уделяется особое внимание.

Еще одним фактором, оказывающим существенное влияние на миникомпьютерную архитектуру, является требование модифицируемости и расширяемости (наращиваемости) миникомпьютерных систем. В отличие от традиционных вычислительных систем, характеризующихся стабильностью конфигурации, минисистемам свойственна изменчивость — они должны допускать варьирование конфигурации в широких пределах с целью удовлетворительной подгонки ее под применение, обеспечивать легкость увеличения ресурсов включением в систему дополнительной аппаратуры, предоставлять возможность сочленения машин в комплексы и простого сопряжения с измерительными приборами, исполнительными механизмами, аппаратурой связи.

§ 4. Особенности архитектуры миникомпьютеров

Самой значительной особенностью архитектуры является, как уже было отмечено, необычное короткое машинное слово — результат приведения этого важнейшего параметра цифровой машины в соответствие с тем, что действительно необходимо для подавляющего большинства ее практических применений, и главный источник эффективности миникомпьютеров. Для современных миникомпьютеров стандартным является слово длиной 16 битов (два байта), но вместе с тем продолжается выпуск машин с 12-битным словом, а в микропроцессорном исполнении наиболее массовыми тиражами выпускаются 8-битные и 4-битные (т. е. байтовые и полубайтовые) машины. Правда, микропроцессоры и микрокомпьютеры принято рассматривать как отдельный класс цифровых машин,

нередко противопоставляемый миникомпьютерам. Однако для такой классификации нет достаточных оснований. Как уже было сказано, мы рассматриваем микрокомпьютеры как технологическую разновидность миникомпьютеров: у них та же (за исключением несущественных отклонений, связанных с жесткой ограниченностью количества междумодульных соединений) архитектура и те же принципы применения. Некоторые из них прямо заимствуют архитектуру обычных миникомпьютеров или являются их микропроцессорными версиями. Например, микропроцессор Harris HM 6100 — аналог миникомпьютера PDP-8, а MicroNOVA и LSI-11 — миникомпьютеры, реализованные на больших интегральных схемах.

Короткое машинное слово, являясь главной особенностью миниархитектуры, обусловило ряд других ее особенностей. Они возникли вследствие невозможности сохранить в условиях короткого слова традиционные форматы команд. Даже при наибольшей для миникомпьютеров длине слова — 16 битов — нельзя прямо обозначить достаточное число операций и адресовать память приемлемой емкости (заметим, что не только мало адресуемых ячеек, но и ячейки малы — не более 16 битов). Например, выделив всего лишь 5 битов для кода операций и отказавшись от указателей индекс-регистров, имеем в качестве адреса 11 битов, т.е. прямо адресуемая память может составлять не более $2^{11} = 2048$ слов, что с точки зрения современных требований крайне мало.

Типичными для миникомпьютерной архитектуры путями преодоления данной трудности являются:

- разделение команд на *адресные* (ссылающиеся на главную память) и *безадресные*, задающие операции над регистрами процессора и, может быть, периферийных устройств без обращения за операндами к главной памяти (обычно адресными командами задаются только пересылки, переходы и строгий минимум других операций, поэтому код операции совсем короткий и почти все слово занято адресной частью, а в безадресной, наоборот, подавляющую часть слова составляет код операции);

- использование страничной, косвенной, относительной и других видов эффективной адресации, а также самомодифицирующихся указателей и стеков;

— использование команд, кодируемых более чем одним словом, в частности, представление адресов отдельными словами;

— послоговое представление программы, при котором операциям и операндам сопоставляются отдельные короткие слова (слоги), сочетания которых интерпретируются процессором как виртуальные команды, причем длина этих команд (число слогов в команде) автоматически варьируется в зависимости от выполняемых операций и от контекста, в котором встретилась та или иная операция.

Еще одним сугубо миникомпьютерным приемом расширения возможностей в условиях короткого слова является разукрупнение команд и операций. Например, вместо команд условного перехода, осуществляющих передачу управления по содержащемуся в команде произвольному адресу, применяют команды вида **skip if** (перескок, если...), вызывающие скачок через очередную команду (пропуск ее) в случае, если, выполнено некоторое условие. Для реализации перехода по произвольному адресу *A* при выполнении условия *X* в такой системе требуется пара команд:

skip if \bar{X}
go to A

где \bar{X} означает не-*X*. Однако важно то, что вместо множества адресных команд условного перехода, соответствующих различным условиям, используется единственная адресная команда безусловного перехода **go to**, а требующееся условие определяется выбором безадресной команды **skip if**. Таким образом удастся сохранить компактность набора адресных команд, обеспечив достаточно богатый набор условий перехода, необходимых для эффективного программирования управляющих процедур.

§ 5. Микропрограммирование

Функционально полный набор операций оказывает-ся тем компактнее, чем из более базовых (менее специализированных) операций он составлен. Это аналогично тому, как при сооружении различных зданий, чем крупнее применяемые блоки, тем шире

должна быть их номенклатура, а можно все построить, используя элементарный кирпич. По сравнению с арифметическими операциями, имеющимися на больших цифровых машинах, миникомпьютерные операции преобразования данных выглядят как кирпичи против крупноблочных конструкций. Это не «полноценные» машинные операции в традиционном понимании, а скорее элементы для их построения (программирования).

Арифметика миникомпьютера представлена операцией сложения и, может быть, вычитания целых двоичных чисел, но имеются более простые машинные операции, такие как побитная инверсия, конъюнкция (иногда и дизъюнкция), установка на ноль (очистка) аккумулятора, прибавление (иногда и вычитание) единицы, циклический сдвиг на одно место влево и вправо. Небольшой набор таких примитивных операций в совокупности с операциями тестирования состояний аккумулятора, а также некоторых отдельных битов, и перехода или перескока в зависимости от результата тестирования является исключительно гибкой основой для реализации не только арифметических, но и любых других преобразований данных. Таким образом, разукрупнением операций достигается компактность их функционально полного набора (что в любом случае полезно, а особенно при нехватке битов для кодирования операций в условиях короткого слова), возможность тонко программировать произвольные, в том числе арифметические, процедуры преобразования данных и, наконец, простота, низкая стоимость и малогабаритность арифметико-логической аппаратуры.

Конечно, перечисленные блага получаются не безвозмездно, а за счет снижения производительности минипроцессора в вычислительных применениях, поскольку арифметические операции реализуются в виде программ. Но, во-первых, для этих применений существует много других (не мини) машин, во-вторых, снижение арифметической производительности не означает невозможности выполнять вычисления — миникомпьютеры широко используются в качестве вычислителей ограниченной мощности, в-третьих, при необходимости обеспечить значительную вычислительную мощность миникомпьютер можно доуком-

плектовать арифметическим процессором, который, не являясь обязательным в стандартной конфигурации машины, подключается к основному процессору как периферийное устройство.

По существу преобразование данных в мини-компьютере, не оснащенном арифметическим процессором, реализуется на микропрограммном уровне. При этом для сокращения длины программы и времени ее выполнения применяются *команды-микропрограммы*. Такая команда задает не одну операцию, а последовательность элементарных операций (микроопераций), выполняемых в установленном для данной команды порядке. В простейшем случае каждой микрооперации в коде команды сопоставлен бит — микрооперация производится, если в ее бите содержится единица. Другими словами, биты команды работают как выключатели микроопераций, причем 1 означает «включить», а 0 — «выключить».

Например, в одной из команд-микропрограмм миникомпьютера PDP-8 4-й бит управляет микрооперацией «очистка аккумулятора», 6-й бит — «инверсия аккумулятора», 11-й — «прибавление единицы», причем операции эти производятся в той последовательности, в которой они здесь названы. Прочие биты этой команды, управляющие другими микрооперациями, для простоты будем считать установленными в нулевое состояние, т.е. выключенными. Если все рассматриваемые биты содержат нули, то все микрооперации выключены, обозначенная командой микропрограмма оказывается пустой и в результате ее выполнения никакие операции над данными произведены не будут. Мнемокод этого варианта команды NOP так и гласит: никакой операции.

Если в состоянии 1 будет только 4-й бит, то выполнится микрооперация «очистка аккумулятора»: $Ac := 0$. Соответственно, если 1 будет только в 6-м бите, то содержимое аккумулятора будет инвертировано: $Ac := \overline{Ac}$, а если 1 будет только в 11-м бите, то команда-микропрограмма будет предписанием прибавить единицу: $Ac := Ac + 1$.

Наличие единиц одновременно в двух или трех рассматриваемых битах вызовет выполнение соответствующих операций в указанной выше последовательности. Так, если 1 будет в 4-м и в 6-м битах, то

аккумулятор будет очищен и затем инвертирован, в результате чего все его биты будут установлены в состояние 1. Если 1 будет в 4-м и в 11-м битах команды, то произойдет очистка и затем прибавление единицы: $Ac := 0$; $Ac := Ac + 1$; и в результате значение аккумулятора будет равно единице. Если 1 будет в 6-м и в 11-м битах, то содержимое аккумулятора будет инвертировано с последующим прибавлением к нему единицы: $Ac := \bar{Ac}$; $Ac := Ac + 1$; т.е. в целом будет произведена операция двоичного дополнения, истолковываемая как изменение знака числа, представленного в дополнительном коде.

Наконец, если 1 будет одновременно в трех рассматриваемых битах команды, то будут произведены последовательно операции очистки, инверсии и прибавления единицы, т.е. будет выполнена микропрограмма: $Ac := 0$; $Ac := \bar{Ac}$; $Ac := Ac + 1$. Первая операция установит в аккумуляторе нуль, вторая — единицы во всех битах, а в результате третьей операции в аккумуляторе снова получится нуль и возникнет единица переноса из старшего бита в связанный с ним бит связи Lk (Link).

Сопоставление каждой операции отдельного бита в команде-микропрограмме оправдано в тех случаях, когда операции независимы и их сочетания в различных комбинациях могут иметь применения. Для управления несовместимыми, взаимно исключающими друг друга микрооперациями используют кодирование их группой битов. При этом нулевое значение кода соответствует случаю, когда ни одна микрооперация не включена, а отличные от нуля значения включают каждое одну из операций. Например, группа из трех битов принимает 8 значений, т.е. за вычетом нулевого значения позволяет обозначить до 7 попарно несовместимых микроопераций. В общем случае команда-микропрограмма содержит как однобитные выключатели независимых микроопераций, так и представленные группами битов переключатели взаимно исключающих друг друга микроопераций.

Команды-микропрограммы используются не только для преобразования данных, но также для программирования условий перескока. Примером может служить другая команда-микропрограмма PDP-8, в

которой биты $K(5:8)$ управляют перескоком следующим образом.

Если $K(8) = 0$, то при $K(5) = 1$ перескок производится по отрицательному содержимому аккумулятора ($Ac < 0$), при $K(6) = 1$ — по нулевому содержимому аккумулятора ($Ac = 0$), при $K(7) = 1$ — по ненулевому (т. е. единичному) состоянию бита связи ($Lk = 1$).

Таким образом, если $K(8) = 0$, то перескок происходит при условии

$$K(5) = 1 \wedge Ac < 0 \vee K(6) = 1 \wedge Ac = 0 \vee \\ \vee K(7) = 1 \wedge Lk = 1.$$

Заметим, что в случае $K(5:8) = 0$ перескок, безусловно, запрещен.

Если $K(8) = 1$, то сформулированное только что условие перескока заменяется его отрицанием, которое по правилу де-Моргана можно преобразовать к виду

$$(K(5) = 0 \vee Ac \geq 0) \wedge (K(6) = 0 \vee Ac \neq 0) \wedge \\ \wedge (K(7) = 0 \vee Lk = 0).$$

Другими словами, если $K(8) = 1$, то при $K(5) = 1$ перескок может произойти только в случае неотрицательного содержимого аккумулятора ($Ac \geq 0$), при $K(6) = 1$ — только в случае $Ac \neq 0$, а при $K(7) = 1$ — только в случае $Lk = 0$, причем, когда единица имеется одновременно более чем в одном из перечисленных битов, то для выполнения перескока требуется совместное удовлетворение условий, связанных с теми битами, в которых находится 1. Например, при $K(5) = K(6) = 1$, $K(7) = 0$ перескок происходит, если выполнены одновременно условия $Ac \geq 0$ и $Ac \neq 0$, что равносильно $Ac > 0$.

Заметим, что в случае $K(5) = K(6) = K(7) = 0$, $K(8) = 1$ условие перескока выполняется независимо от содержимого аккумулятора и бита связи. Таким образом код $K(5:8) = 1$ предписывает безусловный перескок.

Использование команд-микропрограмм позволяет существенно уменьшить длину программы и время ее выполнения по сравнению с таким вариантом, в котором каждая микрооперация была бы задана отдельной машинной командой. Однако значительно больший эффект достигается применением в машине микропрограммного процессора, работающего с

быстродействующей, так называемой управляющей памятью. В техническом отношении это аналогично микропрограммной реализации машинных команд, предложенной в 1951 г. М. Уилксом и широко используемой в современных цифровых машинах, но в миникомпьютерах эта техника приобрела несколько иной и более широкий смысл.

Микропрограммировать выгодно сложную архитектуру, а миникомпьютеры отличаются элементарностью и немногочисленностью операций, поэтому просто микропрограммная реализация машинных операций в миникомпьютере нерациональна. Миниархитектура предоставляет программисту возможность микропрограммирования на уровне машинных команд — это один из главных источников ее эффективности. Можно, разумеется, упрятать микрооперации во внутренний процессор, оснащенный памятью микропрограмм, функционально эквивалентных «полноценным» машинным операциям, но это был бы уже не миникомпьютер, а обычная вычислительная машина с микропрограммной реализацией команд. Другой важный атрибут миникомпьютера — короткое слово — в такой машине при традиционном формате команд сохранить, по-видимому, также не удастся, потому что с укрупнением машинных операций увеличится их численность и, следовательно, необходимая длина кода операции.

Микропрограммная память в миникомпьютере используется не столько в качестве средства экономической реализации сложных машинных команд, сколько для ускоренного выполнения микропрограмм, программируемых пользователем. Устройства микропрограммной памяти, обладающие сравнительно небольшой емкостью (обычно 1—2КВ) и выполняемые большей частью без возможности производить оперативную перезапись информации («только для чтения» или допускающие стирание и запись с помощью специальной аппаратуры), характеризуются в 3—5, а иногда и в 10 раз более высоким быстродействием, чем используемые в миникомпьютерах устройства главной памяти. С учетом того, что слова-команды в микропрограммной памяти, как правило, в 2—3 раза длиннее 16-битного слова миникомпьютера, время выполнения программ с использованием

этой памяти сокращается в среднем в 2 — 4 раза, а в отдельных случаях в 10 и более раз.

Первоначально микропрограммная техника базировалась на управляющей памяти с неизменяемым содержимым и применялась в миникомпьютерах главным образом для убыстрения часто используемых процедур и повышения эффективности программ, написанных на языках высокого уровня, в частности на фортране. Появление управляющей памяти с возможностью перезаписи сделало технически реальным предоставление ее пользователю с целью более тонкого приспособления машины к конкретному применению. Дальнейшим развитием этого процесса явилось выполнение некоторой части управляющей памяти в виде быстродействующего устройства, допускающего как чтение, так и оперативную запись, с предоставлением пользователю средств засылки в него своих микропрограмм и обращения к ним.

ГЛАВА 4

АДРЕСАЦИЯ И УПРАВЛЕНИЕ ХОДОМ ПРОГРАММЫ

Миникомпьютер, как и всякая цифровая машина, выполняет заданную программой обработку (преобразование) данных. Исходные данные и программа вводятся в главную память с периферийных устройств, результаты обработки выводятся из главной памяти на периферийные устройства.

Некоторую часть данных, в частности влияющих на ход программы, процессор получает непосредственно от периферийных устройств, но основную массу данных и управляющей информации он получает из главной памяти.

Часть архитектуры, ответственная за структуру главной памяти и обеспечение доступа к ее элементам (ячейкам), называется *системой адресации*. Понятие «адресация» включает способы задания адреса ячейки памяти, а также механизмы, используемые для организации доступа к ячейкам в определенном порядке.

Система адресации, с одной стороны, обеспечивает доступ к хранящимся в главной памяти значениям операндов при выполнении операций преобразования данных и к ячейкам для записи результатов этих операций, а с другой стороны, она играет важную роль в организации управления ходом программы, поскольку последовательность выполнения команд определяется последовательностью их адресов. От эффективности системы адресации и тесно связанных с ней средств управления ходом программы сильно зависит эффективность архитектуры в целом.

§ 1. Способы и механизмы адресации

Если не заботиться об эффективности, то система адресации машины может быть очень простой (такой она и была у первых машин). Память рассматривается как одномерный массив $m(0:k-1)$, содержащий k однотипных элементов — слов, пронумерованных целыми числами — *адресами*, обычно начиная с нуля. Операцию задают вместе с адресами ее операндов и адресом, по которому будет записан результат. Например, операция сложения двух чисел задается в виде *трехадресной* команды $(+, i, j, s)$ и реализуется как $m(s) := m(i) + m(j)$.

Развитие систем адресации ориентировано на повышение их эффективности, достигаемое главным образом путем сокращения числа обращений к главной памяти и путем все более компактного представления адресной информации в программах. К сожалению, как правило, более эффективные системы адресации являются и более сложными. Стремление адресовать память большой емкости в условиях короткого слова делает проблему адреса в миникомпьютерах особенно острой, и естественно, что миникомпьютерные системы адресации являются наиболее изощренными.

Понятия и термины, используемые в связи с системами адресации, возникли на практике, и истолкования их не всегда однозначны, а иногда и противоречивы. Поэтому наше истолкование не будет, конечно, в согласии со всеми существующими.

В отношении архитектуры систему адресации характеризуют представленными в ней способами адресации и реализующими их механизмами. Способы адресации порождаются комбинацией отличительных признаков, определяющих основные классы адресации в следующих категориях:

1. Различие по рангу. *Рангом* адреса является число обращений к памяти, на которое этот адрес отстоит от выполнения связанной с ним операции. В рассмотренном выше примере адреса i, j, s обладают рангом 1, так как для выборки значения каждого операнда и для записи результата требуется по одному обращению к памяти. Адрес 2-го ранга является номером ячейки, содержащей адрес 1-го ранга. Например, если A — адрес 2-го ранга, то для выборки

значения соответствующего операнда требуется два обращения к памяти: $m(m(A))$. Это определение естественно обобщается на случай n -го ранга, где n — целое положительное число. Однако программа может задавать операнды не только посредством адресов, но и в виде значений, над которыми непосредственно выполняются операции. В таких случаях говорят о *непосредственной* адресации или адресации нулевого ранга. Адрес (адресацию) 1-го ранга называют *прямым (прямой)*. Адреса (адресации) 2-го и выше рангов называют *косвенными*. Косвенную адресацию, при которой не используются адреса ранга выше 2-го, называют *однократной* или *одноуровневой*, в отличие от *многократной (многоуровневой)* косвенной адресации, при которой ранг адреса может быть больше 2 или вообще неограничен.

2. Различие в зависимости от использования регистров. Система адресации имеет отношение как к главной памяти, так и к собственной памяти (регистрам) процессора.

Адресацию, затрагивающую регистры процессора, принято называть *регистровой*. Адресацию к главной памяти, не затрагивающую регистров процессора, естественно назвать *безрегистровой*. Адресация, касающаяся только регистров и не затрагивающая главную память, может быть названа *чисто регистровой*.

Примеры безрегистровой адресации были даны выше. Чисто регистровая адресация используется при программировании операций над значениями, содержащимися в регистрах процессора, т.е. без обращения к главной памяти. Например, команда сложения, в которой первый операнд указан номером регистра 1, а второй операнд и место результата — номером 2, т.е. команда $(+, 1, 2)$, будет выполнена как $R2 := R1 + R2$. (Здесь и в дальнейшем, во избежание нагромождения скобок, принята упрощенная форма обозначения регистров: $R1$ вместо $R(1)$, $R2$ вместо $R(2)$ и т.д. При этом, например, 5-й бит 2-го регистра будет обозначаться $R2(5)$, а не $R(2, 5)$, как следовало бы писать, строго придерживаясь формальных правил.)

Регистровая адресация в миникомпьютерах получила самое широкое применение. Число регистров невелико, поэтому код номера регистра значительно

короче даже очень урезанного адреса главной памяти, а вместе с тем время обращения к регистру пренебрежимо мало по сравнению с временем обращения к главной памяти. Поэтому выгодно, например, сформировать в регистре адрес операнда, находящегося в главной памяти, а затем употреблять номер этого регистра в качестве косвенного адреса в командах, использующих данный операнд. При этом содержащийся в регистре адрес можно модифицировать, скажем, прибавлением единицы перед каждым обращением, если требуется обращаться к последовательным ячейкам памяти, или каким-либо другим необходимым образом.

3. Различие по полноте задания адреса. Полный адрес ячейки памяти называют *абсолютным*, а адресацию заданием полного адреса — *абсолютной* адресацией. Кроме абсолютной, широко применяется адресация приращением адреса, которую естественно назвать *относительной*. Однако термином относительная адресация обозначают обычно только адресацию относительно адреса выполняемой команды (относительно текущего значения программного счетчика PC), т.е. случай, когда абсолютный (исполнительный, эффективный) адрес ЕА связан с задаваемой в программе его частью А соотношением

$$EA = PC + A.$$

В других случаях адресацию приращением называют *индексной* или *базовой*, а регистр, участвующий в вычислении абсолютного адреса, — *индексным* или *базовым* (т.е. содержащим основную часть адреса) регистром. В прошлом название «индексный» относили к регистру, хранящему приращение к заданной в программе основной части адреса и отличали индексный регистр от базового. Однако это различие не регистров, а способов их использования. Регистр, обладающий достаточной длиной, с равным успехом можно использовать как одним, так и другим способом. В миникомпьютерах регистры, участвующие в вычислении адресов, называют индексными, а используют их как в качестве индексных, так и в качестве базовых.

Вычисление адреса может производиться с использованием более чем одного индексного регистра.

Например, существуют системы *двойной индексации*, в которых исполнительный адрес получается как сумма содержимого двух индексных регистров и части адреса, имеющейся в программе:

$$EA = Ri + Rj + A.$$

В случае индексной адресации и адресации относительно счетчика команд программа должна содержать в качестве элементов адреса величину приращения A и номер регистра, а при двойной индексации — номера двух регистров. Так, применительно к последнему примеру адресную информацию в программе задают в виде (i, j, A) . Обычно нулевому значению номера не сопоставляют регистра и интерпретируют его как указание не производить индексацию. Например, если упомянутый только что адрес записать как $(0, 0, A)$, то получится $EA = A$. С другой стороны, если указано нулевое приращение, то исполнительный адрес полностью определяется содержимым индексного регистра или регистров.

Индексная адресация может сочетаться с косвенной. При этом различают *преиндексную* (preindexing) и *постиндексную* (postindexing) адресацию. В случае преиндексной адресации индексируется адрес наибольшего ранга, т.е. отправной, заданный в команде адрес.

В случае постиндексной адресации индексируется адрес 1-го ранга, т.е. заключительный адрес, хранящийся в ячейке главной памяти или в одном из регистров и указывающий ячейку, в которой содержится значение операнда или должен запоминаться результат операции.

В качестве индексных регистров обычно используются *общие* регистры процессора, т.е. регистры общего назначения, над содержимым которых можно производить арифметические и логические операции, а также операции пересылки данных, тестирования и т.п. Но кроме того, применяется автоматическая модификация содержимого индексного регистра (а в некоторых машинах содержимого отдельных ячеек главной памяти) как специальный вид индексации, называемый *автоиндексацией* (самоиндексацией). Как правило, автоиндексация — это прибавление или вычитание единицы (иногда двойки). Автоиндексация

прибавлением (приращением) называется *инкрементной*, вычитанием (убавлением) — *декрементной*. Приращение или убавление может производиться либо перед выборкой содержимого, либо после выборки, т. е. возможны следующие четыре варианта:

$$\begin{aligned}EA &:= R; R := R + 1; \\EA &:= R; R := R - 1; \\R &:= R + 1; EA := R; \\R &:= R - 1; EA := R;\end{aligned}$$

В случае, если для автоиндексации используется не регистр, а ячейка главной памяти, технически проще прибавлять (убавлять) *перед* выборкой, например:

$$m(EA) := m(EA) + 1; EA := m(EA).$$

Специальным случаем автоиндексного регистра является *указатель стека*: в нем либо производится приращение содержимого при каждой загрузке в стек и убавление после каждого изъятия из стека (*преинкремент/постдекремент*), либо, наоборот, убавление при загрузке и прибавление после изъятия (*преддекремент/постинкремент*). Механизм стековой адресации будет рассмотрен позже.

Безрегистровая автоиндексация осуществляется в выделенных для этой цели ячейках памяти и обычно только при обращении к такой ячейке за адресом в процессе косвенной адресации. Ячейка может быть либо автоинкрементной, либо автодекрементной. Автоиндексируемые ячейки выступают в качестве заменителей индексных регистров и весьма эффективны при обработке массивов.

Разновидностью базовой адресации является *страничная* адресация. Исполнительный адрес в случае страничной адресации получается не сложением заданного в адресной части команды приращения с содержимым индексного регистра, а конкатенацией (сочленением) приращения с содержимым регистра. При этом содержимое регистра рассматривается как номер страницы N_p , а приращение — как число без знака, представляющее собой адрес A ячейки в пределах страницы. Конкатенация $N_p A$ адреса A с номером страницы N_p является абсолютным адресом. Если длина абсолютного адреса равна n битам, а адрес

в пределах страницы состоит из m битов, то номер страницы должен содержать $n - m$ битов, причем страниц будет 2^{n-m} , а ячеек на каждой странице — 2^m . Обычно $m = 7-10$, т. е. объем страницы выбирают равным 128, 256, 512 или 1024 ячейки.

В качестве указателей номера страницы могут быть использованы как специально предусмотренные для этой цели регистры, так и регистры, выполняющие кроме того, другие функции. Часто указателем номера страницы служит головная часть программного счетчика, естественно содержащая номер той страницы, на которой находится выполняемая в данный момент команда. Эту страницу называют *текущей*. Простейший вариант страничной адресации обходится без специальных регистров и занимает в командном слове только один бит для указания страницы: при нулевом состоянии этого бита обращение производится к нулевой ($Np = 0$) странице, т. е. имеющееся в команде приращение A интерпретируется как абсолютный адрес, если же бит-указатель содержит единицу, то обращение относится к текущей странице, т. е. исполнительный адрес получается сочленением k головных битов, $k = n - m$, программного счетчика и приращения A :

$$EA := PC(1:k)A.$$

Страничная адресация по сравнению с индексной является более ограниченной, однако ее легче реализовать, так как при формировании исполнительного адреса не требуется операция сложения. Кроме того, чтобы гарантировать невыход за пределы данной страницы, не требуется проверять принадлежность значения исполнительного адреса интервалу соответствующих этой странице адресов — достаточно того что номер данной страницы находится в регистре указателя, на который ссылается команда.

Рассмотренные способы адресации позволяют не только значительно расширить адресное пространство при жестко ограниченной длине слова-команды, но и получить существенную экономию необходимой для размещения программ памяти, а в ряде случаев и машинного времени. Более того, они составляют неотъемлемую основу современной техники программирования: перемещаемость программ, автомати-

ческое распределение и защита памяти, мультипрограммная работа, структурирование данных — все это реализуется в цифровых машинах посредством системы адресации и тем эффективней, чем богаче арсенал способов и механизмов, которыми она представлена в той или иной конкретной машине.

Возможность экономить занимаемую программой память путем сокращенного задания адресов обусловлена тем, что адреса в программах, как правило, собраны в пучки, т.е. их можно получать ограниченной вариацией сравнительно малого числа задаваемых полностью базовых адресов. Эта возможность реализуется при страничной, индексной и относительной адресации, а в том случае, когда значения адреса изменяются с каждым обращением последовательно, особенно эффективна автоиндексная адресация.

Относительная и непосредственная адресация составляет неперемнную принадлежность *позиционно-независимых* программ, очень удобных в применениях реального времени. Такая программа может быть выполнена в произвольном месте памяти без всякой настройки: все ссылки и переходы производятся в ней относительно текущего адреса, а константы заданы непосредственно.

Косвенная адресация позволяет, даже при отсутствии индексных регистров, сочетать длинный адрес с короткой командой: в команде имеется короткая ссылка на ячейку нулевой или текущей страницы, содержащую полный исполнительный адрес. При наличии регистров возможна аналогичная ссылка на адрес, содержащийся в регистре. Производя замену адресов в ячейках (регистрах) косвенной адресации, можно одну и ту же программу применять к данным, расположенным в разных местах памяти, или одной и той же командой перехода с косвенной адресацией производить переход по разным адресам. В более развитых формах косвенной адресации ячейка, на которую ссылается команда, может содержать не просто адрес, а *описатель* (дескриптор) операнда, указывающий, например, его длину в байтах или в битах и тип (целое без знака или со знаком, действительное с фиксированной или с плавающей запятой и т.п.).

Адресные команды миникомпьютеров обычно предоставляют программисту возможность комбинированного применения способов адресации. Даже в такой простой архитектуре, как PDP-8, адресная часть команды содержит, кроме 7-битного адреса ячейки в пределах страницы, бит-указатель вида адреса (прямой/косвенный) и бит-указатель страницы (нулевая/текущая). Комбинациям значений этих битов соответствуют четыре возможности: прямой адрес для нулевой страницы, прямой адрес для текущей страницы, косвенный адрес для нулевой страницы, косвенный адрес для текущей страницы. Кроме того, в случае косвенных адресов $10_8 - 17_8$ на нулевой странице производится автоиндексация: при обращении к любой из ячеек, соответствующих этим адресам, ее содержимое увеличивается на единицу, а затем используется как исполнительный адрес операнда. При многократном выполнении команды, содержащей адрес такой ячейки в качестве косвенного, производится обращение по последовательно возрастающим адресам.

В общем случае с адресной командой связан определенный алгоритм вычисления исполнительного адреса операнда (в некоторых случаях — значения операнда), выполнение которого при конкретных значениях элементов адресной части команды, входящих в него в качестве переменных, регистров и ячеек главной памяти, приводит к значению абсолютного адреса операнда (соответственно к значению операнда). Этот алгоритм автоматически выполняется механизмом адресации машины всякий раз, когда требуется доступ к памяти, и является, в сущности, отражением того, как функционирует данный механизм. Примеры алгоритмов вычисления исполнительного адреса имеются в последующих главах этой книги, посвященных описанию архитектуры наиболее распространенных миникомпьютеров.

Наряду с многочисленными способами задания адресов в команде, а также в виде отдельного слова (например, при косвенной адресации), которые в совокупности естественно назвать *явной* адресацией, широко применяется *неявная* (подразумеваемая) адресация. Механизмы неявной адресации встроены в схемы реализации команд и срабатывают, когда

необходимо, несмотря на то, что специальной информации для управления этими механизмами команды не содержат.

Наиболее известным примером неявной адресации является образование адреса очередной команды прибавлением единицы к текущему значению программного счетчика. Другим примером может служить использование аккумулятора, к которому в процессе выполнения команды обычно производится два обращения (за операндом и для сохранения результата операции), хотя код команды не содержит ни одного бита, указывающего на аккумулятор.

Более сложным механизмом неявной адресации является *стек (магазин)* — организация памяти по принципу «последним прибыл, первым вышел». По этому принципу работают магазины пистолетов-автоматов: при зарядке каждый вводимый патрон проталкивает введенные до него патроны в глубь магазина, становясь на ближайшее к выходу место так, что первым выстреливается патрон, введенный в магазин последним. Магазиновая или стековая память работает совершенно аналогично, но вместо патронов в нее заталкивают слова, а вместо выстрелов производят операции, потребляющие слова в качестве операндов. Одноместная операция потребляет одно слово, а затем заталкивает в магазин слово-результат. Двухместная операция потребляет два слова и заталкивает одно слово-результат.

Технически стековая память обычно устроена не так, как магазин пистолета: слова в ней не сдвигаются, а сидят каждое в той ячейке, в которую оно попало при вводе в стек. Вместо сдвига слов модифицируется указатель стека (*stack pointer*) — регистр, содержащий адрес ячейки, в которой находится последним поступившее в стек слово. Эта указываемая указателем ячейка называется *вершиной* стека. При каждом занесении слова в стек сначала производится единичное приращение содержащегося в регистре-указателе адреса, т.е. формируется адрес первой еще незанятой ячейки, а затем поступающее в стек слово записывается в эту ячейку. При выталкивании слова из стека оно считывается из ячейки-вершины, а затем значение указателя уменьшается на единицу: вершиной становится ячейка,

являвшаяся подвершиной, а прежняя вершина становится первой незанятой ячейкой.

Говорят также, что указываемая указателем стека ячейка находится в окне стека. При занесении слова в стек окно сдвигается на одно место вперед, а при выталкивании — назад.

Засылая в указатель то или иное начальное значение, можно располагать стек в произвольно выбранной области главной памяти. Иногда заполнение стека производят в порядке убывания адресов ячеек, т.е. убавляя значение указателя перед занесением слова и увеличивая после выталкивания. В других реализациях принимают, что указываемая ячейка (вершина стека) является не содержащей последним принятое слово, а первой незанятой ячейкой. При этом модификация указателя производится *перед* выталкиванием из стека и *после* занесения в стек.

Все эти различия носят технический, непринципиальный характер, а существо стека как механизма, обеспечивающего запоминание элементов данных в порядке их поступления и выдачу в обратном порядке, остается одним и тем же. Существенно, что это механизм безадресной организации памяти — наличие указателя в нем обусловлено использованием в качестве основы для его реализации адресуемой памяти, но существуют стеки на основе сдвигающих регистров (слова сдвигаются в них, как патроны в магазинах), работающие без указателей и адресов.

Безадресность стековой организации памяти позволяет осуществить на ее основе эффективное использование неявной адресации. Важнейшим примером этого является вычисление с помощью стека арифметических выражений. Стек можно использовать в вычислителе как дальнейшее развитие идеи аккумулятора.

В отличие от однорегистрового аккумулятора, стек обеспечивает возможность автоматического запоминания всех используемых в процессе вычисления выражения операндов и промежуточных результатов, избавляя от необходимости отсылать последние в главную память, а затем вызывать их при выполнении последующих операций. Это не только

сберегает занимаемую программой память, но и коренным образом упрощает компиляцию участков программы, реализующих вычисление выражений.

В простейшем (нульадресном) стековом вычислителе операции преобразования данных полностью отделены от пересылок и выполняются только над содержимым вершины и подвершины стека, причем результат операции принимает вершина. Например, выражение $(a + b) \times (c + d)$ такой вычислитель вычисляет в польской инверсной записи $ab + cd \times$, интерпретируя ее так:

| | |
|-------------|--------------------------|
| заслать a | в стеке a |
| заслать b | b, a |
| сложить | $b + a$ |
| заслать c | $c, b + a$ |
| заслать d | $d, c, b + a$ |
| сложить | $d + c, b + a$ |
| умножить | $(d + c) \times (b + a)$ |

Стековый вычислитель, в котором допустимо совмещение пересылок с операциями преобразования данных, может интерпретировать ту же запись как более компактную последовательность команд:

| | |
|---------------|--------------------------|
| заслать a | в стеке a |
| прибавить b | $a + b$ |
| заслать c | $c, a + b$ |
| прибавить d | $c + d, a + b$ |
| умножить | $(c + d) \times (a + b)$ |

Рациональное построение процессора на стековой основе позволяет также просто и эффективно решить проблему кодирования команд в условиях очень короткого слова. Устанавливают два вида командных слов: слово-адрес и слово-операция (т.е. слово, обозначающее некоторую операцию). Процессор интерпретирует слово-адрес как команду засылки в стек содержимого указываемой этим адресом ячейки главной памяти. Слово-операция в простейшем случае понимается как команда выполнить обозначенную операцию над содержимым верхних ячеек стека, поместив результат в стек. В других случаях процессор может интерпретировать сочетания слова-операции и одного или нескольких адресных слов как виртуальные команды, совмещающая выборку операндов

из главной памяти с выполнением операций преобразования данных.

В существующих миникомпьютерах (впрочем, как и в цифровых машинах вообще) преобладает архитектура процессоров с одним и более аккумуляторами или с несколькими регистрами общего назначения. Стеки в устройствах преобразования данных еще не заняли своего места. Пока их чаще восхваляют, чем применяют. По-видимому, не просто преодолеть традицию. Как правило, даже убежденные сторонники стековой архитектуры на практике ограничиваются половинчатыми решениями — вводят автоинкрементную и автодекрементную адресацию или в лучшем случае команды засылки (PUSH) в стек и выталкивания (POP) из стека. Этого, очевидно, недостаточно для реализации указанных выше преимуществ стекового вычислителя.

Более успешным является внедрение стека в качестве средства автоматического управления ходом программы, рассматриваемого в следующем параграфе.

Другим представляющим интерес в отношении неявной адресации, но пока почти неиспользуемым адресным механизмом является *очередь*.

Реализовать очередь можно при помощи двух регистров-указателей, работающих в автоинкрементном (или в автодекрементном) режиме. Один из этих регистров, обозначим его R1, ставит в очередь поступающие на ее вход данные, а второй регистр R2 ведает передачей данных из очереди на дальнейшую обработку. Очередь, как и все правильные очереди, работает по принципу «первым пришел — первым вышел».

Вначале очередь пуста: $R1 = R2$. Прием в очередь слова W производится так:

$$R1 := R1 + 1; m(R1) := W;$$

а выборка из очереди в регистр D для дальнейшей обработки соответственно так:

$$R2 := R2 + 1; D := m(R2).$$

Чтобы очередь «не бегала по всей памяти», следует ограничить области принимаемых регистрами R1 и R2 значений. Это легко реализовать путем изоляции

нескольких младших битов данных регистров по цепи переноса, т.е. запретом переноса в последующие старшие биты. Изоляция k младших битов приведет к тому, что в инкрементном (и в декрементном) режиме регистры будут работать как циклические по модулю 2^k счетчики. При этом состояние $R2 = R1 + 1$ в инкрементном режиме ($R2 = R1 - 1$ в декрементном режиме) будет соответствовать переполнению очереди — ситуации, в которой отведенная под очередь область памяти полностью занята поступившими данными. Если в такой ситуации не приостановить дальнейший прием в очередь, то поступающие слова будут попадать в ячейки, хранящие принятые ранее и еще непереданные на обработку слова. Таким образом, для правильного функционирования очереди при наступлении указанного выше состояния прием в очередь необходимо приостанавливать до тех пор, пока это состояние не изменится вследствие выдачи на обработку стоящих в начале очереди слов.

Важным применением очереди в цифровых системах является сопряжение нерегулярного источника данных (сигналов, запросов) с их обработчиком. Очередь выполняет функцию буфера, автоматически выравнивающего входной поток, обеспечивающего обработчику возможность работать в собственном ему ритме.

§ 2. Управление ходом программы

Способы и механизмы адресации имеют непосредственное отношение к управлению ходом программы в процессе ее реализации, поскольку выборка команд производится из памяти и последовательность, в которой будут выполнены команды, определяется в конечном счете последовательностью значений адреса, находящегося в программном счетчике. Как уже было сказано, выборка команд в линейной последовательности в порядке возрастания адресов обеспечивается путем неявной автоинкрементной адресации по программному счетчику. Необходимые по тем или иным причинам отклонения от линейной последовательности выполнения команд также осуществляются с помощью системы адресации, с использованием предоставляемых ею способов и механизмов.

Простейший вид нарушения линейной выборки команд — *безусловный переход* — реализуется засылкой в программный счетчик значения адреса, отличного от того, которое получается в результате очередного приращения находящегося в этом счетчике текущего значения. Адресная часть команды перехода, как и адресная часть любой команды, может содержать либо непосредственно подлежащее засылке в программный счетчик значение абсолютного адреса, либо значения необходимых для вычисления этого адреса элементов — смещения, указателей вида адресации, номера индекса регистра и т. п.

В случае, когда команда содержит абсолютный адрес A той ячейки, из которой должна производиться выборка следующей команды, переход осуществляется просто присваиванием $PC := A$. Это переход с прямой адресацией. Если же команда содержит абсолютный адрес в качестве косвенного, то переход производится на команду, адрес которой находится в ячейке $m(A)$, т. е. выполняется присваивание $PC := m(A)$. Но абсолютный адрес для короткой миникомпьютерной команды обычно слишком длинен, поэтому вместо него указывают номер индексного регистра и приращение A . При этом переход осуществляется присваиванием

$$PC := R_j + A,$$

где R_j — индексный регистр номер j .

Особый интерес представляет переход с относительной адресацией, т. е. тот случай, когда в качестве индексного регистра используется сам программный счетчик:

$$PC := PC + A.$$

Приращение A может интерпретироваться либо как число без знака, но при этом переход возможен только вперед, либо как число со знаком, обеспечивающее возможность перехода как вперед, так и назад относительно текущего значения PC . Важным преимуществом перехода с относительной адресацией является его независимость от абсолютных адресов. Программа, в которой используются только такие переходы, может быть позиционно независимой, т. е. ее можно без настройки выполнять в произвольном месте памяти.

В общем случае переход выполняется как присваивание вида $PC := EA$, где EA — исполнительный адрес, значение которого вычисляется по установленному в каждой конкретной системе адресации алгоритму.

Безусловный переход можно рассматривать лишь как весьма примитивное средство управления — он обеспечивает возможность задания произвольной последовательности выполнения команд, но не обеспечивает автоматического изменения этой последовательности в зависимости от складывающихся по ходу программы условий. Чтобы программировать управление, необходимы средства, позволяющие назначать выполнение или невыполнение определенных действий в зависимости от того, соблюдено или не соблюдено заданное условие. В цифровой машине таким средством является переход по условию (при соблюдении условия), называемый обычно *условным переходом*.

В двоичной машине проверка условия всегда приводит к одному из двух исходов — соблюдено/не соблюдено. Другими словами, результат проверки условия выражается одним битом — да/нет. Охарактеризовав соблюдение проверяемого условия битом b , можно представить условный переход в виде

if b then $PC := EA$ else $PC := PC + 1$,

т. е. переход по адресу EA происходит, если $b = 1$, а в случае $b = 0$ производится обычное прибавление в программный счетчик единицы и, следовательно, продолжается выборка команд в линейном порядке.

Содержание сопоставленного биту b условия может быть самым различным. Например, состояние $b = 1$ может означать, что содержимое аккумулятора равно нулю, или оно может сигнализировать о каком-либо внешнем событии, скажем, о готовности телепринтера принять в буферный регистр очередной символ для печати. Таким образом, процессор получает возможность «ощущать» результаты производимой им обработки данных и использовать эту обратную связь при выборе направления дальнейшей обработки в точках разветвления выполняемой программы. Соответственно, он может контролировать работу периферийных устройств, ориентироваться в обстановке, взаимодействовать с окружающей средой.

В вычислительных приложениях удобны условные переходы по результату сравнения чисел. Например, трехадресная команда условного перехода с адресами A, B, C может формулироваться так:

if $m(A) < m(B)$ **then** $PC := C$ **else** $PC := PC + 1$.

Коротким миникомпьютерным словом не то чтобы трех, но и двухадресную команду закодировать мудрено. Поэтому переход по результату сравнения реализуют двумя командами: первая содержит адреса сравниваемых чисел и предписывает установить в соответствии с результатом сравнения состояние некоторого бита, вторая назначает переход по содержащемуся в ней адресу при условии, что указанный бит находится в требуемом состоянии.

Более развитый вариант подобной системы может включать несколько битов, характеризующих в определенных отношениях результат каждой операции преобразования данных. При этом условные переходы могут производиться при тех или иных сочетаниях значений данных битов. Например, результат арифметической операции над числами, представленными в дополнительном коде, можно характеризовать тремя битами: V (overflow) — переполнение, Z (zero) — нуль, N (negative) — отрицательное, условившись, что

V = 1, если произошло арифметическое переполнение,

Z = 1, если результат, взятый без бита переполнения, равен нулю,

N = 1, если результат (также без бита переполнения) отрицательный.

Полный результат операции, с учетом бита переполнения, с помощью признаков V, Z, N можно охарактеризовать так:

| | |
|---------------------------|-----------------------|
| $(V = N) \wedge (Z = 0)$ | — строго больше нуля, |
| $V = Z$ | — не меньше нуля, |
| $(V = 0) \wedge (Z = 1)$ | — равен нулю, |
| $(V \neq N) \vee (Z = 1)$ | — не больше нуля, |
| $V \neq N$ | — строго меньше нуля, |
| $(V = 1) \vee (Z = 0)$ | — не равен нулю. |

Для реализации переходов по данным условиям надо либо предусмотреть отдельную команду перехода

для каждого из них, либо иметь три команды, снабженные битом инвертирования (отрицания) условия перехода. Каждая из этих трех команд будет при нулевом значении бита инвертирования осуществлять переход по одному из указанных выше условий, а если бит инвертирования содержит единицу, то по другому, являющемуся отрицанием первого. Например, одним условием будет $(V = N) \wedge (Z = 0)$, а другим — $(V \neq N) \vee (Z = 1)$. Различных команд и в этом случае будет, конечно, шесть, но благодаря правильной логической организации набор их воспринимается как более компактный.

Условный переход является необходимым и в принципе достаточным средством управления ходом программы. На его основе реализуются (программируются) любые применяемые в программах схемы управления — ветвление, переключатели, циклы. В миникомпьютерах, как и в других цифровых машинах, условный переход является основным и, в сущности, единственным средством реагирования на возникающие в ходе выполнения программы ситуации. Однако нельзя сказать, что это единственно возможный или наилучший вариант реализации данной функции, по-видимому, это только наиболее простой вариант.

Очень важной разновидностью перехода является переход с запоминанием адреса возврата или *переход на подпрограмму*. В простейшем случае подпрограмма — это обособленный фрагмент программы, автоматически включаемый в нее в процессе выполнения посредством перехода на его начало и возврата после окончания на команду, расположенную непосредственно за командой перехода. Переход на подпрограмму называют также вызовом подпрограммы и обращением к подпрограмме. Программа, вызывающая подпрограмму, называется основной или главной программой. Однако подпрограмма в свою очередь может обратиться к подпрограмме, в частности и к самой себе (*рекурсивное обращение*).

Переход на подпрограмму, как и обычный переход, осуществляется засылкой в программный счетчик адреса команды, которой начинается подпрограмма. Но при этом необходимо сохранить адрес возврата, получаемый прибавлением единицы к текущему значению счетчика: $PC := PC + 1$. Существуют и

практически применяются различные способы сохранения адреса возврата.

Проще всего запомнить его в специально выделенной для этой цели фиксированной ячейке памяти, а в конце подпрограммы поместить команду безусловного перехода, в которой использовать адрес этой ячейки в качестве косвенного. Недостатком такого решения является необеспеченность вызова подпрограммы программой: при попытке запоминания в выделенной ячейке второго адреса возврата уже хранящийся в ней адрес будет утрачен и возврат в главную программу станет невозможным. Этот же недостаток имеет место и в случае сохранения адреса возврата в одном из регистров процессора. Правда, содержимое регистра легче перезапомнить при необходимости освободить регистр для нового использования.

Более совершенный способ заключается в том, что адрес возврата запоминают в начальной ячейке самой подпрограммы, производя переход на следующую за ней ячейку, в которой содержится первая команда подпрограммы. По этому способу переход на подпрограмму с начальным адресом SBR выполняется так:

$$m(SBR) := PC + 1; PC := SBR + 1.$$

Команда возврата должна содержать адрес SBR в качестве косвенного, т. е. производить переход по адресу $m(SBR)$, находящемуся в ячейке SBR. Поскольку каждая подпрограмма сохраняет в себе адрес возврата, при обращении подпрограммы к подпрограмме сохранение адресов возврата обеспечивается. Однако обращение подпрограммы к самой себе по-прежнему невозможно. Нельзя также, чтобы подпрограмма В, к которой обратилась подпрограмма А, вызывала эту подпрограмму А.

Наиболее приемлемое решение проблемы возврата из подпрограммы достигается запоминанием адреса возврата в стеке. Для этого можно воспользоваться тем стеком, с помощью которого в стековом процессоре осуществляется вычисление выражений, но лучше иметь отдельный адресный стек. Приняв, что загрузка в стек выполняется с преинкрементной автоиндексацией его указателя SP, переход на начальный адрес А подпрограммы с запоминанием адреса

возврата $PC + 1$ можно реализовать так:

$$SP := SP + 1; m(SP) := PC + 1; PC := A.$$

Возврат в главную программу по окончании выполнения подпрограммы осуществляется безадресной командой RETURN — «возвратиться», которая вызывает выталкивание адреса возврата из стека в программный счетчик:

$$PC := m(SP); SP := SP - 1.$$

В случае, когда в подпрограмме происходит обращение к другой подпрограмме (или к самой себе), механизм запоминания нового адреса возврата работает точно так же, как и во время предшествующего обращения. При этом находящийся на вершине стека предыдущий адрес возврата проталкивается новым адресом в подвершину. Этот процесс, называемый *гнездованием* обращений, может продолжаться вплоть до переполнения стека, т. е. имеется возможность пользоваться достаточно длинными цепочками вызовов и самовывзовов. Стек запоминает адреса возвратов в порядке их поступления, а затем по мере завершения соответствующих подпрограмм выталкивает их в обратном порядке в программный счетчик. Замечательно то, что этот механизм работает совершенно автоматически — программа только сообщает адреса вызываемых подпрограмм.

Одной из причин выделения подпрограмм в обособленные единицы, для обращения к которым потребовалось разрабатывать описанные выше механизмы перехода с запоминанием адреса возврата, является то, что одна и та же подпрограмма обычно используется в ряде мест программы, а нередко и в нескольких различных программах. Отказавшись от возможности обращаться к подпрограмме, имеющейся в одном экземпляре, приходится вставлять ее копию вместо каждого обращения к ней, вследствие чего программа становится неоправданно громоздкой и труднообозримой. Таким образом, обособление подпрограмм позволяет не только экономить занимаемую программами память, но также снижать трудоемкость разработки и обслуживания программ. Однако подпрограмма, вызываемая в разные места программы, должна быть приспособлена для выполнения на каждом

из этих мест, в частности, ей необходимо производить передачу параметров — брать у вызывающей программы значения аргументов и оставлять результаты своей работы.

Передача параметров производится обычно при помощи ячеек, расположенных непосредственно за командой перехода на подпрограмму. При этом адрес возврата увеличивается на соответствующее число единиц. Доступ к указанным ячейкам реализуется в подпрограмме путем косвенной адресации относительно той ячейки (или регистра), в которой содержится адрес возврата. Поэтому, даже при наличии адресного стека, некоторые процессоры запоминают адрес возврата в регистре, чтобы пользоваться им для обращения к ячейкам, хранящим значения параметров. Прежнее содержимое этого регистра при засылке в него адреса возврата отправляется в стек, а по окончании работы подпрограммы обратно выталкивается из стека в регистр. Другими словами, регистр используется как продолжение (надвершина) стека, благодаря чему способность стека гнездовать обращения полностью сохраняется.

Иной способ передачи параметров при наличии стека заключается в том, что необходимые подпрограмме значения операндов и адреса ячеек засылаются перед обращением к ней в стек. Обратная передача результатов подпрограммы также производится посредством стека.

Использование для передачи параметров того же стека, в котором хранятся адреса возврата, связано с необходимостью существенно усложнить этот стек, например, вводить дополнительный указатель (или указатели), обеспечивающий доступ внутрь стека. Проще отделить передачу параметров от стекования адресов возврата и осуществлять ее при помощи другого стека — того, который используется при выполнении операций преобразования данных. В большинстве случаев при этом подпрограмма, подобно обычным машинным операциям, будет выполняться над содержимым вершины (и, может быть, подвершины) стека, засылая затем в него значение результата (или значения результатов). Имеются, конечно, доводы и в пользу того, чтобы реализовать все рассматриваемые функции в едином стеке (нередко так и делают),

однако функционирование стека в этом варианте не-
померно усложняется — свойственные чистому стеку
простота и автоматизм в значительной степени ока-
зываются утраченными.

§ 3. Команды структурированного программирования

Эффективность стекового механизма вызова и в осо-
бенности гнездования подпрограмм понуждает расши-
рить использование его в качестве средства управле-
ния ходом программы.

На протяжении последнего десятилетия традицион-
ная техника управления ходом программы с помощью
безвозвратных переходов подверглась основательной
критике как первопричина трудности программирова-
ния, отладки, понимания, проверки правильности, об-
служивания и модификации программ. Было установ-
лено, что эффективность и качество труда програм-
миста могут быть существенно повышены, если строить
программу не как кому вздумается, а так, чтобы ход
ее был по возможности прямолинейным и легко
прослеживался, чтобы логическая структура програм-
мы была простой по форме и ясной. Эта налагае-
мая на программиста (для его же блага) дисциплина
называется *структурированным программированием*.

Технически структурированное программирование
выражается в том, что для управления ходом про-
граммы используют в основном три конструкции: ли-
нейную последовательность элементов, ветвление
(IF THEN ELSE) и цикл (DO WHILE). Ветвление
позволяет осуществлять в ходе программы выбор
одной из двух данных подпрограмм в зависимости
от выполнения заданного условия. Цикл предписывает
проверять заданное условие перед выполнением (или
после выполнения) подпрограммы и, пока это условие
удовлетворяется, выполнять ее снова и снова.

Каждая из рекомендованных конструкций обладает
одним входом и одним выходом, поэтому их соеди-
нение в программе друг с другом всегда является
однолинейной (неразветвляющейся) цепочкой — развет-
вления имеют место только внутри модулей ветвления
и цикла. Программа строится путем последователь-
ного соединения и гнездования модулей. Последнее
означает, что имеющиеся в конструкциях модулей

подпрограммы являются, подобно программе в целом, также последовательностями модулей, в которых в свою очередь имеются подпрограммы и т. д.

Благодаря такому строению программы управление действиями, совершаемыми в процессе ее выполнения, сводится к реализации их в линейной последовательности и к гнездованию обращений к подпрограммам, вызываемым, вообще говоря, в зависимости от удовлетворения или неудовлетворения соответствующих условий. При этом выполнение каждой подпрограммы, сколько бы в ней ни содержалось обращений к другим подпрограммам и какой бы глубины ни достигали гнездования подпрограмм, завершается в конце концов автоматическим возвратом в основную программу на продолжение линейной последовательности ее модулей.

Другими словами, программа представляется в виде дерева, стволом которого служит однолинейная последовательность модулей (может быть, включающая только один модуль), а ветвями являются подключенные к этим модулям подпрограммы, также имеющие вид однолинейных последовательностей модулей, к которым подключены ветви-подпрограммы второго яруса и т. д. Кроме того, что такая структура проста для понимания и предотвращает запутывание, она естественно и легко расчленима на самостоятельные части — подпрограммы, которые можно по отдельности программировать, отлаживать, видоизменять или заменять целиком.

Структурированное программирование возникло как средство преодоления трудностей, с которыми сопряжена разработка очень больших программ, и применяется главным образом в связи с языками высокого уровня — структурированный кобол, структурированные расширения фортрана. И, хотя совершенно ясно, что структурированные программы возможны и полезны на всех уровнях, существует мнение, будто на языке ассемблера структурированное программирование нецелесообразно. Если понимать язык ассемблера как строгое отражение в символьной форме языка машины, то с этим мнением придется согласиться. Действительно, как можно структурированно программировать на языке ассемблера, когда в языке машины отсутствуют необходимые для

построения структурированной программы средства управления. Существующие «структурированные ассемблеры» являются скорее компиляторами управляющих конструкций структурированного программирования, чем ассемблерами в традиционном смысле слова.

Чтобы совместить машинную эффективность языка ассемблера с преимуществами структурированного программирования, необходимо иметь указанные управляющие конструкции в языке команд машины. Речь идет о введении дополнительно к используемым в современных машинах командам условного перехода (или вместо них) новых команд управления ходом программы, обеспечивающих возможность эффективно применения структурированного программирования непосредственно на машинном языке. Естественно потребовать, чтобы эти команды прямо соответствовали основным управляющим конструкциям, используемым при построении структурированных программ. Такие команды можно успешно реализовать с помощью стекового механизма обращения к подпрограммам.

Команда ветвления должна производить переход с запоминанием адреса возврата на одну из двух подпрограмм в зависимости от того, выполнено или не выполнено некоторое условие. Обозначив начальные адреса подпрограмм соответственно $A1$ и $A2$ и охарактеризовав выполнение условия битом b , как это делалось выше, можно выразить функцию команды ветвления в виде

$SP := SP + 1; m(SP) := PC + 1; \text{ if } b \text{ then } PC := A1$
 $\text{ else } PC := A2;$

где SP — по-прежнему указатель стека, запоминающего адрес возврата. Для простоты начала подпрограмм представлены абсолютными адресами $A1$ и $A2$, однако в действительности каждый из этих адресов может быть вычислен в соответствии с принятым в машине алгоритмом определения исполнительного адреса на основании содержащихся в команде данных. Таким образом, команда ветвления должна кроме кода операции содержать два адреса или данные (номера регистров, указание вида операции и т. п.), необходимое для вычисления этих адресов.

Наряду с рассмотренной полной командой ветвления целесообразно иметь ее сокращенный вариант, соответствующий ветвлению, в котором одна из ветвей пуста, т. е. не содержит никаких действий. Такая команда равносильна команде условного обращения к подпрограмме с начальным адресом A:

```
if b then begin SP := SP + 1; m(SP) := PC + 1;  
PC := A end else PC := PC + 1.
```

Предполагается, что в конце каждой вызываемой при выполнении рассматриваемых команд подпрограммы стоит безадресная команда RETURN, осуществляющая возврат на продолжение программы, из которой произведено обращение к данной подпрограмме:

```
PC := m(SP); SP := SP - 1.
```

Весьма полезной в структурированных программах оказывается также команда безусловного обращения к подпрограмме. С целью улучшить обозримость программы целесообразно вынести в подпрограммы все второстепенные детали, оставляя в стволе программы (и далее в стволах подпрограмм) только ссылки на укрупненные и функционально важные процедуры, реализуемые в виде обращений к соответствующим подпрограммам.

Команда, задающая цикл, аналогична команде условного обращения к подпрограмме, но отличается от нее тем, что возврат по окончании выполнения подпрограммы производится не на следующую за ней команду, а на нее же, т. е. адресом возврата служит не PC + 1, а PC:

```
if b then begin SP := SP + 1; m(SP) := PC;  
PC := A end else PC := PC + 1.
```

Это вариант цикла с проверкой условий на входе. Чтобы получить цикл с проверкой условий на выходе, надо перед описанной только что командой цикла поставить команду безусловного обращения к подпрограмме, составляющей тело цикла. В результате данная подпрограмма первый раз будет выполнена безусловно, а затем только при соблюдении условия.

ГЛАВА 5

УПРАВЛЕНИЕ ПЕРИФЕРИЕЙ

Миникомпьютеры называют периферийно-ориентированными машинами. В общем это можно понимать как выражение принципиального отличия активно взаимодействующих с окружающей средой миникомпьютеров от традиционных цифровых машин, основной работой которых является трудоемкое внутримашинное перемалывание данных. Но вместе с тем периферийную направленность миникомпьютеров можно усмотреть в особенностях их архитектуры, обусловленных стремлением сделать более непосредственным контакт процессора с внешним миром. Процессор миникомпьютера не отгорожен от периферии каналами ввода-вывода или периферийным процессором — он сам детально занимается операциями с внешними устройствами. Для этого в наборе команд миникомпьютера предусмотрены команды, позволяющие тестировать состояние управляющих регистров внешних устройств и производить над ними необходимые операции.

Таким образом, периферийная работа реализуется в миникомпьютерах на такой же элементарной основе, как и внутрипроцессорное преобразование данных — нет специализированных блоков ввода/вывода (также как нет, например, блока арифметики с плавающей запятой), но имеется набор микроопераций, позволяющий эффективно запрограммировать необходимые в том или ином применении процедуры управления периферией. Этим достигается желаемая универсальность, простота и дешевизна аппаратуры в сочетании с достаточной для большинства применений производительностью. А в тех случаях, когда произво-

длительность стандартного варианта недостаточна, имеется возможность увеличить ее включением в систему специальных устройств или применением в дополнение к основному еще одного миникомпьютера, предназначенного исключительно для управления периферией.

§ 1. Способы ввода/вывода

Специфика периферии заключается в том, что ее устройства работают не синхронно с тактом процессора и, как правило, являются намного менее быстродействующими по сравнению с процессором и главной памятью. Поэтому с точки зрения процессора моменты появления сигналов периферийных устройств непредсказуемы. Чтобы не допустить потери информации, процессор должен либо оставаться в ожидании сигнала (что приводит к значительным потерям времени, так как сигналы редки), либо уметь прерываться при появлении сигнала, т. е. быстро откладывать выполняемую программу и реагировать на поступивший сигнал.

Еще одна возможность, почти неиспользуемая в современных цифровых машинах, состоит в том, что производится запоминание поступивших сигналов в периодически просматриваемом процессором регистре, что аналогично посматриванию на часы с целью не пропустить заданный момент времени, вместо того чтобы завести на это время звонок будильника.

Если достаточно быстрое реагирование процессора на поступивший сигнал каким-либо образом обеспечено, то проблему ввода/вывода можно считать решенной. В общем случае сигнал, поступающий на определенный вход процессора, инициирует сопоставленное этому входу действие — обычно выполнение программы, реагирующей на данный сигнал. Например, сигнал, поступивший от устройства ввода, может интерпретироваться как сообщение, что в буферном регистре этого устройства готова для передачи процессору порция считанных с внешнего носителя данных и процессор может скопировать содержимое этого регистра в один из своих собственных регистров или в ячейку главной памяти.

Буферный регистр является важнейшим узлом в сопряжении процессора с периферийным устройством. По отношению к процессору он является быстродействующим регистром, который способен производить прием и передачу данных наравне с внутренними регистрами процессора. С другой стороны, буферный регистр осуществляет прием данных от связанного с ним внешнего устройства и выдачу полученных от процессора данных этому устройству с той скоростью и в том порядке, в котором необходимо по условиям функционирования устройства. Работа буферного регистра в процессе ввода или вывода характеризуется чередованием двух фаз:

1) передача данных между регистром и внешним устройством, доступ процессора к регистру при этом запрещен;

2) передача данных между регистром и процессором, доступ к регистру со стороны внешнего устройства запрещен.

Синхронизация операций, выполняемых над буферным регистром со стороны процессора, и операций, выполняемых со стороны внешнего устройства, осуществляются с помощью доступного (как и буфер) процессору и устройству бита, называемого обычно *флажком готовности*.

Закончив свои операции по загрузке буфера при вводе или потребив содержимое буфера при выводе, устройство устанавливает бит-флажок в состояние 1 — «поднимает флажок». Поднятие флажка означает окончание фазы устройства в работе буфера и начало фазы процессора: поднятый флажок разрешает доступ к буферу со стороны процессора и запрещает доступ со стороны устройства. Обычно говорят, что поднятием флажка устройство уведомляет процессор о завершении операции ввода-вывода, которую оно выполняло, и о готовности буфера принять от процессора новую порцию данных для вывода или передать введенные устройством данные. Отсюда и название — флажок готовности (Ready Flag, Done Flag).

Процессор по окончании своей операции передачи данных из или в буфер сбрасывает флажок в нулевое состояние. Сброс флажка символизирует окончание фазы процессора и начало фазы устройства. Можно сказать, что это сигнал готовности буфера для

использования устройством, посылаемый устройству процессором, но так не говорят.

Рассмотренная процедура передачи данных посредством буфера и синхронизации с помощью обоюдно управляемого и тестируемого флажка составляет техническую основу всякой передачи данных между несинхронными устройствами.

Иногда наряду с флажком готовности используют флажок *занятости* устройства (Busy Flag), устанавливаемый в состояние 1 на то время, в течение которого доступ к буферу со стороны процессора запрещен — устройство занято выполнением заданной операции. В процессе ввода или вывода состояния флажков готовности и занятости по логике их функционирования всегда будут противоположными и в этом смысле второй флажок представляется излишним. Существует, однако, ситуация, в которой оба флажка сброшены: ни процессор, ни внешнее устройство буфером не занимается — это состояние покоя: ввод/вывод не производится.

В этом состоянии процессор инициирует ввод поднятием флажка занятости, предписывающим устройству начать ввод данных в буфер. Чтобы начать вывод, процессор засылает в буфер выводимые данные и поднимает флажок занятости устройства, запуская последнее в работу. Производя операцию ввода или приняв из буфера выводимые данные, устройство сбрасывает флажок занятости и поднимает флажок готовности, сообщая этим процессору, что наступил его черед действовать. Процессор, приняв из буфера (если происходит ввод) данные, может либо продолжить работу, сбрасывая флажок готовности и поднимая флажок занятости, либо прекратить ввод/вывод, сбросив флажок готовности и оставив сброшенным флажок занятости.

Осуществление синхронизации действий процессора и внешнего устройства с помощью флажка предполагает наличие у участвующих в работе сторон, в частности у процессора, способности как устанавливать флажок в требуемое состояние, так и опознавать (тестировать) состояние флажка. Процессор должен иметь в наборе команд соответственно команды установки состояний флажка и команды условного в зависимости от состояния флажка перехода или пере-

скока. Обычно процессор тестирует состояние флажка готовности в *цикле ожидания*, состоящем из двух команд:

```
L:    SFS          /if F = 1 then SKIP;  
      JMP .-1      /go to L;  
      . . . . .
```

Первая из этих команд — Skip if Flag Set («перескочить, если флажок поднят») — не вызывает никакого действия, пока флажок сброшен. Вторая команда производит безусловный переход на первую. Таким образом, процессор будет повторять данный цикл снова и снова, пока не произойдет поднятие флажка. Первое же выполнение команды при поднятом флажке приведет к перескоку через следующую за ней вторую команду, и процессор, выйдя таким образом из цикла ожидания, приступит к выполнению последующих команд, определяющих характер его реагирования на поднятие флажка готовности данного устройства.

Очевидным недостатком описанного способа восприятия состояний флажка процессором является потеря времени в цикле ожидания. Когда наступит момент ожидаемого поднятия флажка, как правило, с достаточной точностью предсказать нельзя, и поэтому время пребывания в цикле ожидания приходится выбирать с большим запасом.

Предотвратить потери времени в циклах ожидания и существенно улучшить взаимодействие процессора с периферией, в особенности при одновременной работе с многими внешними устройствами, позволяет использование *прерывания*. Запустив в работу устройства ввода/вывода, процессор может не тратить время на слежение за состоянием их флажков, если устроить так, чтобы поднятие флажка вызывало переключение процессора с выполняемой им программы на ту программу, которую ему надлежит выполнять в случае, когда флажок поднят. Как правило, это будет короткая программа, осуществляющая передачу данных между буферным регистром внешнего устройства и одним из внутренних регистров процессора. Выполнив ее, процессор сможет продолжать прерванную программу, адрес возврата в которую он сохраняет. В этом прерывание подобно переходу на подпрограмму, однако

подпрограмма органически связана с вызывающей программой, является ее логическим продолжением, а программа, выполняемая в процессе прерывания, вообще говоря, содержательно не связана с прерываемой программой. Последняя просто временно уступает процессор для другого использования.

К сожалению, прерывание также не обходится без накладных расходов. Программа, овладевшая процессором в результате прерывания, прежде чем воспользоваться регистрами процессора, должна упрятать в ячейки главной памяти их содержимое, принадлежащее прерванной программе, а в конце своей работы восстановить все, как было в момент прерывания, чтобы процессор мог нормально продолжить выполнение прерванной программы. Все это означает непроизводительные траты времени, а также (что даже более существенно) замедление реакции процессора на поступающие от периферийных устройств сигналы.

Тем не менее прерывание является наиболее часто применяемым в современных цифровых машинах приемом реализации взаимодействия процессора с периферийными устройствами.

В тех случаях, когда ввод или вывод данных производится в виде массива значительных размеров (десятки-сотни слов или байтов), накладные расходы, сопутствующие прерыванию, удастся резко снизить, предоставив внешнему устройству *прямой доступ к памяти* (DMA — Direct Memory Access). Обычная процедура ввода/вывода связана с использованием регистров процессора: при вводе содержимое буфера данных копируется в регистр процессора, при выводе содержимое регистра процессора передается в буфер. Однако в конечном счете ввод/вывод осуществляется, как правило, между внешним устройством и главной памятью, т. е. выводимые данные засылаются в регистры процессора из ячеек памяти, а вводимые записываются из регистров в память. Очевидно, что, производя ввод/вывод путем непосредственной передачи данных между буфером внешнего устройства и ячейками главной памяти, можно обойтись без регистров процессора и таким образом избежать необходимости упрятывать, а затем восстанавливать их содержимое.

В современных миникомпьютерах передача слова или байта методом прямого доступа к памяти занимает один цикл работы запоминающего устройства и выполняется без прерывания в обычном смысле этого слова. Происходит только приостановка процессора на время одного цикла памяти, в течение которого производится передача. Говорят, что внешнее устройство захватывает этот цикл; ворует его у процессора. Приготовившись к приему в буфер или к выдаче из буфера данных, устройство предоставляет главной памяти адрес ячейки, участвующей в передаче, и указывает вид операции — чтение/запись. Соответственно, либо считанное из ячейки данное принимается в буфер, либо выданное буфером записывается в ячейку.

Устройство, использующее прямой доступ к памяти, помимо буферного регистра данных и флажков, оснащается регистром адреса и счетчиком переданных слов (байтов). Иницилируя передачу массива данных, процессор пересылает устройству в регистр адреса начальный адрес этого массива, а в счетчик — число подлежащих передаче слов. Если устройство способно как передавать, так и принимать данные и осуществляет поиск местонахождения их на внешнем носителе (например, устройство памяти на магнитных дисках), то процессор указывает ему вид операции: прием или передача, а также информацию, необходимую для поиска, — номер дорожки, сектора и т. п.

Приняв задание, устройство производит поиск (или другую подготовку) и приступает к приему (или к передаче) массива. При приеме/передаче каждого слова устройство модифицирует содержимое адресного регистра, образуя адрес следующего слова, и вычитает единицу из счетчика (практически чаще прибавляют единицу в счетчик, а количество слов задают в виде отрицательного числа). Появление в счетчике нуля является сигналом того, что массив исчерпан. Устройство прекращает работу и возбуждает прерывание процессора, чтобы сообщить ему о выполнении задания.

Реализация ввода/вывода путем прямого доступа внешних устройств к главной памяти в значительной степени сокращает затраты времени процессора на выполнение периферийных операций и в несколько раз увеличивает скорость передачи данных между внутрен-

ней частью машины и периферией. Максимальная скорость передачи, достигаемая в случае безраздельного захвата периферийным устройством всех циклов памяти (процессор в этом случае простаивает), определяется максимально допустимой частотой обращений к памяти $F_{\text{макс}} = 1/T_{\text{цикла}}$ и составляет для современных миникомпьютеров обычно 1–2 MW в секунду. К сожалению, эти преимущества прямого доступа действительны только при передаче достаточно больших массивов данных, в то время как миникомпьютеру чаще приходится обмениваться с периферией отдельными словами и даже битами.

Рассмотренные способы ввода-вывода обычно классифицируют следующим образом:

1. Программируемый ввод/вывод без прерывания — передача данных между буферными регистрами периферийных устройств и регистрами процессора с тестированием флажков готовности в циклах ожидания. Это самый простой и реализуемый минимумом аппаратуры способ взаимодействия с периферией, обеспечивающий вместе с тем сравнительно высокую скорость передачи данных, правда, ценой значительных затрат времени процессора.

2. Программируемый ввод/вывод с использованием прерывания — передача данных между буферными регистрами периферийных устройств и регистрами процессора, при которой устройства уведомляют процессор о своей готовности прерыванием выполняемой программы. Использование прерывания позволяет осуществлять ввод/вывод при меньших затратах времени процессора, однако связано с необходимостью дополнительной аппаратуры и усложняет логику функционирования системы.

3. Ввод/вывод путем прямого доступа к памяти — передача данных между буферными регистрами периферийных устройств и ячейками главной памяти, не затрагивая процессор. Прямой доступ обеспечивает дальнейшую разгрузку процессора от забот, связанных с реализацией передачи данных, перепоручая соответствующие функции специальной аппаратуре, которой должны быть оснащены периферийные устройства. Такое перепоручение функций сравнительно легко осуществляется без ущемления гибкости непосредственно распоряжающегося периферией процессора только

в случае, когда пересылаются массивы данных. Для оперативного обмена одноразовыми пересылками прямой доступ в его современной форме непригоден.

Предоставление внешнему устройству возможности прямого доступа к памяти связано с существенным усложнением его контроллера (блока управления) — необходимы регистр адреса и счетчик переданных слов, а также соответствующие управляющие цепи и дополнительные шины связи с процессором. Известно, что значительную часть этого оборудования можно сделать общей для нескольких внешних устройств в виде так называемого *канала ввода/вывода*.

Канал получает от процессора задание, указывающее номер внешнего устройства (и, если необходимо, адрес зоны или сектора на внешнем носителе), направление передачи данных, число передаваемых слов и адрес начала массива в главной памяти. Выполняя задание, канал запускает в работу указанное устройство, осуществляет по мере его готовности захват циклов главной памяти, модифицирует используемый при этом адрес, ведет счет передаваемых слов и по окончании передачи массива прерыванием уведомляет процессор о том, что задание выполнено.

Существуют *селекторные* и *мультиплексные* каналы ввода/вывода. Селекторный канал в процессе выполнения полученного им задания полностью предоставлен в распоряжение обслуживаемого внешнего устройства и не может заниматься другими подключенными к нему устройствами, пока не завершит заданную работу. Мультиплексный (многоместный) канал обладает несколькими комплектами адресный регистр — счетчик слов и параллельно обслуживает несколько внешних устройств, уделяя поочередно каждому из них отрезок своего времени, т. е. работая по принципу деления времени между подканалами. Пропускная способность канала в целом при этом, очевидно, не может превосходить суммы пропускных способностей подканалов.

Каналы ввода/вывода являются стандартной принадлежностью традиционных цифровых машин и почти не применяются на миникомпьютерах. Причина та же, по которой в процессорах миникомпьютеров нет, например, арифметики с плавающей запятой — для

большинства применений не требуется. Да и не так велика сегодня экономия от обобществления регистров в канале, чтобы жертвовать ради нее гибкостью автономно управляемых устройств.

§ 2. Системы прерывания

Системой прерывания называют совокупность технических средств (аппаратуры и программного оснащения), предназначенных для переключения процессора с выполняемой в текущий момент программы на другую, находящуюся в главной памяти программу при сохранении возможности возврата на продолжение прерванной программы. Указанные средства должны обеспечивать, как минимум, восприятие сигнала, инициирующего прерывание, запоминание адреса очередной команды прерываемой программы в специально отведенной ячейке памяти и автоматический переход на начальную команду запускаемой в работу программы, которую называют обычно *обслуживающей* прерывание. В принципе этот минимум является достаточным для реализации всех открываемых использованием прерывания возможностей, однако эффективность такой простейшей системы для многих применений оказывается неприемлемо низкой.

Важнейшая функция прерывания — обеспечить быструю реакцию процессора на поступающие извне сигналы при небольших затратах процессорного времени. В условиях названного выше минимума средств эта функция удовлетворительно реализуется только для простейших ситуаций с весьма малым числом возбудителей прерывания. Системы, функционирующие в более сложном окружении, включают дополнительные средства — автоматическое опознавание возбудителей прерывания с переходом на сопоставленные им обслуживающие программы, автоматическое управление очередностью обслуживания с учетом присвоенных возбудителям приоритетов, автоматическое упрятывание и восстановление содержимого регистров процессора и т. д.

Применяемые в миникомпьютерах системы прерывания весьма различны как в отношении предоставляемых ими возможностей, так и в отношении тех средств, которыми эти возможности обеспечены. Сле-

дует отметить, что наряду с появлением все более сложных систем широко используются предельно простые и дешевые системы прерывания, скромные возможности которых оказываются вполне достаточными для большинства наиболее массовых применений миникомпьютеров.

Аппаратура простейшей системы прерывания принимает сигналы от всех устройств-возбудителей на один вход, т. е. все устройства подключены к единственной линии запроса прерывания. Если система включена (восприятие запросов не запрещено), то подача на линию сигнала хотя бы одним устройством вызывает по окончании выполняемой процессором команды упрятывание текущего значения программного счетчика и переход на обслуживающую прерывание программу.

Обычно для сохранения содержимого программного счетчика, которое представляет собой адрес очередной в прерванной программе команды (*адрес возврата*), используется нулевая ячейка главной памяти (ячейка с адресом 0). При этом следующая за ней ячейка с адресом 1 содержит либо команду перехода на начало обслуживающей программы, либо адрес начала этой программы. В первом случае упрятывание адреса возврата сопровождается переходом на команду, находящуюся в ячейке 1, т. е. прерывание осуществляется в виде

$$m(0) := PC; PC := 1.$$

Во втором случае автоматически выполняется переход с косвенной по ячейке 1 адресацией:

$$m(0) := PC; PC := m(1).$$

До окончания этих действий система прерывания автоматически выключается, чтобы предотвратить возникновение повторных прерываний при выполнении обслуживающей программы.

Задача обслуживающей программы включает опознание устройства, вызывающего прерывание, посылку ему уведомления о том, что его заявка принята, выполнение сопоставленной этому устройству подпрограммы, связанное, может быть, с необходимостью упрятывания и последующего восстановления содержимого некоторых регистров процессора, переход на

продолжение прерванной программы с включением при этом системы прерывания.

Опознавание устройства-возбудителя осуществляется последовательным тестированием флажков готовности, подключенных к системе прерывания устройств. Как правило, подаваемые на линию запросов прерывания сигналы являются просто повторением состояний флажков готовности: поднятие флажка одновременно служит сигналом запроса прерывания, сброс флажка снимает запрос. Тестирование флажков производится командами условного перехода или перескока в зависимости от состояния флажка. Если флажок данного устройства поднят, то производится переход на подпрограмму, обслуживающую это устройство, в противном случае выполняется следующая по порядку команда перехода по поднятому флажку второго устройства, причем, если и этот переход не состоялся, то следует аналогичная команда для третьего устройства и т. д., пока не встретится устройство с поднятым флажком. Другими словами, программа тестирования флажков имеет вид

```
if F1 = 1 then go to S1; if F2 = 1 then go to S2;...  
if Fn = 1 then go to Sn;
```

где F_n — флажок n -го устройства, S_n — начальный адрес сопоставленной этому устройству обслуживающей подпрограммы.

Короткое слово миникомпьютера не позволяет представить целиком команду типа команд, использованных в данном фрагменте, поэтому тестирование каждого флажка осуществляется парой команд — условный перескок и безусловный переход. При этом рассмотренный фрагмент преобразуется к виду

```
SZF F1  /if F1 = 0 then SKIP;  
JMP S1  /go to S1;  
SZF F2  /if F2 = 0 then SKIP;  
JMP S2  /go to S2;  
. . . . .  
SZF Fn  /if Fn = 0 then SKIP;  
JMP Sn  /go to Sn;  
. . . . .
```

где команда SZF F_n означает перескок по нулевому

значению флажка n -го устройства, а команда JMP Sn — переход на обслуживающую это устройство подпрограмму. Если флажок сброшен, то происходит перескок через команду перехода на команду тестирования очередного флажка, а если флажок поднят, то перескока нет и выполняется переход на обслуживающую подпрограмму.

В том случае, когда поднято более одного флажка, т. е. обслуживания требуют одновременно несколько устройств, предпочтение получает, как видно, то устройство, флажок которого тестируется первым из числа поднятых флажков. Таким образом, порядок, в котором следуют команды тестирования флажков, определяет приоритет устройств в системе прерывания: приоритет устройства тем выше, чем раньше встречается команда, тестирующая флажок этого устройства, в ряду команд тестирования. Первые места в этом ряду естественно предоставляются устройствам, требующим наиболее срочного обслуживания. Например, в случае, когда к процессору подключены телепринтер, печатающий до 10 знаков в секунду, ленточный перфоратор, пробивающий 30 строк в секунду, и считыватель с перфоленты на 300 строк в секунду, то тестирование флажков производится в следующем порядке: сначала флажок считывателя, затем перфоратора, затем телепринтера.

Поскольку при переходе на обслуживающую программу происходит выключение системы прерывания, то заявки от других устройств, если они были в момент, когда произошло прерывание, или появились во время выполнения обслуживающей программы, не могут вызвать новое прерывание до тех пор, пока не будет вновь включена система. Включить систему прерывания до завершения работы обслуживающей программы просто так нельзя, потому что в ячейке $m(0)$ в результате нового прерывания будет уничтожен адрес возврата на продолжение основной программы. При желании разрешить в процессе выполнения обслуживающей программы прерывания по заявкам устройств, обладающих более высоким приоритетом, чем обслуживаемое устройство, в рассматриваемой системе необходимо предусмотреть перезапоминание содержимого ячейки $m(0)$, когда производится прерывание в прерывании, и восстановление прежнего

содержимого после возврата из вторичного прерывания в первичное.

Такая *многоуровневая*, т. е. допускающая прерывание в прерывании, система может быть реализована в условиях описанного выше минимума аппаратуры путем соответствующего усложнения обслуживающей программы. Эта программа должна устанавливать (например, подсчетом производимых при входе в прерывание тестирований флажков) приоритет прерывающего устройства и сравнивать его с приоритетом выполняемой обслуживающей подпрограммы, чтобы решить, вправе или не вправе данное устройство прервать уже производящееся обслуживание другого устройства. Каждая подпрограмма обслуживания устройства, за исключением обладающей наивысшим приоритетом, должна в начале ее работы перезапомнить содержимое ячейки $m(0)$ в имеющуюся у нее для этого ячейку и включить систему прерывания, а по завершении обслуживания, выключив систему прерывания, восстановить содержимое $m(0)$ и только тогда осуществить стандартный выход из прерывания.

Данная реализация многоуровневой системы прерывания на основе аппаратуры одноуровневого варианта потребовала значительного усложнения обслуживающей программы, практически не улучшив времени реагирования на срочные запросы при уже производимом обслуживании устройства, обладающего меньшим приоритетом. Только в случае значительной длины собственно обслуживающей подпрограммы ее прерывание может существенно сократить задержку в обслуживании поступившего срочного запроса. Если же собственно обслуживающая часть невелика, то накладные расходы времени процессора на перезапоминание адреса возврата из ячейки $m(0)$ и сравнение приоритетов конкурирующих устройств легко превосходят время самого обслуживания, так что вместо убыстрения реакции на срочные запросы получается даже замедление. Как всегда, имеется, конечно, возможность оптимизации. Например, можно производить перезапоминание адреса возврата не всякий раз, а только в случае когда выяснилось, что поступил запрос с более высоким приоритетом и прерывание обслуживающей программы должно произойти. Однако коренного улучшения на этом пути достичь нельзя: опти-

мизации, хотя и могут дать некоторый выигрыш, связаны с дальнейшим увеличением накладных расходов и усложнением логики обслуживающей программы, что приводит к дополнительным затратам памяти и времени процессора, к снижению эффективности системы прерывания в целом.

Несмотря на то, что рассмотренный вариант многоуровневой системы прерывания используют на практике, его следует расценивать скорее как пример нерационального распределения функций между аппаратурой и программами. Минимум реализуемых аппаратурой функций, вполне достаточный для практического осуществления одноуровневой системы прерывания, оказывается недостаточным для создания на его основе программными средствами удовлетворительной многоуровневой системы: рассмотренное усложнение обслуживающей программы обеспечивает возможность прерывания в прерывании, но не обеспечивает того убыстрения реакции на срочные запросы, ради которого создавалась многоуровневая система.

Необходимое для достижения быстрой реакции сокращение накладных расходов, которые обременили обслуживающую программу в связи с осуществлением многоуровневой системы прерывания, возможно путем перепоручения тех или иных выполняемых этой программой функций аппаратуре. К таким функциям относятся: упрятывание и восстановление адреса возврата, упрятывание и восстановление содержимого используемых обслуживающей программой регистров процессора, опознание устройства, запросившего прерывание, установление приоритетов устройств и обслуживающих подпрограмм, разрешение приоритетных споров между конкурирующими запросчиками прерывания. В зависимости от того, какие из этих функций перепоручаются аппаратуре, и от того, какие механизмы используются для их выполнения, система прерывания может быть устроена весьма различным образом. Существующие системы действительно характеризуются чрезвычайным разнообразием, и сколько-нибудь полное рассмотрение их здесь не представляется возможным. Типичными примерами используемых на практике систем прерывания могут служить системы описываемых в последующих главах миникомпьютеров. Ниже кратко охарактеризованы технические приемы и механизмы.

при помощи которых достигается эффективная реализация функций, связанных с обслуживанием прерывания в многоуровневой системе.

Присваивание приоритетов устройствам осуществляется либо путем последовательного включения их в линию запроса прерывания (или в линию, по которой процессор производит опрос при выявлении прерывающего устройства), либо путем предоставления каждому устройству отдельной линии, либо путем комбинированного использования обеих возможностей, т. е. применения нескольких линий с последовательным включением устройств в каждой из них или применения одной линии с последовательно включенными в нее «модулями прерывания», к каждому из которых подключено несколько устройств. В случае последовательного включения устройств в линию наивысшим приоритетом обладает ближайшее к началу линии (т. е. к процессору) устройство: запрашивая прерывание, устройство производит отключение следующего за ним участка линии и этим блокирует запросы устройств, занимающих менее приоритетные места. Это экономная, но негибкая система. Предоставление каждому устройству отдельной линии обеспечивает ценой значительного увеличения аппаратуры наибольшую гибкость: процессор получает возможность назначать приоритеты устройств и оперативно изменять их в процессе выполнения программы. Кроме того, в системе с индивидуальными линиями отпадает необходимость производить опрос устройств с целью опознания запросчика — возбуждение линии прямо называет номер прерывающего устройства. Комбинированные варианты приоритетных схем обеспечивают возможность желаемого компромисса между затратами аппаратуры и степенью гибкости.

Наиболее простым способом сокращения времени опроса устройств, по сравнению с рассмотренным выше последовательным тестированием флажков обслуживающей программой, является введение специальной команды, в результате выполнения которой получается номер устройства, обладающего наибольшим среди устройств, поднявших флажки, приоритетом. Другой способ выявления прерывающего устройства заключается в том, что в ответ на запрос прерывания процессор посылает в специальную линию опроса, в

которую последовательно включены устройства, сигнал, разрешающий наиболее приоритетному из запросчиков прерывания выдать код своего номера на шины магистрали, которая соединяет процессор со всеми устройствами. Самый быстрый, но и самый дорогой способ опознавания запросчика — это уже упоминавшееся предоставление каждому устройству отдельной линии запроса прерывания: процессор узнает, какие устройства требуют прерывания непосредственно потому, что возбуждены их линии запроса. При этом он сам должен решить, какому из запросчиков предоставить прерывание в первую очередь, т. е. должна быть специальная аппаратура (или программа) установления приоритетов.

Для устранения задержек, связанных с необходимостью перезапоминания и затем восстановления адреса возврата, в многоуровневых системах прерывания отводят для сохранения адресов возврата не одну, а несколько ячеек памяти, нередко по отдельной ячейке на каждое из подключаемых к процессору устройств. В последнем случае адреса возврата при прерываниях, вызываемых различными устройствами, сохраняются совершенно независимо и необходимость перезапоминания их полностью исключена.

Аналогичным путем устраняется задержка, обусловленная участком обслуживающей программы, которой осуществляет переходы на сопоставленные устройствам подпрограммы. За каждым устройством закрепляют ячейку памяти, содержащую начальный адрес обслуживающей это устройство подпрограммы. Когда по запросу некоторого устройства происходит прерывание, система автоматически помещает текущее значение программного счетчика в сопоставленную этому устройству ячейку для сохранения адреса возврата, а содержащийся во второй закрепленной за устройством ячейке начальный адрес обслуживающей подпрограммы засылает в программный счетчик. Если при этом устройства обладают собственными линиями запроса прерывания, то такая система, в сущности, представляет собой совокупность отдельных для каждого устройства одноуровневых систем, связанных с единым приоритетным механизмом. Другими словами, сокращение накладных расходов на прерывание в прерывании, выражающихся в замедлении реакции

системы и потерях времени процессора, достигнуто ценой многократного увеличения количества используемой в системе аппаратуры.

Рассмотренные меры по убыстрению реакции на запросы направлены главным образом на сокращение задержек, возникших вследствие программной реализации многоуровневой системы прерывания. Но соответствующее усложнение аппаратуры позволяет сократить задержки и в одноуровневой (а следовательно, также и в многоуровневой) системе. Речь идет о той части накладных расходов, которая обусловлена упрятыванием и восстановлением содержимого регистров процессора, используемых обслуживающей программой. Максимальное убыстрение здесь достигается использованием переключения регистров: процессор располагает несколькими комплектами регистров и, осуществляя прерывание, переключается с одного комплекта на другой, а при возврате из прерывания производится обратное переключение.

Менее дорогим (но и менее действенным) усовершенствованием техники сохранения и восстановления содержимого регистров является введение специальных команд, позволяющих одной командой упрятывать или восстанавливать содержимое нескольких регистров. Перспективным путем сокращения рассматриваемого вида накладных расходов представляется исключение необходимости упрятывания и восстановления содержимого регистров, достигаемое надлежащей рационализацией архитектуры процессора. Например, в процессоре с преобразованием данных в стеке можно вообще избежать упрятывания и восстановления содержимого регистров.

Еще один широко распространенный способ убыстрения реакции системы на срочные запросы — это так называемое «маскирование прерываний». Сущность его заключается в том, что процессору предоставляют возможность управления способностью устройств запрашивать прерывание — вводят команды, позволяющие запрещать и разрешать тем или иным устройствам возбуждение линии прерывания, т. е. отключать и подключать данные устройства к этой линии. Каждое устройство (или группа устройств) снабжается битом запрета (или битом разрешения) прерывания: оно имеет право на запрос прерывания только в том слу-

чае, когда бит запрета содержит нуль (бит разрешения содержит единицу). Установка в желаемое состояние бита запрета/разрешения может осуществляться командой, относящейся к одному из устройств, но обычно имеется возможность устанавливать в заданные состояния биты запрета/разрешения одновременно всех устройств. Для этого совокупность битов представляют в виде регистра, в котором каждый бит занимает определенное место в соответствии с приоритетом управляемого им устройства. Содержимое такого регистра, интерпретируемое как булевский вектор, называют *маской*. Если регистр состоит из битов разрешения, то маскирующим значением является нуль, а если из битов запрета, то единица. Как правило, имеются команды засылки и копирования маски, но иногда также некоторые другие, например конъюнкция или дизъюнкция. Программно-управляемое маскирование прерывания процессора теми или иными устройствами придает системе известную гибкость и в ряде случаев позволяет повысить быстроту обслуживания срочных запросов. Так, имеется возможность присвоить любому из устройств высокий приоритет, замаскировав более приоритетные запросы, с другой стороны, можно улучшить реакцию на запросы высокоприоритетного устройства, маскируя запросы более низкого приоритета и т. д.

Рассмотренные методы и механизмы применяются в системах прерывания миникомпьютеров во всевозможных сочетаниях, порождая, как уже было сказано, большое многообразие вариантов, первоосновой которых является описанная в начале данного параграфа простейшая одноуровневая система прерывания, сохраняющая адрес возврата в фиксированной ячейке памяти. В последние годы получают все большее распространение системы прерывания, в которых для сохранения адресов возврата и другой, характеризующей текущее состояние процессора информации используется стек. Применение стекового механизма в системе прерывания в общем аналогично описанному в предыдущей главе применению его в связи с обращением к подпрограммам и позволяет так же просто и эффективно реализовать упрятывание и восстановление состояний процессора и гнездование прерываний в многоуровневой системе. Образцом стековой системы

прерывания может служить описанная в гл. 9 система прерывания миникомпьютера PDP-11, отличающаяся также оригинальной и весьма гибкой приоритетной схемой.

В соответствии с темой этой главы прерывание рассматривалось как средство взаимодействия процессора с периферийными устройствами управления вводом/выводом. Используемое в этих целях прерывание называют обычно внешним, в отличие от внутреннего прерывания, которое происходит при возникновении тупиковых или аварийных ситуаций в самом процессоре, например, при обнаружении ошибок в работе аппаратуры, при некондиционности электропитания, при переполнении или исчерпании страниц памяти, при попытке деления на ноль и т. п. В отличие от внешних, внутренние прерывания неотложны — производятся вне всякой очереди. В многопрограммных системах внутреннее прерывание служит средством осуществления взаимодействия между процессами, обращения к координирующей программе (супервизору). Кроме того, внутреннее прерывание, называемое также trap («западня», «ловушка»), используется для переключения процессора на подпрограммы, реализующие некоторые «стандартные» в данной системе функции. В этом случае команда, вызывающая прерывание (а точнее, переключение процессора на подпрограмму с запоминанием текущего его состояния), указывает подпрограмму, на которую производится переключение, подобно тому, как прерывающее устройство в случае внешнего прерывания, называя свой номер, указывает сопоставленную ему обслуживаемую подпрограмму.

ГЛАВА 6

ОДНОАККУМУЛЯТОРНЫЕ МИНИКОМПЬЮТЕРЫ СЕРИИ DEC PDP-8

Название PDP-8 (Programed Data Processor-8) относится как к цифровой машине, которую фирма Digital Equipment Corporation (DEC) начала выпускать в апреле 1965 г., так и к серии машин, родоначальницей которой явилась PDP-8. Последующие модели этого семейства обозначаются этим же названием, но с добавлением буквы. Например, PDP-8/S — более дешевая и менее быстрая модель с процессором последовательного действия, PDP-8/I — первая модель, выполненная на интегральных схемах. Всего с 1965 по 1974 гг. появились восемь моделей PDP-8 (см. табл. 6.1), различающиеся главным образом технологией выполнения и конструктивным оформлением. Архитектура в основном сохранялась неизменной, поэтому можно говорить об архитектуре PDP-8, имея в виду семейство в целом. Общность архитектуры позволяет причислить к семейству также выпускавшуюся с 1963 г. машину PDP-5, которая является по существу прототипом PDP-8.

Архитектура PDP-8 замечательна не только тем, что она положила начало миникомпьютерной технике, но в гораздо большей степени тем, что прогрессивные принципы и методы этой техники проявились в ней наиболее концентрированным и совершенным образом. Это поистине классическая миниархитектура — предельно простая и эффективная.

Заслуживает внимания то, что ни появление миникомпьютеров с более «богатой» и мощной архитектурой, в том числе семейства PDP-11, созданного самой DEC, ни оказавшееся нестандартным (в связи

Семейство DEC PDP-8

| Модель | Год начала поставки | Начальная цена базового комплекта (тыс. долл.) | Цикл главной памяти (мкс) | Время выполнения команды (мкс) | Примечания |
|---------|---------------------|--|---------------------------|--------------------------------|--|
| PDP-8 | 1965 | 18 | 1,5 | 3,0 | первоначальная модель |
| PDP-8/S | 1966 | 10 | 8,0 | 33,0 | менее быстрая память, процессор последовательного действия |
| PDP-8/I | 1968 | 16 | 1,5 | 3,0 | первая модель на интегральных схемах |
| PDP-8/L | 1968 | 8 | 1,6 | 3,2 | менее быстрый, но более дешёвый вариант на ИС |
| PDP-8/E | 1971 | 5 | 1,2 | 2,6 | на основе OMNIBUS, память до 32К |
| PDP-8/M | 1971 | 3,7 | 1,2 | 2,6 | то же, вариант для комплектации систем |
| PDP-8/F | 1972 | 4 | 1,2 | 2,6 | на основе UNIBUS, память до 16К |
| PDP-8/A | 1974 | 2,6 | 1,5 | 3,0 | на двух платах, память до 32К |

с введением 8-битного байта) 12-битное слово PDP-8, ни даже сказочно дешёвые 8-битные микропроцессоры не смогли подорвать популярность семейства PDP-8. Выпуск этих «примитивных» машин не только не прекратился, но и увеличивается: к середине 1972 г. (т. е. за 7 лет) их было выпущено 12 тыс., в 1974 г. численность семейства превысила 20 тыс., в 1976 г. — 30 тыс., не считая машин с архитектурой PDP-8, выпускаемых другими фирмами в ряде стран.

Почти десять лет спустя с начала производства PDP-8 DEC выпустила очередную модель этой машины под названием PDP-8/A с процессором в виде платы 38 × 23 см, реализованным на малых интегральных схемах, и памятью до 32К слов, выполненной на другой такой же плате. Цена процессора с 1KW памяти — 895 долл., в партии 100 штук — 572 долл. Цена базового комплекта (процессор, 4KW памяти,

телетайп) — 2,6 тыс. долл., а в 1965 г. PDP-8 в таком же комплекте стоила 18 тыс. долл.

Другой современный продукт — микропроцессор на одном чипе Harris Semiconductor HM 6100 (Intersil IM 6100), эмулирующий архитектуру PDP-8, с ценой менее 100 долл. В 1977 г. DEC анонсировала построенную на базе этого микропроцессора минисистему DECstation в объеме видеотерминала, включающую 16KW памяти, два гибких диска и видеодисплей (80 столбцов × 24 строки) с клавиатурой. Decstation оснащена усовершенствованной и расширенной операционной системой OS/78 с языками PAL/8, basic и фортран IV, а для применений реального времени использует RTS/8. Цена DECstation — 7995 долл.

Причину долголетия архитектуры PDP-8 обычно усматривают в том, что для нее имеется обширнейшее программное оснащение. Но это верно лишь отчасти. Главное достоинство этой архитектуры в ней самой, в ее простоте, доступности для понимания и использования.

§ 1. Структура процессора и типы команд

Архитектуру PDP-8 кратко можно охарактеризовать как одноаккумуляторную, с 12-битным словом, страничной и косвенной адресацией.

Регистры процессора, доступные программе:

PC(0 : 11) — программный счетчик,

Ac(0 : 11) — аккумулятор,

Lk(1 : 1) — бит связи (Link),

MQ(0 : 11) — множителя/частного
(Multiplier/Quotient).

Аккумулятор и бит связи сочленены в 13-битный регистр LkAc, причем при выполнении сложения перенос из старшего разряда Ac(0) аккумулятора распространяется в Lk. При выполнении сдвига регистр LkAc замкнут в кольцо (циклический сдвиг) таким образом, что Lk включен между головным Ac(0) и хвостовым Ac(11) битами аккумулятора (рис. 6.1). При сдвиге влево происходит передача из Lk в Ac(11) и из Ac(0) в Lk, т.е. $LkAc := AcLk$. При сдвиге вправо — передача из Lk в Ac(0) и из Ac(11) в Lk, т.е.

$AcLk := LkAc$. Это в сочетании с возможностью тестировать, очищать и инвертировать Lk является мощным средством выделения и обработки отдельных битов аккумулятора.

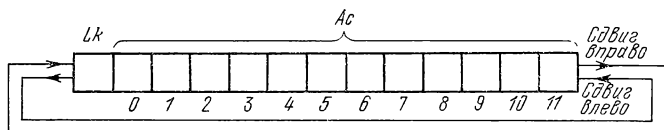


Рис. 6.1. Сочленение $LkAc$ аккумулятора Ac с битом связи Lk .

Пересылки данных возможны между аккумулятором и ячейками главной памяти, между аккумулятором и регистрами периферийных устройств и путем прямого доступа периферийных устройств к главной памяти.

Двенадцатибитный программный счетчик $PC(0:11)$ обеспечивает возможность прямой адресации 4096, т.е. 2^{12} , ячеек главной памяти, хранящих 12-битные слова. Каждая выборка из памяти команды сопровождается увеличением значения счетчика на единицу: $PC := PC + 1$. Переход осуществляется приемом в счетчик адреса, вычисляемого по элементам адресной части команды.

Команда, представляемая 12-битным словом $K(0:11)$, может быть одного из трех типов (соответственно форматов), различаемых по значению головной тройки битов $K(0:2)$:

$K(0:2) < 6$ — адресная, т.е. с обращением к главной памяти, команда;

$K(0:2) = 6$ — команда ввода/вывода, затрагивающая регистры периферийных устройств и процессора;

$K(0:2) = 7$ — команда-микропрограмма, касающаяся только регистров процессора.

§ 2. Адресные команды и система адресации

Тройка битов $K(0:2)$ в адресной команде служит кодом операции, а прочие 9 битов — $K(3:11)$ — составляют адресную часть (рис. 6.2). Так как $K(0:2) < 6$, то имеется всего 6 видов адресных команд (см. табл. 6.2).

Адресные команды PDP-8

| K(0 : 2) | Мнемокод | Операция | Описание |
|----------|----------|-------------------------------|---|
| 0 | AND | конъюнкция | $Ac := Ac \wedge m(EA);$ |
| 1 | TAD | сложение | $LkAc := LkAc + m(EA);$ |
| 2 | ISZ | приращение и перескок по нулю | $m(EA) := m(EA) + 1;$ if $m(EA) = 0$ then $PC := PC + 1;$ |
| 3 | DCA | запоминание и очистка Ac | $m(EA) := Ac;$ $Ac := 0;$ |
| 4 | JMS | переход на подпрограмму | $m(EA) := PC;$ $PC := EA + 1;$ |
| 5 | JMP | переход | $PC := EA;$ |

Мнемонические обозначения операций: AND – соединительный союз; TAD – Two's Complement Addition; ISZ – Increment and Skip if Zero; DCA – Deposit and Clear Ac; JMS – Jump to Subroutine; JMP – Jump to.

Примечание. В описания команд не включено прибавление единицы к PC, автоматически происходящее при выборке каждой команды. Встречающееся в описаниях текущее значение PC соответствует адресу команды, которая находится непосредственно за описываемой, т.е. оно на единицу больше адреса самой описываемой команды.

Адресная часть команды слишком коротка, чтобы ее употреблять для представления абсолютного адреса: $2^9 = 512$. Поэтому применена система, обеспечивающая прямой доступ только к первым 128 ячейкам (нулевая страница) и при данном значении

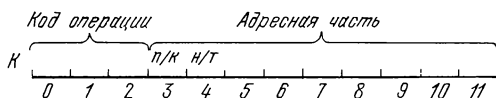


Рис. 6.2. Формат адресной команды PDP-8.

PC к 128 ячейкам с адресами вида PC(0 : 4)K(5 : 11) – текущая страница. Доступ к произвольной ячейке памяти возможен посредством косвенной адресации.

Указателями вида адресации (прямая/косвенная) и адресации, используемой при обращении страницы

(нулевая/текущая), служат биты $K(3)$ и $K(4)$:

$$K(3) = \begin{cases} 0 - \text{прямая адресация,} \\ 1 - \text{косвенная адресация,} \end{cases}$$

$$K(4) = \begin{cases} 0 - \text{нулевая страница,} \\ 1 - \text{текущая страница.} \end{cases}$$

Остальные 7 битов адресной части — $K(5 : 11)$ — составляют адрес ячейки в пределах одной страницы.

Таким образом, в случае $K(3) = 0$ (прямая адресация) при $K(4) = 0$ исполнительным адресом будет просто $K(5 : 11)$, а при $K(4) = 1$ им будет $PC(0 : 4)K(5 : 11)$, т.е. конкатенация 5 головных битов программного счетчика, указывающих номер текущей страницы, и 7 хвостовых битов адресной части команды. Главная память, рассматриваемая с точки зрения программного счетчика как одномерный массив $m(0 : 4096)$, в отношении страничной адресации представляется в виде двумерного массива $m(0 : 31, 0 : 127)$ — 32 страницы по 128 ячеек.

В случае $K(3) = 1$ (косвенная адресация) определенный в предыдущем абзаце адрес указывает ячейку, содержимое которой является исполнительным адресом. Этот 12-битный адрес достаточен для обращения к любой из 4096 ячеек памяти. В особом положении находятся 8 ячеек нулевой страницы с адресами $10_8 - 17_8$. При обращении к какой-либо из них в режиме косвенной адресации ее содержимое автоматически увеличивается на единицу, прежде чем будет использовано как исполнительный адрес. Это автоиндексируемые ячейки, призванные в какой-то степени заменить отсутствующие в PDP-8 индекс-регистры. В режиме прямой адресации они ведут себя как обычные ячейки памяти.

Функционирование адресной системы PDP-8 отражает следующий алгоритм.

Алгоритм вычисления исполнительного адреса EA в PDP-8.

```
if  $K(4) = 0$  then  $EA := K(5 : 11)$ 
    else  $EA := PC(0 : 4)K(5 : 11)$ ;
if  $K(3) = 1$  then
    begin if  $EA \geq 10_8 \wedge EA \leq 17_8$  then
         $m(EA) := m(EA) + 1$ ;  $EA := m(EA)$ 
    end
```

§ 3. Команды-микропрограммы

При $K(0 : 2) = 7$ процессор интерпретирует командное слово как микропрограмму. Отдельным битам слова сопоставлены микрооперации (из набора, представленного табл. 6.3). Процессор выполняет данную микрооперацию при условии, что в соответствующем ей бите содержится цифра 1.

Таблица 6.3

Микрооперации PDP-8

| Мнемокод | Операция | Описание |
|----------|-----------------------------------|--------------------------------------|
| CLA | очистка Ac | $Ac := 0;$ |
| CLL | очистка Lk | $Lk := 0;$ |
| CMA | инверсия Ac | $Ac := \overline{Ac};$ |
| CML | инверсия Lk | $Lk := \overline{Lk};$ |
| IAC | прибавление 1 | $LkAc := LkAc + 1;$ |
| RAL | циклический сдвиг влево | $LkAc := AcLk;$ |
| RTL | два циклических сдвига влево | $LkAc := AcLk; LkAc := AcLk;$ |
| RAR | циклический сдвиг вправо | $AcLk := LkAc;$ |
| RTR | два циклических сдвига вправо | $AcLk := LkAc; AcLk := LkAc;$ |
| BSW | перестановка полуслов (байтов) Ac | $Ac(0 : 11) := Ac(6 : 11)Ac(0 : 5);$ |
| MQA | прием из MQ в Ac | $Ac := Ac \vee MQ;$ |
| MLQ | передача из Ac в MQ | $MQ := Ac; Ac := 0;$ |
| OSR | прием из SR в Ac | $Ac := Ac \vee SR;$ |
| SMA | перескок по минусу в Ac | if $Ac < 0$ then $PC := PC + 1;$ |
| SZA | перескок по нулю в Ac | if $Ac = 0$ then $PC := PC + 1;$ |
| SNL | перескок по $Lk \neq 0$ | if $Lk = 1$ then $PC := PC + 1;$ |
| HLT | останов | stop; |

Элементы мнемоники: CL — Clear; CM — Complement; IAC — Increment Ac; RAL (RAR) — Rotate Ac Left (Right); RTL — Rotate Twice Left; MQ — Multiplier/Quotient (регистр множителя/частного); SR — Switch Register (регистр ключей на панели управления); BSW — Byte Swap; SMA (SZA) — Skip on Minus (Zero) Ac; SNL — Skip on Non-Zero Link; HLT — Halt.

См. примечание к табл. 6.2.

Имеются три соответствия микрокоманд битам слова и, следовательно, три группы микропрограммируемых команд. Они различаются значениями битов $K(3)$ и $K(11)$.

1) $K(0:2) = 7 \wedge K(3) = 0$ — команды, оперирующие с As и Lk ,

2) $K(0:2) = 7 \wedge K(3) = 1 \wedge K(11) = 0$ — команды, тестирующие As и Lk ,

3) $K(0:2) = 7 \wedge K(3) = 1 \wedge K(11) = 1$ — команды расширенной арифметики.

Соответствие микроопераций битам команд 1-й группы показано на рис. 6.3.

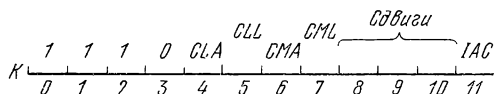


Рис. 6.3. Формат команды-микропрограммы первой группы.

Для битов 8, 9, 10 нет простого соответствия и применяется следующий код:

$$K(8:10) = \begin{cases} 101 - RTR \\ 100 - RAR \\ 011 - RTL \\ 010 - RAL \\ 001 - BSW \end{cases}$$

Выполнение микроопераций производится в определенной временной последовательности, а именно: первыми выполняются CLA , CLL , затем CMA , CML , затем IAC и, наконец, RAR , RAL , RTR , RTL , BSW . Таким образом, всякая команда 1-й группы реализуется в виде следующей микропрограммы:

```

if  $K(0:2) = 7 \wedge K(3) = 0$  then
  begin if  $K(4) = 1$  then  $CLA$ ; if  $K(5) = 1$ 
    then  $CLL$ ;
    if  $K(6) = 1$  then  $CMA$ ; if  $K(7) = 1$ 
      then  $CML$ ;
    if  $K(11) = 1$  then  $IAC$ ;
    if  $K(8:10) = 1$  then  $BSW$ 
    else if  $K(8:10) = 2$  then  $RAL$  else
      if  $K(8:10) = 3$  then  $RTL$ 
  end

```

else if $K(8 : 10) = 4$ then RAR else
if $K(8 : 10) = 5$ then RTR

end

Для записи команды-микропрограммы на языке ассемблера достаточно перечислить мнемокоды микроопераций, которые должны быть выполнены.

Таблица 6.4

Комбинированные микрооперации PDP-8

| Мнемокод | Операция | Описание |
|----------|-------------------------|--------------------------------------|
| NOP | ничего не делать | No Operation |
| CIA | дополнение Ac | $Ac: = \overline{Ac}$; IAC; |
| STL | установить Lk | $Lk: = 0$; $Lk: = \overline{Lk}$; |
| STA | установить Ac | $Ac: = 0$; $Ac: = \overline{Ac}$; |
| GLK | передать Lk в Ac | $Ac: = 0$; $LkAc: = AcLk$; |
| LAS | загрузить Ac с SR | $Ac: = 0$; $Ac: = Ac \vee SR$; |
| SKP | безусловный перескок | $PC: = PC + 1$; |
| SZL | перескок по $Lk = 0$ | if $Lk = 0$ then $PC: = PC + 1$; |
| SNA | перескок по $Ac \neq 0$ | if $Ac \neq 0$ then $PC: = PC + 1$; |
| SPA | перескок по $Ac \geq 0$ | if $Ac \geq 0$ then $PC: = PC + 1$; |
| CAM | очистка Ac и MQ | $Ac: = 0$; $MQ: = 0$; |
| SWP | обмен между Ac и MQ | $Ac: = :MQ$ |

Например, запись CMA IAC предписывает инвертировать содержимое Ac и прибавить 1, т. е. результирующей операцией будет изменение знака числа в дополнительном коде.

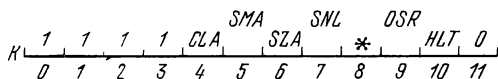


Рис. 6.4. Формат команды-микропрограммы второй группы.
* В случае $K(8) = 1$ условие перескока инвертируется.

Существует мнемокод NOP (No Operation), порождающий команду 1-й группы с нулями во всех битах, кроме трех первых.

Команда-микропрограмма 2-й группы показана на рис. 6.4. Если $K(8) = 1$, то условие перескока

через очередную команду, выражающееся в виде

$$K(5) = 1 \wedge Ac < 0 \vee K(6) = 1 \wedge Ac = 0 \vee K(7) = 1 \wedge Lk = 1,$$

заменяется его отрицанием:

$$(K(5) = 0 \vee Ac \geq 0) \wedge (K(6) = 0 \vee Ac \neq 0) \wedge (K(7) = 0 \vee Lk = 0).$$

Таким образом, в случае $K(5 : 7) = 0 \wedge K(8) = 1$ перескок производится при любых сочетаниях Ac и Lk . Эта команда безусловного перескока на языке ассемблера обозначается мнемокодом **SKP**.

Последовательность микроопераций при выполнении команды 2-й группы такова: перескоки, затем **CLA**, затем **OSR**, затем **HLT**. Это равносильно следующей микропрограмме:

```
if K(0 : 2) = 7  $\wedge$  K(3) = 1  $\wedge$  K(11) = 0 then
begin if K(5) = 1  $\wedge$  Ac < 0  $\vee$  K(6) = 1  $\wedge$  Ac = 0  $\vee$ 
      K(7) = 1  $\wedge$  Lk = 1 then
  begin if K(8) = 0 then PC := PC + 1 end
  else if K(8) = 1 then PC := PC + 1;
  if K(4) = 1 then Ac := 0;
  if K(9) = 1 then Ac := Ac  $\vee$  SR;
  if K(10) = 1 then HLT
end
```

Команда-микропрограмма 3-й группы используется главным образом в связи с устройством расширенной арифметики, которое может быть включено в состав машины в качестве необязательного добавления и здесь не рассматривается. В стандартном процессоре реализован ослабленный вариант этой команды, обеспечивающий доступ к регистру **MQ**, которым можно пользоваться для временного запоминания операндов. Управление микропрограммой осуществляют биты $K(4)$, $K(5)$ и $K(7)$. При этом в неиспользуемых битах должны содержаться нули (рис. 6.5).

Выполнение этого варианта команды равносильно выполнению следующей микропрограммы:

```
if K(0 : 2) = 7  $\wedge$  K(3) = 1  $\wedge$  K(11) = 1 then
begin if K(4) = 1 then CLA;
      if K(5) = 1  $\wedge$  K(7) = 1 then SWP
```

```

else if K(5) = 1  $\wedge$  K(7) = 0 then MQA
else if K(5) = 0  $\wedge$  K(7) = 1 then MQL
end

```

Заметим, что комбинация CLA MQA выполняется как $A_c := MQ$, а комбинация CLA SWP — как $A_c := MQ; MQ := 0$.

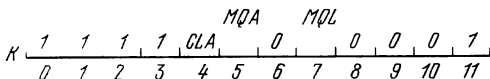


Рис. 6.5. Формат команды-микропрограммы третьей группы.

В качестве примеров использования рассмотренных команд приведем несколько простейших программ на языке ассемблера PAL III.

Программа вычисления разности $c = a - b$.

| Метки | Команды | Комментарии |
|--------|---------|--|
| * 200 | | |
| START, | CLA CLL | /A _c := 0; L _k := 0; |
| | TAD B | /A _c := A _c + m(B); |
| | CIA | /A _c := - A _c ; |
| | TAD A | /A _c := A _c + m(A); |
| | DCA C | /m(C) := A _c ; A _c := 0; |
| | HLT | /STOP; |
| A, | . . . | /значение a; |
| B, | . . . | /значение b; |
| C, | . . . | /значение c; |
| \$ | | |

Программа вычисляет разность $a - b$ путем изменения знака числа b (получения дополнения b) и последующего прибавления к полученному числу a . Найденная разность записывается в ячейку c . Тестирование переполнения не производится.

Звездочка * является указанием ассемблеру использовать в качестве начального адреса программы 200₈. Запятая является разделителем, отделяющим метку. Метку START ассемблер заменит адресом 200₈. Косая черта отделяет комментарий. Знак \$ уведомляет ассемблер о конце текста программы.

Команды отделены друг от друга тем, что каждая написана на отдельной строке (машине это сообщается посылкой символа «возврат каретки» при

переходе на каждую новую строку). Можно записывать команды в строку, используя в качестве разделителя точку с запятой: START, CLA CLL; TAD B; CIA; TAD A; и т. д.

Используемыми символами могут быть любые сочетания букв и цифр, начинающиеся буквой, но ассемблер учитывает только 6 первых знаков. Символы отделяются друг от друга пробелами и знаками-разделителями — точкой с запятой, запятой и др. Числа, представляемые последовательностями цифр, воспринимаются как восьмеричные, если нет указания в виде псевдокоманды DECIMAL считать их десятичными. Это указание отменяет псевдокоманда OCTAL.

Программа, формирующая матрицу битов с единицами на диагонали.

```
*200
START,   CLA CLL      /Ac: = 0; Lk: = 0;
          TAD M0       /Ac: = m(M0);
          DCA P        /m(P): = Ac; Ac: = 0;
          TAD W        /Ac: = m(W);
CYCLE,   DCA I P      /m(m(P)): = AC;
          Ac: = 0;
          TAD I P      /Ac: = m(m(P));
          ISZ P        /m(P): = m(P) + 1;
          RAL          /LkAc: = AcLk;
          SZL          /if Lk = 0 then
                      PC: = PC + 1;
          HLT          /STOP;
          JMP CYCLE    /go to CYCLE;
W,       0001         /значение первой
                      строки матрицы;
P,       /текущий адрес строки;
M0,      /адрес начальной
$,       строки;
```

Эта программа записывает в последовательные ячейки памяти, начиная с той, адрес которой содержится в ячейке M0, серию двоичных слов вида

```
000000000001
000000000010
000000000100
```

```
010000000000
100000000000
```

Запись осуществляется многократным выполнением последовательности команд CYCLE (цикл). Адрес очередной строки матрицы находится в ячейке P, содержимое которой при каждом повторении цикла увеличивается на единицу командой ISZ. Обращение к ячейке-строке производится методом косвенной адресации посредством ячейки P. Значение очередной строки записывается в соответствующую ячейку командой DCA I P.

Буква I означает косвенную адресацию, т.е. $K(3) = 1$.

Поскольку при записи в память происходит очистка аккумулятора, содержимое его восстанавливает команда TAD I P. Значение для следующей строки получается командой RAL — циклический сдвиг LkAc влево. Когда сдвигаемая по аккумулятору единица попадает в Lk, повторения цикла прекращаются.

Программа, суммирующая содержимое 1024 последовательных ячеек.

| | | |
|--------|----------|---------------------------------------|
| * 10 | | |
| AJ, | 0 | /указатель адреса очередной строки |
| * 200 | | |
| START, | CLA CLL | /Ac: = 0; Lk: = 0; |
| | TAD C1 | /Ac: = m(C1); |
| | DCA CJ | /m(CJ): = Ac; Ac: = 0; |
| | TAD A1 | /Ac: = m(A1); |
| | DCA AJ | /m(AJ): = Ac; Ac: = 0; |
| | TAD I A1 | /Ac: = m(m(A1)); |
| | TAD I AJ | /m(AJ): = m(AJ) + 1; |
| | | Ac: = Ac + m(AJ); |
| | ISZ CJ | /m(CJ): = m(CJ) + 1; |
| | | if m(CJ) = 0 then SKIP; |
| | JMP. -2 | /go to PC - 3; |
| | DCA SUM | /m(SUM): = Ac; Ac: = 0; |
| | HLT | /STOP; |
| C1, | 6001 | /начальное значение счетчика, |
| CJ, | | . /счетчик, |
| A1, | | . /начальный адрес, |
| SUM, | | . /сумма |

Точку в выражении операнда команды ассемблер заменяет адресом этой команды. Так, команда JMR.—2 задает переход на PC-3 (см. примечание к табл. 6.2).

Выборка суммируемых значений, за исключением первого, осуществляется с помощью автоиндексируемой ячейки AJ (адрес 10_8). Счетчик CJ считает слагаемые, начиная с 2-го, и обнуляется после 1024-го. Переполнение аккумулятора игнорируется, т.е. суммирование ведется по модулю 2^{12} .

§ 4. Команды ввода/вывода

При $K(0 : 2) = 6$ командное слово интерпретируется процессором как команда ввода/вывода. Такая команда содержит 6-битный $K(3 : 8)$ номер устройства, к которому она относится, и 3-битный $K(9 : 11)$ код операции, которую надлежит выполнить с этим устройством. При различных значениях номера устройства один и тот же код операции может иметь различный смысл, указывать различные операции. Устройствам, для управления которыми мало 8 операций, кодируемых 3-битным кодом (например, магнитные ленты, диски, считыватели перфокарт), сопоставляется более одного номера. Таким образом, код операции в совокупности с номером устройства понимается как команда, предписывающая некоторую операцию с тем или иным органом определенного устройства. На языке ассемблера такая команда обозначается единым мнемоническим словом, например: KCF (Clear Keyboard Flag) — «сбросить флажок клавиатуры», KSF (Skip on Keyboard Flag) — «перескочить по флажку клавиатуры», ADRB (Read A/D Converter Buffer) — «считать с буфера аналого-цифрового преобразователя» и т.п.

Команда ввода/вывода, выданная процессором в магистраль, к которой подключены своими блоками управления (контроллерами) периферийные устройства, воспринимается и выполняется только тем из них, к которому относится указанный в этой команде номер. Каждое устройство обладает буфером данных, т.е. регистром, в который принимается слово, выданное процессором на вывод, или из которого процессор получает слово в случае ввода. Кроме

того, устройство содержит бит готовности — «флажок», состояние которого процессор может тестировать выполнением команды перескока по флажку данного устройства.

Поднятие флажка сигнализирует о готовности устройства к передаче данных. В случае ввода поднятый флажок означает, что вводимое слово уже находится в буфере и процессор может произвести считывание с буфера соответствующей командой. В случае вывода поднятый флажок указывает на то, что буфер освобожден и процессор может переслать в него очередное слово для вывода.

При считывании с буфера и при пересылке в буфер процессор сбрасывает флажок и ждет, пока устройство вывода не перенесет полученное от процессора слово из буфера на внешний носитель или пока устройство ввода не произведет очередное считывание с внешнего носителя в буфер. Ожидание процессор осуществляет путем циклического повторения команды, тестирующей состояние флажка.

Примером такого цикла может служить следующий фрагмент программы, выполняющей ввод с буфера клавиатуры (он же является буфером считывания с перфоленты):

| | |
|----------|--|
| KCC | /очистить аккумулятор, сбросить флажок и инициировать считывание перфоленты, |
| KSF | /если флажок поднят, перескочить через очередную команду, |
| JMP. — 1 | /перейти на предыдущую команду, |
| KRB | /переслать содержимое буфера в аккумулятор, сбросить флажок. |

Очистив аккумулятор и инициировав ввод при сброшенном флажке, процессор выполняет циклически команды KSF и JMP. — 1, пока устройство ввода не произведет загрузку в буфер полученного с клавиатуры или считанного с перфоленты кода, о чем устройство сигнализирует поднятием флажка. Команда KSF, выполненная при поднятом флажке, осуществляет выход из цикла на команду KRB, завершающую ввод считыванием с буфера в аккумулятор и сбросом флажка.

Аналогичный цикл ожидания используется в случае вывода на телетайп (печать/перфорация ленты):

| | |
|------------|---|
| CLA CLL | /очистить аккумулятор и линк, |
| TLS | /переслать в буфер содержимое аккумулятора, сбросить флажок; произведя печать (перфорацию), поднять флажок, |
| TAD CHRACT | /загрузить в аккумулятор код выводимой литеры, |
| TSF | /если флажок поднят, перескочить через очередную команду, |
| JMP. — 1 | /перейти на предыдущую команду, |
| TSF | /переслать в буфер содержимое аккумулятора, сбросить флажок; произведя печать (перфорацию), поднять флажок. |

По условиям работы телетайпа загрузка буфера должна производиться в определенный промежуток времени, начинающийся с поднятия флажка готовности, т.е. поднятие флажка является сигналом загружать буфер.

Чтобы получить этот сигнал для вывода первой литеры, в начале приведенного фрагмента программы осуществлена загрузка буфера нулевым кодом. Этому коду не сопоставлено никакой литеры, и загрузка им буфера не вызывает ни печати, ни перфорации, однако поднятие флажка она вызывает. Для загрузки второй и всех последующих литер сигналом служит поднятие флажка, возникающее по окончании печати или перфорации предшествующей литеры.

§ 5. Система прерывания и прямой доступ к памяти

Рассмотренный способ ввода/вывода, называемый программированным или управляемым программой вводом/выводом, связан со значительными потерями времени процессора в циклах ожидания. Чтобы не упустить вводимых данных или момента готовности устройства к выводу очередной литеры, процессор должен заблаговременно переходить в цикл ожидания и поэтому практически лишен возможности совме-

щать ввод/вывод с выполнением другой полезной работы.

Эта возможность обеспечивается при использовании прерывания. Если система прерывания включена, то поднятие флажка любым подключенным к ней периферийным устройством автоматически вызывает по окончании выполнения текущей команды переход на подпрограмму при $EA = 0$, т.е. производится запоминание адреса возврата в нулевой ячейке памяти и выборка очередной команды из первой ячейки:

$m(0) := PC; PC := 1;$

При этом происходит выключение системы прерывания и новое прерывание не может возникнуть, пока система не будет включена специальной командой из группы команд ввода/вывода — **ION** (**Interrupt turn ON** — «прерывание включить»). Имеется также противоположная по смыслу команда — **IOF** (**Interrupt turn OFF** — «прерывание выключить»). При включении машины система прерывания автоматически выключается, а флажки всех периферийных устройств сбрасываются.

Команда, считанная при входе в прерывание из ячейки с адресом 1, является командой перехода на начало подпрограммы, реагирующей на прерывание (обслуживающей прерывание). Обычно содержимое первых ячеек памяти таково:

| | | |
|----|---------|---|
| 0, | 0 | /сохранение адреса возврата; |
| 1, | JMP I 2 | /переход с косвенным адресом; |
| 2, | SERV | /адрес начала обслуживающей подпрограммы. |

Подпрограмма, обслуживающая прерывание, должна:

- упрятать (сохранить в ячейках памяти) содержимое Ac и Lk ,
- определить устройство, вызвавшее прерывание поднятием флажка,
- выполнить операции по обслуживанию этого устройства,
- восстановить упрятанное содержимое Ac и Lk ,
- произвести переход на продолжение прерванной программы, включив систему прерывания.

Примером такой подпрограммы может служить подпрограмма, обслуживающая прерывание в случае, когда к системе подключены только два периферийных устройства: клавиатура телетайпа (KB — Keyboard) и печатающее устройство (TP — Teleprinter).

Начало подпрограммы, обслуживающей прерывание.

| | | |
|-------|--------|------------------------------|
| SERV, | DCA A | /m(A): = Ac; Ac: = 0; (упря- |
| | | тывание Ac), |
| | RAL | /LkAc: = AcLk; |
| | DCA B | /m(B): = Ac; Ac: = 0; (упря- |
| | | тывание Lk), |
| | KSF | /перескок, если поднят фла- |
| | | жок клавиатуры, |
| | SKP | /безусловный перескок, |
| | JMP KB | /переход на обслуживание |
| | | клавиатуры, |
| | TSF | /перескок, если поднят фла- |
| | | жок телепринтера, |
| | SKP | /безусловный перескок, |
| | JMP TP | /переход на обслуживание |
| | | телепринтера, |
| | HLT | /стоп. |

Здесь А, В — имена ячеек памяти, используемых для сохранения содержимого аккумулятора и линка, KB, TP — начальные адреса подпрограмм, обслуживающих соответственно клавиатуру и телепринтер.

Подпрограмма, обслуживающая клавиатуру, считывает содержимое буфера клавиатуры в аккумулятор, сбрасывая флажок, а затем из аккумулятора — в очередную ячейку буфера ввода в главной памяти.

Подпрограмма, обслуживающая телепринтер, посылает содержимое очередной ячейки буфера вывода в аккумулятор, а затем из аккумулятора в буфер телепринтера, сбрасывая его флажок. По окончании каждой из этих подпрограмм производится переход на подпрограмму выхода из прерывания, которая восстанавливает упрятанные значения аккумулятора и линка, включает систему прерывания и производит переход с косвенной адресацией относительно нулевой ячейки, содержащей адрес возврата в прерванную программу.

Окончание подпрограммы, обслуживающей прерывание.

| | | |
|-------|---------|--|
| EXIT, | CLA CLL | /Ac: = 0; Lk: = 0; |
| | TAD B | /Ac: = m(B); |
| | RAR | /AcLk: = LkAc; (восста- новление Lk), |
| | TAD A | /Ac: = m(A); (восстановление Ac), |
| | ION | /включить систему преры- вания, |
| | JMP I O | /переход по адресу воз- врата. |

Команда ION выполняется с задержкой — система прерывания будет включена лишь после выполнения следующей за ION команды, т.е. после того, как произойдет возврат в прерванную программу.

Подпрограмму SERV, определяющую устройство, которым вызвано прерывание, нетрудно расширить применительно к любому числу обслуживаемых устройств. Устройства, требующие безотлагательного обслуживания, должны, разумеется, занимать первые места в цепочке проверок, а «неторопливые» устройства (клавиатура, телепринтер) могут находиться в ее конце, не рискуя потерей информации. В случае, если окажутся поднятыми флажки двух или более устройств, первым обслуживается устройство, стоящее ближе других к началу цепочки проверок, а затем остальные в порядке занимаемых ими в этой цепочке мест. На данной основе строится программируемая система приоритета прерываний, в которой устройства с высоким приоритетом не только обслуживаются в первую очередь, но и могут прерывать уже начавшееся обслуживание устройства более низкого приоритета. Последнее обеспечивается тем, что подпрограммы обслуживания с низким приоритетом производят включение системы прерывания не после завершения обслуживания, а в самом начале его. При этом возникает необходимость упрятывания более чем одной пары значений аккумулятора и линка, а также перезапоминания содержащегося в нулевой ячейке адреса возврата.

При работе с быстродействующими периферийными устройствами (диски, магнитные ленты) исполь-

зуют ввод/вывод путем прямого доступа к памяти (DMA — Direct Memory Access, фирменное название — Data Break). Передача данных при этом производится непосредственно между периферийным устройством и главной памятью, не затрагивая процессор. Передается группа следующих друг за другом слов (массив). Устройство в результате выполнения процессором специальной подпрограммы получает адрес ячейки главной памяти, с которой начинается поле размещения массива, число передаваемых слов, номер зоны периферийной памяти, с которой будет производиться обмен, и направление передачи данных: ввод в главную память или вывод из нее на внешний носитель.

Помимо буферного регистра для приема/выдачи передаваемых слов, устройство (точнее, его контроллер) имеет регистр адреса и счетчик, в которые помещаются полученные от процессора уменьшенный на 1 адрес начала массива в главной памяти и отрицательное число, указывающее, сколько слов надо передать. Передачу каждого слова устройство производит в следующей последовательности: прибавлением 1 в регистр адреса формируется адрес ячейки главной памяти, с которой будет происходить обмен, прибавлением 1 в счетчик учитывается количество переданных слов — нуль в счетчике получается при выполнении последней передачи. Будучи готовым произвести очередную передачу слова, устройство захватывает («ворует») у процессора один цикл главной памяти, в течение которого обращается к указанной в его регистре адреса ячейке. В соответствии с заданным направлением передачи либо содержимое этой ячейки копируется в буфер устройства (вывод), либо происходит запись в нее содержимого буфера (ввод). По окончании передачи, произведенной при нулевом значении счетчика слов, устройство возбуждает прерывание, уведомляющее процессор, что заданная передача массива данных выполнена.

Существует и другой вариант реализации прямого доступа к памяти, отличающийся тем, что периферийное устройство не имеет регистра адреса и счетчика слов, а использует для выполнения их функций две ячейки главной памяти, указывая адреса этих ячеек. В таком варианте передача каждого слова

занимает не один, а три цикла памяти: цикл модификации счетчика, цикл модификации адреса и цикл собственно передачи слова.

Запросы циклов устройствами при прямом доступе к памяти удовлетворяются вне всякой очереди. Другими словами, при прямом доступе устройство обладает более высоким приоритетом, чем процессор и любое устройство — запросчик прерывания.

В режиме прямого доступа к памяти одновременно могут работать несколько устройств — на PDP-8/E до 12. При этом может случиться, что два (или более) устройства попытаются захватить один и тот же цикл памяти. Чтобы исключить возможность такой коллизии, каждому устройству прямого доступа сопоставлена одна из 12 приоритетных линий. При наличии нескольких претендентов цикл достается подключенному к линии с наибольшим приоритетом.

§ 6. Добавочное и периферийное оборудование

Базовая конфигурация PDP-8 включает процессор, 4KW память и телетайп в качестве устройства ввода/вывода, обеспечивающего возможность ввода с клавиатуры и путем считывания с бумажной перфоленты и вывод путем перфорации ленты и/или печатания. Расширение системы производится добавлением как внутреннего, так и периферийного оборудования. Внутреннее расширение может быть осуществлено наращиванием главной памяти до 32KW, добавлением блока расширенной арифметики, датчиков (часов) реального времени и устройства, сигнализирующего о ненормальности режима электропитания. Для расширения периферийной части системы имеется большое число (более 70 наименований) устройств ввода/вывода, периферийной памяти и связного оборудования.

Главная память наращивается модулями по 4KW. Поскольку 12-битное слово PDP-8 позволяет адресовать не более 4K ячеек, то вместе с добавочными модулями вводятся два трехбитных регистра — IF и DF, используемые для наращивания 12-битного адреса. В регистр IF (Instruction Field) помещают номер модуля, из которого выбираются команды

и операнды при выполнении команд AND, TAD, ISZ, DCA с прямой адресацией. Регистр DF (Data Field) указывает модуль, из которого выбираются операнды в случае косвенной адресации, причем адрес операнда выбирается из модуля, указываемого регистром IF, а соответствующее этому адресу значение (или ячейка, в которую производится запись содержимого аккумулятора командой DCA) находится в модуле, указываемом регистром DF.

Нумерация модулей начинается с нуля: 0, 1, 2, ..., 7. В начале работы номера используемых модулей засылают в регистры IF и DF с помощью ключей панели управления, а в процессе работы засылки производятся специальными командами CIF и CDF. Имеется также аппарат автоматического упрятывания содержимого регистров IF и DF при входе в прерывание и ряд команд, предназначенных для управления последовательностью переключения модулей памяти с учетом текущего состояния системы. Все эти команды, в том числе CIF и CDF, входят в группу команд ввода/вывода.

Оборудование для укомплектования периферийной части PDP-8 включает устройства периферийной памяти и ввода/вывода данных, аппаратуру сопряжения с линиями связи, аппаратуру сопряжения с измерительными приборами и исполнительными механизмами.

В число устройств периферийной памяти входят:

- диск-картридж RK8 (емкость — $0,83\text{M} \times 12$ бит, среднее время обращения — 130 мс, к одному контроллеру подключаются 4 диска, общая емкость — $3,32\text{MW}$),

- быстрый диск с произвольным доступом RF08 (емкость — $32\text{K} \times 13$ бит, среднее время обращения — 20 мс, расширяем до 128KW),

- магнитная лента DEStape TU 56 (емкость — $190\text{K} \times 12$ бит, скорость передачи — 100 тыс бит/с., к одному контроллеру подключается до 8 лентопряжек).

В качестве пультового устройства ввода/вывода, кроме телетайпа ASR 33, имеется LA 30 DECwriter — матричный бесшумный принтер со скоростью 30 зн/с, 80 знаков в строке, клавиатура с 96 знаками в коде ASCII. В отличие от телетайпа,

ЛА 30 не имеет перфоратора и считывателя перфоленты.

Быстродействующее перфолентное оборудование включает считыватель PR8-E (300 зн/с) и перфоратор PP8-E (50 зн/с), а также их комбинацию — PC8-E.

В качестве устройств ввода/вывода используются также следующие:

Считыватель перфокарт CR8-E (200 карт/мин).

Алфавитно-цифровой дисплей VT05 (20 строк по 72 знака).

Осциллографический дисплей VC8-E (1024×1024 точек).

Устройство ввода графической информации VW01 (разрешающая способность — 10 битов по X и по Y).

Устройства графического вывода барабанного типа XY8-EA на основе бумажной ленты шириной 30 см и XY8-EB на основе ленты шириной 76 см.

Планшетные устройства графического вывода XY8-EH, XY8-EJ, XY8-EK на основе Houston Instrument DP-10.

Строчный принтер LE 8 (длина строки — 80 или 132 знака, набор знаков — 64 или 96, скорость — от 245 до 1100 строк в минуту).

В качестве средств сопряжения с линиями связи используется следующее специальное оборудование:

Аппаратура сопряжения с линиями синхронной связи DP8-EA для работы с модемами типа Bell System 200-Series и DP8-EB для работы с модемами Bell System 300-Series.

Аппаратура сопряжения с линиями асинхронной связи: KL8-EA, KL8-EB, ..., KL8-EG со скоростями передачи от 110 до 2400 бод.

Аппаратура сопряжения с лабораторными приборами включает: преобразователь аналог-цифра AD8-EA (выход — 10 битов, частота отсчетов более 50 кгц), обслуживающий с аналоговыми мультиплексорами AM8-EA, AM8-EB до 16 каналов; устройство DRB-EA цифрового ввода/вывода по 12 одноканальным каналам; лабораторную монтажную панель — шасси для компактного и удобного размещения лабораторных приборов.

Аппаратура сбора данных и управления представлена аналоговой мультиплексорной системой с диа-

пазоном входных сигналов от 10 мв до 100 в при количестве каналов от 4 до 1024 и длине слова на выходе 10—11 битов, системой преобразования цифра-аналог с числом каналов до 64 и длиной слова на входе 10—12 битов, универсальной системой цифрового управления, содержащей 256 адресуемых каналов, каждый из которых может служить для ввода или для вывода 12-битных слов, т. е. для получения информации о состояниях 12 двузначных объектов или для управления этими объектами.

В качестве периферийного устройства к PDP-8/E имеется процессор с плавающей запятой FPP-12, использование которого позволяет ускорить вычисления с плавающей запятой приблизительно в 40 раз по сравнению с реализацией их путем программирования на процессоре PDP-8/E. Процессор FPP-12 выполняет операции сложения, вычитания, умножения и деления над 36-битными числами (3 слова по 12 битов): 24 бита — мантисса, 12 битов — порядок.

Еще одно специальное периферийное устройство — межпроцессорный буфер DB8-E — обеспечивает возможность обмена данными между двумя PDP-8/E в виде передаваемых впараллель 12-битных слов с частотой 5 кгц.

Все перечисленные выше устройства обеспечены соответствующими контроллерами, программным оснащением и всеми принадлежностями, необходимыми для подключения их к машине.

§ 7. Программное оснащение PDP-8

К машинам семейства PDP-8 имеется обширное, тщательно разработанное и проверенное большой практикой программное оснащение, пополнение и развитие которого продолжается на протяжении полутора десятка лет. В дополнение к сотням программ, созданным программистами самой фирмы Digital Equipment Corp., существует и постоянно пополняется библиотека DECUS (DEC Users' Society) — общества пользователей машин PDP.

DEC различает в составе программного оснащения три категории программ: системные, служебные и прикладные. К системным относятся средства программирования и управления работой цифровой

системы: редакторы, загрузчики, ассемблеры, компиляторы, интерпретаторы, отладочные программы, мониторы. Служебные программы обеспечивают ввод/вывод и преобразование представления данных (десятичного в двоичное, двоичного в десятичное и т. п.), выполнение операций с плавающей запятой, реализацию математических функций, а также тестирование аппаратуры и диагностику неисправностей. Прикладные программы охватывают широкое разнообразие применений миникомпьютеров, в частности, к ним относятся специализированные системы программного оснащения, такие как системы обработки деловой информации на основе языка дибол (dibol — Digital Equipment Corp. Business Oriented Language), обучающие EDUSYSTEMS, лабораторные LAB-8, типографского набора Typeset 8, сбора данных и управления в режиме реального времени INDAC и др.

Достаточно мощный арсенал средств программирования доступен даже на базовой конфигурации PDP-8, включающей процессор, 4KW память и теле-тайп. В него входят: символьный редактор, ассемблеры PAL III и MACRO-8, интерпретаторы «разговорных» языков программирования focal (FOrmula CALculator) и basic, фортран-компилятор, загрузчики RIM и BIN (Binary), отладочные программы DDT и ODT, а также большой набор служебных программ.

Управление последовательностью работ, программируемых на языке ассемблера базовой конфигурации, осуществляется вручную. Первоначально оператор вводит с панели управления минимальный загрузчик RIM, используемый для ввода с перфоленты загрузчика BIN, осуществляющего ввод абсолютного двоичного кода, в котором представлены системные программы и программы, получаемые с их помощью. Для подготовки разрабатываемой программы в машину с помощью загрузчика BIN вводится символьный редактор, обеспечивающий возможность ввода программы в символьном представлении с клавиатуры или с перфоленты, внесения необходимых изменений и получения перфоленты с исходной программой, готовой для ассемблирования. Затем, снова используя BIN, вводят ассемблер и за два-три прохода получают программу на машинном языке, которая после

отладки ее с помощью ODT или DDT готова для использования в виде перфоленты, загрузку которой осуществляет все тот же BIN.

Системы программирования в режиме диалога focal-8 и basic-8, а также 4KW фортран-компилятор обладают встроенными средствами управления, выполняющими необходимый в каждом случае минимум функций операционной системы. Наличие этих средств вместе с простотой и естественностью языков focal и basic обеспечивает общедоступность систем focal-8 и basic-8 для программирования и выполнения вычислений непрограммистами. Многие системы невычислительного назначения (программируемые эксперименты, испытательно-измерительные комплексы), использующие в качестве языков программирования модифицированные или расширенные версии языков focal и basic, построены на основе этих систем.

Имеется несколько вариантов систем focal и basic. Минимальный вариант этих систем, рассчитанный на одного пользователя (один телетайп), реализован на базовой конфигурации аппаратуры. При наличии 8KW памяти система focal допускает одновременную работу четырех пользователей, а система basic — пяти пользователей. Добавление к 8KW диска или магнитной ленты DEStape позволяет увеличить число пользователей focal до семи, а basic до шестнадцати.

Конфигурация с 8KW (без периферийной памяти) достаточна для работы с ассемблером SABR (Symbolic Assembler for Binary Relocatable programs) и 8KW фортран-компилятором, использующим SABR в качестве выходной ступени. SABR — это однопроходный ассемблер, выдающий переместимые программы, загрузку которых осуществляет 8KW загрузчик. Важным достоинством SABR является возможность странично-независимого программирования: ассемблер автоматически производит межстраничные ссылки и переходы при переполнении страниц памяти.

Конфигурации, включающие устройства периферийной памяти, как правило, оснащаются мониторными системами. Простейшая клавиатурно-ориентированная дисковая мониторная система может работать уже при 4KW главной памяти, предоставляя пользователю возможность эффективно управлять прохождением работ и обеспечивая его достаточно

полным набором средств программирования, содержащим редактор, ассемблер PAL-D (гибрид PALIII и макро-8), фортран D и динамический отладчик DDT-D. Система включает специальную программу (Bilder), осуществляющую генерацию конкретной версии монитора для данного комплекта аппаратуры.

Мониторы разделения времени TSS/8 и TSE (Time-Sharing PDP-8/E) требуют минимум 12KW главной памяти и диск. Обеспечивается работа одновременно до 16 пользователей, которым предоставляется возможность ассемблирования, компилирования, редактирования, загрузки, отладки и прогонки программ в режиме on-line с использованием системной библиотеки подпрограмм, периферийной памяти, а также диалогового программирования на языках focal и basic.

Наиболее развитой является система OS/8 (в первоначальной версии PS/8 — Programming System/8), характеризующаяся чертами, свойственными операционным системам больших машин. Она может работать на PDP-8/E с минимумом 8KW памяти при наличии дисков или магнитных лент DECtape. Система обеспечивает аппаратно-независимый доступ к периферийным устройствам (до 15 различных устройств, в том числе до 8 лентопротяжек и до 4 дисководов), управление файлами на этих устройствах, работу с языковыми процессорами 8KW focal, 8KW фортран, PAL-8 SABR, редактором, загрузчиками и средствами отладки. Система OS/8 адаптируема к различным периферийным устройствам и расширяется по мере наращивания используемой конфигурации аппаратуры. Она способна работать с емкостью главной памяти от 8KW до 32KW.

ГЛАВА 7

ДВУХАККУМУЛЯТОРНЫЕ МИНИКОМПЬЮТЕРЫ HEWLETT-PACKARD

Миникомпьютеры Hewlett-Packard являются типичными образцами двухаккумуляторной миниархитектуры, которую можно рассматривать в качестве первой ступени дальнейшего развития и усложнения одноаккумуляторной миниархитектуры семейства DEC PDP-8.

Компания Hewlett-Packard к тому времени, когда она включилась в производство миникомпьютеров (1966 — 1967 гг.), являлась одним из крупнейших производителей электронных измерительных приборов (осциллографов, генераторов, анализаторов спектра и т. п.). Поэтому первые выпущенные ею миникомпьютеры HP 2116, HP 2115 и HP 2114 предназначались главным образом для сопряжения с измерительными приборами с целью программированного управления экспериментом, автоматического сбора и обработки результатов измерений. В этом важном секторе применения миникомпьютеров Hewlett-Packard стала ведущей фирмой.

В последующие годы компания неуклонно расширяла производство цифровой техники различного назначения, от карманных и настольных калькуляторов до многоязычных мультипрограммных минисистем обработки данных HP 3000, HP 3000-II, сопоставимых по возможностям с системами, реализуемыми на базе средних и больших цифровых машин, но существенно более дешевых и простых в отношении обслуживания и использования.

Техническую основу этих и других систем Hewlett-Packard составляют две серии миникомпьютеров:

НР 2100, начало которой положили упомянутые выше первые модели, и НР 21МХ, характерная интенсивным использованием микропрограммирования.

Несмотря на то, что от модели к модели происходило обогащение архитектуры и поздние машины серии 2100, например, появившиеся в 1974 г. модели 2105 и 2108, а тем более машины серии 21МХ, значительно отличаются от первых моделей, однако все они способны выполнять программы, написанные на языке ассемблера для их предшественников, и совместимы с ранее созданными периферийными устройствами.

В настоящей главе архитектура машин Hewlett-Packard рассматривается в наиболее простом (первоначальном) ее варианте, реализованном в машинах НР 2114 — НР 2116.

§ 1. Регистры процессора и форматы команд

В отношении архитектуры миникомпьютеры Hewlett-Packard представляют собой двухаккумуляторные 16-битные машины с многократной косвенной и страничной адресацией.

Процессор интерпретирует машинное слово либо как 16-битную команду $K(15:0)$, либо как адрес $A(15:0)$ ячейки главной памяти, в котором старший бит $A(15)$ указывает вид адреса: $A(15) = 0$ — прямой, $A(15) = 1$ — косвенный, а остальная часть слова представляет собой число без знака — собственно адрес ячейки. В качестве данного слово обрабатывается как число со знаком в двоичном дополнительном коде и как булевский вектор.

Главная память является массивом $m(0:k-1)$ 16-битных слов, максимальное число которых $k_{\text{макс}} = 2^{15} = 32\text{К}$. В системе страничной адресации память рассматривается как двумерный массив $m(0:q-1, 0:1023)$, в котором страницы содержат по 1024 слова, т. е. по K слов, а максимальное число страниц равно $q_{\text{макс}} = 32$.

В более поздних моделях операционная система осуществляет с помощью специальной аппаратуры расширение памяти до 1024 страниц, т. е. до 1МВ.

Доступными программе регистрами процессора являются: 15-битный программный счетчик PC(14:0), два 16-битных аккумулятора: A(15:0) и B(15:0), однобитные регистры: удлинитель аккумуляторов Ex (Extend) — аналог бита связи Lk в PDP-8 и индикатор переполнения Ov (Overflow).

Однобитный регистр Ex выполняет функцию соединителя в кольцо аккумулятора A (B) при операциях циклического сдвига, выполняемых с участием этого регистра, и принимает значение 1, если в результате операции сложения или приращения содержимого аккумулятора A (или B) возникает отличный от нуля перенос из A(15) или из B(15). Программа может тестировать значение бита Ex, устанавливая его равным 1 или 0 и инвертировать.

Бит переполнения Ov принимает значение 1, если при выполнении сложения или приращения содержимого аккумулятора A (или B) имеет место арифметическое переполнение, т. е. если в результате неотрицательных (содержащих в 15-м бите нуль) чисел получается отрицательное (в 15-м бите единица) число или в результате сложения отрицательных чисел получается неотрицательное число.

Аккумуляторы A и B обладают адресами в системе адресации главной памяти: адресом аккумулятора A является 0, адресом аккумулятора B является 1. Это позволяет производить над содержимым аккумуляторов двухместные операции, а также одноместные операции, определенные над ячейками памяти при помощи одноадресных команд. Например, команда ADB 0 («сложение в аккумуляторе B, адрес нуль») выполняется как

$$B := B + A,$$

поскольку адрес 0 принадлежит аккумулятору A. Другой пример: команда ISZ (Increment and Skip if Zero — «прирастить содержимое адресуемой ячейки, и если в результате получится нуль, то перескочить через следующую команду»). Употребленная с адресом 1, т. е. в виде ISZ 1 эта команда предписывает выполнить следующее:

$$B := B + 1; \text{ if } B = 0 \text{ then } PC := PC + 1.$$

Мы принимаем, что подразумеваемое при последо-

вательной выборке команд приращение программного счетчика РС производится сразу же после выборки текущей команды, поэтому явно обозначенное в данном примере прибавление единицы к РС в процессе выполнения команды является дополнительным, вызывающим перескок через следующую по порядку адресов команду.

В отличие от безадресных команд приращения содержимого аккумуляторов INA и INB, при выполнении которых бит-удлинитель E_x принимает значение 1 в случае ненулевого переноса из старшего (15-го) бита аккумулятора, а бит переполнения O_v фиксирует арифметическое переполнение аккумулятора, адресная команда приращения ISZ как содержимого ячейки памяти, так и содержимого аккумулятора, указываемого его адресом, не затрагивает регистров E_x и O_v .

В машинах серии HP 2100 используются три основных вида команд: адресные, регистровые и

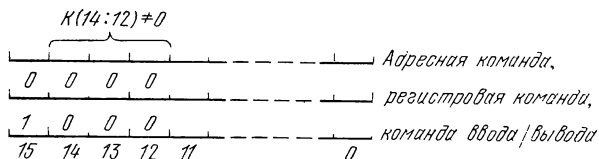


Рис. 7.1. Виды команд в миникомпьютерах Hewlett-Packard.

ввода/вывода (рис. 7.1). Эти виды различаются по следующим признакам:

- $K(14:12) \neq 0$ — адресная команда,
- $K(15:12) = 0$ — регистровая команда,
- $K(15:12) = 10_8$ — команда ввода/вывода.

Регистровые команды и команды ввода/вывода разделяются на группы в зависимости от значения бита $K(10)$. Первая группа регистровых команд ($K(10) = 0$) включает команды для микропрограммирования сдвигов в аккумуляторах, вторая группа ($K(10) = 1$) — команды модификации содержимого аккумуляторов и регистра E_x , а также команды перескока по заданному условию. Команды ввода/вывода при $K(10) = 0$ относятся к расширенной арифметике, а при $K(10) = 1$ являются действительными команда-

ми управления периферией, но в их число включены также команды, манипулирующие битом переполнения Ov.

§ 2. Адресные команды

Команда интерпретируется процессором как адресная, если код командного слова $K(15:0)$ удовлетворяет условию $K(14:12) \neq 0$. При этом 4 бита $K(14:11)$ рассматриваются как код операции, а остальные 12 битов — как адресная часть (рис. 7.2).

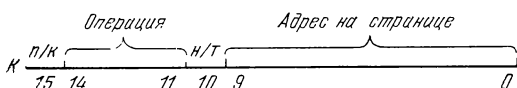


Рис. 7.2. Формат адресной команды.

С учетом условия $K(14:12) \neq 0$ имеется всего 14 значений кода операции для адресных команд. Коды, мнемонические обозначения и описания этих команд представлены в табл. 7.1.

Первые три команды задают булевские операции «И», «ИЛИ с исключением» («неэквивалентность») и «ИЛИ», выполняемые побитно над содержимым аккумулятора A и указываемой адресной частью команды ячейки памяти. Следующие три команды — «переход на подпрограмму», «безусловный переход» и «приращение с перескоком по нулю» — не затрагивают аккумуляторов, а касаются только главной памяти и программного счетчика PC .

Последние восемь команд задают операции сложения, сравнения, засылки в аккумулятор и запоминания содержимого аккумулятора в ячейке главной памяти. В этих командах собственно операция указывается тремя битами $K(14:12)$, а четвертый бит — $K(11)$ — представляет собой номер того аккумулятора, к которому относится команда: $K(11) = 0$ — аккумулятор A , $K(11) = 1$ — аккумулятор B . Заметим, что в этой роли бит $K(11)$ выступает также в регистровых командах и в командах ввода/вывода.

В процессе выполнения всякой адресной команды производится вычисление исполнительного адреса EA по имеющимся в адресной части команды элементам:

Адресные команды машин серии HP 2100

| $K(14:11)_8$ | Мнемокод | Операция | Описание |
|--------------|----------|--------------------------------------|---|
| 2 | AND | конъюнкция | $A := A \wedge m(EA);$ |
| 3 | XOR | сложение битов | $A := A \oplus m(EA);$ |
| 4 | IOR | дизъюнкция | $A := A \vee m(EA);$ |
| 5 | JSB | переход на под- программу | $m(EA) := PC;$ $PC := EA + 1;$ |
| 6 | JMP | переход | $PC := EA;$ |
| 7 | ISZ | приращение и пе- реросток по нулю | $m(EA) := m(EA) + 1;$ if $m(EA) = 0$ then $PC :=$ $PC + 1;$ |
| 10 | ADA | сложение в A | $CrA := A + m(EA);$ if $Cr = 1$ then $Ex := 1;$ if $OVERFL(A) = 1$ then $Ov := 1;$ |
| 11 | ADB | сложение в B | $CrB := B + m(EA);$ if $Cr = 1$ then $Ex := 1;$ if $OVERFL(B) = 1$ then $Ov := 1;$ |
| 12 | CPA | сравнение с A | if $A \neq m(EA)$ then $PC := PC + 1;$ |
| 13 | CPB | сравнение с B | if $B \neq m(EA)$ then $PC := PC + 1;$ |
| 14 | LDA | загрузка в A | $A := m(EA);$ |
| 15 | LDB | загрузка в B | $B := m(EA);$ |
| 16 | STA | запоминание A | $m(EA) := A;$ |
| 17 | STB | запоминание B | $m(EA) := B;$ |

Обозначения. A, B — аккумуляторы, Cr — цифра переноса из A(15) или из B(15). Ex — бит-удлиннитель аккумуляторов, Ov — бит переполнения, EA — исполнительный адрес.

указателю вида адресации K(15), указателю страницы K(10) и адресу в пределах страницы K(9:0).

Указатель вида адресации, обозначаемый условно п/к (прямая/косвенная), принимает два значения:

$$K(15) = \begin{cases} 0 - \text{прямой адрес,} \\ 1 - \text{косвенный адрес.} \end{cases}$$

Указатель страницы, обозначаемый н/т (нулевая/текущая), также принимает два значения:

$$K(10) = \begin{cases} 0 - \text{нулевая страница,} \\ 1 - \text{текущая страница.} \end{cases}$$

Адрес в пределах страницы состоит из десяти битов $K(9:0)$ и принимает 1024 значения, интерпретируемые как целые числа без знака в диапазоне от 0 до 1777_8 . При $K(10) = 0$, т. е. при обращении к нулевой странице, исполнительным адресом является просто $K(9:0)$. В случае же $K(10) = 1$, т. е. при обращении к текущей странице, исполнительный адрес получается конкатенацией пяти старших битов программного счетчика $PC(14:10)$, содержащих номер текущей страницы (страницы, с которой считана выполняемая в текущий момент команда), и указанного в команде адреса на странице $K(9:0)$:

$$EA := PC(14:10)K(9:0).$$

В случае прямой адресации, т. е. когда $K(15) = 0$, полученный таким образом адрес является исполнительным адресом. Если же $K(15) = 1$, то адрес является косвенным, т. е. указывает ячейку главной памяти, к которой надо обратиться, чтобы продолжить вычисление исполнительного адреса. При этом содержимое данной ячейки — 16-битное адресное слово $W(15:0)$ — рассматривается как исполнительный адрес, если в его головном бите содержится нуль, а если $W(15) = 1$, то 15 младших битов $W(14:0)$ являются новым косвенным адресом, по которому находится следующее адресное слово и т. д. Другими словами, головной бит адресного слова $W(15)$, как и головной бит адресной команды $K(15)$, указывает вид адреса л/к — прямой/косвенный, благодаря чему осуществляется многократная косвенная адресация.

Рассмотренная процедура получения исполнительного адреса описывается следующим алгоритмом.

Алгоритм вычисления исполнительного адреса EA на машинах Hewlett-Packard.

```
if  $K(10) = 0$  then  $EA := K(9:0)$ 
else  $EA := PC(14:0)K(9:0)$ ;
if  $K(15) = 1$  then
begin
L:    $W := m(EA)$ ;  $EA := W(14:0)$ ;
if  $W(15) = 1$  then go to L
end
```

Прямо адресуемая часть главной памяти (нулевая и текущая страницы) составляет в машинах

Hewlett-Packard 2KW. Обращение к недоступным прямо в данный момент страницам осуществляется с помощью косвенной адресации через ячейки нулевой и текущей страниц.

В большинстве адресных команд исполнительный адрес является адресом ячейки, из которой производится выборка операнда, используемого затем в операции с одним из аккумуляторов. Выполнение команд сложения ADA и ADB связано с возможностью возникновения ненулевого переноса из старшего бита аккумулятора, а также арифметического переполнения, т. е. получения результата, выходящего за пределы диапазона представимых в аккумуляторе чисел (от -32768 до $+32767$). При возникновении ненулевого переноса из старшего бита происходит установка в состояние 1 бита-удлинителя Eх, а в случае переполнения устанавливается 1 в бит переполнения Ov. Другие адресные команды не касаются битов Eх и Ov.

Команды сравнения CPA, CPB (Compare to A, to B) предписывают перескок через команду, расположенную непосредственно за выполняемой, при условии неравенства друг другу значений операндов, считанного из ячейки памяти и находящегося в аккумуляторе. Содержимое как ячейки, так и аккумулятора сохраняется неизменным.

Команда ISZ (Increment and Skip if Zero) осуществляет прибавление 1 к содержимому указанной в ней ячейки памяти, и если в результате этого содержимое станет нулем, то производит перескок через следующую команду. Поскольку аккумуляторы А и В обладают, подобно ячейкам памяти, адресами, команда ISZ может использоваться также применительно к ним, причем, в отличие от регистровых команд приращения содержимого аккумуляторов, она не касается битов Eх и Ov.

Команды безусловного перехода (JMP) и перехода на подпрограмму (JSB) идентичны по содержанию соответствующим командам PDP-8. В команде JMP исполнительный адрес EA указывает ячейку, из которой производится выборка следующей команды. В команде JSB исполнительный адрес является адресом ячейки, в которой сохраняется текущее значение программного счетчика PC в качестве адреса возврата

с подпрограммы, а начальная команда подпрограммы находится в следующей ячейке, и принимаемый в РС ее адрес получается прибавлением к исполнительному адресу ЕА единицы. Возврат по окончании подпрограммы производится командой перехода JMP, содержащей адрес ячейки, в которую помещен адрес возврата в качестве косвенного.

В отличие от используемой в PDP-8 команды STA, при выполнении которой запись содержимого аккумулятора в память сопровождается очисткой аккумулятора, команды STA и STB в машинах Hewlett-Packard осуществляют запоминание содержимого соответственно аккумуляторов А и В, не изменяя состояния последних. Это, пожалуй, наиболее заметное отличие описываемых команд от соответствующих команд PDP-8.

Другое существенное расхождение имеется в правилах, регламентирующих функционирование бита-удлинителя Ех, которые, как нетрудно заметить, не вполне совпадают с правилами функционирования бита связи Lk в PDP-8.

На языке ассемблера адресная команда представляется в виде строки, в которой содержатся в общем случае четыре расположенных друг за другом слева направо элемента: метка (не обязательно), мнемокод операции, обозначение операнда, комментарий (не обязательно). Эти элементы отделяются друг от друга одним или несколькими пробелами.

Метка может содержать буквы, цифры и точку, в сумме не более 5 знаков, и не должна начинаться цифрой.

Мнемокод операции всегда состоит из трех букв.

Обозначение операнда обычно записывается в виде так называемого адресного выражения, за которым может следовать отделенный от него запятой символ косвенной адресации I. Адресное выражение — это, как правило, либо метка, либо метка \pm целое число, либо метка \pm метка, либо «звездочка» \pm целое число, но допустимы и более сложные конструкции. Примеры:

STA BUFF + 3

LDB DATA — V4

ISZ TABLE, I

JMP * — 4

В командах сложения, сравнения, засылки в аккумулятор, а также AND, XOR и IOR имеется возможность непосредственно задавать значение операнда («литерал»).

Например, команда

LDA = D2769

засылает в аккумулятор А целое число 2769, а команда

ADB = B20

прибавляет к содержимому аккумулятора В число 20₈. Символы = D и = B означают соответственно: «целое десятичное», «целое восьмеричное». Имеются также другие типы литералов: = F — «десятичное с плавающей запятой», = A — «символы кода ASCII», = L — «выражение».

§ 3. Регистровые команды

Регистровые команды миникомпьютеров Hewlett-Packard — это команды-микропрограммы, аналогичные командам-микропрограммам PDP-8.

Отличие состоит в том, что аккумуляторов теперь два и необходимо указывать, к какому из них относится команда (система построена так, что микрооперации, включенные в команду, не могут относиться к разным аккумуляторам). Кроме того, с увеличением длины слова до 16 битов задаваемые командами микропрограммы были усложнены.

Регистровые команды опознаются по нулевому значению четырех старших битов $K(15:12) = 0$ и разделяются в зависимости от значения десятого бита на две группы:

$$K(10) = \begin{cases} 0 - \text{группа сдвига и вращения (циклического сдвига),} \\ 1 - \text{группа преобразования и перескоков.} \end{cases}$$

Другими словами, имеются две команды-микропрограммы: первая предназначена главным образом для осуществления сдвига в аккумуляторах, иногда с участием бита-удлинителя E_x , вторая — для преобразования и тестирования содержимого аккумуляторов и бита E_x .

На рис. 7.3 бит K(11) указывает аккумулятор, к которому относятся предписанные командой действия:

$$K(11) = \begin{cases} 0 - \text{аккумулятор А,} \\ 1 - \text{аккумулятор В.} \end{cases}$$

Тройка битов K(8:6), обозначенная на рисунке SH1, и тройка K(2:0), обозначенная SH2, представляют

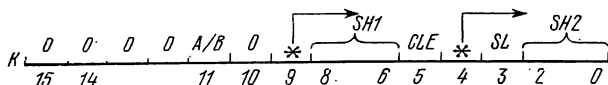


Рис. 7.3. Формат команды-микропрограммы сдвига и вращения.

собой коды микроопераций сдвига. Эти микрооперации (их всего 8) описаны в табл. 7.2. В процессе выполнения команды-микропрограммы могут быть произведены две микрооперации сдвига над аккумулятором, указанным битом K(11). Микрооперация SH1

Таблица 7.2

Микрокоманды сдвига в машинах серии HP 2100

| K(8:6), K(2:0) | Мнемокод | Операция | Описание |
|-------------------|------------|--|--|
| 0 | ALS BLS | арифметический сдвиг влево | Ac(14): = 0; Ac(14 : :0): = Ac(13 : :0)Ac(14); |
| 1 | ARS BRS | арифметический сдвиг вправо | Ac(14:0): = Ac(15:1); |
| 2 | RAL RBL | циклический сдвиг влево | Ac: = Ac(14 : :0)Ac(15); |
| 3 | RAR RBR | циклический сдвиг вправо | Ac: = Ac(0)Ac(15 : :1); |
| 4 | ALR BLR | сдвиг влево с очист- кой головного бита | Ac(15:14): = 0; Ac: = Ac(14 . :0)Ac(15); |
| 5 | ERA ERB | сдвиг в кольце ExAc вправо | AcEx: = ExAc; |
| 6 | ELA ELB | сдвиг в кольце ExAc влево | ExAc: = AcEx; |
| 7 | ALF BLF | циклический сдвиг на 4 места влево | Ac: = Ac(11 : :0)Ac(15:12); |

производится при условии $K(9) = 1$, а микрооперация SH2 — при условии $K(4) = 1$. Иначе говоря, биты $K(9)$ и $K(4)$, помеченные на рис. 7.3 звездочкой, выполняют функцию выключателей операций соответственно SH1 и SH2: если выключатель в состоянии 0, операция не выполняется (выключена).

Бит $K(5)$ является выключателем микрооперации CLE (Clear Ex — «очистить Ex»), т. е. она выполняется при условии $K(5) = 1$:

if $K(5) = 1$ **then** $Ex := 0$.

Наконец, биту $K(3)$, обозначенному на рисунке буквами SL, соответствует микрооперация SLA (или SLB) — Skip if LSB of A (LSB of B) = 0 — перескока по нулевому состоянию младшего бита аккумулятора A или B в зависимости от значения $K(11)$:

if $K(3) = 1 \wedge (K(11) = 0 \wedge A(0) = 0 \vee K(11) = 1 \wedge B(0) = 0)$ **then** $PC := PC + 1$.

Микрооперации команды-микропрограммы сдвига выполняются в той последовательности, в которой расположены сопоставленные им биты и тройки битов в командном слове, читаемом слева направо, т. е.

SH1, CLE, SL, SH2.

Запись команды-микропрограммы сдвига на языке ассемблера представляет собой последовательность мнемокодов конкретных микроопераций, разделенных запятыми. Например: RAL, CLE, SLA, RAR — «вращать A влево; очистить Ex; если $A(0) = 0$, то перескок; вращать A вправо».

Ассемблер преобразует эту запись в командное слово, включая в него коды указанных микроопераций. Разумеется, нельзя смешивать в одной команде операции, выполняемые над аккумуляторами A и B. Приведенный пример команды, включающей максимально возможное число микроопераций, не является типичным — обычно возможности микропрограммы используются не полностью. В частности, команда может быть не связана с операциями сдвига, например: CLE — «очистить Ex».

Команда сдвига, в которой $K(9) = K(5:3) = 0$, не включает никаких операций и равносильна коман-

де $K(15:0) = 0$, которая также входит в группу команд сдвига и обозначается NOP — No Operation.

Функционирование процессора при выполнении команды-микропрограммы сдвига описывает следующий алгоритм:

```

procedure SRG (Ac); binary array Ac (15:0);
begin
  procedure SH (c); binary array c (2:0);
  begin
    if c = 0 then begin Ac (14): = 0;
    Ac (14:0): = Ac (13:0) Ac (14) end else
    if c = 1 then Ac (14:0): = Ac (15:1) else
    if c = 2 then Ac: = Ac (14:0) Ac (15) else
    if c = 3 then Ac: = Ac (0) Ac (15:1) else
    if c = 4 then begin Ac (15:14): = 0;
    Ac: = Ac (14:0) Ac (15) end else
    if c = 5 then AcEx: = ExAc else
    if c = 6 then ExAc: = AcEx else
    if c = 7 then Ac: = Ac (11:0) Ac (15:12)
    end SH;
    if K (9) = 1 then SH (K (8:6));
    if K (5) = 1 then Ex: = 0;
    if K (3) = 1  $\wedge$  Ac (0) = 0 then PC: = PC + 1;
    if K (4) = 1 then SH (K (2:0));
    end SRG;
    if K (11) = 0 then SRG (A) else SRG (B);
  
```

Вторая команда-микропрограмма позволяет задавать преобразования и перескоки. Как и в команде сдвигов, в ней $K(15:12) = 0$, однако десятый бит содержит не 0, а 1 (рис. 7.4).

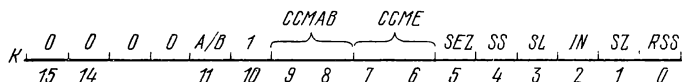


Рис. 7.4. Формат команды-микропрограммы преобразования и перескоков.

Поле $K(9:8)$ является указателем операции, выполняемой над аккумулятором А, если $K(11) = 0$, или В,

если $K(11) = 1$:

$$K(9:8) = \begin{cases} 0 - \text{ничего не делать,} \\ 1 - \text{CLA или CLB (Clear - «очистить} \\ \quad \text{A (или B))}, \\ 2 - \text{CMA или CMB (Complement -} \\ \quad \text{«инвертировать»),} \\ 3 - \text{CCA или CCB (Clear and} \\ \quad \text{Complement - «очистить и} \\ \quad \text{инвертировать»).} \end{cases}$$

Операция очистки устанавливает в указанном аккумуляторе нуль: $Ac := 0$. Инверсия означает замену нулей единицами, а единиц нулями: $Ac := \overline{Ac}$. Очистка и инверсия; $Ac := 0$; $Ac := \overline{Ac}$; устанавливает все биты аккумулятора в состояние 1.

Поле $K(7:6)$ задает аналогичные операции над битом-удлинителем Ex :

$$K(7:6) = \begin{cases} 0 - \text{ничего не делать,} \\ 1 - \text{CLE (Clear Ex - «очистить Ex»),} \\ 2 - \text{CME (Complement Ex - «инвертировать} \\ \quad \text{Ex»),} \\ 3 - \text{CCE (Clear and Complement Ex -} \\ \quad \text{«установить Ex = 1»).} \end{cases}$$

Бит $K(2)$ является выключателем операции приращения содержимого аккумуляторов INA или INB (Increment A или Increment B):

if $K(2) = 1$ **then begin** **if** $K(11) = 0$
then INA **else** INB **end.**

Биты $K(5)$, $K(4)$, $K(3)$ и $K(1)$ представляют собой выключатели перескоков соответственно по условиям:

SEZ — Skip if Ex is Zero («перескочить, если $Ex = 0$ »),

$SSA(SSB)$ — Skip if Sign bit of A (of B) is zero («перескочить, если $Ac(15) = 0$, т. е. если знак Ac не минус»),

$SLA(SLB)$ — Skip if Least significant bit is zero («перескочить, если $Ac(0) = 0$, т. е. если в Ac четное число»).

$SZA(SZB)$ — Skip if A (B) is zero («перескочить, если $Ac = 0$ »).

Наконец, бит $K(0)$, находясь в состоянии 1, инвертирует сформулированные только что условия

перескоков. Мнемокод микрооперации инвертирования условий — RSS (Reverse Skip Sense «инвертировать смысл перескока»). Если команда не содержит RSS, то $K(0) = 0$ и условие перескока в целом выражается формулой

$$K(5) = 1 \wedge E_x = 0 \vee K(4) = 1 \wedge A_c(15) = 0 \vee \\ \vee K(3) = 1 \wedge A_c(0) = 0 \vee K(1) = 1 \wedge A_c = 0.$$

Если же команда в записи на ассемблере содержит мнемокод RSS, то данное условие заменяется его отрицанием (инверсией):

$$(K(5) = 0 \vee E_x = 1) \wedge (K(4) = 0 \vee A_c(15) = \\ = 1) \wedge (K(3) = 0 \vee A_c(0) = 1) \wedge (K(1) = \\ = 0 \vee A_c \neq 0).$$

Команда, в которой не содержится ни один мнемокод условия перескока (т. е. в которой $K(5) = K(4) = K(3) = K(1) = 0$), но содержится RSS, как видно из этой формулы, производит безусловный перескок.

Последовательность выполнения микроопераций в команде-микропрограмме преобразования и перескоков такова:

$$\begin{array}{c} L \\ CMA(B), SEZ, CME, SSA(B), SLA(B), INA(B), \\ C \\ C \\ SZ(A), RSS, \end{array}$$

т. е. сначала производится (если оно задано) преобразование содержимого A (или B), затем тестирование E_x , затем преобразование E_x , затем тестирование A(15) или B(15) и A(0) или B(0), затем приращение A (или B) и проверка условия $A = 0$ (или $B = 0$) и, наконец, с учетом наличия или отсутствия RSS выясняется, надо ли производить перескок. Мнемокоды необходимых микроопераций размещаются в записи команды на языке ассемблера в порядке выполнения этих микроопераций.

Последовательность операций, предписываемая рассмотренной командой-микропрограммой, в общем случае выражается следующим алгоритмом.

Алгоритм реализации команды-микропрограммы преобразования и перескоков в миникомпьютерах серии HP 2100.

```

procedure ASG (Ac); binary array Ac(15:0);
begin binary SE, SS, SL, SZ, Cr;
if K(8) = 1 then Ac := 0;
if K(9) = 1 then Ac :=  $\overline{\text{Ac}}$ ;
if K(5) = 1  $\wedge$  Ex = 0 then SE := 1 else SE := 0;
if K(6) = 1 then Ex := 0;
if K(7) = 1 then Ex :=  $\overline{\text{Ex}}$ ;
if K(4) = 1  $\wedge$  Ac(15) = 0 then SS := 1 else SS := 0;
if K(3) = 1  $\wedge$  Ac(0) = 0 then SL := 1 else SL := 0;
if K(2) = 1 then
  begin CrAc := Ac + 1; if Cr = 1 then Ex := 1;
  if OVERFL(Ac) then Ov := 1
  end;
  if K(1) = 1  $\wedge$  Ac = 0 then SZ := 1 else SZ := 0;
  if (K(0) = 0  $\wedge$  (SE = 1  $\vee$  SS = 1  $\vee$  SL = 1  $\vee$  SZ = 1))  $\vee$ 
    (K(0) = 1  $\wedge$  SE = 0  $\wedge$  SS = 0  $\wedge$  SL = 0  $\wedge$  SZ = 0) then PC := PC + 1
  end ASG;
if K(11) = 0 then ASG(A) else ASG(B);

```

§ 4. Команды ввода/вывода и система прерывания

Командное слово, в котором $K(15:12) = 10_8$ и $K(10) = 1$, интерпретируется как команда ввода/вывода (рис. 7.5).

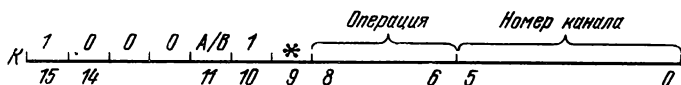


Рис. 7.5. Формат команды ввода/вывода.

Поле $K(5:0)$, условно обозначенное на рисунке словами «номер канала», принимает 64 значения, сопоставленные подключенным к машине устройствам ввода/вывода и некоторым другим объектам, управляемым командами ввода/вывода.

Поле $K(8:6)$ указывает операцию, выполняемую с устройством, номер которого задан полем $K(5:0)$.

Бит К(9) в состоянии 1 предписывает сбросить флажок готовности устройства по окончании выполняемой операции.

Бит К(11) осуществляет свою стандартную функцию, указывая участвующий в операции аккумулятор: К(11) = 0 — аккумулятор А, К(11) = 1 — аккумулятор В. Если операция не использует аккумулятор, состояние бита К(11) может быть произвольным. Исключения составляют операции STC и CLC (см. табл. 7.3).

Для процессора каждое периферийное устройство представлено своим номером («номером канала») и тремя регистрами: регистром буферирования данных DB, длина которого зависит от типа устройства и может достигать 16 битов, и двумя одноканальными регистрами — флажком готовности Fg и битом управления Sb.

Из 64 значений «номера канала» первые восемь (с 0 по 7) зарезервированы для специальных целей:

0 — используется в командах включения и выключения системы прерывания,

1 — используется в командах передачи данных между регистром панели управления и аккумуляторами, а также в командах изменения и тестирования состояния бита переполнения Ov.

2, 3 и 6, 7 — заняты в каналах прямого доступа к памяти (DMA): 2, 3 — запуск устройств, а 6, 7 — передача данных,

4 — в системе прерывания при нарушении электропитания,

5 — в системе защиты памяти и контроля по четности.

Следующие 56 номеров предназначены для обозначения подключенных к машине устройств ввода/вывода и периферийной памяти. Номер, присвоенный устройству, определяет его приоритет в системе прерывания, а именно: чем меньше номер, тем выше приоритет. Присвоенный устройству номер является функцией местоположения интерфейсной карты этого устройства, т. е. определяется тем, в какое из гнезд для включения интерфейсных карт она вставлена.

Устройство сигнализирует об окончании производившейся им операции поднятием флажка готовности: при вводе «поднятие флажка» говорит, что в буферном регистре готова для передачи процессору очередная

порция данных, а при выводе, что этот регистр готов принять данные от процессора. Флажок остается в состоянии 1, пока не будет сброшен командой CLF (Clear Flag) или другой командой ввода/вывода, в которой $K(9) = 1$. Имеются также команды поднятия флажка — STF (Set Flag) и тестирования его состояния — SFC (Skip if Flag Clear — «перескочить, если флажок сброшен») и SFS (Skip if Flag Set — «перескочить, если флажок поднят»).

Бит управления (Control bit) устанавливается в состояние 1 командой STC (Set Control) и сбрасывается в состояние 0 командой CLC (Clear Control). Установка бита управления в состояние 1 запускает устройство в работу и предоставляет ему право запрашивать прерывание: если бит управления находится в состоянии 1, то поднятие устройством флажка автоматически вызывает запрос прерывания. Прерывание происходит по окончании текущего такта процессора или по завершении выполняемой команды, если система прерывания не выключена и если нет неудовлетворенных запросов от устройств, обладающих более высоким приоритетом.

Номер устройства, получившего разрешение на прерывание, используется как адрес ячейки главной памяти, содержащей команду перехода на подпрограмму обслуживания данного устройства. Адрес возврата запоминается в теле этой подпрограммы. Таким образом, опрос устройств, выполняемый программой обслуживания прерывания на машинах PDP-8 с целью выявить прерывающее устройство, производить не надо — определение подлежащего обслуживанию устройства производит аппаратура в соответствии с зафиксированной в ней иерархией приоритетов. Это позволяет обеспечить более быструю реакцию процессора на прерывания и увеличить пропускную способность системы ввода/вывода.

Если в процессе обслуживания некоторого устройства ввода/вывода приходит запрос от устройства, обладающего более высоким приоритетом, и система прерывания не выключена, то выполняемая обслуживающая программа будет прервана и отложена до завершения отеснившей ее более срочной обслуживающей программы. Наивысшим приоритетом в системе прерывания обладает использующая «номер канала» 4

сигнализация о неисправности электропитания (номера 0 — 3 связаны с подсистемами, не вызывающими прерываний). Затем идет защита памяти, а за ней два канала прямого доступа. Распределение последующих уровней приоритета (номера с 8-го по 63-й) определяется схемой сопряжения периферийных устройств с процессором, в частности, расстановкой интерфейсных карт.

Команды ввода/вывода, их коды, мнемоника и описания соответствующих операций приведены в табл. 7.3. Подразумевается, что флажок готовности Fg, буферный регистр данных DB и бит управления Cb принадлежат каналу, номер которого содержит поле K(5:0) командного слова.

В командах STC и CLC, манипулирующих битом управления, в необычной для него роли выступает бит K(11): если $K(11) = 0$, то производится $Cb := 1$, а если $K(11) = 1$, то $Cb := 0$.

Особое значение имеют команды с номерами канала 00: STF производит включение системы прерывания, CLF — выключение, а CLC — сброс битов управления во всех каналах в состояние 0, причем последнее можно сочетать с выключением системы прерывания, употребив команду CLF, в которой $K(9) = 1$.

Как уже было сказано, в число команд ввода/вывода включены не относящиеся к вводу/выводу команды установки и тестирования состояния бита переполнения Ov: STO, CLO, SOC и SOS. Они кодируются как соответствующие команды установки и тестирования флажка готовности в канале с номером 1, но так как этот номер связан с битом переполнения, то заданные операции производятся не над флажком, а над битом Ov процессора. Команды LIA, LIB, MIA, MIB с номером канала 1 производят считывание кода, набранного ключами на панели управления, а команды OTA и OTB — передачу содержимого аккумуляторов в индикаторный регистр панели управления (последнее только на модели HP2114A).

Передача данных между 16-битными аккумуляторами A, B и буферными регистрами DB, длина которых в зависимости от типа устройства может составлять от одного до 16 битов, производится из или в младшие биты аккумуляторов: биту DB(0)

Команды ввода/вывода HP 2100 (K(15:12) = 10₈, K(10) = 1)

| K(8:6) | K(9)/ K(11) | Мнемокод | Команда | Описание операции |
|---------------------------------|----------------|----------|---|--|
| 0 | | HLT | остановить процессор | STOP; if K(9) = 1 then |
| 1 | 0/ | STF | Set Flag — поднять флажок | Fg: = 0; Fg: = 1; |
| 1 | 1/ | CLF | Clear Flag — сбросить флажок | Fg: = 0; |
| 2 | 0/ | SFC | Skip if Flag Clear — перескочить, если флажок сброшен | if Fg = 0 then PC: = PC + 1; |
| 3 | 0/ | SFS | Skip if Flag Set — перескочить, если флажок поднят | if Fg = 1 then PC: = PC + 1; |
| 4 | /0 | MIA | Merge Input into A — наложить копию бу- фера на A | A: = A ∨ DB; if K(9) = 1 then Fg: = 0; |
| 4 | /1 | MIB | Merge Input into B — наложить копию бу- фера на B | B: = B ∨ DB; if K(9) = 1 then Fg: = 0; |
| 5 | /0 | LIA | Load Input into A — скопировать буфер в A | A: = DB; if K(9) = 1 then Fg: = 0; |
| 5 | /1 | LIB | Load Input into B — скопировать буфер в B | B: = DB; if K(9) = 1 then Fg: = 0; |
| 6 | /0 | OTA | Output from A — ско- пировать A в буфер | DB: = A; if K(9) = 1 then lg: = 0; |
| 6 | /1 | OTB | Output from B — ско- пировать B в буфер | DB: = B; if K(9) = 1 then Fg: = 0; |
| 7 | 0 | STC | Set Control bit — уста- новить бит управле- ния в 1 | Cb: = 1; if K(9) = 1 then Fg: = 0; |
| 7 | /1 | CLC | Clear Control bit — сбросить бит управле- ния | Cb: = 0; if K(9) = 1 then Fg: = 0; |
| Операции над битом переполнения | | | | |
| 1 | 0 | STO | Set Overflow — устано- вить Ov = 1 | Ov: = 1; |

| К (8 : 6) | К (9)/ К (11) | Мнемокод | Команда | Описание операции |
|-----------|------------------|----------|---|---------------------------------|
| 1 | 1/ | CLO | Clear Overflow – очистить Ov | Ov := 0; |
| 2 | | SOC | Skip if Overflow Clear – перескочить, если Ov = 0 | if Ov = 0 then PC := PC + 1; |
| 3 | | SOS | Skip if Overflow Set – перескочить, если Ov = 1 | if Ov = 1 then PC := PC + 1; |

сопоставляется бит A(0) (или B(0)), биту DB(1) – бит A(1) (или B(1)) и т. д.

Процедура передачи данных между аккумуляторами и буферными регистрами периферийных устройств реализуется с использованием прерывания. В случае ввода процессор командой STC запускает устройство в работу, устанавливая $Cb = 1$, $Fg = 0$. Устройство переносит данные с внешнего носителя в буфер и поднимает флажок. Предполагается, что система прерывания не выключена, поэтому поднятие флажка вызывает прерывание выполняемой процессором программы и переход на программу обслуживания устройства, которая осуществляет пересылку данных из буфера в аккумулятор.

В случае вывода процессор командой ОТА или ОТВ производит пересылку выводимой порции данных из аккумулятора в буфер и командой STC запускает устройство в работу. Устройство, приняв данные из буфера, поднимает флажок готовности. Поднятие флажка вызывает прерывание, в процессе которого производится переход на обслуживающую данное устройство программу. Эта программа загружает буфер новой порцией выводимых данных и повторно иницирует вывод, а если подлежащие выводу данные исчерпаны, то уведомляет главную программу о завершении вывода.

На языке ассемблера запись команды ввода/вывода состоит из мнемокода операции, номера канала в виде десятичного или восьмеричного числа, или символического имени, или выражения допускаемой ассемблером формы.

Например, запись

STC 9

означает «Set Control bit 9», т. е. в канале номер 9 установить бит управления в состояние 1, что вызовет передачу одной порции данных между буферным регистром и собственно устройством ввода/вывода, включенным в этот канал.

Если команда, помимо основной операции, предписывает сбросить флажок готовности, то в ее запись должна быть добавлена буква С, отделенная от номера канала запятой.

Например, команда «скопировать буфер канала 10₈ в аккумулятор А и сбросить флажок» записывается так:

LIA 10B, C

Буква В, присоединенная справа к номеру канала, указывает на то, что этот номер записан в восьмеричной системе счисления.

§ 5. Периферийное оборудование и программное оснащение миникомпьютеров Hewlett-Packard

Рассмотренная в предыдущих параграфах архитектура реализована в 1967—1968 гг. в миникомпьютерах HP 2114, 2115, 2116, а затем эмулировалась на микропрограммных миникомпьютерах HP 2100 А, 2105, 2108, разработанных в начале 70-х годов, и в расширенной версии на микропрограммных миникомпьютерах введенной в 1974 г. новой серии HP 21МХ. Миникомпьютеры серий 2100 и 21МХ используются в многотерминальных системах разделения времени серии HP 2000 и в системах для автоматических измерений и регулирования серии HP 9600, а миникомпьютеры 21МХ — также в системе для обработки в реальном масштабе времени HP 1000. Для укомплектования указанных систем, а также выпускаемой наряду с ними многоязычной мультипрограммной системы HP 3000 фирма Hewlett-Packard выпускает в достаточно полной номенклатуре необходимые периферийные устройства и аппаратуру для сопряжения этих устройств с процессором.

В числе устройств периферийной памяти имеются: магнитные диски с неподвижными головками HP 2766 (емкость 0,5 МВ, среднее время доступа 8,7 мс), диски-картриджи HP 7901 (2,45 МВ, 35 мс), HP 7900 (4,9 МВ, 33 мс, до 4 дисководов на один контроллер) и HP 7905 (15 МВ, 33 мс, до 8 дисководов на контроллер), пакетные диски HP 2883 (23,4 МВ, 33 мс) и HP 7920 (50 МВ, 33 мс, до 8 дисководов на контроллер), стандартная (IBM — совместимая) магнитная лента HP 7970 (800 и 1600 бит/дюйм), магнитные кассеты HP Mini-Cartridge емкостью 110 КВ.

Устройства перфолентного ввода/вывода представлены считывателем HP 2748 В (500 зн/с) и перфоратором HP 2895 В (75 зн/с).

Перфокартный ввод/вывод обеспечивается считывателем-перфоратором HP 2894А (считывание — до 200 карт/мин, перфорация — до 75 карт/мин) и считывателем перфокарт HP 2893А (600 карт/мин). Имеется также оптический считыватель карт, отмеченных карандашом. HP 7260А (до 300 карт/мин).

В качестве печатающих устройств имеются: строчный матричный принтер HP 2607А (200 строк/мин, 132 позиции в строке, 64-значный алфавит или 165 строк/мин, 132 позиции, 128-значный алфавит), строчный ленточный принтер HP 2613А (300 строк/мин, 136 позиций, 64 знака или 240 строк/мин, 136 позиций, 96 ASCII знаков), строчный ленточный принтер средней скорости HP 2617А (600 строк/мин, 136 позиций, 64 знака или 436 строк/мин, 136 позиций, 96 знаков), быстрый строчный принтер HP 2618А (1800 строк/мин, 132 позиции, 64 знака или 925 строк/мин, 132 позиции, 96 ASCII знаков), печатающие терминалы HP 2762 (120 зн/с, 120 позиций, алфавит содержит и прописные, и строчные буквы) и HP 2605А (10, 15, 30 зн/с). Помимо устройств текстовой печати, выпускаются построители графиков HP 7202А и 7203А.

Серия электронно-лучевых дисплейных с клавиатурой терминалов HP 2640 (24 строки по 80 позиций, сменный 128-значный алфавит, микропроцессорное управление, от 1 КВ до 8 КВ собственной памяти, максимальная скорость приема/передачи данных — 2400 бит/с) включает, в частности, «русский терминал» HP 2640С, алфавит которого состоит из двух наборов,

переключаемых специальной клавишей или соответствующим управляющим кодом, — 64-значного с прописными латинскими буквами и 128-значного с прописными и строчными буквами русского алфавита. Более «интеллектуальными» терминалами являются HP 2645A Display Station и HP 2641A APL Display Station, обладающие памятью 4—12 КВ и двумя встроенными кассетными лентопротяжками, что обеспечивает возможность автономного редактирования текстов. Имеется также ряд печатающих терминалов: HP 2749B и HP 2754B (10 зн/с), HP 2762A (30 зн/с, 75 позиций), HP 2762B (120 зн/с, 120 позиций) и др.

Аппаратура сопряжения с аналоговыми и цифровыми объектами в лабораторных и промышленных измерительных и регулирующих системах включает устройство аналогового ввода/вывода HP 2313B (до 528 дифференциальных аналоговых входов и до 44 аналоговых выходов, точность 12 битов, частоты сканирования при уровне сигнала 10 В — до 45 кгц, при уровне сигнала менее 800 мВ — до 8 кгц), интегрирующий цифровой вольтметр HP 2320A (разрешающая способность 1 мкВ на 0,1 В, число входных каналов — до 200, скорость обегания — до 14 каналов в секунду), интегрирующий цифровой вольтметр HP 2323A (разрешающая способность 1 мкВ на 0,1 В, число входных каналов — до 1000, скорость обегания — до 40 каналов в секунду), система цифрового ввода/вывода (разрешающая способность 12 битов, до 2880 цифровых входов/выходов), преобразователь цифра/аналог HP 12555B для вывода информации в графическом виде на экраны электронно-лучевых осциллографов и двухкоординатные построители графиков.

Аппаратура сопряжения с линиями связи включает асинхронный однолинейный адаптер к модемам Bell 103 или 203 HP 12587B, асинхронный 16-линейный мультиплексор HP 12920B для работы с модемами тех же типов или с местными терминалами, синхронный однолинейный к модему Bell 201A адаптер 12618A и ряд других устройств.

Характерным для Hewlett-Packard является выпуск главным образом готовых, укомплектованных мини-компьютерных систем, а не отдельных процессоров и периферийных устройств для использования в системах, создаваемых OEM-фирмами или самими

пользователями. Наиболее известными сериями систем, поставляемых Hewlett-Packard, являются HP 9600, HP 2000, HP 3000.

Серия 9600 включает специализированные одно-процессорные системы для научных измерений и управления экспериментом HP 9602A, 9603A, 9604A, а также систему для управления производственными процессами HP 9611A. Все эти системы оснащены аппаратурой аналогового и цифрового сопряжения с контролируемыми и управляемыми ими приборами и машинами. В качестве основного программного оснащения на них используются операционные системы реального времени RTE (Real-Time Executive).

Имеются четыре системы RTE: RTE-B, RTE-C, RTE II и RTE III.

Система RTE-B (Real-Time BASIC) обеспечивает возможность разговорного программирования на языке BASIC и совместное выполнение до 16 различных работ, инициируемых по сигналам датчика времени или в зависимости от внешних событий. Система рассчитана на конфигурацию, не содержащую периферийной памяти, но включающую не менее 16 КВ главной памяти.

Система RTE-C также функционирует без периферийной памяти при наличии минимум 8 КВ главной памяти. Она характеризуется более быстрой реакцией на запросы извне и при наличии достаточной памяти может работать с большим числом заданий, чем RTE-B. В качестве языков программирования в ней используются фортран, алгол и язык ассемблера. Система RTE-C совместима с более мощными системами RTE II и RTE III.

Система RTE II представляет собой дисковую операционную систему реального времени с пакетной обработкой на заднем плане и с возможностью одновременного обслуживания нескольких терминалов пользователей.

Как и в RTE-C, языками программирования являются фортран, алгол и язык ассемблера. В качестве необязательного дополнения система может включать автономный монитор пакетной обработки Batch Spool Monitor, обеспечивающий автоматическое управление файлами на диске и более независимую организацию работ заднего плана.

Наиболее развитой из операционных систем реального времени для миникомпьютеров Hewlett-Packard является система RTE III, работающая с главной памятью от 32 KW до 256 KW и памятью на дисках, которая может включать до 8 дисков-картриджей или пакетных дисководов. Она обеспечивает совместное выполнение до 64 программ пользователей, предоставляя каждой из них до 16 KW памяти. Языками программирования являются фортран, алгол, HP-ассемблер. Кроме того, система может включать много-терминальную подсистему разговорного программирования на языке basic.

Кроме перечисленных систем RTE, имеется BCS (Basic Control System), которая может работать при наличии 4 KW главной памяти. Она также обеспечивает быструю реакцию на прерывания и буферирование ввода/вывода, но не обладает способностью планирования работ во времени. Языки программирования — фортран, алгол и язык ассемблера.

Рассмотренные операционные системы способны взаимодействовать друг с другом, образуя распределенные системы Distributed Systems. Распределенная система включает узловые (Central) комплексы аппаратуры, работающие под управлением RTE II или RTE III, с каждым из которых связано несколько спутников, оснащенных системами RTE-B, RTE-C или BCS. Спутниками служат перечисленные выше системы серии HP9600 без периферийной памяти, а в качестве узловой используется более мощная, обладающая дисками и производительными устройствами ввода/вывода система HP9700A Distributed Systems Central.

Операционные системы спутников снабжены связанными программами SCE (Satellite Communication Executive), а сама узловая система, соответственно, CCE (Central Communication Executive), обеспечивающей взаимодействие с спутниками, с другими узловыми системами и с вышестоящей системой обработки данных, в качестве которой могут использоваться IBM 360/370 или HP 3000. Связь при расстояниях до 3 км может осуществляться по проводным линиям без модемов со скоростями передачи более 30 000 бит/с, а с использованием модемов практически на любые расстояния в пределах телефонной сети.

Достоинства распределенной системы заключаются как в оперативности сбора и обработки данных, поступающих от спутников в узловые системы и далее в главный центр обработки и управления, так и в оптимальности использования ресурсов: спутники имеют доступ к файлам на дисках узловой системы и пользуются ее устройствами ввода/вывода, а также услугами ее процессора. Узловая система, в частности, осуществляет генерацию операционных систем спутников и их загрузку с помощью имеющегося у нее кросс-загрузчика.

Серия HP 2000 систем разделения времени и удаленного ввода работ в большие вычислительные системы базируется на двухпроцессорной конфигурации с периферийной памятью на дисках и на магнитных лентах. Один процессор выполняет общесистемные функции, другой — связные. Оба являются миникомпьютерами серии 2100 или 21MX с памятью для системного — 32 KW, а для связного — 32 KW в системах, обслуживающих до 32 терминалов, и 8 KW или 16 KW в системах, обслуживающих до 16 терминалов. Для сопряжения связного процессора с терминалами используется один или два 16-канальных мультиплексора.

Основной операционной системой для серии HP 2000 является система разделения времени с языком программирования BASIC, но используются и другие операционные системы, работающие с языками фортран, алгол, HP-ассемблер, — дисковые системы пакетной обработки DOS, RTE и BCS.

Наиболее мощными системами Hewlett-Packard являются системы серии HP 3000 (HP 3000-I, 3000CX, 3000-II), основанные на высокопроизводительных микропрограммных процессорах с возможностью расширения главной памяти до 512 KB и памяти на дисках до 400 MB. Используемая в них многоязыковая, мультипрограммная с виртуальной памятью операционная система MPE (Multiprogramming Executive) обеспечивает возможность совмещения режима реального времени с пакетной обработкой и обслуживанием до 64 терминалов. Языки программирования: basic (интерпретатор и компилятор), фортран, RPG, кобол, APL и машинно-ориентированный язык высокого уровня SPL (System Programming Language). Система ос-

нащена широким набором специализированных пакетов программ для деловых, научных и учебных применений. В частности, имеются пакеты: IMAGE и QUERY — управления базой данных, SORT — сортировки и упорядочения файлов, STAR — статистического анализа, EDIT — редактирования текстов и управления файлами и др. В качестве дополнения к MPE имеются связанные программы Distributed System Software, обеспечивающие взаимодействие систем HP 3000 друг с другом, а также с системами реального времени HP 1000 и системами редактирования и ввода данных HP 2026 в сетях распределенной обработки.

ГЛАВА 8

ЧЕТЫРЕХАККУМУЛЯТОРНЫЕ МИНИКОМПЬЮТЕРЫ DG NOVA

Четырехаккумуляторная архитектура миникомпьютеров семейства NOVA, выпускаемых корпорацией Data General (DG), представляет собой превосходный образец последовательного и в ряде отношений завершенного развития тех принципов, на которых базируется архитектура PDP-8. Операции преобразования данных в процессоре NOVA полностью отделены от обращений к главной памяти, несмотря на то, что в результате увеличения длины слова с 12 до 16 битов условия размещения кода операции и адреса в слове-команде стали менее трудными. Этот шаг оправдан, в частности, следующими двумя обстоятельствами. Сочетание в команде-микропрограмме основной операции (задаваемой в PDP-8 адресной командой) с операциями циклического сдвига и тестирования результата на выполнение условий перескока существенно повысило эффективность микропрограммирования. Использование четырех аккумуляторов с единым битом связи обеспечило возможность реализации многих часто встречающихся процедур при минимуме обращений к главной памяти.

Эффективность архитектуры семейства NOVA проявляется в компактности машинных программ, в экономических преимуществах машин этого семейства на широком спектре применений. Важными свидетельствами совершенства этой архитектуры являются ее долголетие и стабильность.

Корпорация Data General основана в 1968 г. группой инженеров, вышедших из DEC и из Fairchild Semiconductor, во главе с Edson de Castro —

одним из ведущих разработчиков PDP-8. Поставка первых моделей семейства NOVA началась в 1969 г. В настоящее время это семейство включает не менее дюжины моделей единой архитектуры, различающихся циклом и максимальной емкостью памяти, используемой технологией и конструктивным оформлением.

Представителями этого семейства являются: NOVA, SUPERNOVA, SUPERNOVA SC, NOVA 1200, NOVA 800, NOVA 2, NOVA 3, MICRONOVA. Архитектура семейства NOVA используется в ряде машин, выпускаемых другими фирмами, например в миникомпьютерах фирмы Rolm, рассчитанных на эксплуатацию в суровых условиях (по военным нормам).

С 1975 г. параллельно с производством машин семейства NOVA DG начала выпуск миникомпьютеров повышенной мощности — Eclipse, возможности которых при сохранении в качестве основы четырехаккумуляторной архитектуры машин NOVA обогащены применением микропрограммирования, расширением набора команд, введением быстрого процессора с плавающей запятой.

§ 1. Виды команд и регистры процессора

Длина слова в машинах семейства NOVA — 16 битов. Как элемент программы, слово может быть командой или (в случае косвенной адресации) адресом.

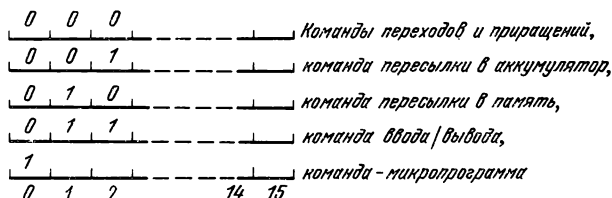


Рис. 8.1. Виды команд в миникомпьютерах NOVA.

Существуют четыре вида (формата) слов команд К(0:15) (рис. 8.1):

- 1) команды переходов и приращений,
- 2) команды пересылок,
- 3) команды ввода/вывода,

4) команды-микропрограммы, оперирующие с регистрами процессора.

Первые три вида команд имеют $K(0) = 0$, четвертый вид — $K(0) = 1$. Дальнейшее различие происходит в зависимости от $K(1:2)$:

в командах переходов и приращений $K(1:2) = 0$,
в командах пересылок

$$K(1:2) = \begin{cases} 1 - \text{пересылка в аккумулятор (Load Ac),} \\ 2 - \text{пересылка в память (Store Ac),} \end{cases}$$

в командах ввода/вывода $K(1:2) = 3$.

Только команды пересылок, переходов и приращений затрагивают главную память, т.е. адресуются к ее ячейкам, являются адресными. Команды ввода/вывода касаются регистров процессора и периферийных устройств, а команды-микропрограммы — только регистров процессора.

Доступными программе регистрами процессора являются: программный счетчик $PC(1:15)$, четыре 16-битных аккумулятора: $Ac0(0:15)$, $Ac1(0:15)$, $Ac2(0:15)$, $Ac3(0:15)$ и однобитный регистр $Cr(1:1)$ — Carry flag — аналог регистра Link в PDP-8. Регистр Cr принимает перенос из старших (нулевых) битов аккумуляторов и служит соединителем нулевого бита с хвостовым при выполнении циклического сдвига влево или вправо. Значение Cr можно устанавливать, изменять и тестировать.

§ 2. Система адресации и адресные команды

В командах, касающихся главной памяти, младшие 11 битов $K(5:15)$ составляют адресную часть, которая имеет следующую структуру (рис. 8.2).

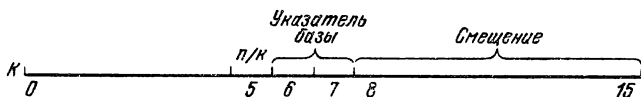


Рис. 8.2. Структура адресной части в командах миникомпьютеров NOVA.

Три первых бита $K(5:7)$ указывают вид адресации:

$$K(5) = \begin{cases} 0 - \text{прямая,} \\ 1 - \text{косвенная,} \end{cases}$$

$$K(6:7) = \begin{cases} 0 - \text{абсолютная,} \\ 1 - \text{относительно PC,} \\ 2 - \text{относительно As2,} \\ 3 - \text{относительно As3.} \end{cases}$$

Остальные 8 битов — $K(8:15)$ — составляют «смещение».

В случае $K(6:7) = 0$, т. е. при абсолютной адресации, код $K(8:15)$ интерпретируется как двоичное число без знака, являющееся адресом ячейки главной памяти в диапазоне $0-377_8$. В случае $K(6:7) \neq 0$ код $K(8:15)$ рассматривается как смещение — двоичное число со знаком, представленное в дополнительном коде. Для получения адреса ячейки это число надо прибавить к содержимому PC, или As2, или As3, в зависимости от значения $K(6:7)$. Таким образом, адрес получается смещением в пределах -200_8 , $+177_8$ относительно значения, находящегося в указанном $K(6:7)$ регистре.

При $K(5) = 0$ определенный таким способом адрес является окончательным, т. е. указывает ячейку, к которой прямо относится обозначенная командой операция. Если же $K(5) = 1$, то содержимое адресуемой ячейки рассматривается как адрес $A(0:15)$, причем при $A(0) = 0$ этот адрес является окончательным, а при $A(0) = 1$ — косвенным, т. е. возможна многократная косвенная адресация.

Посредством прямой адресации процессору доступны 256 первых ячеек памяти (абсолютная адресация, $K(6:7) = 0$) и по 256 ячеек, адресуемых относительно PC, As2 и As3. Посредством косвенной адресации доступны 32К ячеек от 0 до 77777_8 .

Особыми являются 16 ячеек главной памяти с адресами 20_8-37_8 . В процессе вычисления исполнительного адреса ячейки 20_8-27_8 работают как автоинкрементные, а ячейки 30_8-37_8 — как автодекрементные. Содержимое первых при обращении к ним за адресным словом автоматически увеличивается на единицу, а содержимое вторых соответственно убавляется на единицу. Будучи прямо адресуемыми, эти ячейки работают обычным образом.

Рассмотренная процедура вычисления исполнительного адреса $EA(1:15)$ в машинах NOVA может быть описана в виде следующей программы:

```

if K (6 : 7) = 0 then EA := K (8 : 15) else
if K (6 : 7) = 1 then EA := PC +
+ sgnvl(K (8 : 15)) else
if K (6 : 7) = 2 then EA := Ac2 +
+ sgnvl(K (8 : 15)) else
if K (6 : 7) = 3 then EA := Ac3 +
+ sgnvl(K (8 : 15));
if K (5) = 1 then
begin
L: if EA  $\geq$  208  $\wedge$  EA  $\leq$  278 then
m(EA) := m(EA) + 1 else
if EA  $\geq$  308  $\wedge$  EA  $\geq$  378 then
m(EA) := m(EA) - 1;
A := m(EA); EA := A (1 : 15);
if A (0) = 1 then go to L
end

```

Функция `sgnvl` интерпретирует двоичное слово как целое число со знаком, представленное в дополнительном коде, в отличие от общепринятой в двоичных машинах интерпретации адресных слов как целых чисел без знака.

Арифметические операции при вычислении исполнительного адреса EA производятся по модулю 2^{15} , т. е. в качестве EA используются 15 младших битов результата. Поэтому адресом, на единицу большим 77777_8 , будет 0, а адресом, на единицу меньшим нулевого, будет 77777_8 . Следует иметь в виду, что приращение адреса 77777_8 или убавление адреса 0 в хранимом в памяти адресном слове изменяет значение бита A(0), который служит признаком «прямой/косвенный адрес».

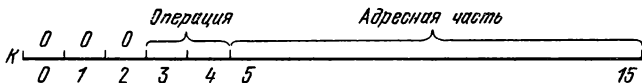


Рис. 8.3. Формат команд переходов и приращений.

Адресными командами в машинах NOVA являются команды переходов, приращений и пересылок (рис. 8.3 и 8.4).

Код операции K(3:4) принимает следующие значения:

$$K(3:4) = \begin{cases} 0 - \text{JMP, безусловный переход,} \\ 1 - \text{ISZ, приращение и перескок} \\ \quad \text{по нулю,} \\ 2 - \text{DSZ, убавление и перескок по нулю,} \\ 3 - \text{JSR, переход на подпрограмму.} \end{cases}$$

Адрес возврата при переходе на подпрограмму сохраняется в Ас3.

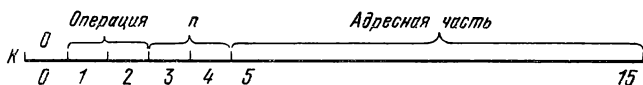


Рис. 8.4. Формат команд пересылок.

Формат команд пересылок отличается тем, что K(3:4) указывает номер n аккумулятора, участвующего в пересылке (см. рис. 8.4).

Этих команд две:

$$K(1:2) = \begin{cases} 1 - \text{LDA, загрузить аккумулятор,} \\ 2 - \text{STA, запомнить содержимое аккумулятора.} \end{cases}$$

Описание см. в табл. 8.1.

Запись рассмотренных команд на языке ассемблера осуществляется при следующих соглашениях о представлении адресов.

Абсолютный адрес пишут просто в виде восьмеричного числа. Например, команда

JMP 62

предписывает переход по адресу 50 ($62_8 = 50$).

В случае относительной адресации пишут два числа, разделенных запятой: первое — восьмеричное число в прямом коде — указывает величину и знак смещения, второе указывает регистр, относительно которого производится смещение, т.е. 1 указывает РС, 2 — Ас2, 3 — Ас3. Например,

JMP -36, 1

означает переход на 30 ячеек назад ($36_8 = 30$) относительно текущего значения РС. Эту же команду можно

Адресные команды машин NOVA

| Мнемокод | Операция | Описание |
|----------|-------------------------------|--|
| LDA | загрузить Ас | $Acn := m(EA);$ |
| STA | запомнить Ас | $m(EA) := Acn;$ |
| JMP | переход | $PC := EA;$ |
| ISZ | приращение и перескок по нулю | $m(EA) := m(EA) + 1;$ if $m(EA) = 0$ then $PC := PC + 1;$ |
| DSZ | убавление и перескок по нулю | $m(EA) := m(EA) - 1;$ if $m(EA) = 0$ then $PC := PC + 1;$ |
| JSR | переход на подпрограмму | $Ac3 := PC + 1; PC := m(EA);$ |

Мнемоника: LDA — Load Ac, STA — Store Ac, JMP — Jump, ISZ — Increment and Skip if Zero, DSZ — Decrement and Skip if Zero, JSR — Jump to Subroutine, *n* — номер аккумулятора, EA — исполнительный адрес.

Примечание. В описания команд не включено прибавление единицы к PC, которым завершается выполнение каждой команды, за исключением команд перехода и перехода на подпрограмму. Встречающееся в описаниях текущее значение PC соответствует адресу ячейки, в которой находится описываемая команда.

записать более компактно, обозначив текущее значение PC точкой:

JMP. — 36

Косвенная адресация обозначается @. Например, команда

ISZ @300

предписывает приращение содержимого и перескок по нулевому значению ячейки, доступ к которой осуществляется через ячейку 192 ($300_8 = 192$), может быть, путем многократной косвенной адресации.

В командах пересылок номер аккумулятора, участвующего в пересылке, помещается перед адресом и отделяется от него запятой. Например, команда

LDA 2, 20, 3

предписывает скопировать в Ac2 содержимое ячейки.

адрес которой равен $As3 + 20_8$, а команда

STA 1, @ — 5, 2

предписывает скопировать содержимое $As1$ в ячейку памяти, адресуемую косвенно посредством ячейки $As2 - 5$.

Язык ассемблера позволяет использовать в адресных выражениях имеющиеся в программе метки. Например, в программе, содержащей помеченную команду

L: LDA 3, @0, 2

возможна команда, адресующая относительно метки L:

STA 2, L + 7

Ассемблер заменит выражение $L + 7$ либо соответствующим абсолютным адресом (если $L + 7 < 256$), либо смещением относительно PC (если $PC - 128 \leq L + 7 \leq PC + 127$), а в случае, когда ни первое, ни второе невозможно, напечатает сообщение об ошибке.

Иллюстрацией возможностей рассмотренных способов адресации может служить техника обращения к подпрограммам. Переход на подпрограмму производится командой JSR A, где A — адресное выражение, указывающее начальную ячейку подпрограммы. Команда JSR засылает PC в $As3$ и производит переход на начало подпрограммы. Сохранение адреса возврата в аккумуляторе (а не в теле подпрограммы) позволяет сделать подпрограмму чистой (повторно входимой) и упрощает передачу параметров. Однако при необходимости использования в подпрограмме аккумулятора $As3$ содержащийся в нем адрес возврата надо перезапомнить в ячейке главной памяти. Возврат по окончании подпрограммы, осуществляемый обычно командой JMP 0, 3, в этом случае следует производить, используя косвенную адресацию посредством ячейки, скажем, B, хранящей адрес возврата: JMP @B.

Переход на подпрограмму обычно выполняется также с помощью косвенной адресации посредством некоторой ячейки абсолютно адресуемого участка памяти, в которой находится адрес начала подпрограммы. Обозначив адрес этой ячейки буквой C, а адрес начала подпрограммы — буквой A, покажем, как программируется обращение к подпрограмме

с передачей в Ac0 и Ac1 значений параметров, адреса которых расположены непосредственно после команды JSR, причем результат, получающийся в Ac1 по окончании выполнения подпрограммы, пересылается в память на место второго параметра.

Пример обращения к подпрограмме A с передачей параметров.

| | | |
|----|--------------|-------------------------|
| L: | JSR @C | |
| | . | ; адрес 1-го параметра, |
| | . | ; адрес 2-го параметра |
| | . | и результата, |
| | | ; продолжение програм- |
| | | мы, |
| | . | |
| | . | |
| | . | |
| | . | |
| A: | LDA 0, @0, 3 | ; засылка 1-го парамет- |
| | | ра в Ac0, |
| | LDA 1, @1, 3 | ; засылка 2-го парамет- |
| | | ра в Ac1, |
| | . | |
| | . | |
| | . | |
| | . | |
| | STA 1, @1, 3 | ; запоминание резуль- |
| | | тата, |
| | JMP 2, 3 | ; возврат на L + 1 + 2. |

§ 3. Команда-микропрограмма

Операции преобразования данных в машинах NOVA полностью отделены от операций обращения к главной памяти и выполняются над аккумуляторами Ac0, Ac1, Ac2, Ac3 и битом переноса Cr. Программирование этих операций осуществляется с помощью команды-микропрограммы, которая предписывает определенную последовательность операций над содержимым бита переноса и одного-двух указанных в ней аккумуляторов, а также назначает условие, при выполнении которого производится перескок через следующую команду. Последовательность операций в общем случае включает: предварительную операцию над битом переноса, основную операцию над аккумуляторами и битом переноса, сдвиг результата основной операции в кольце аккумулятор — бит переноса влево или вправо и тестирование результата на соответствие заданному условию.

Головной бит команды-микропрограммы (рис. 8.5) содержит единицу: $K(0) = 1$ — это ее отличие от других команд. Две пары битов — $s = K(1:2)$ и

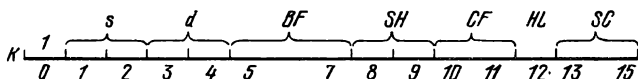


Рис. 8.5. Формат команды-микропрограммы.

$d = K(3:4)$ — представляют собой номера аккумуляторов отправления s (source) и назначения d (destination). Вид основной операции BF (Base Function) указывают следующие три бита команды: $BF = K(5:7)$.

Если основная операция является одностепенной, то она производится (рис. 8.6) над операндом As , получаемым из аккумулятора отправления Acs , а ее

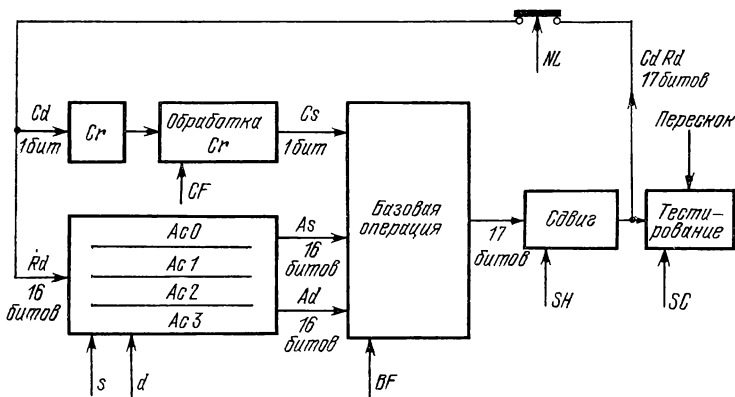


Рис. 8.6. Схема выполнения команды-микропрограммы

результат после обработки сдвигом посылается в аккумулятор назначения Ac_d . Двухместная основная операция производится над операндами As и Ad , получаемыми из аккумуляторов Acs и Ac_d , а ее результат, также через сдвигатель, посылается в аккумулятор назначения Ac_d .

Операнд As при выполнении основной операции сочленяется с Cs — модифицированной операцией CF (Carry Function) значением бита переноса Cr , образуя 17-битное слово $CsAc$. Таким образом, основная операция имеет вид $BF(CsAc, Ad)$, где $Cs = CF(Cr)$,

**Основные операции команды-микропрограммы
в машинах NOVA**

| BF | Мнемокод | Операция | Описание |
|----|----------|----------------------|-------------------------------------|
| 0 | COM | инверсия | $CdRd := Cs\overline{As}$ |
| 1 | NEG | изменение знака | $CdRd := Cs\overline{As} + 1;$ |
| 2 | MOV | пересылка | $CdRd := CsAs;$ |
| 3 | INC | прибавление 1 | $CdRd := CsAs + 1;$ |
| 4 | ADC | прибавление инверсии | $CdRd := Ad + Cs\overline{As};$ |
| 5 | SUB | вычитание | $CdRd := Ad + Cs\overline{As} + 1;$ |
| 6 | ADD | сложение | $CdRd := Ad + CsAs;$ |
| 7 | AND | конъюнкция | $Cd := Cs; Rd := Ad \wedge As;$ |

Таблица 8.3

Операции над битом переноса в машинах NOVA

| CF | Мнемокод | Операция | Описание |
|----|----------|-------------|------------------------|
| 0 | | сохранение | $Cs := Cr;$ |
| 1 | Z | установка 0 | $Cs := 0;$ |
| 2 | 0 | установка 1 | $Cs := 1;$ |
| 3 | C | инверсия | $Cs := \overline{Cr};$ |

а операнд Ad игнорируется, если операция BF является одностепенной. Виды операций BF и CF представлены в табл. 8.2 и 8.3.

Полученное в результате выполнения основной операции 17-битное слово обрабатывается циклическим сдвигом, который в зависимости от значения управляющей им пары битов $SH = K(8:9)$ либо производит циклический сдвиг на одно место влево, либо производит аналогичный сдвиг вправо, либо переставляет местами головной и хвостовой байты в слове Ad , либо сохраняет результат неизменным. Операции сдвигателя охарактеризованы в табл. 8.4.

После сдвигателя следует тестирование результата на соответствие условию перескока SC (Skip Condition), которое задано тремя хвостовыми битами $K(13:15)$. Перечень условий приведен в табл. 8.5. Если результат удовлетворяет указанному в команде

Операции сдвига в команде-микропрограмме машин NOVA

| SH | Мнемокод | Операция | Описание |
|----|----------|--------------------------|-------------------------|
| 0 | | ничего не делать | |
| 1 | L | циклический сдвиг влево | $CdRd = RdCd$; |
| 2 | R | циклический сдвиг вправо | $RdCd = CdRd$; |
| 3 | S | перестановка байтов | $Rd(0:7) = :Rd(8:15)$: |

Таблица 8.5

Условия перескока в команде-микропрограмме машин NOVA

| SC | Мнемокод | Условие перескока | Описание операции |
|----|----------|------------------------|--|
| 0 | | никогда | |
| 1 | SKP | безусловно | $PC := PC + 1$; |
| 2 | SZC | $Cd = 0$ | if $Cd = 0$ then $PC := PC + 1$; |
| 3 | SNC | $Cd = 1$ | if $Cd = 1$ then $PC := PC + 1$; |
| 4 | SZR | $Rd = 0$ | if $Rd = 0$ then $PC := PC + 1$; |
| 5 | SNR | $Rd \neq 0$ | if $Rd \neq 0$ then $PC := PC + 1$; |
| 6 | SEZ | $Cd = 0$ или $Rd = 0$ | if $Cd = 0 \vee Rd = 0$ then $PC := PC + 1$; |
| 7 | SBN | $Cd = 1$ и $Rd \neq 0$ | if $Cd = 1 \wedge Rd \neq 0$ then $PC := PC + 1$; |

условию, то происходит перескок, т.е. следующая команда будет пропущена.

Выполнение команды-микропрограммы завершается приемом результата в Cr и Acd : Cr принимает значение головного бита, Acd — 16 остальных битов. Прием результата производится при условии $K(12) = 0$. Если же $K(12) = 1$ ($NL = 1$, No Load), прием результата не производится, содержимое аккумуляторов и бита переноса сохраняется таким, каким оно было до выполнения команды. В этом случае указываемые командой операции производятся с целью проверить выполнение условия перескока и, если оно удовлетворено, осуществить перескок.

Рассмотренную интерпретацию команды-микропрограммы выражает следующий алгоритм.

```

begin boolean array As(0 : 15), Ad(0 : 15),
Rd(0 : 15), Cs(1 : 1), Cd(1 : 1);
ASGN:  if K(1 : 2) = 0 then As := Ac0 else
        if K(1 : 2) = 1 then As := Ac1 else
        if K(1 : 2) = 2 then As := Ac2 else
        if K(1 : 2) = 3 then As := Ac3;
        if K(3 : 4) = 0 then Ad := Ac0 else
        if K(3 : 4) = 1 then Ad := Ac1 else
        if K(3 : 4) = 2 then Ad := Ac2 else
        if K(3 : 4) = 3 then Ad := Ac3;
        if K(10 : 11) = 0 then Cs := Cr;
Z:      if K(10 : 11) = 1 then Cs := 0;
O:      if K(10 : 11) = 2 then Cs := 1;
C:      if K(10 : 11) = 3 then Cs :=  $\overline{Cr}$ ;
COM:    if K(5 : 7) = 0 then CdRd := CsAs;
NEG:    if K(5 : 7) = 1 then CdRd := Cs $\overline{As}$  + 1;
MOV:    if K(5 : 7) = 2 then CdRd := CsAs;
INC:    if K(5 : 7) = 3 then CdRd := CsAs + 1;
ADC:    if K(5 : 7) = 4 then CdRd := Cs $\overline{As}$  + Ad;
SUB:    if K(5 : 7) = 5 then CdRd :=
        CsAs + 1 + Ad;
ADD:    if K(5 : 7) = 6 then CdRd := CsAs + Ad;
AND:    if K(5 : 7) = 7 then begin Rd := As  $\wedge$  Ad;
        Cd := Cs end;
L:      if K(8 : 9) = 1 then CdRd := RdCd;
R:      if K(8 : 9) = 2 then RdCd := CdRd;
S:      if K(8 : 9) = 3 then Rd(0 : 7) := :Rd(8 : 15);
SKP:    if K(13 : 15) = 1 then PC := PC + 1;
SZC:    if K(13 : 15) = 2  $\wedge$  Cd = 0 then
        PC := PC + 1;
SNC:    if K(13 : 15) = 3  $\wedge$  Cd = 1 then
        PC := PC + 1;
SZR:    if K(13 : 15) = 4  $\wedge$  Rd = 0 then
        PC := PC + 1;

```

```

SNR:    if K(13 : 15) = 5  $\wedge$  Rd  $\neq$  0 then
        PC := PC + 1;
SEZ:    if K(13 : 15) = 6  $\wedge$  (Cd = 0  $\vee$  Rd = 0) then
        PC := PC + 1;
SBN:    if K(13 : 15) = 7  $\wedge$  Cd = 1  $\wedge$  Rd  $\neq$  0 then
        PC := PC + 1;
LOAD:   if K(12) = 0 then
        begin Cr := Cd;
        if K(3 : 4) = 0 then Ac0 := Rd else
        if K(3 : 4) = 1 then Ac1 := Rd else
        if K(3 : 4) = 2 then Ac2 := Rd else
        if K(3 : 4) = 3 then Ac3 := Rd
        end
        end

```

Запись команды-микропрограммы на языке ассемблера производится в следующей последовательности. Первым пишут мнемокод основной операции, затем указывают операцию над битом переноса, затем сдвиг и запрет загрузки результата в виде знака #. Если операцию над битом переноса или сдвиг выполнять не надо, а также если загрузка результата не запрещена, то соответствующие мнемнообозначения не пишутся. Далее записываются разделяемые запятыми номер аккумулятора отправления, номер аккумулятора назначения и мнемокод условия перескока, если перескок возможен. Комментарий отделяется точкой с запятой.

Например:

```

SUB      2, 1      ; Ac1 := Ac1 - Ac2;
MOVCR    2, 0      ; Cr := Cr;
                        Ac0Cr := CrAc2;
ADCZ #    1, 0, SZC ; if Ac0  $\leq$  Ac1 then
                        PC := PC + 1;

```

Примеры программ, иллюстрирующие характер программирования на языке ассемблера для машин NOVA.

Программа вычисления разности $c = a - b$.

```

.LOC      200
START:   LDA 1, A    , Ac1 := m(A);
          LDA 0, B    ; Ac0 := m(B);

```

| | | | | |
|----|------|------|------------------------|----------------|
| | SUB | 0, 1 | ; $Ac1 := Ac1 - Ac0$; | |
| | STA | 1, C | ; $m(C) := Ac1$; | |
| | HALT | | ; stop | |
| A: | . | . | . | ; значение a |
| B: | . | . | . | ; значение b |
| C: | . | . | . | ; значение c |

Программа формирования матрицы битов с единицами на диагонали.

Матрица состоит из 16 16-битных слов. Первое слово M0(0:15) содержит единицу в хвостовом бите M0(15), а в остальных его битах — нули. Второе слово M1(0:15) содержит единицу только в бите M1(14), третье — только в M2(13) и т.д. Последнее слово M15(0:15) содержит единицу только в головном бите M15(0).

```

.LOC      200
START:    LDA      2, A0      ;  $Ac2 := m(A0)$ ;
          STA      2, 21      ;  $m(21) := Ac2$ ;
          LDA      1, W       ;  $Ac1 := m(W)$ ;
          STA      1, 0, 2    ;  $m(Ac2) := Ac1$ ;
          MOVZL    1, 1, SNR  ;  $Cr := 0$ ;  $CrAc1 :=$ 
                                $Ac1Cr$ ; if  $Ac1 \neq 0$ 
                               then  $PC := PC + 1$ ;
          HALT      ; stop;
          STA      1, @21     ;  $m(21) := m(21) +$ 
                                $+ 1$ ;  $m(m(21)) :=$ 
                                $Ac1$ ;
          JMP      -3         ; go to  $PC - 3$ ;
W:         00000i
A0:        . . .             ; адрес M0

```

В этой программе ячейка с адресом 21 использована в качестве автоинкрементного указателя адреса текущей строки матрицы.

Программа, суммирующая содержимое N последовательных ячеек памяти, начиная с ячейки, адрес которой находится в ячейке A1:

```

LOC      200      0,C1      ;  $Ac0 := m(C1)$ ;
START:    LDA      0,CJ      ;  $m(CJ) := Ac0$ ;
          STA      0,CJ
          LDA      2,A1      ;  $Ac2 := m(A1)$ ;
          STA      2,21      ;  $m(21) := Ac2$ ;
          LDA      0,0,2     ;  $Ac0 := m(Ac2)$ ;

```

| | | | |
|------|------|--------|---|
| L: | LDA | 1, 21 | ; m(21) := m(21) + 1; Ac1 := m(m(21)); |
| | ADDZ | 1, 0 | ; Cr := 0; |
| | DSZ | , CJ | Ac0 := Ac0 + Ac1; |
| | | | ; m(CJ) := m(CJ) - 1; if m(CJ) = 0 then PC := PC + 1; |
| | JMP | L | ; go to L; |
| | STA | 0, SUM | ; m(SUM) := Ac0; |
| | HALT | | ; stop |
| C1: | . | . | ; значение N - 1 |
| CJ: | . | . | ; счетчик |
| A1: | . | . | ; адрес первого слагаемого |
| SUM: | | | ; сумма |

§ 4. Команды ввода/вывода и управление прерыванием

Признак команды ввода/вывода — $K(0:2) = 3$ (рис. 8.7). Команды этого вида обеспечивают управление периферийными устройствами, тестирование

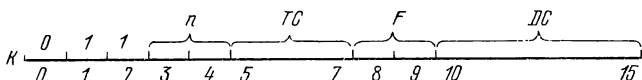


Рис. 8.7. Формат команды ввода/вывода.

их состояния и передачу данных между регистрами этих устройств и аккумуляторами процессора.

Обозначение (код) периферийного устройства, к которому относится команда, — DC (Device Code) — составляют шесть ее хвостовых битов $K(10:15)$. Из 64 значений кода DC нулевое не используется, а 77_8 обозначает сам процессор, в частности, при управлении системой прерывания. Номер n аккумулятора, участвующего в передаче данных, содержится в $K(3:4)$.

Три следующих бита — $K(5:7)$, обозначенные TC (Transfer Code), — указывают регистр периферийного устройства, участвующий в передаче, и направление передачи: ввод, т.е. из устройства в процессор, или вывод, т.е. из процессора в устройство (см.

**Функция ТС, управляющая передачей данных между
аккумуляторами и буферными регистрами**

| ТС | Мнемокод | Операция | Описание |
|----|--|------------------------|-------------|
| 0 | NIO | передача не происходит | |
| 1 | DIA | прием из буфера А | $Asn: = A,$ |
| 2 | DOA | передача в буфер А | $A: = Asn,$ |
| 3 | DIB | прием из буфера В | $Asn: = B,$ |
| 4 | DOB | передача в буфер В | $B: = Asn,$ |
| 5 | DIC | прием из буфера С | $Asn: = C;$ |
| 6 | DOC | передача в буфер С | $C: = Asn;$ |
| 7 | Команды перескока по состоянию (см. табл. 8.8) | | |

Мнемоника: NIO – NO I/O transfer, DIA – Data In A, DOA – Data Out A, DIB – Data In B и т.д.

табл. 8.6). Двум значениям ТС, а именно 0 и 7, соответствует отсутствие передачи: при $K(5:7) = 0$ команда осуществляет только управление устройством, при $K(5:7) = 7$ тестирует состояние устройства, не производя передачи данных. Номер аккумулятора в этих двух случаях не используется.

Пара битов $K(8:9)$, обозначенная F (Function), в случае $K(5:7) \neq 7$, задает функцию управления состоянием устройства, а в случае $K(5:7) = 7$ определяет условие перескока по состоянию устройства (см. табл. 8.7 и 8.8).

Каждое периферийное устройство, подключаемое к магистрали ввода/вывода, имеет в своем контроллере 6-входный дешифратор кода устройства, один-три буферных регистра для приема/выдачи данных и три бита (флажка) состояния: «занят» (Busy), «готов» (Done), «прерывание запрещено» (Interrupt Disable). Устройство воспринимает те команды, в которых значение $K(10:15)$ совпадает с кодом этого устройства, а также команды управления прерыванием.

Флажки «занят» и «готов» характеризуют состояние устройства следующим образом. Когда оба флажка сброшены, устройство бездействует. Чтобы запустить устройство в работу, процессор соответствующей командой поднимает его флажок «занят». В состоянии «занят» устройство в случае ввода считывает

Функция F, управления флажками устройств, $TC \neq 7$, $DC \neq 77_8$

| F | Мнемокод | Операция | Описание |
|---|----------|--|-----------------------|
| 0 | | никакой операции | |
| 1 | S | запуск устройства в работу | Done: = 0; Busy: = 1; |
| 2 | C | перевод в состояние бездействия | Done: = 0; Busy: = 0; |
| 3 | P | возбуждение специальной управляющей линии — эффект определяется видом устройства | |

Примечание. При $DC = 77_8$: S — включение. C — выключение системы прерывания.

Условия перескока по состоянию устройства, $TC = 7$, $DC \neq 77_8$

| F | Мнемокод | Операция | Описание |
|---|----------|---------------------------|--------------------------------|
| 0 | SKPBN | перескок, если «занят» | if Busy = 1 then PC: = PC + 1; |
| 1 | SKPBZ | перескок, если не «занят» | if Busy = 0 then PC: = PC + 1; |
| 2 | SKPDN | перескок, если «готов» | if Done = 1 then PC: = PC + 1; |
| 3 | SKPDZ | перескок, если не «готов» | if Done = 0 then PC: = PC + 1; |

Примечание. При $DC = 77_8$: Busy — включенность системы прерывания. Done — неисправность питания.

вводимые данные с внешнего носителя и помещает их в свой буфер, а в случае вывода переносит содержимое буфера на внешний носитель. Закончив ввод/вывод, устройство сбрасывает флажок «занят» и поднимет флажок «готов». Состояние «готов» в случае ввода означает готовность устройства передать содержимое своего буфера процессору, а в случае вывода — готовность принять от процессора в буфер очередную порцию выводимых данных. Команда, которой процессор передает содержимое буфера в один из аккумуляторов или загружает буфер содержимым аккумулятора, может в качестве функции управления F

сбросить флажок «готов» и поднять флажок «занят», снова запустив устройство в работу, или сбросить флажок «готов», не поднимая флажка «занят», вследствие чего устройство перейдет в состояние бездействия. Команда с $K(5:7) = 0$ осуществляет управление флажками, не производя передачи данных.

Состояние флажка «прерывание запрещено» управляется специальной командой ($DC = 77_8$, $TC = 4$), которая производит установку этих флажков одновременно на всех периферийных устройствах, причем состояние каждого флажка определяется одним из битов находящейся в аккумуляторе 16-битной маски. Если флажок «прерывание запрещено» на данном устройстве сброшен, то при каждом поднятии на этом устройстве флажка «готов» автоматически запрашивается прерывание процессора. Если же флажок «прерывание запрещено» поднят, то устройство не может возбуждать прерывание и процессор должен тестировать состояние флажка «готов» на этом устройстве с помощью цикла ожидания. Тестирование состояний флажков «готов» и «занят» осуществляется командами перескока, в которых $K(5:6) = 7$, номер аккумулятора $K(3:4)$ несуществен, а биты функции управления $K(8:9)$ указывают условие перескока (см. табл. 8.8).

Запись команды ввода/вывода на языке ассемблера производится в следующей последовательности. Первым пишут мнемокод управления передачей данных, добавляя к нему, если необходимо, букву S, C или P, управляющую состоянием устройства, вторым словом записываются разделенные запятой номер аккумулятора и буквенный или цифровой код устройства. Примеры:

| | | |
|------|-------|-------------------|
| DIA | 0,PTR | ; Ac0: = PTR A; |
| | | Done (PTR): = 0; |
| DIAS | 2,PTR | ; Ac2: = PTR A; |
| | | Done(PTR): = 0; |
| | | Busy(PTR): = 1; |
| DOAC | 1,TTO | ; TTOA: = Ac1; |
| | | Done(TTO): = 0; |
| | | Busy(TTO): = 0; |
| NIOS | PTR | ; Done(PTR): = 0; |
| | | Busy(PTR): = 1; |

Команды тестирования флажков состоят из мнемокода, выражающего условие перескока, и кода устройства, к которому относится команда.

Например:

```
SKPDN    PTRi    ; if Done(PTR) = 1 then  
PC := PC + 1.
```

Особый смысл имеют команды, в которых в качестве кода устройства записано CPU, соответствующее цифровому коду 77₈ и означающее процессор — Central Processing Unit. Для этих команд наряду с записью их по изложенным выше правилам установлена специальная мнемоника, отражающая их не соответствующее стандартной записи содержание.

Буквы S и C, используемые в стандартной мнемонике для управления состоянием флажков «занят» и «готов», в командах с CPU означают: S — включить систему прерывания, C — выключить систему прерывания. Команду NIOS CPU, включающую систему прерывания, допустимо обозначать специальным мнемокодом INTEN — Interrupt Enable, а команду NIOC CPU, выключающую систему прерывания, — INTDS, что значит Interrupt Disable.

Команды передачи данных между буферами A, B, C, с одной стороны, и аккумуляторами процессора — с другой, при использовании их применительно к CPU имеют следующий смысл и специальную мнемонику:

DIA *n*, CPU — считать в Acp значение, представленное набором ключей панели управления, мнемокод — READS (Read Switches);

DIB *n*, CPU — принять в Acp(10:15) код ближайшего к процессору по положению на магистрали ввода/вывода устройства, требующего прерывания, мнемокод — INTA (Interrupt Acknowledge);

DIC 0, CPU — сбросить флажки «занят», «готов» и «прерывание запрещено» на всех устройствах. Мнемокод IORST (I/O Reset) соответствует команде DICC 0, CPU, которая, кроме того, выключает систему прерывания;

DOV *n*, CPU — установить флажки «прерывание запрещено» в соответствии с состояниями сопоставленных устройствам битов маски, содержащейся в Acp, мнемокод MSKO, означает Mask Out;

DOC 0,CPU — остановить процессор, выдав на индикаторы панели управления данную команду и адрес следующей за ней команды. Мнемокод — HALT.

Специальная мнемоника, в отличие от стандартной, не позволяет наряду с основной операцией производить включение и выключение системы прерывания добавлением к мнемокоду букв S или C. Например, команда, останавливающая процессор, возможна в трех модификациях: 1) DOC 0,CPU, 2) DOCS 0,CPU и 3) DOCC 0,CPU, а мнемокод HALT соответствует только первой из них. Мнемокод IORST соответствует команде DICC 0,CPU, которая, в частности, выключает систему прерывания, однако имеют место два других варианта этой команды: DIC 0,CPU — сбросить флажки всех устройств, не выключая системы прерывания, DICS 0,CPU — сбросить флажки устройств и включить систему прерывания.

Устройство запрашивает прерывание, если на нем флажок «готов» поднят, а флажок «прерывание запрещено» сброшен. Затребованное прерывание происходит по окончании выполнения текущей команды при условии, что система прерывания включена, нет заявок на прямой доступ к памяти и нет запросов прерывания с более высоким приоритетом, чем присвоенный данному устройству.

Иерархию приоритетов определяют следующие обстоятельства. При запросе кода прерывающего устройства командой INTA процессор из нескольких одновременно требующих прерывания устройств получает код того, которое расположено на магистрали ввода/вывода физически ближе к процессору, чем другие, требующие прерывания устройства. При тестировании флажков «готов» приоритет определяется порядком, в котором программа перебирает устройства.

Способность устройства прерывать программу, обслуживаемую прерыванием, вызванное другим устройством, определяется тем, каким битом маски сопоставлены эти устройства (приоритет убывает с увеличением номера бита), и состоянием битов (устройствам, сопоставленным битам, содержащим единицы, прерывание запрещено). Следует иметь в виду, что одному биту маски может соответствовать несколько устройств.

Начиная прерывание, процессор выключает систему прерывания, запоминает очередное значение программного счетчика $PC + 1$ в ячейке 0 и производит переход по адресу, хранящемуся в ячейке 1, на программу обслуживания прерывания. Эта программа упрятывает содержимое необходимых ей аккумуляторов и бита переноса, определяет устройство, которое надлежит обслуживать, используя команду INTA или цепочку команд, тестирующих флажки устройств, и выполняет соответствующие функции обслуживания. На время своей работы обслуживающая программа может оставить систему прерывания выключенной, но если необходимо разрешить какие-либо срочные прерывания, то она, замаскировав прерывания от устройств, не требующих срочного обслуживания, и перезапомнив содержащийся в ячейке 0 адрес возврата, включает систему прерывания. Перед выходом из прерывания система прерывания должна быть выключена, чтобы не допустить прерывания в период выхода.

Действия, выполняемые при выходе из прерывания, включают восстановление упрятанного при входе содержимого аккумуляторов и бита переноса, размаскирование замаскированных прерываний, включение системы прерывания и переход на продолжение прерванной программы. Включение системы прерывания командой INTEN происходит не сразу, а только после выполнения следующей за ней команды перехода по адресу возврата, иначе очередное прерывание могло бы преждевременно заслать в ячейку 0 свой адрес возврата, сделав возврат из текущего прерывания невыполнимым.

Для высокоскоростных периферийных устройств, таких как магнитные диски и магнитные ленты, в машинах семейства NOVA предусмотрен прямой доступ к памяти (фирменное название Data channel — канал данных). Контроллер устройства, пользующегося каналом данных, помимо буферных регистров и битов-флажков, имеет регистр адреса и счетчик передаваемых слов. Получив от центрального процессора задание на ввод/вывод, указывающее, в частности, адрес начала передаваемого массива данных в главной памяти и количество слов в нем, устройство осуществляет передачу слов, не запрашивая прерывания,

а захватывая циклы памяти по одному на каждое передаваемое слово. Максимальная скорость передачи зависит от модели процессора и типа памяти и составляет обычно от 0,2 до 0,5 млн. слов в секунду, а в случае высокоскоростного канала, допускающего захват циклов памяти не только по окончании очередной команды, но и во время ее выполнения, передача может производиться со скоростью 1 млн. слов в секунду и более. Кроме считывания и записи данных по указываемым устройством адресам главной памяти, каналы машин NOVA способны производить прибавление единицы, а в некоторых моделях также прибавление полученного от устройства числа к содержимому указанной ячейки, причем результат сложения сообщается устройству в виде 16-битного слова с сигналом переполнения, если оно произошло.

Заявки на предоставление цикла памяти для прямого доступа удовлетворяются в первую очередь (в высокоскоростном канале даже с приостановкой выполнения команды). Если одновременно имеются заявки от нескольких устройств, то очередность предоставления циклов памяти определяется физической близостью этих устройств к процессору на магистрали ввода/вывода. Процессор не может продолжить прерванную работу, пока не удовлетворит все имеющиеся заявки на предоставление циклов, а по завершении очередной команды также все заявки на прерывание, выполняемые вслед за предоставлением циклов.

§ 5. Периферийное оборудование миникомпьютеров NOVA

Характерные для миникомпьютеров модульность и наращиваемость оборудования, а также гибкая приспособляемость конфигурации системы к конкретному применению в полной мере присущи машинам семейства NOVA. Базовый комплект, включающий процессор, 4KW (в некоторых моделях 8KW) памяти и телетайп, легко расширяется подключением добавочных блоков памяти, специализированных процессорных блоков, устройств периферийной памяти, ввода/вывода, сопряжения с линиями связи, с измерительными

приборами и исполнительными механизмами. Машины, как предназначенные для монтирования в стандартной стойке, так и выпускаемые в настольном исполнении, имеют свободные места для установки добавочных плат памяти, операционных блоков и аппаратуры сопряжения с периферией.

Архитектура машин NOVA обеспечивает адресацию главной памяти емкостью до 32 KW (до 64 KB), поэтому принципиальных трудностей при расширении памяти в этих пределах не возникает. Практически память увеличивают добавлением плат емкостью 4KW, 8KW и 16KW. Все другие расширения осуществляются в отношении архитектуры с помощью команд ввода/вывода, т.е. центральный процессор взаимодействует с добавляемыми устройствами как с периферийными.

Непосредственными добавлениями к процессору являются: датчик реального времени (Real Time Clock), аппаратура сигнализации о неисправности питания и автоматического перезапуска (Power Monitor and Autorestart), аппаратура распределения и защиты памяти, аппаратура умножения и деления, устройство арифметики с плавающей запятой FPU (Floating Point Unit).

Датчик реального времени, включаемый в работу командой ввода/вывода DOA и, RTC, прерывает центральный процессор с заданной частотой, в качестве которой можно использовать частоты 10, 100, 1000 гц и частоту переменного тока сети питания.

Аппаратура сигнализации о неисправности питания и автоматического перезапуска уведомляет процессор посредством системы прерывания о критическом отклонении напряжения питания от номинального значения, чтобы соответствующая обслуживающая программа упрятала в память содержимое его регистров для восстановления состояния процессора, когда питание станет нормальным.

Аппаратура распределения и защиты памяти, используемая в качестве необязательного добавления на некоторых моделях семейства NOVA, обеспечивает возможность распределения памяти между пользователями при многопрограммной работе и защиту от недозволенного доступа. Физическая память разделена на блоки по 4KW. Каждому пользователю может

быть выделен один или более блоков, и операционная система с помощью аппаратуры распределения отображает имеющиеся в рабочих программах пользователя адреса в физические адреса соответствующих блоков. С целью защиты память разделена на страницы по 256 слов.

Попытка пользователя произвести запись на странице, которая запрещена, вызывает прерывание и передачу управления операционной системе.

Аппаратура умножения/деления выполняет задаваемые при помощи команд ввода/вывода MUL и DIV операции умножения и деления чисел без знака. Сомножителями являются числа, содержащиеся в Ac1 и Ac2. Произведение получается удвоенной длины и прибавляется к содержимому Ac0: $Ac0Ac1 := Ac0 + Ac1 * Ac2$. Деление $Ac0Ac1 / Ac2$ выполняется только в случае $Ac2 > Ac0$, причем частное получается в Ac1, остаток в Ac0, делитель сохраняется в Ac2. В машинах, не укомплектованных аппаратурой умножения/деления, эти действия реализуются подпрограммами.

Устройство арифметики с плавающей запятой выполняет арифметические операции над числами с плавающей запятой обычной (32 бита, 6—7 десятичных цифр) и удвоенной (64 бита, 15 десятичных цифр) точности. Использование этого устройства во много раз увеличивает вычислительную мощность процессора: умножение чисел удвоенной точности на машине с циклом памяти 0,8 мкс оно выполняет менее чем за 20 мкс.

Набор устройств ввода/вывода, используемых с машинами семейства NOVA, включает дополнительно к телетайпу перфолентное оборудование (считыватель со скоростью 400 знаков в секунду и перфоратор со скоростью 60 знаков в секунду), считыватели с перфокарт, построчные и матричные побуквенные принтеры, барабанные и планшетные устройства графического вывода, электронно-лучевые дисплеи. В качестве устройств периферийной памяти имеется широкий выбор магнитных дисков, в том числе быстродействующие с фиксированными головками (Novadisks), емкостью до 1,5MB при среднем времени доступа 8,4 мс, картриджи, емкостью до 5MB, и пакетные, емкостью до 192MB при среднем времени

доступа 43 мс. Кроме того, используются устройства 7- и 9-дорожечной магнитной ленты и магнитные кассеты (Nova Cassettes).

Периферийное оборудование для работы в измерительных и управляющих системах представлено рядом преобразователей аналог/цифра с разрешающей способностью от 8 до 14 битов и временем преобразования соответственно от 8 до 100 мкс. Мультиплексная система преобразования позволяет обслуживать до 512 аналоговых каналов. Система преобразования цифра/аналог обслуживает до 24 каналов.

Имеется аппаратура для подключения машин NOVA к каналам ввода/вывода IBM 360/370 с целью использования их в качестве связанных процессоров, сборщиков данных и для выполнения других подобных функций.

Имеется также устройство межпроцессорного обмена — МСА (Multiprocessor Communications Adapter), с помощью которого возможно объединение в единую систему до 15 машин семейства NOVA и обеспечение обмена данными между ними со скоростью до 1 MB в секунду.

Другой способ комплексирования машин, используемый в системе Dual NOVA (сдвоенная NOVA), состоит в том, что две машины работают с общей памятью на магнитных дисках. Такая система обладает повышенной надежностью (если одна из машин выходит из строя, система продолжает работать с оставшейся машиной), возможностью выполнения работ в сокращенные сроки за счет распараллеливания обработки и повышенной мощностью, которая на некоторых применениях практически равна сумме мощностей соединенных машин.

Связное оборудование машин семейства NOVA включает контроллеры для управления асинхронной связью со скоростью до 9600 бит/с (до 64 линий), однолинейный контроллер синхронной связи со скоростью до 50 тыс. бит/с и многолинейные (до 64 линий) контроллеры синхронной связи со скоростью до 9600 бит/с.

МСА подключается к магистрали ввода/вывода машины на правах периферийного устройства, пользующегося каналом данных, т. е. прямым доступом к памяти.

§ 6. Программное оснащение миникомпьютеров NOVA

Программное оснащение семейства NOVA характерно не столько многочисленностью и разнообразием программ, сколько их совместимостью и согласованностью, сбалансированностью и единством системы в целом. Как правило, программа может работать в любой операционной системе, располагающей необходимыми для ее выполнения ресурсами. Ослабленные варианты программ являются совместимыми подмножествами более полных их вариантов. Программы, разработанные для одной из моделей семейства, выполнимы на всех моделях, обладающих требуемыми ресурсами, поскольку архитектура всех моделей одинакова.

Для машин семейства NOVA имеются три основные операционные системы: RDOS (Real-time Disk Operating System), RTOS (Real Time Operating System), SOS (Stand-alone Operating System).

RDOS представляет собой наиболее развитую операционную систему, сочетающую режим реального времени с обслуживанием терминалов пользователей и с пакетной обработкой на заднем плане. Она предоставляет пользователям обычные для дисковых ОС возможности работы с файлами, обеспечивает удобное управление вводом/выводом и линиями связи. Система может работать на любой модели семейства при наличии не менее 12KW главной памяти и не менее 128KW памяти на диске, датчика реального времени и телетайпа.

RTOS, представляющая собой совместимое подмножество RDOS, работает при наличии минимум 4KW главной памяти, датчика реального времени и телетайпа. Она обеспечивает выполнение программ пользователя в режиме реального времени и совместную работу процессора и периферийных устройств.

SOS также является совместимым подмножеством RDOS и может работать при наличии 12KW (для простейшей версии — 8KW) главной памяти и телетайпа. Система обеспечивает возможность удобного редактирования, ассемблирования и выполнения программ, вводимых с перфоленты, магнитной ленты или магнитных кассет.

В качестве средств программирования используются работающие в любой из перечисленных операционных систем ассемблеры (абсолютный, расширенный с ассемблированием переместимых программ и макро), фортран- и алгол-компиляторы, а также basic — интерпретатор, имеющийся в двух вариантах — для одного пользователя и для многотерминального разделения времени (до 16 терминалов). Имеются кросс-ассемблеры на фортране для IBM 360, CDC 6600 и UNIVAC 1108.

Языки программирования представлены их полными и расширенными версиями.

Фортран IV на основе ANSI стандарта 1966 г. с добавлением арифметики комплексных чисел удвоенной точности, с обобщением индексных выражений и увеличением размерности массивов до 128, специально приспособленный к применениям реального времени и к включению подпрограмм, написанных на языке ассемблера (имеется также дальнейшее расширение под названием фортран 5).

Алгол 60 с возможностью простого бесформатного ввода/вывода, манипулирования битами и строками, высокоточной арифметики (30 десятичных цифр), рекурсивных и повторно-входимых процедур и ряда других свойств.

Basic с использованием всех возможностей периферии машин NOVA, в том числе возможности манипулирования файлами.

Помимо преимуществ, предоставляемых расширенными языками, удобство программирования обеспечивается наличием редакторов и отладочных программ, а также развитой диагностикой компиляторов (например, компилятор фортран 5 в процессе компиляции способен выдавать 250 различных сообщений об ошибках).

Библиотека прикладных программ семейства NOVA содержит большое количество программ различного назначения, в частности, пакеты связанных, вычислительных, обработки данных, управления производственными процессами, учебных и игровых программ.

ГЛАВА 9

МНОГОРЕГИСТРОВЫЕ МИНИКОМПЬЮТЕРЫ DEC PDP-11

Первые машины семейства PDP-11 корпорации Digital Equipment появились в начале 70-х годов. В отличие от серии PDP-8, представляющей собой многократное повторение по существу одной и той же машины в различных технологических вариантах, семейство PDP-11 задумано и осуществляется как ряд машин, объединенных единой архитектурой, но значительно отличающихся друг от друга в отношении производительности и цены. Семейство PDP-11 неуклонно расширяется и включает уже около двадцати моделей, покрывающих обширный диапазон мощностей — от микрокомпьютера LSI-11, поставляемого в минимальном варианте с 2КВ памяти, до «супермини» PDP-11/70 с памятью 2МВ и производительностью на уровне средних моделей IBM S/360-370.

В фирменных проспектах только младшие модели семейства PDP-11 называются миникомпьютерами, а к средним и старшим моделям применяется термин «системные компьютеры». Однако это разделение, введенное через несколько лет после появления семейства, по-видимому, с рекламными целями, является произвольным — все члены семейства представляют собой 16-битные машины единой миникомпьютерной архитектуры и различаются в основном по производительности процессоров и максимальной емкости памяти.

Младшие и средние машины выпускаются в двух вариантах: в завершенном оформлении для конечного пользователя (PDP-11/10, 11/20, 11/40) и в виде «полуфабрикатов» для комплектования систем (PDP-

11/04, 11/05, 11/15, 11/35). Старшие модели (PDP-11/45, 50, 55, 60, 70) независимо от назначения поставляются в завершенном варианте. К машинам PDP-11 имеется богатый выбор периферийных устройств, выпускаемых как самой Digital Equipment, так и многими другими фирмами, специализирующимися на производстве минипериферии.

Замечательной чертой семейства PDP-11 является тщательно соблюденная на всех уровнях совместимость аппаратуры, программного оснащения, а также кодов и форматов представления данных на различных носителях. Например, данные, накопленные на бобине магнитной ленты или в кассете самой младшей моделью семейства, могут быть считаны и обработаны на самой старшей модели. Программы, разработанные в одной операционной системе, могут без переделки использоваться в других операционных системах. Пользователь, эксплуатирующий машину младшей или средней модели, может по мере необходимости наращивать ее мощность добавлением новых устройств или даже установкой процессора более старшей модели без риска лишиться возможности выполнения ранее созданных программ.

Разработанная на основе обстоятельного анализа существовавших к концу 60-х годов разновидностей цифровых машин и тенденций их развития архитектура PDP-11 явилась образцовой для 70-х годов, подобно тому, как образцом архитектуры предыдущего десятилетия служила IBM S/360. Она оказывает сильное влияние на новейшие разработки цифровых машин, в частности, микрокомпьютеров, ее изучают, заимствуют, цитируют в учебниках. В результате сравнения ряда получивших широкое признание архитектур цифровых машин, выполненного объединенным комитетом армии и ВМФ США по выбору архитектуры для машин военного применения, предпочтение было отдано архитектуре семейства PDP-11.

§ 1. Форматы команд и данных

Архитектуру миникомпьютеров семейства DEC PDP-11 кратко можно охарактеризовать как 16-битную, с многорегистровой схемой процессора, переменной длиной команд и единой магистралью.

Важными особенностями архитектуры PDP-11¹ является всесторонне развитая система адресации и использование стекового механизма при обращениях к подпрограммам и при обслуживании прерываний.

Основная единица представления данных и управляющей информации в машинах PDP-11 — 16-битное слово, но наряду с этим процессор может выполнять операции с байтами, а команды могут состоять как из одного, так и из двух, а также из трех 16-битных слов. Второе и третье слова команды — это либо адреса, либо непосредственно заданные значения операндов. Таким образом, 16-битное слово интерпретируется процессором или как команда (точнее, первая часть команды), или как адрес, или как значение операнда, которое в свою очередь может быть двоичным числом без знака, двоичным числом со знаком в дополнительном коде, булевским вектором и парой байтов. Кроме того, возможны специальные интерпретации, например, слово состояния процессора.

Нумерация битов в машинном слове PDP-11 осуществляется справа налево, начиная с нуля, т.е. слово W описывается как W(15:0).

Ядром процессора PDP-11 является массив, состоящий из восьми 16-битных регистров общего назначения (общих регистров), пронумерованных, начиная с нуля: R0, R1, R2, ..., R6, R7. Содержимое каждого из этих регистров может быть использовано как значение операнда, или как адрес (составляющая для вычисления адреса) ячейки главной памяти. Выбором способа адресации имеется возможность производить операции над содержимым как регистров, так и ячеек главной памяти с запоминанием результата либо в регистре, либо в ячейке. Другими словами, всякая команда пересылки или преобразования данных в зависимости от используемой адресации может быть выполнена в любой из четырех возможных ее разновидностей: регистр — регистр, регистр — память, память — регистр, память — память. Указание операндов в основном (первом) командном слове осуществляется 6-битными полями адресации. В таком поле 3 младших бита составляют номер регистра, а 3 старших бита определяют способ адресации. Система адресации будет рассмотрена ниже.

Имеются двухоперандные и однооперандные команды преобразования и тестирования данных (рис. 9.1 и рис. 9.2). Двухоперандная команда $K(15:0)$ включает два поля адресации: $K(11:6)$ и $K(5:0)$. Первое содержит указание для вычисления

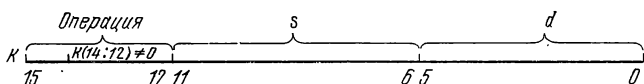


Рис. 9.1. Формат двухоперандной команды PDP-11 (буквами s и d обозначены адресные поля операндов отправления и назначения).

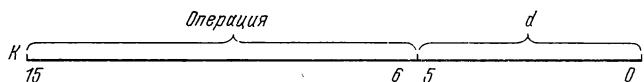


Рис. 9.2. Формат однооперандной команды PDP-11 (буквой d обозначено адресное поле операнда назначения).

адреса отправления (source address), второе — для адреса назначения (destination address). Однооперандная команда включает одно поле адресации, посредством которого определяется адрес назначения, являющийся, разумеется, и адресом отправления.

В двухоперандном формате для кода операции остается 4 бита — $K(15:12)$, т.е. 16 значений. Однако два значения — $K(15:12) = 0$ и $K(15:12) = 10_8$ не используются в качестве кодов операций двухоперандных команд, а указывают на то, что команда не является двухоперандной. При этом код операции наращивается до 10 битов, а в ряде случаев (безадресные команды) занимает все 16-битное слово. В однооперандном формате 10 старших битов — $K(15:6)$ — составляют код операции, а 6 младших — $K(5:0)$ — адрес назначения.

Команда, не являющаяся двухоперандной, может быть однооперандной командой преобразования и/или тестирования данных, командой условного (безусловного) перехода, перехода на подпрограмму, возврата с подпрограммы, установки состояния битов, определяющих условие при переходе, либо одной из команд управления режимом работы процессора — «стоп», «ожидание», «сброс», «возврат из прерывания» и т.п.

Операции преобразования-тестирования данных как одно-, так и двухоперандные (за исключением сложения и вычитания) существуют в двух вариантах: оперирующие с 16-битными словами и оперирующие с 8-битными байтами. Соответственно, каждую из указанных операций представляют две команды — одна для обработки слов, другая для обработки байтов. Команды обработки байтов отличаются от соответствующих команд обработки слов значением головного бита $K(15)$: в командах обработки слов $K(15) = 0$, а в командах обработки байтов $K(15) = 1$. В языке символьного ассемблера мнемокод команды обработки байтов получается добавлением к мнемокоду соответствующей команды обработки слов буквы В (Byte). Например: MOV — «переслать слово», MOVB — «переслать байт», CMP — «сравнить слова», CMPB — «сравнить байты» и т. п.

При выполнении операций над байтами выборка операндов из главной памяти осуществляется побайтно (байт является минимальной адресуемой единицей). В регистрах операция производится над младшим байтом $R(7:0)$.

Главная память рассматривается, с одной стороны, как массив 8-битных байтов $Bm(0:177777_8)$, а с другой стороны, как массив 16-битных слов $m(0:177776_8)$, пронумерованных только четными числами. При этом в каждом слове адрес его младшего байта совпадает с адресом слова в целом, а адрес старшего байта на единицу больше адреса слова.

Например, в слове с адресом 0 байт, соответствующий младшей половине слова, обладает адресом 0, а байт, составляющий старшую половину, обладает адресом 1.

Команды, адреса и прочие элементы программы представимы только целыми словами, поэтому после каждой выборки значение программного счетчика РС автоматически увеличивается на 2: $PC := PC + 2$. Длина операндов (байт/слово), как уже было сказано, определяется в зависимости от кода операции.

Операнды длиной более одного слова используются в связи с операциями, выполняемыми специальным процессором арифметики с плавающей запятой, в котором число с обычной точностью представляется двумя словами, а с удвоенной точностью — четырьмя

словами. В большинстве моделей такой процессор является необязательным дополнением к основному составу машины. В этой книге он не рассматривается.

§ 2. Способы адресации

В архитектуре PDP-11 представлены две независимые системы адресации: одна используется в командах преобразования-тестирования данных, а также в команде безусловного перехода JMP и в команде перехода на подпрограмму, другая — в командах условного перехода и в команде безусловного перехода BR (Branch unconditionally — ответить безусловно). Кроме того, имеется стек, автоматически запоминающий адреса возврата при переходах на подпрограммы и в случае прерываний.

Система адресации, примененная в командах преобразования данных, характерна многообразием включенных в нее способов образования адреса и интенсивным использованием общих регистров процессора. Задание адреса операнда в команде осуществляется при помощи 6-битного поля (в двухоперандной команде два поля), а в случае индексной адресации, кроме того, 16-битного адресного слова (в двухоперандной команде может быть два адресных слова).

Три младших бита адресного поля составляют номер того регистра, содержимое которого используется при вычислении адреса операнда, а три старших бита указывают способ вычисления. Младший бит старшей тройки, находясь в состоянии 1, означает косвенную адресацию. Два других бита составляют код, четырем значениям которого сопоставлены следующие виды адресации: 0 — регистровая, 1 — автоинкрементная, 2 — автодекрементная, 3 — индексная. В зависимости от значения младшего бита (0 или 1) адресация каждого из этих четырех видов может быть либо прямой, либо косвенной. Таким образом всего имеется 8 различных способов вычисления исполнительного адреса, сопоставленных 8 значениям тройки старших битов ms адресного поля (см. табл. 9.1).

Регистровая прямая адресация ($ms = 0$) означает, что указываемая командой операция должна выпол-

Способы адресации в машинах PDP-11

| мс | Мнемокод | Название адресации | Описание |
|----|----------|----------------------------|---------------------------------------|
| 0 | Rn | регистровая прямая | ORN: = n; |
| 1 | @ Rn | регистровая косвенная | EA: = Rn; |
| 2 | (Rn) + | автоинкрементная прямая | EA: = Rn; Incr(Rn); |
| 3 | @ (Rn) + | автоинкрементная косвенная | EA: = m(Rn); Rn: = Rn + 2; |
| 4 | — (Rn) | автодекрементная прямая | Decr(Rn); EA: = Rn; |
| 5 | @ — (Rn) | автодекрементная косвенная | Rn: = Rn — 2; EA: = m(Rn); |
| 6 | X(Rn) | индексная прямая | EA: = Rn + m(PC); PC: = PC + 2; |
| 7 | @X (Rn) | индексная косвенная | EA: = m(Rn + m(PC)); PC: = PC + 2; |

Случаи, когда в качестве Rn используется PC, т. е. n = 7:

| | | | |
|---|-------|-------------------------|--|
| 2 | # A | непосредственная | EA: = PC; PC: = PC + 2; |
| 3 | @ # A | абсолютная | EA: = m(PC); PC: = PC + 2; |
| 6 | A | относительная | EA: = m(PC); PC: = PC + 2; EA: = EA + PC; |
| 7 | @ A | относительная косвенная | EA: = m(PC); PC: = PC + 2; EA: = m(EA + PC); |

Примечание. Описание функций Incr(Rn), Decr(Rn):
 Incr(Rn): if K(15) = 1 \wedge K(15:12) \neq 16₈ \wedge n < 6 then Rn: = Rn + 1 else Rn: = Rn + 2; Decr(Rn): if K(15) = 1 \wedge K(15:12) \neq 16₈ \wedge n < 6 then Rn: = Rn — 1 else Rn: = Rn — 2;

няться над содержимым регистра Rn, номером которого ORN является значение n младшей тройки битов адресного поля. Короче говоря, операции производятся над регистрами — имя регистра является именем переменной, текущее значение которой содержит этот регистр.

Например, операция R1 := 0 очистки регистра R1 в мнемонических обозначениях языка ассемблера

записывается в виде

CLR %1.

Если учесть, что знак % заменяет обозначение регистра — R, а мнемокод CLR является сокращением слова CLeaR, то данная мнемоника гласит: Clear R1, т.е. «очистить R1».

Двухоперандная команда вычитания с прямой регистровой адресацией обоих операндов записывается в виде, например:

SUB R3, R2.

Здесь указанный первым R3 — регистр отправления, R2 — регистр назначения.

Операция выполняется так:

$R2 := R2 - R3,$

т.е. вычитаемым является операнд отправления, а уменьшаемым — операнд назначения.

Регистровая косвенная адресация ($mc = 1$) заключается в том, что содержимое названного в команде регистра рассматривается как адрес той ячейки главной памяти, над которой должна производиться операция. Запись на языке ассемблера отличается от рассмотренной только тем, что мнемоники прямой регистровой адресации наличием впереди знака косвенной адресации (@).

Например, запись

INC @R5

означает INCrement @R5, т.е. прирастить содержимое ячейки, адресуемой косвенно по регистру R5. Эта команда выполняется следующим образом:

$m(R5) := m(R5) + 1.$

Другой пример косвенной регистровой адресации — пересылка (копирование) содержимого ячейки, указываемой одним регистром, в ячейку, указываемую другим регистром:

MOV @R3, @R4.

Это выполняется так:

$m(R4) := m(R3).$

В двухоперандной команде адресация операндов независима и может осуществляться для одного операнда одним, а для другого — другим способом. Например, в команде

ADD @R3, R2

операнд отправления адресуется косвенно, а операнд назначения — прямо. Команда предписывает прибавить к содержимому регистра R2 содержимое ячейки памяти, указываемой регистром R3:

$$R2 := R2 + m(R3).$$

Чтобы получить сумму содержимого регистра и указываемой другим регистром ячейки в этой ячейке, применяется косвенная адресация операнда назначения:

ADD R2, @R4.

Эта команда выполняется так:

$$m(R4) := m(R4) + R2.$$

Команда пересылки байта

MOVB @R4, R4

копирует содержимое однобайтной ячейки памяти $Bm(R4)$, рассматриваемое как число со знаком $sgnvl(Bm(R4))$, в регистр назначения R4:

$$R4 := sgnvl(Bm(R4)).$$

При этом во все биты старшего байта регистра R4 распространяется значение головного («знакового») бита в $Bm(R4)$. Такое наращивание байта до полного слова является исключительной принадлежностью операции **MOVB**. Все другие операции над байтами касаются только младшей половины регистра, не затрагивая его старшего байта. Например, операция очистки, задаваемая командой

CLRB R5

очищает только младший байт регистра R5, оставляя в сохранности содержимое старшего байта этого регистра:

$$R5(7:0) := 0.$$

Автоинкрементная прямая адресация ($mc = 2$) — точнее, поставтоинкрементная прямая — так же, как регистровая косвенная, использует содержимое регистра в качестве адреса операнда, однако затем производится приращение этого содержимого прибавлением единицы в случае адресации байта и прибавлением двойки в случае адресации слова. С учетом того, что в командах обработки слов, за исключением команды вычитания (код операции 168), $K(15) = 0$, а также того, что по регистру R6, который является указателем SP стека, и по регистру R7, который является программным счетчиком PC, можно адресовать только слова, процедура автоинкрементной прямой адресации выражается следующим образом:

$EA := R_n$; if $K(15) = 1 \wedge K(15:12) \neq 16_8 \wedge n < 6$ then $R_n := R_n + 1$ else $R_n := R_n + 2$.

Эта адресация особенно эффективна при обработке массивов и таблиц. Кроме того, она в совокупности с рассматриваемой ниже автодекрементной адресацией обеспечивает возможность легко программировать стеки.

На языке ассемблера автоинкрементная прямая адресация по регистру R_n записывается в виде

$(R_n) +$.

Знак $+$, поставленный *после* заключенного в скобки обозначения регистра, указывает на то, что регистр используется в поставтоинкрементном режиме.

Например, команда

ADD (R3) +, R2

предписывает выполнить следующее:

$R2 := R2 + m(R3)$; $R3 := R3 + 2$;

т.е. прибавить к содержимому регистра назначения R2 содержимое ячейки, указываемой регистром R3, и прирастить значение этого регистра-указателя, прибавив к нему двойку. Чтобы получить в регистре R2 сумму k чисел, находящихся в последовательных ячейках главной памяти, начиная с ячейки, адрес которой содержится в регистре R3, достаточно просто выполнить данную команду k раз. При каждом выполнении ее в результате автоматического

приращения регистра-указателя R3 исполнительный адрес будет последовательно увеличиваться, всякий раз указывая ячейку памяти, в которой содержится очередное слагаемое.

Поэлементное сложение двух k -элементных массивов осуществляется выполнением k раз команды

ADD (R2) + , (R3) +

при условии, что в регистре R2 находится адрес начального элемента массива отправления, а в регистре R3 — адрес начального элемента массива назначения. Команда всякий раз выполняется так:

$m(R3) := m(R3) + m(R2); R2 := R2 + 2;$
 $R3 := R3 + 2.$

Автоинкрементная косвенная адресация ($mc = 3$) осуществляется также с последующим приращением, однако содержимое регистра используется не как прямой, а как косвенный адрес операнда. Запись на языке ассемблера содержит впереди символ косвенной адресации @:

@(Rn) +.

Вычисление соответствующего исполнительного адреса выполняется так:

$EA := m(Rn); Rn := Rn + 2.$

Например, команда приращения значения операнда, косвенно адресуемого с помощью автоинкрементной по регистру R4 адресации

INC @(R4) +,

вызывает выполнение следующих действий:

$m(m(R4)) := m(m(R4)) + 1; R4 := R4 + 2.$

Аналогичная команда, относящаяся к операнду-байту:

INCB @(R4) +,

осуществляется, соответственно, так:

$Bm(m(R4)) := Bm(m(R4)) + 1; R4 := R4 + 2.$

Автодекрементная прямая адресация ($mc = 4$) — это преавтодекрементная прямая адресация. Сначала

содержимое указанного в адресном поле регистра убавляется вычитанием единицы (если адрес относится к байту) или двойки, а затем используется в качестве адреса ячейки памяти. В языке ассемблера символом предекрементности служит знак «минус», поставленный перед заключенным в скобки обозначением регистра:

— (Rn).

Действия по нахождению исполнительного адреса в случае прямой автодекрементной адресации можно описать следующим образом:

if K(15) = 1 \wedge K(15:12) \neq 16₈ \wedge n < 6 **then**
Rn := Rn - 1 **else** Rn := Rn - 2; EA := Rn.

В противоположность автоинкрементной адресации автодекрементная адресация связана с последовательным убавлением адреса. Кроме того, она является не пост-, а преиндексацией, т. е. при реализации стека, в котором указатель указывает последнюю занятую (а не следующую за ней свободную) ячейку она должна использоваться в командах засылки в стек, причем заполнение стека будет происходить путем убывания адреса ячейки-вершины. Например, если в качестве указателя стека применен регистр R5, то засылка в стек содержимого регистра R2 осуществляется командой

MOV R2, — (R5),

которая реализуется следующей последовательностью действий:

R5 := R5 - 2; m(R5) := R2.

Обратная операция выталкивания из данного стека в регистр R2 задается командой с поставтоиндексной адресацией операнда отправления

MOV (R5) +, R2,

которая вызовет пересылку из стека в регистр R2 и освободит ячейку, являющуюся вершиной стека:

R2 := m(R5); R5 := R5 + 2.

Автодекрементная косвенная адресация (mc = 5) как и косвенная автоинкрементная, заключается в употреб-

лении содержимого регистра в качестве косвенного адреса.

Запись на языке ассемблера

@ — (Rn)

вызывает следующее вычисление исполнительного адреса

$R_n := R_n - 2; EA := m(R_n).$

Например, команда сложения

ADD R3, (@ — (R0))

выполняется так:

$R_0 := R_0 - 2; m(m(R_0)) := m(m(R_0)) + R_3.$

Индексная прямая адресация ($m_6 = 6$) заключается в том, что адрес операнда вычисляется как сумма содержимого регистра R_n , номер которого n указан в поле адресации, и адресного слова — второго или третьего слова команды. Если в команде имеется один операнд с индексной адресацией, то адресным служит второе слово, а если операндов с индексной адресацией два, то адресным для операнда отправления служит второе слово, а для операнда назначения — третье слово команды.

На языке ассемблера прямая индексная адресация указывается в виде записи в скобках обозначения регистра, перед которой стоит значение X , помещаемое ассемблером во второе или в третье слово команды.

$X (R_n).$

Вычисление эффективного адреса при выполнении команды с прямой индексной адресацией операнда осуществляется следующим образом:

$EA := R_n + m(PC); PC := PC + 2;$

где PC — программный счетчик. Следует иметь в виду, что приращение значения PC производится сразу же после выборки команды, т.е. текущее значение PC во время выполнения команды равно значению адреса следующего за выполняемой командой слова программы, а после выборки этого слова — значению адреса следующего за ним (третьего) слова.

Например, команда сложения с прямой индексной адресацией обоих операндов

ADD 20 (R0), 1020 (R2)

в рабочей программе будет представлена тремя словами, причем значением второго слова является 20, и значением третьего слова — 1020. Выполнение этой команды можно описать следующим образом, введя переменную TMP для временного запоминания значения операнда:

$$\begin{aligned} \text{TMP} &:= m(R0 + m(PC)); \text{PC} := \text{PC} + 2; \\ m(R2 + m(PC)) &:= m(R2 + m(PC)) + \text{TMP}; \\ \text{PC} &:= \text{PC} + 2. \end{aligned}$$

Заметим, что $m(PC)$ в первой строке доставляет значение 20, а после прибавления к PC двойки, во второй строке $m(PC)$ означает выборку третьего слова команды, содержащего 1020.

В качестве значения адресного слова в команде с индексной адресацией может быть указан нуль: 0 (Rn) — в этом случае прямая индексная адресация дает тот же результат, что и косвенная регистровая @Rn, но, в отличие от последней, требует два командных слова.

Индексная косвенная адресация ($m_c = 7$) отличается от прямой индексной тем, что вычисленный рассмотренным только что способом адрес EA является не прямым, а косвенным адресом операнда, т.е. адресом, по которому находится прямой адрес. Косвенная индексная адресация обозначается на языке ассемблера в виде

@ X (Rn).

Исполнительный адрес операнда вычисляется так:

$$\text{EA} := m(Rn + m(PC)); \text{PC} := \text{PC} + 2.$$

Например, команда пересылки

MOV @ 2040 (R2), R4,

в которой операнд отправления адресуется посредством косвенной индексной адресации, выполняется следующим образом:

$$\text{R4} := m(m(R2 + m(PC))); \text{PC} := \text{PC} + 2.$$

При выполнении этой записи значение переменной $m(PC)$, содержащееся во втором слове команды, будет равно 2040, сумма $R2 + 2040$ явится косвенным адресом операнда, по которому будет найден прямой адрес $EA = m(R2 + 2040)$ и, наконец, значение операнда — $m(EA)$.

Подытоживая рассмотрение способов адресации, используемых в командах преобразования-тестирования данных в машинах PDP-11, приведем общий алгоритм вычисления исполнительного адреса EA операнда или номера ORN регистра, в котором находится операнд.

Алгоритм вычисления исполнительного адреса EA или номера ORN регистра, содержащего значение операнда, по элементам mc и n адресного поля команды преобразования-тестирования данных в машинах PDP-11.

```

if  $mc = 0$  then  $ORN := n$  else
if  $mc = 1$  then  $EA := R_n$  else
if  $mc = 2$  then
    begin  $EA := R_n$ ; if  $K(15) = 1 \wedge K(15:12) \neq 16_8 \wedge n < 6$  then
         $R_n := R_n + 1$  else  $R_n := R_n + 2$ 
    end else
if  $mc = 3$  then begin  $EA := m(R_n)$ ;  $R_n := R_n + 2$ 
end else
if  $mc = 4$  then
    begin if  $K(15) = 1 \wedge K(15:12) \neq 16_8 \wedge n < 6$ 
        then  $R_n := R_n - 1$  else
         $R_n := R_n - 2$ ;  $EA := R_n$ 
    end else
if  $mc = 5$  then begin  $R_n := R_n - 2$ ;
 $EA := m(R_n)$  end else
if  $mc = 6$  then begin  $EA := R_n + m(PC)$ ;
 $PC := PC + 2$  end else
if  $mc = 7$  then begin  $EA := m(R_n + m(PC))$ ;
 $PC := PC + 2$  end

```

В том случае, когда используемый при индексной адресации регистр содержит нуль, значение присоеди-

ненного к команде адресного слова является абсолютным адресом, прямым или косвенным, если адресация косвенная. Однако прямая абсолютная адресация осуществима и другим, рассматриваемым ниже более экономным путем — как косвенная автоинкрементная относительно регистра РС.

Программным счетчиком РС в машинах PDP-11 является, как известно, регистр R7. Это означает, что помимо возложенных на него специальных функций программного счетчика, данный регистр способен выполнять также все функции общего регистра, в частности, функции, выполняемые общими регистрами в системе адресации. Однако то, что регистр R7 является программным счетчиком придает осуществляемым с его участием способам адресации специфический характер. Особый интерес представляет автоинкрементная и индексная адресация с регистром R7.

Автоинкрементная прямая адресация с регистром R7 в фирменных руководствах DEC названа *непосредственной* адресацией на том основании, что значение операнда, адресуемого этим способом, находится непосредственно при команде, в следующем за ней слове программы. На языке ассемблера значение А операнда также включается непосредственно в команду, причем перед ним ставят знак # вот так:

А.

На самом деле данному способу адресации свойственны бóльшие возможности, чем непосредственной адресации в обычном ее понимании, адресуемая ячейка доступна не только для чтения, но и для записи. Запись в нее производится в случае непосредственной адресации операнда назначения. Другое дело, целесообразно ли пользоваться этой возможностью. Как автоинкрементная прямая эта адресация выражается в виде

(РС) +

и реализуется, соответственно, так:

EA := РС; РС := РС + 2.

Поскольку в РС при выполнении команды находится адрес следующей за ней ячейки, то полу-

ченный адрес ЕА как раз и является адресом этой ячейки.

Примером команды с непосредственной адресацией операнда может служить команда вычитания

SUB # 4,R3,

предписывающая вычесть четверку из содержимого регистра R3:

$$R3 := R3 - 4; PC := PC + 2.$$

В действительности процессор реализует это в виде

$$R3 := R3 - m(PC); PC := PC + 2.$$

Автоинкрементная косвенная адресация с регистром R7 является *абсолютной* адресацией. Запись на языке ассемблера

@ # A,

где A — значение абсолютного адреса, помещаемое ассемблером в следующую за командой ячейку программы, т.е. присоединяемое в качестве второго (или третьего) слова команды. В качестве исполнительного адреса, вычисляемого по правилу автоинкрементной косвенной адресации, используется содержимое именно этой ячейки:

$$EA := m(PC); PC := PC + 2.$$

Пример команды с абсолютной адресацией — безусловный переход по адресу 2020:

JMP 2020,

выполнение которой можно описать следующим образом:

$$EA := m(PC); PC := PC + 2; PC := EA.$$

Индексная прямая адресация с регистром R7 является *относительной*, потому что исполнительный адрес операнда, записанного на языке ассемблера в виде

X (PC),

вычисляется как смещение на величину X относительно текущего значения PC:

$$X := m(PC); PC := PC + 2; EA := PC + X.$$

Ассемблер допускает другую, более удобную форму задания операнда символьным или числовым именем (меткой) ячейки, содержащей значение операнда. В рабочей программе осуществляется индексная адресация этой ячейки относительно текущего значения РС.

Например, команда, записанная на языке ассемблера в виде

INC A,

в рабочей программе будет заменена двухсловной командой с индексной относительно РС адресацией и значением индекса $X = A - (IA + 2)$, где IA — адрес ячейки, запоминающей значение X адресного слова команды.

Относительная адресация очень полезна как средство обеспечения позиционной независимости программ.

Индексная косвенная адресация с регистром R7 представляет собой *косвенную относительную* адресацию вида $@X(PC)$. На языке ассемблера операнд задают в виде $@A$, где A — символьный или числовой косвенный адрес, которому в рабочей программе соответствует значение X , вычисляемое ассемблером по приведенной выше формуле. Исполнительный адрес операнда содержится в ячейке, обладающей адресом A .

Независимая система относительной адресации применена в командах условного, а также безусловного перехода типа Branch (рис. 9.3). Старшие 8 битов

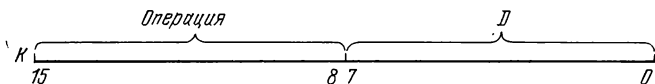


Рис. 9.3. Формат команды перехода.

$K(15:8)$ команды этого типа составляют код операции, а младшие 8 битов $K(7:0)$ — адресную часть (Offset), которая интерпретируется как представленное в дополнительном коде число со знаком, выражающее смещение относительно текущего значения РС, измеренное в машинных словах. При этом, в случае если переход производится, новое значение

PC определяется так:

$$PC := PC + 2 * \text{sgnvl}(K(7:0)).$$

Иначе говоря, переход возможен в пределах от -128 до $+127$ слов относительно текущего значения программного счетчика или, соответственно, в пределах от -256 до $+254$ байтов, однако только на четное число байтов, т.е. на целое число машинных слов.

На языке ассемблера в командах типа Branch, как и в использующих относительную адресацию командах преобразования-тестирования данных, задается символьный или числовой адрес (метка) команды, на которую производится переход. Если этот адрес находится в допустимых пределах относительно адреса IA самой команды перехода, т.е. если $-254 \leq A - IA \leq 256$, то ассемблер порождает машинную команду перехода с соответствующим относительным адресом (Offset). В противном случае печатается сообщение об ошибке.

§ 3. Команды преобразования-тестирования данных

Операции, задаваемые командами преобразования-тестирования данных, выполняются над операндами, выборка которых из регистров процессора или из ячеек главной памяти производится в соответствии с рассмотренными в предыдущем параграфе способами адресации. Операнд указывается в команде при помощи 6-битного адресного поля, содержащего код адресации (mc) и номер регистра (n). Двухоперандные команды имеют по два адресных поля: одно для задания операнда отправления — s (source), другое для задания операнда назначения — d (destination). Операнд однооперандной команды условно называется операндом назначения. Буквы s и d, которыми обозначаются операнды при описании операций, можно рассматривать как имена регистров (ячеек памяти), содержащих значения соответствующих операндов и принимающих результат операции (см. табл. 9.2 и 9.3).

Каждая операция сопровождается тестированием полученного результата и выработкой «кода условия»,

Однооперандные команды PDP-11

| Код K(15:6) ₈ | Мнемокод | Команда | Описание операции |
|-----------------------------|----------|--|----------------------|
| 0050 | CLR | Clear — очистить | d := 0; |
| 1050 | CLRB | Clear Byte — очистить байт | |
| 0051 | COM | Complement — инвертировать | d := d; |
| 1051 | COMB | Complement Byte — инвертировать байт | |
| 0052 | INC | Increment — прирастить | d := d + 1; |
| 1052 | INCB | Increment Byte — прирастить байт | |
| 0053 | DEC | Decrement — убавить | d := d - 1; |
| 1053 | DECB | Decrement Byte — убавить байт | |
| 0054 | NEG | Negate — изменить знак | d := d + 1; |
| 1054 | NEGB | Negate Byte — изменить знак байта | |
| 0055 | ADC | Add Carry — сложить с переносом | d := d + C; |
| 1055 | ADCB | Add Carry Byte — сложить с переносом байт | |
| 0056 | SBC | Subtract Carry — вычесть перенос | d := d - C; |
| 1056 | SBCB | Subtract Carry Byte — вычесть перенос из байта | |
| 0057 | TST | Test — тестировать | 0 - d |
| 1057 | TSTB | Test Byte — тестировать байт | |
| 0060 | ROR | Rotate Right — циклически сдвинуть вправо | dC := Cd |
| 1060 | RORB | Rotate Right Byte — циклически сдвинуть вправо байт | |
| 0061 | ROL | Rotate Left — циклически сдвинуть влево | Cd := dC; |
| 1061 | ROLB | Rotate Left Byte — циклически сдвинуть влево байт | |
| 0062 | ASR | Arithmetic Shift Right — арифметически сдвинуть вправо | d(14:0) C; := d; |
| 1062 | ASRB | Arithmetic Shift Right Byte — арифметически сдвинуть вправо байт | |
| 0063 | ASL | Arithmetic Shift Left — арифметически сдвинуть влево | Cd := 2 × d; |
| 1063 | ASLB | Arithmetic Shift Left Byte — арифметически сдвинуть влево байт | |
| 0001 | JMP | Jump — перейти | PC := EA; |
| 0003 | SWAB | Swap Bytes — переставить байты | d := d(7:0) d(15:8); |

Двухоперандные команды PDP-11

| Код K(15:12) ₈ | Мнемокод | Команда | Описание операции |
|------------------------------|----------|--|--------------------------|
| 01 | MOV | Move — переслать | $d := s;$ |
| 11 | MOVB | Move Byte — переслать байт | |
| 02 | CMP | Compare — сравнить | $s - d;$ |
| 12 | CMPB | Compare Byte — сравнить байты | |
| 03 | BIT | Bit Test — тестировать биты | $s \wedge d;$ |
| 13 | BITB | Bit Test Byte — тестировать биты в байте | |
| 04 | BIC | Bit Clear — очистить биты | $d := d \wedge \bar{s};$ |
| 14 | BICB | Bit Clear Byte — очистить биты в байте | |
| 05 | BIS | Bit Set — установить биты | $d := d \vee s;$ |
| 15 | BISB | Bit Set Byte — установить биты в байте | |
| 06 | ADD | Add — сложить | $d := d + s;$ |
| 16 | SUB | Subtract — вычесть | $d := d - s;$ |

состоящего из четырех битов, обозначенных буквами Z, N, C, V, и характеризующих результат операций следующим образом:

Z (Zero) принимает значение 1, если результат (без учета переполнения) равен нулю, т.е. если $R(15:0) = 0$ при обработке слов или $R(7:0) = 0$ при обработке байтов;

N (Negative) принимает значение 1, если результат (без учета переполнения) отрицателен, т.е. если $R(15) = 1$ в случае слова или $R(7) = 1$ в случае байта;

C (Carry) принимает значение цифры переноса из старшего бита;

V (Overflow) принимает значение 1, если имеет место арифметическое в дополнительном коде переполнение.

Значения битов «кода условия» используются при выполнении условных переходов как информация, на основе которой принимается решение, производить переход или не производить. Отсюда и название «код условия».

Имеются команды, выполнение которых не производит никакого результата, кроме установки битов

«кода условия», т. е. полученный результат тестируется, но не запоминается. Например, команда CMP (Compare — «сравнить») выполняется путем вычитания *s* — *d* и выработки «кода условия», однако запоминание значения полученной разности не происходит и таким образом операнд назначения, обычно утрачиваемый при запоминании результата остается в сохранности. Вместе с тем информация, воспринятая битами «кода условия» достаточна для того, чтобы путем выполнения соответственно подобранной команды условного перехода установить, равны или не равны значения операндов и какое из них является меньшим или большим.

Почти все команды преобразования-тестирования данных, задающие операции над словами, имеются также в разновидности, оперирующей с байтами. Исключение составляют команды сложения, вычитания, перехода на подпрограмму, безусловного перехода (эти две не относятся по существу к командам преобразования данных, но включены в их число, потому что используют те же способы адресации) и перестановки байтов в слове-операнде — операции, задаваемые этими командами потребляют в качестве операндов только целые машинные слова. Как уже было сказано, адреса ячеек главной памяти интерпретируются как адреса слов или как адреса байтов в зависимости от того в какой команде эти адреса использованы — относящейся к словам или относящейся к байтам.

Некоторые команды используют не только значения адресуемых операндов *s* и *d*, но также значение бита переноса *C*, который, как и три других бита «кода условия», выступает в качестве подразумеваемого (неявно адресуемого) операнда.

Кроме того, что значения битов «кода условия» вырабатываются при выполнении команд преобразования-тестирования данных, имеется специальная команда-микропрограмма, позволяющая присвоить любому из этих битов значение 0 или 1. Конкретные варианты этой команды обозначаются на языке ассемблера соответствующей мнемоникой.

Например: SEC (Set *C* — «установить *C* = 1»), CLC (Clear *C* — «очистить *C*»), CLV (Clear *V* — «очистить *V*») и т. д.

§ 4. Средства управления ходом программы

Архитектура PDP-11 предоставляет богатые возможности управления ходом программы в виде 14 команд условного перехода, (см. табл. 9.4), которые в фирменных описаниях названы командами условного ветвления (Conditional branch), стекового механизма гнездования подпрограмм и прерываний, а также ряда специальных команд, обеспечивающих легкость и быстроту программированного реагирования на те или иные возможные в ходе выполнения программы ситуации.

Каждая команда условного перехода предписывает переход на одну из команд, адрес которой относительно адреса самой команды условного перехода отстоит на более, чем на -254 или $+256$. Переход на большее расстояние реализуется с помощью команды JMP, обладающей возможностями адресации, свойственными командам преобразования-тестирования данных.

С каждой из команд условного перехода связано определенное условие, при выполнении которого происходит переход. Это условие сформулировано в виде отношения, которым связываются значения битов «кода условия» Z, N, C, V, но поскольку значения данных битов характеризуют результат предыдущей операции преобразования данных, то по существу условие налагает определенные ограничения на величину этого результата, и следовательно, на соотношение величин операндов, над которыми произведена конкретная операция. Например, задаваемый командой BEQ переход по условию $Z = 1$ (результат равен нулю) после операции вычитания свидетельствует о том, что операнды этой операции равны друг другу.

Наличие команды сравнения (вычитания без запоминания результата) позволяет путем сочетания ее с командами перехода по различным условиям, связывающим значения Z, N, C, V, конструировать аналоги команд перехода по результату сравнения двух чисел. Мнемоника команд условного перехода PDP-11 отражает соотношения между сравниваемыми числами и вместе с тем свойства результата, соответствующие условиям переходов. В приведенном выше

Команды условных переходов PDP-11

| Код | Мнемокод | Команда | Описание операции $D = 2 \times \text{sgnvl}(K(7:0))$ |
|------|---------------|--|--|
| 0004 | BR | Branch unconditionally – перейти безусловно | $PC := PC + D;$ |
| 0010 | BNE | Branch if. Not Equal (zero) – перейти по не равенству (нулю) | if $Z = 0$ then $PC := PC + D;$ |
| 0014 | BEQ | Branch if Equal (zero) – перейти по равенству (нулю) | if $Z = 1$ then $PC := PC + D;$ |
| 0020 | BGE | Branch if Grater or Equal (zero) – перейти, если больше или равно (нулю) | if $N = V$ then $PC := PC + D;$ |
| 0024 | BLT | Branch if Less then (zero) – перейти, если меньше чем (нуль) | if $N \neq V$ then $PC := PC + D;$ |
| 0030 | BGT | Branch if Grater then (zero) – перейти, если больше чем (нуль) | if $(Z = 0) \wedge (N = V)$ then $PC := PC + D;$ |
| 0034 | BLE | Branch if Less or Equal (zero) – перейти, если меньше или равно (нулю) | if $(Z = 1) \vee (N \neq V)$ then $PC := PC + D;$ |
| 1000 | BPL | Branch if Plus – перейти, если не отрицательно | if $N = 0$ then $PC := PC + D;$ |
| 1004 | BMI | Branch if Minus – перейти, если отрицательно | if $N = 1$ then $PC := PC + D;$ |
| 1010 | BHI | Branch if Higher – перейти, если выше | if $(C \vee Z) = 0$ then $PC := PC + D;$ |
| 1014 | BLOS | Branch if Lower or Same – перейти, если не выше | if $(C \vee Z) = 1$ then $PC := PC + D;$ |
| 1020 | BVC | Branch if overflow Clear – перейти, если нет переполнения | if $V = 0$ then $PC := PC + D;$ |
| 1024 | BVS | Branch if overflow Set – перейти по переполнению | if $V = 1$ then $PC := PC + D;$ |
| 1030 | BHIS (BCC) | Branch if Higher or Same – перейти, если не ниже | if $C = 0$ then $PC := PC + D;$ |
| 1034 | BLO (BCS) | Branch if Lower – перейти, если ниже | if $C = 1$ then $PC := PC + D;$ |

примере перехода по $Z = 1$ команда BEQ гласит: «перейти по равенству (нулю)», что следует понимать (в зависимости от того, тестируется равенство операндов или равенство нулю результата) либо как «перейти, если значения операндов равны», либо как «перейти, если значение результата равно нулю».

Среди команд условного перехода имеются команды, в которых условие формулируется применительно к интерпретации двоичных слов как чисел без знака, и команды, предназначенные для сопоставления чисел со знаком, представленных в дополнительном коде. Условия для чисел со знаком выражены с помощью слов «больше», «меньше» (Greater, Less), а те же условия для чисел без знака — с помощью слов «выше», «ниже» (Higher, Lower). Например, команда BLO («перейти, если ниже»), употребленная после команды сравнения, вырабатывающей «код условия» для разности $s - d$, производит переход, если s и d , рассматриваемые как числа без знака, удовлетворяют соотношению $s < d$. Другими словами, команда BLO в действительности означает: «перейти, если интерпретируя s и d как числа без знака, имеем: s меньше («ниже») чем d ». Впрочем, эта же команда допускает и более простое истолкование и имеет соответственно еще одно мнемоническое обозначение — BCS Branch if Carry Set — «перейти, если $C = 1$ ». На самом деле переход производится при условии $C = 1$, но при вычитании чисел без знака результат удовлетворяет этому условию только в том случае, когда уменьшаемое меньше вычитаемого, т.е. $s < d$.

Приведем примеры простейших программ на языке ассемблера PAL-11.

Программа вычисления разности $c = a + b$:

| | | |
|--------|----------|------------------|
| | . = 200 | ; m(C) := m(A) |
| START: | MOV A, C | |
| | SUB B, C | ; m(C) := m(C) - |
| | | - m(B) |
| | HALT | ; stop |
| A: | . . . | ; значение a |
| B: | . . . | ; значение b |
| C: | . . . | ; значение c |
| | . END | |

Здесь запись $. = 200$, в которой точка символизирует программный счетчик PC, задает начальное значение $PC = 200$, другими словами, заменяет метку START адресом 200_8 . Запись $.END$ означает конец текста программы.

Программа, формирующая матрицу битов с единицами на диагонали.

```

                                R0 = %0
                                R1 = %1
                                . = 200
START:    CLC                                ;очистить бит
                                                ;переноса C
                                MOV # 1,R0    ;R0:= 1
                                MOV M0, R1    ;R1:= M0
CYCLE:    MOV R0,(R1) +                    ;m(R1):= R0;
                                                ;R1:= R1 + 2
                                ROL R0        ;CR0:= R0C
                                BCC CYCLE     ;if C = 0 then go
                                                ;to CYCLE
                                HALT          ;stop

M0:       . . .
                                .END         ;адрес начальной
                                                ;строки матрицы

```

Первые две строки данной программы — $R0 = \%0$ и $R1 = \%1$ — вводят идентификаторы R0 и R1 для регистров процессора, обозначаемых в языке ассемблера символами $\%0$ и $\%1$. Команда $MOV \# 1,R0$ засылает в регистр R0 константу 000001_8 , которая является значением начальной строки формируемой матрицы и используется для получения других строк вращением влево в кольце CR0 (бит переноса C — регистр R0). Формирование матрицы оканчивается, как только сдвигаемая в CR0 единица достигнет бита C.

Программа суммирования содержимого N последовательных ячеек.

```

                                R0 = %0
                                R1 = %1
                                R2 = %2
                                . = 200
START:    MOV N,R0                        ; R0:= m(N)
                                MOV A1,R1  ; R1:= m(A1)
                                CLR R2     ; R2:= 0

```

| | | |
|------|----------------|--|
| L: | ADD (R1) + ,R2 | ; R2 := R2 + + m(R1); R1 := R1 + 2 |
| | DEC R0 | ; R0 := R0 - 1 |
| | BGT L | ; if R0 > 0 then go to L |
| | MOV R2,SUM | ; m(SUM) := R2 |
| | HALT | ; stop |
| N: | . . . | ; число суммируе- мых ячеек |
| A1: | | ; адрес первой ячейки |
| SUM: | . . . | ; сумма |
| | .END | |

Сравнение данных программ с соответствующими программами для PDP-8, приведенными в гл. 6, показывает, что программы на PAL-11 представляются меньшим числом строк, однако это не означает экономии памяти даже по числу занятых программой слов, а тем более по числу битов.

Наряду с командами условного перехода в арсенале средств управления PDP-11 имеется аппарат автоматического запоминания адреса возврата при переходе на подпрограмму, реализованный на основе стека. Наличие автоинкрементной и автодекрементной адресации позволяет легко программировать стек, используя в качестве указателя один из общих регистров. Но помимо этой возможности, имеется стек, который не надо специально программировать — засылка в этот стек производится автоматически при выполнении команд перехода на подпрограммы и при переходе в прерывания, а выталкивание из него — при выполнении команд возврата с подпрограмм и из прерываний. Указателем этого системного или адресного стека служит регистр R6, обозначаемый также SP (Steck Pointer), подобно тому как регистр R7 обозначается PC. Он нормально выполняет все функции общего регистра, за исключением побайтной автоинкрементной и автодекрементной адресации, при которой приращения (убавления) в R6, как и в R7, производятся всегда двойками. Функционирование адресного стека рассмотрено ниже на примере команды перехода на подпрограмму.

Команда перехода на подпрограмму JSR задается в виде

JSR R_n, SUBR,

где R_n — обозначение используемого регистра, представляемого в коде команды номером n,

SUBR — обозначение начального адреса подпрограммы, который в коде команды представляется 6-битным адресным полем с помощью рассмотренных ранее способов адресации.

Выполнение команды JSR заключается в том, что содержимое регистра R_n засылают в адресный стек, текущее значение счетчика команд (адрес возврата) помещают в регистр R_n, а вычисленный по имеющимся в команде данным начальный адрес подпрограммы SUBR принимают в счетчик команд PC. Например, в случае относительной адресации начала подпрограммы приведенная выше команда JSR реализуется так:

$$X := m(PC); PC := PC + 2; SP := SP - 2;$$
$$m(SP) := R_n; R_n := PC; PC := PC + X.$$

Здесь переменная X принимает хранящееся во втором слове команды относительное значение адреса начала подпрограммы, равное SUBR — PC — 2, где PC — текущее значение программного счетчика, равное абсолютному адресу второго слова команды JSR.

После выполнения подпрограммы возврат на продолжение вызвавшей ее программы производится командой Return from Subroutine («возврат из подпрограммы»), которой оканчивается всякая подпрограмма. Эта команда, записываемая на языке ассемблера в виде

RTS R_n

вызывает передачу в PC хранящегося в R_n адреса возврата и восстановление прежнего значения R_n, которое при переходе на подпрограмму было упрятано в системный стек:

$$PC := R_n; R_n := m(SP); SP := SP + 2.$$

Сохранение адреса возврата в регистре процессора, а не в стеке, осуществлено с целью обеспечить доступ подпрограммы к параметрам, расположен-

ным в вызывающей программе вслед за командой вызова. При наличии параметров сохраненный в регистре адрес является таким образом не адресом возврата, а адресом первого параметра. Подпрограмма для доступа к параметрам использует автоинкрементную адресацию по регистру Rn, хранящему «адрес возврата», — прямую, если параметр является значением операнда, или косвенную, если параметр является адресом. При этом каждая пересылка параметра сопровождается автоматическим приращением «адреса возврата», т.е. после пересылки всех имеющихся параметров он станет действительным адресом возврата — адресом первой после списка параметров команды продолжения программы.

Например, программа, вызывающая подпрограмму SUBR с передачей ей двух параметров и с запоминанием возврата в регистре R4, будет иметь вид

```
JSR R4, SUBR
адресное слово
1-й параметр
2-й параметр
продолжение программы
. . . . .
```

```
SUBR:  MOV (R4) +, R1
        MOV @(R4) +, R2
        . . . . .
        RTS  R4
```

После выполнения команды JSR в регистре R4 будет находиться адрес ячейки, содержащей 1-й параметр. При пересылке его, осуществляемой с помощью автоинкрементной адресации (R4) + , произойдет приращение «адреса возврата»: $R4 := R4 + 2$, а при пересылке 2-го параметра такое же приращение еще раз. В результате регистр R4 будет содержать адрес продолжения программы, и выполнение команды RTS R4 приведет к переходу на продолжение.

Другой используемый в машинах PDP-11 способ передачи параметров заключается в том, что перед вызовом подпрограммы их засылают в стек. При этом доступ реализуется путем индексной или автоиндексной адресации относительно регистра, в ко-

торый скопировано значение SP, указывающее начало списка параметров.

В случае подпрограммы без параметров, или если передача параметров производится через стек, нет необходимости запоминать адрес возврата в регистре — целесообразно сохранять его в стеке. Это осуществимо с помощью команды JSR, в которой в качестве регистра Rn, запоминающего адрес возврата, используется R7, т.е. PC. Такая команда

JSR PC, SUBR

выполняется следующим образом:

$X := m(PC); PC := PC + 2; SP := SP - 2;$
 $m(SP) := PC; PC := PC + X.$

Адрес возврата оказывается в магазине, а PC принимает начальный адрес подпрограммы. Возврат в этом случае производится командой

RTS PC,

которая в данном варианте выполняется так:

$PC := PC; PC := m(SP); SP := SP + 2.$

Еще один интересный пример использования команды JSR представляет собой организация взаимодействия сопрограмм — программ, которые поочередно вызывают друг друга, причем каждый вызов программы V программой U является вместе с тем возвратом в программу U из программы V, т.е. переход производится по адресу, который был сохранен в качестве адреса возврата при вызове программой V программы U. Предположим, что в рассматриваемый момент выполняется программа U, а вершина стека содержит адрес возврата в программу V. В этой ситуации команда

JSR PC, @ (SP) +

вызовет следующие действия:

$EA := m(SP); SP := SP + 2; SP := SP - 2;$
 $m(SP) := PC; PC := EA.$

Кратко это можно выразить так: $PC := : m(SP)$, т.е. программный счетчик и вершина стека взаимно обмениваются их содержимым — текущий адрес (адрес

следующей команды) выполняемой программы становится адресом возврата, а прежний адрес возврата становится текущим адресом.

Кроме рассмотренного выше механизма обращения к подпрограммам, в машинах PDP-11 имеется реализованная на основе того же системного стека возможность переключения на подпрограммы, эмулирующие некоторые функции (макрооперации). Такое переключение производят команды TRAP и EMT (Emulator Trap), каждая из которых существует в 256 разновидностях, обеспечивая возможность доступа к соответствующему числу различных подпрограмм.

В процессе выполнения этих команд производится упрятывание в стек содержимого регистра состояния процессора PS (рис. 9.4) и текущего значения программного счетчика, т. е. адреса возврата. Новое содержимое регистра состояния и программного счетчика

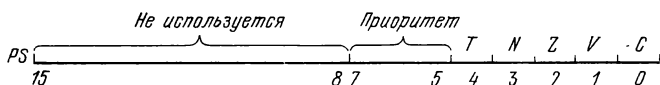


Рис. 9.4. Формат слова состояния процессора PDP-11, содержащего приоритет процессора, Т-бит и «код условия» NZVC.

принимается из ячеек главной памяти: команде TRAP и команде EMT сопоставлено по две ячейки, хранящих значение так называемого trap-вектора, которым определено слово состояния процессора, и адрес команды, на которую производится переключение. Эмулируемую функцию указывает код конкретной команды TRAP или EMT. Команды EMT используются системными программистами, а команды TRAP предназначены для применения пользователями.

Переход с сохранением слова состояния процессора и адреса возврата, называемый в фирменных руководствах trap (ловушка, западня), используется также как внутреннее прерывание программы при возникновении аварийных ситуаций и при обнаружении ошибок — переполнение стека, нечетное число в качестве адреса слова, недозволенная команда и т. п. Кроме того, trap служит средством отладки: в регистре состояния процессора имеется так называемый Т-бит (4-й бит слова состояния), который будучи установленным в состояние 1, вызывает после вы-

полнения текущей команды trap-переход на специальную подпрограмму, осуществляемый при помощи отдельного trap-вектора.

В качестве команды возврата, завершающей trap-подпрограммы, используется команда возврата из прерывания RTI (Return from Interrupt).

§ 5. Управление периферией и система прерывания

Организация управления периферией в машинах PDP-11 существенно отличается от общепринятой в других цифровых машинах. Достаточно сказать, что в наборе команд PDP-11 нет команд ввода/вывода, а вместе с тем система управления периферийными устройствами является исключительно гибкой и эффективной. Дело в том, что построение машины в целом подчинено оптимизации ее периферийных функций (периферийная ориентированность миниархитектуры получила в PDP-11 наиболее полное выражение), и практически все команды процессора применимы к регистрам периферийных устройств в той же мере, как и к ячейкам главной памяти. Понятие адреса, которое в традиционном толковании равнозначно номеру ячейки главной памяти, обобщено в PDP-11 так, что адресуемыми стали также регистры процессора и периферийных устройств. Другими словами, номера всех доступных программе регистров отображены наряду с адресами ячеек главной памяти в единое адресное пространство.

Физически общность адресной системы обеспечивается с помощью единой магистрали — UNIBUS (ЮНИБАС), представляющей собой асинхронный коммутатор, посредством которого взаимодействуют друг с другом все устройства машины — центральный процессор, главная память, устройства ввода/вывода, периферийной памяти, сопряжения с линиями связи и т. п. ЮНИБАС, кроме очевидных преимуществ единой адресной системы, сообщает машине особую гибкость в отношении возможности перестраивать конфигурацию аппаратуры: введение в систему устройств производится путем простого подключения их к ЮНИБАСу, причем в каждое гнездо ЮНИБАСа может быть включено любое из устройств, комплекующих машины PDP-11.

Длина адреса в ЮНИБАСе — 18 битов, т. е. общий объем адресного пространства — 256К и соответственно вся адресуемая память — 256КВ или 128KW. Последние 4KW (адреса с 760000₈ по 777776₈) являются регистрами периферийных устройств и процессора.

Процессоры младших моделей семейства PDP-11, интерпретирующие 16-битное машинное слово-адрес просто как двоичное число без знака, могут адресовать всего 32KW (с 0 по 177776₈). При этом адреса последних 4KW (с 160000₈ по 177776₈), соответствующие регистрам, автоматически преобразуются в сопоставленные регистрам адреса ЮНИБАСа с 760000₈ по 777776₈ по правилу: если в трех старших битах 16-битного адреса содержатся единицы, два бита, добавляющиеся слева в 18-битном адресе ЮНИБАСа, также должны содержать единицы.

Процессоры старших моделей, адресующие до 128KW, оборудуются системой расширения адресного пространства, которая интерпретирует старшие три бита A(15:13) 16-битного слова-адреса A(15:0) как номер, определяющий один из 8-ми 12-битных регистров SR — указателей сегментов памяти. При этом соответствующий исполнительный адрес EA ЮНИБАСа вычисляется так:

$$EA := 2^6 * SR(A(15:13)) + A(12:0).$$

Передача данных через ЮНИБАС производится следующим образом. Устройство-приказчик (Master) выдает впраллель на 18 адресных шин ЮНИБАСа код адреса требуемой ячейки памяти или регистра и одновременно возбуждает шину, определяющую характер передачи: 1) получение копии хранящегося по данному адресу слова, 2) считывание хранящегося слова без перезаписи его по данному адресу (так как затем последует запись нового слова), 3) запись по данному адресу выданного приказчиком слова, 4) запись по данному адресу выданного приказчиком байта. Ответчиком (Slave) является устройство, содержащее регистр или ячейку, чей адрес выдан приказчиком. Опознав принадлежащий ему адрес, ответчик выполняет указанную приказчиком операцию и сигнализирует ему о ее выполнении. В том случае когда передача данных производится от приказчика к от-

ветчику (запись слова или байта), приказчик одновременно с выдачей адреса выдает на шины данных, которых в ЮНИБАСе 16, код передаваемого слова или байта. Ответчик осуществляет запись этого слова (байта) по заданному адресу, причем выданный им ответный сигнал воспринимается приказчиком как подтверждение приема данных. Если же данные передаются от ответчика к приказчику, то ответчик вместе с сигналом о выполнении выдает на шины данных код считанного им по заданному адресу слова. В обоих случаях момент завершения передачи определяется приказчиком.

Такая (описанная весьма упрощенно) процедура передачи, образно называемая «рукопожатием» (handshake) обеспечивает возможность подключения к ЮНИБАСу устройств различного быстродействия без каких-либо дополнительных мер по их согласованию или синхронизации. В принципе приказчиком может быть любое из подключенных к ЮНИБАСу устройств, если только оно не является пассивным в силу своего функционального назначения, например, как устройство главной памяти. Таким образом, передача данных может производиться между подключенными к ЮНИБАСу устройствами, минуя процессор.

Наиболее существенным недостатком одномагистральной организации машины является, пожалуй, невозможность увеличения производительности путем совмещения во времени внутримашинных и периферийных операций — ЮНИБАС, хотя и обладает сравнительно высоким быстродействием (до 2,5MW в секунду), представляет собой «узкое место» в архитектуре PDP-11. На старших моделях семейства, начиная с PDP-11/45, помимо ЮНИБАСа имеется вторая магистраль, связывающая непосредственно процессор с быстродействующей частью главной памяти, благодаря чему замедление, вносимое ЮНИБАСом, в значительной степени устранено.

Поскольку через ЮНИБАС одновременно не может производиться более одной передачи, то в каждый данный момент лишь одному из устройств предоставляется право быть приказчиком. Параллельно с происходящей в данный момент передачей принимается решение, какому из претендующих на доступ к ЮНИБАСу устройств следующим предоставить

право быть приказчиком. Принятие этого решения осуществляется с помощью приоритетной системы, которая устроена и действует описанным ниже образом.

Приоритет, присвоенный центральному процессору, представлен тремя битами PS(7:5) слова состояния процессора (см. рис. 9.4), которое, подобно другим регистрам процессора, обладает адресом ЮНИБАСа (777776₈) и, следовательно, над ним могут производиться машинные операции. Таким образом, программа может устанавливать и изменять значение приоритета процессора в пределах от 0 до 7.

Другим устройствам, способным выступать в качестве приказчиков, приоритеты присвоены включением каждого из них в одну из пяти приоритетных линий запроса доступа к ЮНИБАСу: BR4 (Bus Request 4), BR5, BR6, BR7 и NPR (Non-Processor Request). В каждую линию может быть включено любое число устройств, причем в случае, когда на доступ к ЮНИБАСу в данной линии претендует одновременно несколько устройств, первой удовлетворяется заявка устройства, находящегося ближе других к процессору по положению в линии.

Устройство, требующее права доступа к ЮНИБАСу, становится приказчиком только в том случае, когда нет заявок от устройств, обладающих более высоким приоритетом, и от устройств, расположенных на одной с данным устройством линии ближе к процессору. Процессор при этом также должен обладать меньшим приоритетом, ибо если он будет обладать тем же приоритетом, что и рассматриваемое устройство, то получит предпочтение как ближайший на данной приоритетной линии.

Заявки по линии NPR характеризуются самым старшим приоритетом и удовлетворяются вне всякой очереди, даже если приоритет процессора равен 7. Однако по линии NPR производятся только заявки от устройств, осуществляющих передачу данных путем прямого доступа к памяти, не затрагивая процессор. При этом устройству предоставляется право приказчика для однократной передачи слова или байта.

Удовлетворение заявок, поступающих по другим (не NPR) линиям приоритета, связано с прерыванием программы, выполняемой процессором. Поэтому при-

нятие таких заявок возможно лишь при условии, что приоритет линии выше текущего значения приоритета процессора.

Устройство, овладевшее ЮНИБАСом с помощью одной из линий BR4 — BR7, выдает на его адресные шины адрес своего *вектора прерывания*, точнее, адрес первого из двух соседних слов, составляющих этот вектор. Первое слово вектора прерывания является начальным адресом сопоставленной прерывающему устройству подпрограммы, а второе слово — соответствующим словом состояния процессора. Процессор засылает в системный стек текущее содержимое своего регистра состояния PS и вслед за ним текущее значение программного счетчика PC. В PC принимается первое слово вектора прерывания, в PS — второе. При этом процессору присваивается указанное в новом слове состояния значение приоритета.

Таким образом происходит прерывание выполнявшейся процессором программы с запоминанием слова состояния и адреса возврата в системном стеке. Процессор переключается на выполнение подпрограммы, начальным адресом которой является первое слово вектора, указанное устройством — возбудителем прерывания. Эта обслуживающая прерывание подпрограмма заканчивается командой RTI — Return from Interrupt («возвратиться из прерывания»), выполнение которой восстановит упрятанное в стек при входе в прерывание значение программного счетчика и регистра состояния процессора.

Если принятое в регистр состояния процессора при переходе в прерывание слово имеет в битах PS(7:5) максимальное значение приоритета — 7, то обслуживающая программа гарантирована от прерывания. Если же в этом слове $PS(7:5) < 7$, то не исключено, что в процессе ее выполнения по заявке устройства, обладающего более высоким приоритетом, произойдет новое прерывание и программа, обслуживавшая первое устройство, будет отложена с запоминанием слова состояния и адреса возврата в системном стеке поверх уже хранимых в нем слова состояния и адреса возврата от предыдущего прерывания. Выполнение новой обслуживающей программы, если она не обладает наивысшим приоритетом, также может быть прервано и отложено. Тогда

в стеке окажется третья пара «слово состояния — адрес возврата». Этот процесс может продолжаться, но затем после завершения обработки наиболее срочного прерывания произойдет выталкивание из стека адреса возврата и слова состояния предшествовавшей обслуживающей подпрограммы, а когда и она будет завершена, возобновится выполнение прерванной ею подпрограммы и т. д., пока не наступит возврат на продолжение основной программы.

Такое *гнездование* прерываний может сочетаться с переходами на подпрограммы и переключениями типа `trap`. Стек автоматически запоминает и восстанавливает в обратной последовательности все состояния и адреса возвратов, а кроме того, может использоваться также для передачи параметров подпрограмм, как описано в предыдущем параграфе.

Гнездование прерываний возникает в результате соперничества периферийных устройств, добивающихся овладения ЮНИБАСом (а через него и временем процессора) по приоритетным линиям BR4 — BR7. Кратковременные занятия ЮНИБАСа по линии NPR для прямого доступа к памяти не затрагивают систему прерывания.

Каждое периферийное устройство представлено в системе ЮНИБАСа двумя или более 16-битными регистрами. Эти регистры по функциональному назначению разделяются на регистры управления и состояния — CSR (Control and Status Register) и буферы данных — DBR (Data Buffer Register).

Посредством регистра или регистров управления и состояния программа управляет периферийным устройством и получает необходимую информацию о его работе. Отдельные биты и поля битов в регистрах этого рода выполняют функции флажков занятости или готовности устройства, указывают заданную операцию или номер блока, сообщают об обнаружении ошибки и т. п. Программа осуществляет управление устройством, устанавливая те или иные значения управляющих битов и полей. Устройство в свою очередь отвечает соответствующей установкой битов и полей состояния, которые программа тестирует, предпринимая при помощи команд условного перехода необходимые в той или иной конкретной ситуации действия.

Регистры-буферы служат для временного запоминания данных, передаваемых от процессора устройству в случае вывода и от устройства процессору в случае ввода. Побайтная передача данных производится при помощи младшей половины буферного регистра, т. е. DBR(7:0).

Пересылка данных между буферами и регистрами процессора или ячейками главной памяти осуществляется командами MOV и MOVB. Например, пересылку из буфера в регистр R2 задает команда

MOV DBR, R2,

где DBR — адрес буферного регистра.

Управление периферийным устройством в простейшем случае обеспечивается с помощью битов ENABLE, BUSY и DONE (или READY) регистра управления и состояния.

Запуск устройства в работу осуществляется установкой в состояние 1 бита ENABLE, занимающего в регистре нулевую позицию — CSR(0). Это достигается, например, выполнением команды

BIS # 1, CSR,

формирующей в регистре CSR дизъюнкцию его содержимого и 16-битного слова, представляющего константу 000001₈: производит присваивание CSR(0):=1, сохраняя значения прочих битов регистра неизменными.

Устройство ввода, получив разрешение начать работу, устанавливает в состояние 1 бит CSR(11), сигнализируя этим о том, что оно занято (BUSY), и сбрасывает в нулевое состояние бит CSR(7) готовности (DONE). Затем, поместив в буферный регистр вводимое данное, устройство сбрасывает BUSY (а, возможно, также ENABLE) и устанавливает DONE в состояние 1, что служит сигналом процессору скопировать содержимое буфера. Процессор следит за состоянием бита DONE, являющегося головным битом младшего байта CSR, в цикле

```
L:  TSTB  CSR      ; N := Z := C := V :=  
                                := 0; if CSR(7) = 1 then  
                                N := 1;  
                                if CSR(7:0) = 0 then  
                                Z := 0;
```

| | |
|--------------|--------------------------|
| BPL L | ; if N = 0 then PC := L; |
| MOVB DBR, Rn | ; Rn := |
| | := sgnv1(DBR(7:0)); |

Запуская в работу присваиванием $ENABLE := 1$ устройство вывода, процессор при этом загружает в его буферный регистр выводимое данное. Устройство, приступая к работе, сбрасывает $CSR(7)$ на нуль и устанавливает 1 в $CSR(11)$, а скопировав находящееся в буфере данное, сбрасывает на нуль $CSR(11)$ и устанавливает 1 в $CSR(7)$, что является сигналом **READY** — готовности принять от процессора в буфер очередную порцию данных.

Уведомление о готовности периферийного устройства продолжить вывод или передать из буфера введенное данное может восприниматься процессором как в цикле ожидания, так и посредством прерывания. Чтобы устройство могло сигнализировать о готовности прерыванием, процессор устанавливает имеющийся в регистре CSR бит **INTERRUPT ENABLE** («прерывание разрешено») в состояние 1.

§ 6. Периферийное оборудование миникомпьютеров PDP-11

К миникомпьютерам PDP-11 имеется широкий выбор специально разработанных минипериферийных устройств, поставляемых как самой Digital Equipment, так и многими фирмами — производителями минипериферии. Ниже кратко охарактеризованы характерные представители устройств ввода/вывода, периферийной памяти, сопряжения с линиями связи, с лабораторным и промышленным оборудованием.

Перфолентное оборудование представляют считыватель PR11 (300 зн/с), считыватель/перфоратор PC11 (считывает 300 зн/с, перфорирует 50 зн/с) и теле-тайпный терминал LT33, обеспечивающий перфорацию ленты и ввод с клавиатуры, а также ввод с перфоленты и вывод на перфоленту и печать со скоростью 10 зн/с.

Печатающие устройства: матричные принтеры типа DECwriter LA30 (30 зн/с, 80 позиций в строке) и LA36 (с выбором скорости — 10, 15, 30 зн с. 132

позиции в строке), LS11 (60 строк/мин, 132 позиции в строке), строчные барабанные принтеры типа LP11 (с 80 или 132 позициями в строке, 64 или 96 знаков в алфавите, скорость от 170 до 1200 строк/мин, но обычно 200—300 строк/мин). Электро-статический принтер и построитель графиков LV11 (500 строк/мин, 132 позиции в строке, 96 знаков в алфавите, 122 тыс. точек в секунду, 1024 точки в строке).

Видеотерминалы с клавиатурой алфавитно-цифровые — VT05 (20 строк по 72 позиции) и VT50 DECscore (12 строк по 80 позиций), графические — GT40, GT42, GT44.

Считыватели перфокарт — CR11 (285 карт/мин) и CD11 (1200 карт/мин), считыватель карт с отметками карандашом — CM11 (285 карт/мин).

Устройства памяти на магнитной ленте: кассеты TA11 (92 КВ, 560 байтов в секунду) используются взамен перфоленты, форматная лента TC11, включающая контроллер и до 8 вдвоенных лентопряжек TU56 DECtape (578 блоков по 256 слов на бобине диаметром 9,9 см, до 5 килослов в секунду), TM11 с лентопротяжкой TU10 Magtape (стандартная IBM бобина, 20 МВ, 36 килобайтов в секунду), TU16 Magtape (36 МВ, 72 килобайта в сек).

Устройства памяти на магнитных дисках: RC11 — подсистема, включающая контроллер и до 4-х дисководов с неподвижными головками RS64 (128 КВ, среднее время доступа — 17 мс), RF11 — контроллер и до 8-ми дисководов с неподвижными головками RS11 (512КВ, 17 мс), RK11 — контроллер и до 8 картриджей RK05 (2,4МВ, 70 мс), RS03, RS04 — дисководы с неподвижными головками на 512КВ и 1МВ, среднее время доступа 8,5 мс, RP03, RP04 — съемные диски с емкостью пакета 40МВ и 88МВ, среднее время доступа 36 мс.

Связное оборудование включает: адаптеры асинхронной связи — однолинейный DL11, двухлинейный DC11, однолинейный высокоскоростной DP11, мультиплексоры асинхронной связи на 16 линий — DJ11 и DH11 с программируемыми параметрами, адаптеры для телеграфных линий DC08 и H316, «нулевые» модемы DF11 и H312-A, модем акустического сопряжения DF01, интегральные модемы для преобразо-

вания TTL-импульсов в сигналы звуковой частоты, адаптер H313-A для сопряжения с модемами Bell 103, адаптеры синхронной связи — однолинейные DU11 и DQ11, многолинейный DV11 с прямым доступом к памяти, устройство автоматического набора номера абонента DN11, арифметический блок обнаружения ошибок в сообщениях KG11 и др.

Оборудование для сбора экспериментальных данных и управления лабораторными приборами и промышленными объектами: подсистема аналого-цифрового преобразования AD01 (от 4 до 32 входных каналов, точность 10 битов), подсистема цифро-аналогового преобразования AA11 (до четырех 12-битных аналого-цифровых преобразователей), лабораторная подсистема LPS11 (аналого-цифровой преобразователь 12 битов, до 16 входных каналов, датчик реального времени, 16-битный цифровой ввод/вывод, контроллер для электронно-лучевого дисплея с двумя 12-битными цифро-аналоговыми преобразователями), подсистема аналогового ввода низкого уровня AFC11 (до 1024 дифференциальных аналоговых входов, максимальное значение шкалы низкого уровня — 10 мВ, высокого — 100 В, выход — 13 битов, скорость обегания — 200 каналов в секунду), подсистема цифрового управления UDC11 (до 252 прямо адресуемых цифровых воспринимающих и управляющих 16-битных модулей ввода/вывода или до 4032 индивидуальных цифровых точек, включая выходы реле, контакты, цифро-аналоговые преобразователи и т. п.), подсистема управления промышленными объектами ICS и ее «удаленный» вариант ICR, работающий на расстоянии до 2 км от центрального процессора (до 256 точек ввода/вывода, аналого-цифровые и цифро-аналоговые преобразователи, счетчики, восприятие и выдача сигналов постоянного и переменного тока).

§ 7. Программное оснащение семейства миникомпьютеров PDP-11

Миникомпьютеры PDP-11 обеспечены широким выбором системных и прикладных программ, от рассчитанных на минимальные конфигурации аппаратуры до multifunctional систем с мощными язы-

ковыми процессорами, работающих на старших моделях семейства. Несмотря на большое разнообразие имеющихся программ и различие масштабов, весьма строго удовлетворяется условие совместимости: как правило, программы, разработанные в одной из систем, могут работать в других системах, если обеспечены требуемые ресурсы. Соответственно, совместимы форматы представления данных на всех используемых носителях (перфолента, магнитная лента в кассетах и на бобинах, различные виды магнитных дисков). Далее следует перечисление и краткая характеристика системного программного оснащения для различных конфигураций аппаратуры.

Перфолентная система, работоспособная при наличии 4KW главной памяти и телетайпа, включает символьный ассемблер PAL-11, редактор текста ED-11, начальный и абсолютный загрузчики, отладочные программы ODT-11 и ODT-11X, программы выдачи на печать или перфорацию содержимого памяти в двоичном и восьмеричном кодах Dumpab и Dumpptt, программу обслуживания ввода/вывода IOX (Input/Output Executive) и пакет программ, реализующих операции с плавающей запятой и элементарные математические функции, FPP-11 (Floating-Point Package). Кроме программирования на языке ассемблера, для минимальной конфигурации с 4KW памяти имеется система разговорного программирования на языке basic.

За перфолентной системой по требующимся ресурсам и предоставляемым возможностям следует система с магнитными кассетами CAPS-11 (Cassette Programming System-11), работающая при наличии 8KW главной памяти, терминала типа LA36 DECwriter и кассет TA11. Система CAPS-11 включает редактор, ассемблер переместимого кода, загрузчик, отладчик и ряд служебных программ, управляемые посредством резидентного монитора с клавиатуры терминала. Наличие монитора и использование кассет в качестве периферийной памяти существенно увеличивают эффективность CAPS-11 по сравнению с перфолентной системой. Как и перфолентная система, CAPS-11 включает basic.

Следующей по уровню ресурсов и возможностей является клавиатурно управляемая дисковая опера-

ционная система PDP-11 DOS, требующая 8KW главной памяти, магнитный диск и дополнительно магнитную ленту DECtape.

Помимо указанных выше средств разработки программ в DOS используется библиотечка LIBR-11 и пакет программ для работы с файлами PIP (Peripheral Interchange Program).

Еще одна простая операционная система, работающая при наличии 8KW главной памяти и магнитной ленты DECtape или диска, — RT-11. Эта система обеспечивает разработку программ одним пользователем или однопрограммную работу в режиме реального времени. В усиленном варианте RT-11 F/B (Foreground/Background), требующем не менее 16KW главной памяти, система сочетает режим реального времени с одновременной разработкой программ или пакетной обработкой «на заднем плане». Языками программирования являются basic, focal, фортран и макроассемблер macro-11.

Операционная система для совместного обслуживания нескольких пользователей в режиме разделения времени RSTS/E (Resource Sharing Timesharing System/Extended) может работать на PDP-11/35 и более старших моделях при наличии не менее 32KW главной памяти и дисков, причем PDP-11/35 или 11/40 обслуживает до 24 терминалов, PDP-11/45 до 32, PDP-11/70 до 63. Языком программирования является basic-plus — расширенная версия языка basic, а также кобол в пакетном режиме.

Семейство операционных систем реального времени RSX-11 (Real Time Executive) охватывает широкий диапазон конфигураций аппаратуры и предоставляемых возможностей. Минимальная версия RSX-11S может работать при 8KW главной памяти без диска и магнитной ленты, выполняя функции монитора реального времени, но без возможности разработки программ. Максимальная версия RSX-11D работает с памятью до 124KW, сочетая режим реального времени с обслуживанием нескольких терминалов пользователей и с пакетной обработкой. Машины, работающие с операционными системами RSX-11, могут с помощью пакета связанных программ взаимодействовать друг с другом по линиям связи, образуя единую сеть DECnet.

Для старших моделей семейства PDP-11 имеется многофункциональная операционная система IAS (Interactive Application System), обеспечивающая возможность одновременно разработки и выполнения программ в режиме реального времени, обслуживание терминалов и пакетную обработку. Система требует не менее 64KW главной памяти и подсистемы памяти на дисках. На PDP-11/70 она обслуживает до 16 терминалов. Языки программирования: basic, фортран, кобол, маско-11.

ПОСЛЕСЛОВИЕ

Рассмотренные в четырех последних главах архитектуры миникомпьютерных семейств PDP-8, HP 2100, NOVA и PDP-11 представляют собой своего рода ступени развития миниархитектуры от совсем скромной одноаккумуляторной с минимумом возможностей адресации и примитивной системой прерывания до роскошной регистровой с изощренной адресацией, векторным механизмом прерывания и автоматическим гнездованием подпрограмм в стеке. Схема этого развития обычна: начали с самого необходимого и простого, а затем добавляли и усложняли, стремясь расширить возможности. Такому ходу дела способствует удешевление из года в год цифровой аппаратуры, позволяющее строить все более мощные и сложные машины при неизменной и даже снижающейся цене.

Разумеется, против большей мощности и новых возможностей, предоставляемых за ту же цену, возражений быть не может. Другое дело — сложность. Она явно не на пользу. Ведь простота — важнейший принцип миникомпьютеров, один из основных источников их силы. На ранней стадии она проявлялась в значительной степени как простота аппаратуры, обеспечившая минимизацию стоимости и требуемый уровень надежности. В дальнейшем, по мере усовершенствования и удешевления технологии производства, акцент естественно сместился на простоту понимания и применения машины. К сожалению, разработчики миникомпьютеров не восприняли это и по инерции употребляют возросшие возможности технологии не для упрощения, а для усложнения архитектуры. Красноречивое свидетельство тому — непомер-

ная сложность PDP-11 в сравнении с «детской» простотой PDP-8.

Понятно, что сложность архитектуры не самоцель, а побочный результат стремления повысить эффективность, увеличить гибкость и мощность машины. В том случае, когда машина используется через посредство языков высокого уровня, архитектура машины скрыта от пользователя и ее сложность не вызывает трудностей. Но для миникомпьютеров такой случай не типичен. Эффективность миникомпьютера в значительной мере обусловлена тонкой настройкой на конкретное применение, достигаемой путем непосредственного использования особенностей его архитектуры, т. е. путем искусного программирования на языке ассемблера. Ясно, что чем сложнее архитектура, тем труднее и дороже надлежащее программирование. Поэтому следует строго оберегать простоту архитектуры, не поддаваться соблазну выгод, влекущих сложность.

Но практика не согласуется с данным выводом: подобно большим цифровым машинам, миникомпьютеры развиваются в направлении все большей сложности архитектуры. Вследствие такого развития трудности разработки прикладного программного оснащения стали главным препятствием, сдерживающим новые применения миникомпьютеров, причем усилия по преодолению этих трудностей путем создания языков и систем программирования не обеспечивают и не могут обеспечить удовлетворительного решения данной проблемы. Ведь если особенности архитектуры машины не отражены в языке программирования, то программист не может ими воспользоваться, а если они отражены в достаточной степени, то программирование становится таким же трудоемким, как на языке ассемблера.

Очевидно, решение проблемы находится не в языках и системах, создаваемых в качестве надстройки над архитектурой машины, а в самой архитектуре, которую надо не «обогащать» увеличением числа регистров процессора и раздуванием набора команд, а совершенствовать, в частности, в направлении уменьшения трудоемкости составления эффективных программ. Одна из возможностей такого совершенствования связана с реализацией в архитектуре машины

идей структурированного программирования в духе § 3 гл. 4. Практика показала^{1,2)}, что архитектура этого рода в сочетании с бесскобочными формами выражений и послоговым представлением команд³⁾ является хорошей основой для построения высокоэффективных и легко программируемых машин, а также для преодоления трудностей, обусловленных страничной структурой памяти миникомпьютеров.

¹⁾ Брусенцов Н. П. Структурированное программирование и стековая архитектура. — Программирование, 1977, № 4, с. 72—74.

²⁾ Брусенцов Н. П., Рамиль Альварес Х. Структурированное программирование на малой цифровой машине. — В кн.: Вычислительная техника и вопросы кибернетики, вып. 15, Изд-во МГУ, 1978, с. 3—8.

³⁾ Брусенцов Н. П. Стековые машины с изменяемой адресностью команд. — В кн.: Вычислительная техника и вопросы кибернетики, вып. 13, Изд-во МГУ, 1977, с. 97—112.

ПРИЛОЖЕНИЕ

ОБЩИЕ ПРАВИЛА ЗАПИСИ ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА

Язык ассемблера, т. е. язык, на котором составляется исходный, перерабатываемый затем ассемблером текст программы, отличается от непосредственного языка машины главным образом тем, что программу представляют на нем не числовым (двоичным или восьмеричным) кодом, а с помощью буквенно-цифровых обозначений (символов) операций, операндов и других элементов машинных команд. Символы операций обладают мнемоникой, поэтому человеку работать с ними проще, чем с числовыми кодами операций. Символы операндов и адресов, кроме того, избавляют от бремени абсолютной адресации, которая перепоручена ассемблеру.

Несмотря на различия, обусловленные особенностями архитектуры машин и неидентичностью отдельных обозначений, языки символьных ассемблеров в основном устроены единообразно, подчинены в общем единым правилам. В качестве набора используемых знаков принят алфавит ASCII, включающий латинские буквы, десятичные цифры, разделители (пробел, точка, запятая, двоеточие, точка с запятой, скобки, кавычки, математические и некоторые другие специальные знаки), а также особые знаки управления устройством вывода — «возврат каретки», «перевод строки» и т. п. Знаки полного алфавита ASCII представлены семибитным кодом и для их передачи, хранения и обработки, как правило, используются восьмибитные байты. Однако ассемблеры работают с неполным алфавитом, в котором имеются заглавные, но нет строчных букв, благодаря чему достаточно

шестибитного байта, используемого, например, на машинах PDP-8.

Символ в языке ассемблера — это последовательность из букв, цифр, а иногда и некоторых других знаков, начинающаяся не цифрой. Длина символа обычно не ограничена, но ассемблеры принимают во внимание только 5—6 первых знаков, т. е. символы, совпадающие по этим знакам, для ассемблера не различимы друг от друга. Например, ассемблер, опознающий символы по шести первым знакам, воспринимает символы LABEL1, LABEL2, LABEL12 соответственно как LABEL1, LABEL2, LABEL1, а ассемблер, опознающий по пяти первым знакам, воспринимает любой из них как LABEL. Символ в зависимости от положения, занимаемого им в записи команды, а также от присоединенных к нему знаков-разделителей, воспринимается ассемблером либо как обозначение (мнемокод) операции, либо как обозначение операнда (как элемент обозначения операнда), либо как метка, которой помечена команда, чтобы на нее можно было ссылаться в других командах. Кроме того, символы могут использоваться в директивах, управляющих работой ассемблера, называемых также псевдокомандами. Различают постоянные и определяемые пользователем символы. К постоянным, т. е. обладающим для данного ассемблера фиксированным значением, относятся символы директив и мнемокоды операций. Прочие используемые в программе символы должны быть определены в ней самой.

Программу на языке ассемблера (как и в машинном коде) записывают в виде последовательности команд, располагаемых по одной на строку. При этом запись команды состоит из четырех полей, в которых размещаются соответствующие компоненты записи:

| | | | |
|-------|----------|---------|-------------|
| метка | операция | операнд | комментарий |
|-------|----------|---------|-------------|

Метка, если она имеется, занимает самое левое поле и сопровождается двоеточием (в ассемблерах для PDP-8 — запятой). Ассемблер сопоставляет метке адрес ячейки, занимаемой в памяти машины командой, которая помечена этой меткой. Все ссылки на данную команду, осуществляемые в исходной программе посредством метки, ассемблер переводит в обращение

ния по адресу этой команды. Во избежание неопределенности нельзя одной и той же меткой помечать более чем одну команду. Однако пометить команду несколькими различными метками можно — всем этим меткам ассемблер присвоит в качестве значения один и тот же адрес помеченной ими команды. Метки являются также средством доступа к значениям операндов, располагаемых в помеченных строках подобно командам.

Поле «операция» в записи команды предназначается для мнемокода операции, а в записи директивы — для символа, обозначающего указание, выполняемое в процессе ассемблирования, например: END — конец ассемблируемой программы. Разделителями между символом, находящимся в поле «операция», и меткой дополнительно к сопровождающему метку двоеточию (для PDP-8 — запятой) служат пробелы, употребляемые в необходимом количестве для выравнивания находящихся друг под другом мнемосимволов в столбец по левому краю. Аналогично осуществляется выравнивание левого края столбца символов в поле «операнд», а также в поле «комментарий». Оформление текста программы в виде четко разделенных четырех столбцов, обеспечиваемое обычно с помощью табуляции, существенно облегчает чтение и понимание программы.

Поле «операнд» предназначено для задания операнда (операндов) операции, указанной в поле «операция». Операнды задаются в виде ссылок на ячейки главной памяти, на регистры процессора или на регистры периферийных устройств, содержащие значения операндов. Если же местонахождение значения операнда (значений операндов) определено мнемокодом операции (например: CMA — «инвертировать содержимое аккумулятора», KRB — «скопировать в аккумулятор содержимое буфера клавиатуры»), то поле «операнд» может быть пустым. Вообще же запись в поле «операнд» интерпретируется применительно к заданной операции.

Ссылкой на регистр процессора может служить либо просто цифра — номер этого регистра (как в ассемблере для машин NOVA), либо номер регистра с помещенным перед ним специальным знаком (например, % в языке PDP11: %2 означает «регистр 2»).

Регистры периферийных устройств, как правило, обозначаются буквенными или буквенно-цифровыми символами, например: TT1, DSC, MTA, RBUF, CRB1.

Ссылка на ячейку главной памяти в общем случае является адресным выражением с присоединяемыми к нему указателями способов адресации. Адресное выражение может содержать определенные в тексте исходной программы символы, целые числа, а также обозначение собственного адреса команды (обычно это точка или звездочка), соединенные знаками «+», «-». В частности, выражением может быть просто символ или целое число (абсолютный адрес).

Примеры адресных выражений:

STACK — ITEM3

BUFF + 5

· — 1

POLE5

123

В качестве указателя косвенной адресации используется либо помещаемый перед адресным выражением знак @ (NOVA, PDP-11), либо буква I — Indirect, располагаемая перед адресным выражением и отделяемая от него пробелом (PDP-8), или помещаемая после адресного выражения и отделяемая от него запятой (Hewlett — Packard).

Примеры:

@PRTC

@%4

I TEMP

ADR, I

Индексная адресация задается ссылкой на соответствующий индекс-регистр, т. е. номером этого регистра. Например, команду запоминания содержимого аккумулятора Ac2 в ячейке с индексированным посредством Ac3 адресом START на языке ассемблера машин NOVA записывают в виде

STA 2,START,3

Запись аналогичной команды на языке PAL-11 имеет

вид

MOV %2, START(%3)

Знаком непосредственной адресации в PAL-1 служит #. Например, прибавление числа 5 к содержимому регистра 2 задается командой

ADD # 5,%2

Подобная функция на HP-ассемблере, а именно прибавление числа 5 к содержимому аккумулятора A, выражается в виде

ADA = D5

Адресацию относительно собственного адреса команды задают выражением, в котором этот адрес условно обозначается точкой (в языке HP-ассемблера — звездочкой). Например:

JMP .-1

Присваивание значений вводимым пользователем символам производится следующими способами.

Во-первых, значение символа будет автоматически определено ассемблером, если этот символ употреблен в качестве метки. Просматривая исходный текст программы, ассемблер ведет счет ячейкам памяти, последовательно занимаемым командами, в результате чего определяются адреса ячеек, а следовательно, и значения соответствующих меток. Начальное значение счетчика ячеек указывают ассемблеру директивой, которая в случае, когда это значение равно, например, 300₈, записывается в зависимости от используемого языка в одном из следующих видов:

| | | |
|------|-----|-----------|
| *300 | | (PDP-8), |
| .LOC | 300 | (NOVA), |
| | 300 | (PDP-11). |

Если такая директива отсутствует, ассемблер ведет счет ячеек с некоторого фиксированного значения, например, с 200₈ в ассемблере для PDP-8.

Во-вторых, имеется возможность присваивать значения символам директивой прямого присваивания, имеющей вид

⟨символ⟩ = ⟨значение⟩.

Например:

POINT3 = 7425.

Четвертое поле в строке, представленной на языке ассемблера программы — «комментарий» — предназначено для пояснений к находящейся в первых трех полях записи команды, директивы или числа. Поле комментария отделяется от поля операнда точкой с запятой (в языке ассемблера для PDP-8 — косой чертой), а заканчивается возвратом каретки и переводом строки. Комментарий не воспринимается ассемблером, поэтому в нем допустим произвольный текст, в котором можно употреблять, как правило, любые имеющиеся в алфавите печатающего устройства знаки, за исключением возврата каретки, перевода строки и других управляющих знаков, вызывающих окончание строки.

ЛИТЕРАТУРА

Arnold W. F. Boolean logic IC acts as controller in industrial jobs.—*Electronics*, Sept. 15, 1977, v. 50, № 19, p. 16E, 17E.

Ashenhurst R. L., Vonderohe R. H. MISS — a hierarchical multi-minicomputer system.—*Infotech State of the Art Report «Minicomputer Systems»*, 1977, v. 2, p. 1–40.

Benbasat I., Goldstein R. C. Data base systems for small business: miracle or mirage? — *Data Base*, Summer 1977, v. 9, № 1, p. 5–8.

Brunskill R. H. The evolution of minicomputer architecture.—*Infotech State of the Art Report «Minicomputer Systems»*, 1977, v. 2, p. 61–79.

Bruun R. Push-down stack firmware extends minicomputer language capability.—*Infosystems*, June 1973, v. 20, № 6, p. 46–50.

Bruun R. Researchers pioneer application advances with minis.—*Infosystems*, Oct. 1973, v. 20, № 10, p. 44–47, 52.

Burns R., Savitt D. Microprogramming, stack architecture ease minicomputer programmer's burden.—*Electronics*, Febr. 15, 1973, v. 46, № 4, p. 95–101.

Burr W. E., Smith W. R. Comparing architectures.—*Datamation*, Febr. 1977, v. 23, № 2, p. 48–52.

Bursky D. Focus on data-acquisition equipment.—*Electronic Design*, June 7, 1974, v. 22, № 12, p. 70–85.

Carren D. M. Multiple minis for information management.—*Datamation*, Sept. 1975, v. 21, № 9, p. 54–58.

Caswell S. A. Word processing meets dp.—*Computer Decisions*, Febr. 1977, v. 9, № 2, p. 52, 54, 56.

Clayman S. G. Minicomputers in the office.—*Data Management*, March 1975, v. 13, № 3, p. 22.

Cluley J. C. A survey of minicomputer I/O architecture and interfaces.—*Infotech State of the Art Report «Minicomputer Systems»*, 1977, v. 2, p. 95–110.

Cohen E. Symmetric multi-mini-processors: a better way to go? — *Computer Decisions*, Jan. 1973, v. 5, № 1, p. 16–20.

Conway J. Computers and peripherals — get ready, world the day of the micropriced system is at hand! *EDN*, July 20, 1977, v. 22, № 13, p. 114–117.

Conway J. W. Approach to unified bus architecture sidesteps inherent drawbacks.—*Computer Design*, Jan. 1977, v. 16, № 1, p. 71–76.

Copeland J. R., Fraade D. J. Batch process control with minicomputers.—*Instrument and Control Systems*, Nov. 1973, v. 45, № 11, p. 65–66.

Data General Corp. How to use the Nova computers.— Southboro, Mass., 1972.

David J. Role of minicomputers.— Data Management, Febr. 1975, v. 13, № 2, p. 16–19.

Dickey S. An integrated approach to minicomputer network project design and management.— Infotech State of the Art Report «Minicomputer Systems», 1977, v. 2, p. 129–154.

Digital Equipment Corp. PDP-8/E and PDP-8/M small computer handbook.— Maynard, Mass., 1972.

Digital Equipment Corp. Introduction to programming PDP-8 family computers.— Maynard, Mass., 1969.

Digital Equipment Corp. Programming languages. PDP-8 handbook series.— Maynard, Mass., 1972.

Digital Equipment Corp. PDP-11/15, 20, R20 processor handbook.— Maynard, Mass., 1972.

Digital Equipment Corp. PDP-11 peripherals handbook.— Maynard, Mass., 1973.

Digital Equipment Corp. PDP-11 software handbook.— Maynard, Mass., 1975.

Digital Equipment Corp. PDP-11 computer family products and services.— Maynard, Mass., 1975.

Dimauro J. Input-output the basic options.— EE Systems Engineering Today, May 1973, v. 32, № 5, p. 97–98.

Eckhouse R. H. Minicomputer systems: organization and programming (PDP-11).— Prentice-Hall Inc., 1975.

Edward J. S. The million-word minicomputer main memory.— Hewlett—Packard J., Oct. 1974, v. 26, № 2, p. 19–20.

Evans R. W. Towards multi-minicomputer system in engineering application.— Infotech State of the Art Report «Minicomputer Systems», 1977, v. 2, p. 155–164.

Falk H. A checkup: minicomputer software.— IEEE Spectrum, Febr. 1974, v. 11, № 2, p. 52, 54, 56.

Ferreira R. C., Vojnovic D. A classification of minicomputer multiprocessor systems.— Infotech State of the Art Report «Minicomputer Systems», 1977, v. 2, p. 165–192.

Floam G. Putting a data base on a mini.— Datamation, June 1976, v. 22, № 6, p. 97–102.

Frankenberg R. J. Unraveling the mystery in user microprogramming.— Mini-Micro Systems, June 1977, v. 10, № 6, p. 28–30, 33; July 1977, v. 10, № 7, p. 46–48, 50; Sept. 1977, v. 10, № 9, p. 54, 55, 58–60.

Gayston D. Minicomputers: the commercial view.— Data Systems, Sept. 1973, v. 14, № 7, p. 8.

Gibbons F. M. The mini's impact on data base management systems.— Mini-Micro-Systems, Nov. 1976, v. 9, № 11, p. 26, 28, 29.

Goodell W. Minis gang up.— Computer Decisions, June 1974, v. 6, № 6, p. 32–34.

Hanks D. Programming considerations for minicomputers.— Computers and People, Jan. 1974, v. 23, № 1, p. 11–13.

Hansen J. R. Minis move to the user.— Infosystems, June 1977, v. 24, № 6, p. 43–48, 141.

Healey M. Minicomputers and microprocessors.— Hodder and Stoughton, London, 1977.

Hewlett-Packard Co. A pocket guide to HP computers.— Palo Alto, Calif., 1971.

Hewlett-Packard Co. HP 3000 computer systems. — Palo Alto, Calif., 1977.

Hewlett-Packard Co. Distributed Systems. — Palo Alto, Calif., 1975.

Hobbs L. C. The role of minicomputers and microprocessors in distributed systems. — Infotech State of the Art Report «Distributed Systems», 1976, p. 288—302.

Hodges D. A. Trends in computer hardware technology. — Computer Design, Febr. 1976, v. 15, № 2, p. 77—85.

House D. L. Micro level architecture in minicomputer design. — Computer Design, Oct. 1973, v. 12, № 10, p. 75—80.

Jarosz M. B. Minicomputers-microcomputers-peripherals. — Mini-Micro-Systems, May 1976, v. 9, № 5, p. 79—88.

Kallis S. A. Mini, midi, vici. — Data Management, Febr. 1976, v. 14, № 2, p. 26—27.

Karush A. What is a minicomputer? — The Office, Aug. 1972, v. 76, № 2, p. 12—14.

Katz R. What is a minicomputer. — EE Systems Engineering Today Febr. 1973, v. 31, № 2, p. 117—118.

Kinnucan P. Another bandwagon begins to roll. — Mini-Micro-Systems, July 1977, v. 10, № 7, p. 20, 22.

Kloeppe G. Economic and technical developments in minicomputers influencing distributed systems. — Infotech State of the Art Report «Minicomputer Systems», 1977, v. 2, p. 195—197.

Kluchman A. Z. Here's what the mini can do for you. — Electronic Design, April 15, 1971, v. 19, № 8, p. 48—57.

Knoblock D. E. I/O channels for minis: fast and simple. — EE Systems Engineering Today, May 1973, v. 32, № 5, p. 102—103.

Kocher C. P. Interrupt driven I/O. — EE Systems Engineering Today, May 1973, v. 32, № 5, p. 100—101.

Korn A. G. A survey of minicomputer applications. — Computer and Electrical Engineering, Jan. 1975, v. 2, № 1, p. 3—33.

Korn G. Q. Minicomputers for engineers and scientists. — McGraw-Hill Book Co., 1973.

Koudela J. The past, present, and future of minicomputers. — Proc. of the IEEE, Nov. 1973, v. 61, № 11, p. 1526—1534.

Leis C. T. Minicomputer hardware architecture. — Proc. of the IEEE, Nov. 1973, v. 61, № 11, p. 1535—1538.

Levine S. T. Decentralized networks allowing the computer to be moved to the job. — Electronic Design, April 26, 1974, v. 22, № 9, p. 66—71.

Lewis T. G. Evolving minicomputer architecture. — Infotech State of the Art Report «Minicomputer Systems», 1977, v. 2, p. 199—221.

Maguire R. B., Symes L. R. Effect of laboratory facilities on computer science curriculum. — SIGCSE Bulletin, Febr. 1977, v. 9, № 1, p. 1—5.

Manildi A. B. Inexpensive digital storage media. — Electronic Design, Dec. 6, 1976, v. 24, № 25, p. 86—88.

Matlack R. The mighty mini. — EE Systems Engineering Today, Febr. 1973, v. 31, № 2, p. 114—116.

McClellan S. T. Will success spoil the minicomputer industry? — Mini-Micro-Systems, May 1977, v. 10, № 5, p. 30—32.

McDermott J. Take-your-pick software is making the mini mighty but watch out — it costs. — Electronic Design, April 26, 1974, v. 22, № 9, p. 78—83.

McDonald A. R. Minicomputers... their places in the sun.—Data Management, Febr. 1976, v. 14, № 2, p. 16–22.

Mills D. L. Executive systems and software development for minicomputers.—Proc. of the IEEE, Nov. 1973, v. 61, № 11, p. 1556–1562.

Milstead F. C., Neely G. L., Hall J. L. Get the facts behind mini specs.—Electronic Design, April 29, 1971, v. 19, № 9, p. C20–C27.

Molinary F., Fishman A. Building an analog peripheral inside a minicomputer chassis.—Electronics, Aug. 22, 1974, v. 47, № 17, p. 104–107.

Mueller G. E. Whither the large computer.—Mini-Micro-Systems, Jan. 1977, v. 10, № 1, p. 67–68.

Nielsen J. R., Kaplan D. S. Data entry and communications systems have network capabilities.—Hewlett-Packard J., March 1978, v. 29, № 7, p. 21–26.

Notley J. P. W. Minicomputers for industrial automation.—Automation, Sept. 1972, p. 15–20.

Oliver S. R. (Mini-) computing today.—Infotech State of the Art Report «Minicomputer Systems», 1977, v. 2, p. 223–231.

Osbrink N. K. Minis, minis everywhere, when it comes to control of industrial processes.—Electronic Design, April 26, 1974, v. 22, № 9, p. 118–121.

Pantages A. IBM's Series/1 minicomputers.—Datamation, Dec. 1976, v. 22, № 12, p. 146–147.

Poitras J., Watts C. Maximize a mini: add a disk.—Data Management, March 1976, v. 14, № 3, p. 15–17.

Poppendieck M., Desautels E. J. Memory extension techniques for minicomputers.—Computer, May 1977, v. 10, № 5, p. 68–75.

Potts H. The mini computer.—Data Systems, March 1975, v. 16, № 3, p. 10–19.

Purser W. F. C. Real-time executives for minicomputer systems—implications for hardware architecture.—Infotech State of the Art Report «Minicomputer Systems», 1977, v. 2, p. 233–247.

Riley W. B. Minicomputers go into action in a myriad of applications.—Electronics, March 29, 1973, v. 46, № 7, p. 72–81.

Sakakihara P. M. Distributed Systems/3000.—Hewlett-Packard Journ., March 1978, v. 29, № 7, p. 7–14.

Schwager A. O. The Hewlett-Packard distributed system network.—Hewlett-Packard J., March 1978, v. 29, № 7, p. 2–6.

Scott A. W. Minicomputer network software—requirements and implementation in DECnet.—Infotech State of the Art Report «Minicomputer Systems», 1977, v. 2, p. 319–343.

Shanzer H. M. Minicomputers in communications.—Data Management, May 1975, v. 13, № 5, p. 16.

Shatzer R. R. Distributed Systems/1000.—Hewlett-Packard J., March 1978, v. 29, № 7, p. 15–20.

Shunfenthal M. Minicomputers are banding together to replace central processors.—Electronic Design, Jan. 1977, v. 25, № 1, p. 46, 48.

Smith W. E. Centralization vs. decentralization.—Data Management, Jan. 1977, v. 15, № 1, p. 24–25.

Smythe C. Brewery's mini costs small beer.—Data Systems, Sept. 1973, v. 14, № 7, p. 23.

Spangle C. W. Minicomputers: their expanding role. — Computers and people, Dec. 1976, v. 25, № 12, p. 16–18.

Sprague M. E. How to buy computer peripherals. — Data Management, March 1976, v. 14, № 3, p. 18–21.

Stedman J. M. Microprogramability lets user tailor new minicomputer to his requirements. — Electronics, May 2, 1974, v. 47, № 9, p. 87–93.

Stedman J. M. A user-oriented family of minicomputers. — Hewlett-Packard J., Oct. 1974, v. 26, № 2, p. 2–6.

Stone H. S., Siewiorek D. P. Introduction to computer organization and data structures: PDP-11 edition. — N. Y.: McGraw-Hill Book Co., 1975.

Szuprowicz B. O. The minicomputer revolution. — Modern Data, June 1973, v. 6, № 6, p. 26.

Tanenbaum A. S. Structured computer organization. — New Jersey: Prentice-Hall Inc., 1976.

Thomas R. T. Microprogram control in minicomputer systems. — Infotech State of the Art Report «Minicomputer Systems», 1977, 2, p. 345–356.

Vaughan W. C. M. The mini's most important part may be its OS. — Electronic Design, July 5, 1975, v. 23, № 14, p. 52–55.

Wagner F. V. Is decentralization inevitable? — Datamation, Nov. 1976, v. 22, № 11, p. 86, 87, 91, 93, 97.

Weitzman C. Minicomputer systems. Structure, implementation and application. Englewood Cliffs, N. J.: Prentice-Hall, 1974.

Wilkinson P. T. Interrupt handling strategies for minicomputers. — Infotech State of the Art Report 13: «Minicomputers», 1973, p. 439–513.

Williamson F. K. Some considerations in minicomputer design. — Infotech State of the Art Report «Minicomputer Systems», v. 2, 1977, p. 357–370.

Zornes J. A. Data base management systems on minicomputers. — Data Base, Summer 1977, v. 9, № 1, p. 9–13.

Auerbach on minicomputers. — N. Y.: Petrocelli Books, 1974.

CDC to cash in on peripheral needs of IBM minicomputers. — Electronics, Jan. 19, 1978, v. 51, № 2, p. 30–40.

Digital announces a PDP-8 with an enormous memory. — Mini-Micro-Systems, Oct. 1977, v. 10, № 10, p. 65.

Minicomputers in the digital laboratory program. COSINE Task Force VII — Minicomputers. — Computer, Jan. 1973, v. 6, № 1, p. 28–42.

Networked minicomputers. — Datamation, Febr. 1975, v. 21, № 2, p. 40–56.

Tandem Computer's Non Stop «Miniplex». — Modern Data, Febr 1976, v. 9, № 2, p. 57–59.

The future of minicomputers. — Automation, Nov./Dec. 1974, v. 9, № 11/12, p. 28–35.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

- Адаптивность 11
Адрес возврата 101, 145, 191, 239
Адресация (адрес) 84
— абсолютная 87, 187, 228
— автодекрементная 89, 187, 222
— автоиндексная 88, 134
— автоинкрементная 89, 187, 221
— базовая 86
— двойная индексная 88
— индексная 87, 224
— косвенная 86, 134, 162, 186, 217
— непосредственная 86, 227, 263
— неявная 92
— относительная 87, 187, 228
— постиндексная 88
— преиндексная 88
— прямая 87, 134, 162, 187, 217
— регистровая 86, 217
— страничная 89, 133, 162
Адресное пространство 41, 243
Адресные команды 76, 132, 160, 186
Аппаратура минисистемы 35
Архитектура миникомпьютеров 74, 256
Ассемблер 59
Базовая конфигурация 55
Байт 41
Безадресные команды 76, 135, 165, 192
Бит 9, 65
Буферный регистр 111
Ввод/вывод
— программируемый 112, 142
— прямым доступом 114, 148
— с прерыванием 113, 145
Вектор прерывания 247
Ветвление 72
Встроенные программы 36
Вычисление адреса 134, 162, 188, 226
Гнездование 103, 105, 248
Датчик времени 29, 53, 207
Диски магнитные 47, 150, 178, 208, 251
Дисплейные терминалы 46, 178, 251
Дополнительный код 66
Единая магистраль 40, 243
Емкость памяти 42, 212
Загрузчик 55
«Закон Гроша» 13
Интерактивный режим 60
Интерпретация 60, 65
Канал ввода/вывода 117
Кассеты магнитные 46, 209, 251
Килобайт (KB) 42
Килослово (KW) 42
Команда-микропрограмма 79, 135, 165, 192
Команда цикла 108
Контроллер 10, 116
Кросс-ассемблер 59, 211
Лента магнитная 48, 150, 251
Линк 80, 131
Магазин (стек) 93
Маска 126, 203
Машинное слово 11, 41, 75
Микропрограммирование 77, 82
Микропроцессор 8, 39
Модем 50
Мультиплексор 30, 51, 52

Надежность миникомпьютеров 13, 41
Наращиваемость 11
Нацеленность на применение 32

Обратная связь 7, 23, 30, 60
Обращение к подпрограмме 101, 192, 239
Общие регистры 88, 214
Операционные системы 54, 155, 180, 210
Очередь 96

Память 41
— внешняя (периферийная) 36, 46
— главная (внутренняя) 41, 69, 149
— управляющая 36, 83
Передача параметров 192, 240
Перескок 77
— безусловный 138, 195
— условный 77, 137, 169, 194
Переход 70
— безусловный 71, 98, 217
— на подпрограмму 101, 133, 239
— условный 77, 99, 235
Периферийное оборудование 44, 150, 177
Перфолента 45
Позиционно-независимая программа 91, 229
Преобразователи А/Ц и Ц/А 52, 151, 179
Прерывание 68
Принтеры 46, 151, 178, 250
Присваивание 68
Программное оснащение 35, 54, 152, 252
Прямой доступ к памяти 114, 148
«Псевдомини» 21

Разделение времени 57
Ранг адреса 85
Распределенная обработка 13, 181
Режим реального времени 12, 56

Связное оборудование 49, 151, 179, 209
Система адресации 84, 133, 161, 186, 217
— прерывания 118, 145, 171, 246
— программирования 58, 154, 182, 253

Слово 64
Совместимость 63, 210, 213
Стек 93, 238
Структурированное программирование 105, 258
«Супермини» 26

Текущая страница 90
Теледоступ 28
Телетайп 44, 150

Указатель стека 89, 93, 238
Устройства ввода/вывода 45, 151, 178, 208, 250
— периферийной памяти 47, 150, 178
— расширенной арифметики 33, 149, 208
— сопряжения с приборами 151, 179, 252

Флажок 143, 172
— готовности 111, 200
— занятости 112, 200
— запрета прерывания 200

Центральная часть минисистемы 36
Центральный процессор 37
Цикл ожидания 113, 143
— памяти 42
Циклический сдвиг 135, 166, 195

«Чип» 39, 43
Числовая интерпретация слова 66
Чистая подпрограмма 191

Эффективность миникомпьютеров 22

ЮНИБАС 40, 243

Язык ассемблера 58, 139, 259
Языки высокого уровня 59, 61, 182, 211

Николай Петрович Брусенцов

МИНИКОМПЬЮТЕРЫ

(Серия: «Библиотечка программиста»)

М., 1979 г., 272 стр. с илл.

Редактор Н. Н. Васина

Техн. редактор Е. В. Морозова

Корректор Н. Б. Румянцева

ИБ № 11064

Сдано в набор 21.12.78. Подписано к печати 15.05.79.
Т-08989. Формат 84×108¹/₃₂. Бумага машиномелованная.
Гарнитура Таймс. Высокая печать. Условн. печ. л. 14,28.
Уч.-изд. л. 14,44. Тираж 30000 экз. Заказ № 417. Цена
книги 90 коп.

Издательство «Наука»

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

Ордена Октябрьской Революции, ордена Трудового Крас-
ного Знамени Ленинградское производственно-техническое
объединение «Печатный Двор» имени А. М. Горького
«Союзполиграфпрома» при Государственном комитете СССР
по делам издательств, полиграфии и книжной торговли
197136, Ленинград, П-136, Гатчинская, 26

90 к.

