## §3. АРХИТЕКТУРА МИКРОПРОЦЕССОРОВ

### 3.1. ИСТОРИЯ РАЗВИТИЯ МИКРОПРОЦЕССОРОВ

В 1959 г. инженеры фирмы «Texas Instruments» разработали способ, как разместить внутри одного полупроводникового кристалла несколько транзисторов и соединить их между собой - родилась первая интегральная микросхема (ИМС). По сравнению с функционально теми же устройствами, собранными из отдельных транзисторов, резисторов и т.п.. ИМС обладает значительными преимуществами: меньшими габаритами, более высокой надежностью и т.д. Неудивительно, что количество выпускаемых микросхем стало быстро возрастать, а их ассортимент неуклонно расширяться. Последнее обстоятельство создавало ряд трудностей для потребителей. Важно даже не столько то, что стремительно возраставшее количество типов ИМС затрудняло ориентацию в море наименований. Значительно большим недостатком была узкая специализация ИМС, из-за которой объем их выпуска не мог быть большим, а значит стоимость одной микросхемы оставалась высокой. Улучшить ситуацию позволило бы создание универсальной логической ИМС, специализация которой определялась бы не заложенной на заводе внутренней структурой, а заданной непосредственно самим потребителем программой работы.

Таким образом, оказывается, что первые микропроцессоры (МП) появились совсем не для миниатюризации ЭВМ, а в целях создания более дешевой логической микросхемы, легко адаптируемой к потребностям пользователя.

История создания первого в мире микропроцессора достаточно поучительна. Летом 1969 г. японская компания «Busicom», разрабатывавшая новое семейство калькуляторов, обратилась за помощью в фирму «Intel». К тому времени «Intel» просуществовала всего около года, но уже проявила себя созданием самой емкой на тот момент микросхемы памяти. Фирме «Busicom» как раз и требовалось изготовить микросхемы, содержащие несколько тысяч транзисторов. Для реализации совместного проекта был привлечен инженер фирмы «Intel» М.Хофф. Он познакомился с разработками «Busicom» и предложил альтернативную идею: вместо 12 сложных специализированных микросхем создать одну программируемую универсальную - микропроцессор. Проект Хоффа победил и фирма «Intel» получила контракт на производство первого в мире микропроцессора.

Практическая реализация идеи оказалась непростым делом. В начале 1970 г. к работе подключился Ф.Фаджин, который за 9 месяцев довел процессор от описания до кристалла (позднее Ф.Фаджин основал фирму «Zilog», создавшую замечательный 8-разрядный процессор Z80, который и сейчас успешно работает во многих домашних компьютерах). 15 ноября 1971 г. «Intel 4004» - так назвали процессор -был представлен общественности.

Поскольку для хранения одной цифры калькулятору требуется 4 бита (именно столько необходимо для изображения десятичных цифр «8» и «9»), «Intel 4004» был четырехразрядным процессором. Следующий микропроцессор предназначался для установки в терминал и должен был обрабатывать символьную информацию. Поскольку каждый символ кодируется одним байтом, следующая модель «Intel 8008» стала 8-разрядной; она появилась в апреле 1982 г. По-прежнему этот процессор был заменой «аппаратной логики», но отдельные энтузиасты уже пытались собрать на нем компьютер. Результаты были скорее демонстрационными, нежели полезными, но микрокомпьютерная революция уже началась.

А в апреле 1974 г. компания «Intel» совершила новый качественный скачок: ее изделие с маркой «Intel 8080» стало первым в мире процессором, походившим на «настоящую» вычислительную машину. Отметим любопытную деталь: хотя процессор. и обрабатывал 8-разрядные данные, но адрес ОЗУ был двухбайтовым! Таким образом, 8080 мог иметь до 64 килобайт памяти, что по тем временам казалось программистам недостижимым пределом.

Дальнейшее развитие событий происходило прямо-таки с фантастической скоростью, даже если сравнивать с темпами динамично развивающейся вычислительной техники. За десятилетие был пройден путь от изобретения 4-разрядного МП до достаточно сложной 32-разрядной архитектуры. Было ликвидировано отставание микропроцессорной техники от обычных ЭВМ и началось интенсивное вытеснение последних (все ЭВМ четвертого поколения собраны на базе того или иного микропроцессора!). Для иллюстрации укажем, что первый МП 4004 содержал 2200 транзисторов, МП 8080 - 4800, МП «Intel 80486» - около 1,2 миллиона, а современный «Pentium» - около 3 миллионов!

История развития микропроцессоров представляет собой достаточно интересную самостоятельную тему. Здесь упомянем только, что пионер в создании процессорных микросхем фирма «Intel» по-прежнему сохраняет свои лидирующие позиции в этой области. Ее программно-совместимое семейство последовательно усложняющихся МП (16-разрядные 8086, 80286 и 32-разрядные 80386, 80486, «Pentium») являются «мозгом» значительной части использующихся компьютеров. Именно на базе этих микропроцессоров собраны все широко распространенные в нашей стране IBM-совместимые компьютеры.

Другую ветку обширного микропроцессорного семейства образуют МП фирмы «Motorola»: ее изделия работают в известных компьютерах «Apple», а также в более простых - «Atari», «Commador», «Amiga» и др. Процессоры «Motorola» ничуть не хуже, а порой даже заметно лучше производимых компанией «Intel». Но на стороне последней - огромные производственные мощности транснационального гиганта IBM и десятки южно-азиатских фирм, буквально наводнившие мир дешевыми IBM-совместимыми компьютерами.

В 1993 г. фирма «Motorola» совместно с IBM и «Apple» разработала новый процессор «PowerPC». Этот процессор имеет очень хорошие технические характеристики, но самое главное в нем - он может эмулировать работу компьютеров и «Apple», и IBM. Очевидно, что это событие еще более обострит конкурентную борьбу на рынке микропроцессоров.

Завершая краткий исторический экскурс, попробуем определить некоторые новые направления развития МП в ближайшем будущем. Характерной чертой последних моделей процессоров является возможность работы в многозадачном режиме, который фактически стал нормой для современных ЭВМ. Развивается RISC-архитектура микропроцессоров (процессоры с минимальным числом команд). Такой МП работает необычайно быстро и способен выполнить любую из своих немногочисленных команд за один машинный такт, в то время как обычно на выполнение простой операции требуется 4-5 тактов. Ярким примером достоинств RISC-архитектуры является уже упоминавшийся процессор «PowerPC». Следует особо подчеркнуть, что успехи RISC-подхода оказывают существенное влияние и на конструирование CISC-процессоров (процессоры с полным набором команд). Так, существенное ускорение классических CISC МП старших моделей семейства «Intel» достигается за счет конвейерного выполнения команд, заимствованного из RISC МП.

И, наконец, нельзя не упомянуть о транспьютерах, содержащих в процессорном кристалле собственное ОЗУ от 2 до 16 кбайт и каналы связи с внешним ОЗУ и с другими транспьютерами. Теоретические возможности этих ИМС, реализующих алгоритмы параллельных вычислений, поражают воображение. Однако потребуется значительное время, прежде чем они смогут быть практически реализованы.

Не следует думать, что бурное развитие микропроцессоров требуется только для вычислительных машин, где МП используются уже не только в качестве центрального процессора, но и в качестве контроллеров для управления сложными периферийными устройствами типа винчестера или лазерного принтера. Все большее число ИМС ставится в изделия, напрямую не связанные с ЭВМ, в том числе и бытовые: лазерные аудио- и видеопроигрывателц, телетекст и пейджинговая связь, программируемые микроволновые печи и стиральные машины, а также многие другие. Очевидно, что число таких управляемых микропроцессорами устройств будет все время возрастать.

### 3.3. ВНУТРЕННЯЯ ОРГАНИЗАЦИЯ МИКРОПРОЦЕССОРА

Перечислим основные функции микропроцессора:

• выборка команд из ОЗУ;

•декодирование команд (т.е. определение назначения команды, способа ее исполнения и адресов операндов);

• выполнение операций, закодированных в командах;

• управление пересылкой информации между своими внутренними регистрами, оперативной памятью и внешними (периферийными) устройствами;

• обработка внутрипроцессорных и программных прерываний;

• обработка сигналов от внешних устройств и реализация соответствующих прерываний;

• управление различными устройствами, входящими в состав компьютера.

Внутреннее устройство микропроцессоров очень сложно (вспомним три миллиона транзисторов в «Pentium»). Даже если попытаться рассмотреть наиболее общую схему основных функциональных узлов, и то получится достаточно сложная картина. К тому же внутреннее устройство МП сильно зависит от его марки, а стало быть изучение структуры одного процессора не обязательно помогает понять работу другого. Следует признать нецелесообразным для пользователя (и даже, может быть, для программиста) изучение инженерных деталей процессора современной ЭВМ, и ограничиться, как это принято делать, только теми функциональными узлами, которые доступны программно. При таком подходе оказывается, что МП имеют много общего, и становятся отчетливо видны некоторые закономерности их внутреннего устройства. Кроме того, исчезает пугающая сложность и возникает приятное и полезное чувство, что компьютер - это не какая-то там «вещь в себе» и его поведение можно понять.

Итак, что же представляет собой микропроцессор с точки зрения программиста? Рассмотрение начнем в наиболее общем виде, не конкретизируя пока тип МП.

Ответ на поставленный вопрос, как ни странно, будет чрезвычайно прост: для программиста любой процессор состоит из набора регистров памяти различного назначения, которые определенным образом связаны между собой и обрабатываются в соответствии с некоторой системой правил. Конечно, программисту доступна не вся внутренняя память процессора: есть множество рабочих (программно-недоступных) регистров, использующихся только во время выполнения команд и т.п.; их мы рассматривать не будем.

Обсуждение внутренних регистров микропроцессора начнем с наиболее важных: *счетчика адреса команд, указателя стека* и *регистра состояния.* Наличие счетчика адреса команд, как уже говорилось выше, было предложено еще в работахфонНеймана. Роль *счетчика* состоит в сохранении адреса очередной команды программы и автоматическом вычислении адреса следующей. Благодаря наличию программного счетчика в ЭВМ реализуется основной цикл исполнения последовательно расположенных команд программы. Заметим, что не во всех МП счетчик команд программно доступен.

В *указателе стека* хранится адрес начала специальным образом организуемого участка памяти стека, описанного в следующем параграфе; роль указателя стека в функционировании процессора достаточно велика.

Наконец, *регистр состояния процессора.* Для разных МП он может называться по-разному (например, слово состояния процессора, регистр флагов и т.п.), но суть его всегда одна: в этом регистре хранятся сведения о текущих режимах работы процессора. Сюда же помещается информация о результатах выполняемых команд, например, равен ли результат нулю, отрицателен ли он, не возникли ли ошибки в ходе операции и т.п. Использование и анализ информации в этом очень важном регистре происходит побитно; иными словами, каждый бит регистра состояния имеет самостоятельное значение. Содержание регистра состояния МП всегда старается сохранить в первою очередь сразу после значения командного счетчика.

Помимо рассмотренных выше, каждый МП имеет набор *рабочих регистров,* в которых хранятся текущие обрабатываемые данные или их адреса в ОЗУ. У некоторых процессоров регистры функционально равнозначны (классическим примером служит процессор машин семейства PDP). в других (к ним принадлежит все интел-ловское семейство МП) назначение регистров достаточно жестко оговаривается. В последнем случае выделяется особый регистр, который принято называть *аккумулятором..* В нем производятся все основные операции и сохраняется их результат

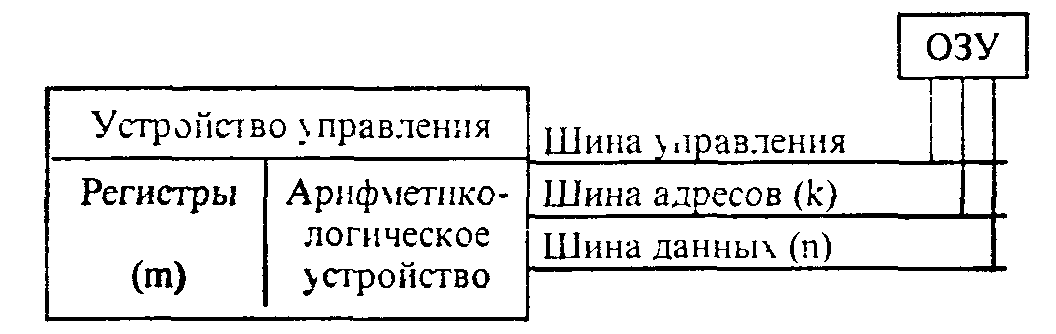
Завершая разговор о регистрах, укажем на существование определенных связей между ними: информация из одного может передаваться в другой. Для машин с равноправными регистрами передача данных возможна между любыми регистрами, для остальных - между строго определенными парами. Так или иначе (в худшем случае за несколько машинных команд) информациюиз одного регистра МП всегда можно перенести в другой.

*Разрядность регистров* МП существенно влияет на возможности всей ЭВМ. Поэтому уточним понятие «разрядность ЭВМ». Оно включает

• разрядность внутренних регистров микропроцессора(m);

• разрядность шины данных (n);

• разрядность шины адреса(k).



*Рис. 4.12.* Обмен информацией между процессором и основной памятью

Схема обмена информацией между микропроцессором и ОЗУ представлена на рис. 4.12. Данные поступают из ОЗУ в регистры процессора и наоборот по шине данных; по шине адреса передается информация о месте хранения данных в ОЗУ. Разрядности этих шин могут не совпадать: например, упомянутый выше МП «Intel 8088» характеризовался значениями m/n/k = 16/8/20. Когда говорят, не вникая в детали, «16-разрядная ЭВМ», то имеют в виду значение т. Поскольку объем адресного пространства ОЗУ, контролируемого МП, равен 2k, то понятно стремление увеличить разрядность шины адреса.

### 3.3. РАБОТА МИКРОПРОЦЕССОРА С ПАМЯТЬЮ. МЕТОДЫ АДРЕСАЦИИ

Адресное пространство МП состоит из множества ячеек памяти ОЗУ, из которых он может брать информацию или засылать ее. Как говорилось выше, начиная с 4-го поколения доминирует байтовая организация памяти, и минимально адресуемой единицей является байт. Например, для ОЗУ емкостью 1024 кбайта адреса байтов таковы:

00000 00001 ... FFFFF

(как принято, записываем их в шестнадцатеричнои системе; адрес последнего байта есть 1024\*1024 -1=1 048 576 - 1 = FFFFF). Длина же ячейки («машинного слова») может быть как один, так и несколько байтов в зависимости от типа процессора и команды, обрабатывающей соответствующую информацию.

При обмене информацией с памятью процессор обращается к ячейкам ОЗУ по их номерам (адресам). Способы задания требуемых адресов в командах ЭВМ принято называть *методами адресации.* От видов и разнообразия методов адресации существенно зависит эффективность работы программы с данными, особенно если последние организованы в определенную структуру.

Для того, чтобы процессор мог извлечь данные из ячейки ОЗУ или поместить их туда, необходимо где-то задать требуемый адрес. Если адрес находится в самой команде, то мы имеем дело с *прямой адресацией.* Поскольку при подобном способе слишком сильно возрастает длина команды, то, чтобы избежать этого неприятного эффекта, при обращении к ОЗУ процессор использует метод *косвенной адресации.* Идея состоит в том, что адрес памяти предварительно заносится в один из регистров МП, а в команде содержится лишь ссылка на этот регистр. Если учесть, что при хранении адреса в регистре его еще очень удобно модифицировать (скажем, циклически увеличивая на заданную величину), становится понятным, почему косвенная адресация нашла такое широкое применение.

Приведем описание наиболее распространенных вариантов ссылок на исходную информацию (учитывая, что терминология для разных МП может различаться, названия методов адресации не приводятся):

1) данные находятся в одном из регистров МП;

2) данные входят непосредственно в составкоманды, т.е. размещаются после кода операции (операции с константой);

3) данные находятся в ячейке ОЗУ, адрес которой содержится в одном из регистров МП;

4) данные находятся в ячейке ОЗУ, адрес которой вычисляется по формуле

*адрес = базовый адрес + смещение.*

Базовый адрес хранится в одном из регистров МП и является начальной точкой массива данных. Смещение может быть как некоторой константой, так и содержимым другого регистра. Часто такой способ доступа к ОЗУ называют *индексным,* так как это похоже на нахождение элемента в одномерном массиве по его индексу.

Следует подчеркнуть, что здесь описаны лишь наиболее общие методы адресации. У конкретных моделей МП существуют некоторые особенности адресации ОЗУ. Кроме того, имеющиеся методы адресации могут быть комбинированными. Так, например, в процессорах семейства PDP возможна двойная косвенная адресация: данные хранятся в ячейке ОЗУ, адрес которой хранится в ячейке, адрес которой находится в указанном регистре (как тут не вспомнить детский стишок о синице, которая ворует пшеницу, которая в темном чулане хранится, в доме, который построилДжек?).

Методы адресации могут быть и более экзотическими. Рассмотрим, например, широко распространенный *сегментный* способ, принятый в процессорах фирмы «Intel». Известный американский программист Питер Нортон метко назвал эют способ словом «клудж» (от английского «kludge» - приспособление для временного устранения проблемы). Сегментный метод адресации был предложен для первого 16-разрядного МП 8086 для того. чтобы, используя 16-разрядные регистры, можно было получить 20-разрядный адрес и тем самым расширить максимально возможный объем ОЗУ с 64 кбайт до 1 Мбайта. Суть метода состоит в том, что адрес ОЗУ вычисляется как сумма двух чисел (сегмента и смещения), причем одно из них сдвинуто влево на 4 двоичных разряда, т.е. умножено на 16. Пусть, например, сегмент в шестнадцатеричном виде равен АООО, а смещение - 1000. Общепринятая запись такого адреса имеет вид АООО: 1000. Итоговый адрес равен

А000()

+ 1000

А1000

«Такой способ адресации более 64 кбайт памяти кажется довольно странным, однако он работает», - не без некоторой иронии замечает по этому поводу Питер Нортон.

Добавим, что, помимо всего прочего, при сегментной адресации один и тот же адрес ОЗУ может быть представлен несколькими комбинациями чисел (в самом деле, одну и ту же сумму можно получить разными способами)

Поскольку современные интелловские процессоры, начиная с 80386, стали 32-разрядными, их регистров хватает, чтобы адресовать до 4 Гбайт памяти Сегментный способ при этом становится излишним, хотя и сохраняется как вариант ради обеспечения программной совместимости с предыдущими моделями.

Нам осталось рассмотреть еще один очень специфический, но интересный и часто используемый на практике способ адресации данных в ОЗУ. Речь пойдет о работе со стеком. Сам термин происходит от английского слова «stack», имеющего множество значений, причем ни одно из них не подходит достаточно точно. Чаще всего в литературе упоминается магазин револьвера, который по очереди обеспечивает доступ к каждому патрону. Даже если не учитывать ненужную «милитаризацию» терминологии, такой перевод явно не подходит к современной реализации стека. Во-первых, стек не является круговым, а во-вторых, информация в стеке микрокомпьютера не смещается (для точности аналогии пришлось бы признать, что магазин покоится, а вращается револьвер!). Поэтому возьмем термин «стек» как есть, без перевода, и перейдем к определению.

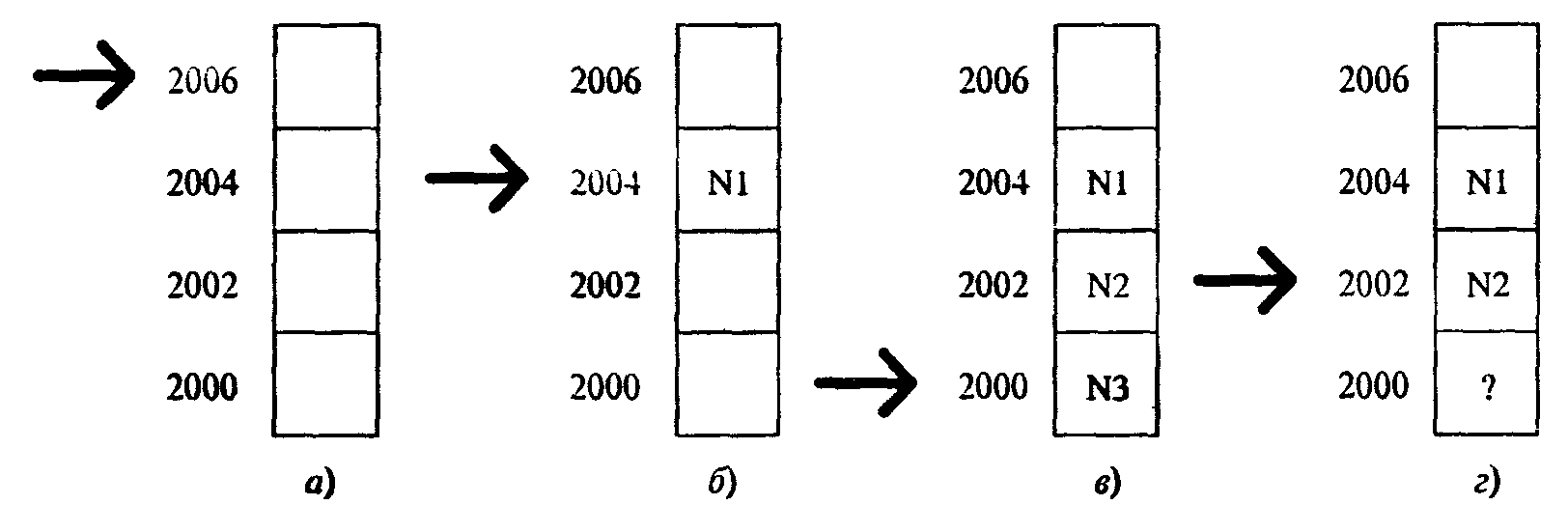
*Стек -* это неявный способ адресации, при котором информация записывается и считывается только последовательным образом с использованием, так называемого, указателя стека.

Обратимся к примеру. Пусть требуется на время сохранить значения трех целочисленных переменных N1, N2 и N3, а затем их все сразу восстановить. Попробуем воспользоваться для этого стековой памятью. Прежде всего запомним, что стек всегда имеет единственный вход и выход информации - для .хранения его адреса и нужен указатель стека. Пусть для определенности он сейчас содержит адрес 2006 (рис. 4.13, а). Тогда по команде «записать в стек N1» (обратите внимание, что в команде не фигурирует явно ни адрес ОЗУ, ни регистр МП!) процессор проделает следующее:

1) уменьшит указатель стека на 2 (целое число занимает в памяти 2 байта);

2) запишет N1 по полученному адресу 2004 (рис.4.13, б).

Аналогично при выполнении команд «записать в стек N2» и «записать в стек N3» значения этих переменных попадут в ячейки 2002 и 2000, причем указатель стека станет равным 2000 (рис. 4.13, в).



*Рис. 4.13.* Пример для изучения принципа работы стека

Теперь займемся извлечением информации. Как видно из рис. 4.13, в, указатель сейчас «направлен» на значение переменной N3, а значит считывать придется, начиная с него. Выполним команду «прочитать из стека N3». При этом процессор

1) считает в N3 значение из стека;

2) увеличит указатель на 2 (рис. 4.13, г).

Аналогично прочитаем N2 и N1, после чего стек опустеет и вернется к начальному состоянию, изображенному на рис. 4.13, а.

*Примечание.* Обратим внимание на то, что значение в стеке после считывания, конечно же, не исчезает, как это условно показано на рис. 4.13, г, но есть причины полагать, что мы его там можем больше не увидеть. Дело в том, что процессор имеет право иногда временно использовать стек для своих «внутренних» нужд. При этом некоторые ячейки, адреса которых меньше текущего указателя стека, естественно, изменятся. Следовательно, во избежание неприятностей лучше всегда считать, что после считывания информации в стековой памяти она пропадает.

Подводя итоги рассмотрения примера, обратим внимание на следующее. Та информация, которая заносится в стек первой, извлекается последней и наоборот. Обычно об организации стека говорят: «первым пришел - последним ушел». Удобно представить себе аналогию с детской пирамидкой, нижнее кольцо которой невозможно снять до тех пор, пока не будут сняты все остальные.

Еще раз подчеркнем, что в командах работы со стеком адрес ОЗУ не фигурирует в явном виде. Но при этом молчаливо предполагается, что указатель стека уже задан. Как правило, инициализацию указателя стека компьютер делает сам, но не мешает поинтересоваться, так ли это в вашем конкретном случае и достаточно ли памяти под стек имеется при этом. Если указатель стека определен неправильно, то запись в стек может разрушить полезную информацию в ОЗУ и вызвать непредсказуемые последствия.

Интересно, что со стековым способом организации мы имеем дело в жизни гораздо чаще, чем это может показаться на первый взгляд. Во-первых, дисциплине «первым пришел - последним ушел» подчиняются скобки в арифметических выражениях: первой всегда закрывается «самая внутренняя», т.е. как раз последняя скобка. Если вы знакомы с программированием на любом алгоритмическом языке, то без труда найдете и еще два примера: вложенные циклы и подпрограммы.

Итак, мы подробно рассмотрели общие принципы записи информации в стек. Остается сказать, что стековый способ организации ОЗУ используется в вычислительной технике очень широко. На аппаратном уровне процессор «обучен» сохранять в стеке текущий адрес программы при выполнении команды перехода с возвратом, т.е. при вызове подпрограммы. Часто программа предварительно заносит в тот же самый стек необходимые для подпрограммы параметры: так реализуется, например, вызов процедур и функций с параметрами в языке «Паскаль». Кроме того. стек часто используется для временного сохранения значений тех или иных внутренних регистров процессора. Наконец, процессор активно использует стек при реализации прерываний от внешних устройств (об этом будет рассказано в следующем параграфе).

### 3.4. ФОРМАТЫ ДАННЫХ

Почти каждая команда процессора нацелена на обработку данных, местонахождение которых определяется значениями адресов операндов. Для понимания работы процессора существенно представлять, какого рода данные он может обрабатывать.

Если в ходе исполнения программы ее остановить, прочитать содержимое какой-нибудь ячейки памяти ОЗУ, к которой в это мгновение обращается процессор, и попытаться понять, что именно хранится в этой ячейке, то это чаще всего невозможно сделать, не расшифровав код выполняемой операции - сам по себе хранимый в ячейке двоичный код может быть и числом, и командой, и кодом символа, и чем-то еще. Все дело в том, как его интерпретирует работающая с ним команда.

Перечислим некоторые форматы данных, типичные для 16- разрядной ЭВМ.

***8- битовые целые числа без знака.*** Каждое такое число занимает 1 байт и воспринимается процессором как целое положительное число. Следовательно, диапазон представимости чисел в этом формате от 00000000 до 11111111, т.е. от 0 до FF в шестнадцатеричной системе (от 0 до 255 в десятичной).

***8- битовые целые числа со знаком.*** В этом случае величина числа задается семью битами, а значение старшего бита определяет знак числа (0 - положительное, 1 -отрицательное). Например, в этом формате код 01101011 означает число +6В.

Однако код 11101011 не означает, как можно подумать, число -6В, так как для кодирования отрицательных чисел применяется, так называемый, дополнительный код. Он образуется следующим образом:

• находится восьмиразрядное двоичное представление абсолютной величины числа;

• найденный код инвертируется, т.е. в нем нули заменяются на единицы и наоборот;

• к полученному коду арифметически прибавляется единица.

Например, процесс получения дополнительного кода десятичного числа -75 таков:

01001011 10110100 10110101

/

Использование дополнительного кода облегчает организацию арифметических действий в процессоре (который, как правило, не имеет даже аппаратно реализованной операции вычитания - все удается свести к сложению целых чисел).

Обратная процедура - восстановление значения числа подополнительному коду -осуществляется по тому же правилу, что и прямая.

Диапазон представимости чисел в этом формате: от-128 до +127.

***16- битовые целые числа со знаком и без знака.*** Они в точности аналогичны 8-битовым, но код имеет вдвое большую длину. Соответственно, многократно возрастает диапазон представимости: для чисел без знака от 0000 до FFFF (т.е. от О до 65535 в десятичной системе), для чисел со знаком - от -8000 до +7FFF (т.е. в десятичной системе от -32768 до +32767).

***8- битовые символы.*** В этом формате двоичный код интерпретируется обрабатывающей его командой как код символа. При работе с персональными ЭВМ обычно используется система кодирования ASCII, о которой говорилось в главе 1. В этой системе стандартизированы (закреплены за определенными символами) коды, у которых значение старшего бита равно 0; все прочие коды остаются за символами национальных алфавитов и дополнительными специальными символами.

***Битовые поля.*** В этом формате значащим является не весь 8- или 16- разрядный код в целом, а каждый из составляющих его битов. Один из примеров битового поля - содержимое регистра состояния процессора (см. выше). Другой пример -форма хранения множеств в языке «Паскаль».

Существуют и другие форматы данных - двоично-десятичные числа, строки и т.д.

### 3.5. ОБРАБОТКА ПРЕРЫВАНИЙ

Важную роль в работе современного МП играют прерывания. Они всегда нарушают естественный ход выполнения программы для осуществления неотложных действий, связанных, например, с реакцией на щелчок мыши или сбой в цепи электропитания.

События, вызывающие прерывания, можно разделить на две группы: *фатальные* и *нефатальные.* На фатальные (неотвратимо наступающие) процессор может реагировать единственным способом: прекратить исполнение программы, проанализировать событие и принять соответствующие меры (чаще всего - сообщить причину прерывания пользователю и ждать его реакции). Однако часто можно с остановкой программы повременить: запомнить, что прерывание было, и продолжать исполнять программу. Например, сложение с переполнением разрядной сетки -фатальное событие, после которого остановка неизбежна; попытка вывода на принтер, не готовый к приему информации, может быть отложена (с сохранением этой информации).

Основные виды прерываний - *внутрипроцессорные прерывания* и *прерывания от внешних устройств.* Первые связаны с возникновением непреодолимого препятствия при выполнении программы. Причин может быть много: из памяти выбрана команда с несуществующим кодом или адресом, в ходе исполнения команды возникло переполнение разрядной сетки ЭВМ или произошла попытка записи в оперативную память, отведенную другой задаче. В большинстве подобных случаев дальнейшее выполнение программы становится невозможным и управление передается системе, обеспечивающей прохождение задач (чаще всего это - операционная система), которая и принимает меры по обработке внешней нештатной ситуации.

Гораздо реже, но возможны, нефатальные внутрипроцессорные прерывания, например, специальные отладочные прерывания. Такие прерывания обеспечивают возможность пошагового исполнения тестируемой программы под контролем специальных программных средств.

Прерывания второй группы возникают по требованиям какого-либо из многочисленных внешних устройств. Такое событие обычно не приводит к фатальному завершению задачи,напротив, это вполне нормальная ситуация. Поэтому при таком прерывании МП принимает меры для того, чтобы обеспечить дальнейшее выполнение программы: запоминает в стеке текущее значение счетчика команд и содержимое регистра состояния. Затем происходит переход на подпрограмму обработки данного прерывания. После ее выполнения процессор восстанавливаетизстека значения своих регистров и продолжает выполнение прерванной программы.

Прерывания от внешних устройств подробно рассматриваются в следующем пункте.

Отметим, что термин «прерывание» часто используется еще в одном значении. Речь идет о, так называемых, программных прерываниях. Например, для IBM-совместимых компьютеров существуют многочисленные команды прерывания 1NT с самыми разнообразными номерами. Следует понимать, что 1NT - это одна из инструкций процессора; чтобы она заработала, ее код должен содержаться в программе. В противоположность этому, «настоящие» прерывания возникают аппаратно и не требуют наличия каких-то специальных команд в тексте прерываемой программы. Более того, аппаратное прерывание может произойти между двумя любыми командами программы.

К программным относятся и межмашинные прерывания, возникающие в локальной сети при обмене информацией между компьютерами.

Запрет нефатального прерывания называется *маскировкой;* маскировка задается программистом или системной программой. Делается это либо с помощью установления вида битового поля в специальном регистре маски прерываний, в котором значения разрядов (0 или 1) связаны соответственно с отсутствием или наличием маскировки закрепленного за этим разрядом прерывания, либо с аналогичным использованием разрядов регистра состояния процессора.

После получения сигнала о незамаскированном прерывании процессор делает следующее:

• запоминает состояние прерванной программы;

• распознает источник прерывания;

• вызывает и выполняет специальную системную программу обработки прерываний;

• восстанавливает состояние прерванной программыи, при возможности, продолжает ее исполнение.

Для распознавания источника прерывания анализируются некоторые биты регистра состояния процессора, состояние внешних устройств и т.д. Для запоминания состояния прерванной программы чаще всего используется стек. Выше мы уже говорили о стеке в ракурсе организации особого способа адресации. Кроме того, стек используется процессором как для организации механизма прерываний, так и для обработки обращения к подпрограммам, передачи параметров и временного хранения данных.

Назначение программы обработки - понять и в удобной для пользователя форме вывести (обычно на экран) сообщение о причине прерывания и (иногда) дать рекомендации по возможной реакции на эту причину.

Все это разительно отличается от ситуации с машинами 1-го и 2-го (а отчасти и 3-го) поколений, когда пользователь без помощи системного программиста в большинстве случаев не мог разобраться в причине события, вызвавшего прерывание его программы на этапе исполнения. Такая ситуация стала нетерпимой для пользователей персональных компьютеров, и чем совершеннее анализатор программы обработки прерываний, тем выше уровень «дружелюбности» пользовательского интерфейса.

### 3.6. РАБОТА МИКРОПРОЦЕССОРА С ВНЕШНИМИ УСТРОЙСТВАМИ

Выше было описано, как процессор обменивается информацией с ее наиболее важным и оперативным источником - памятью. Рассмотрим теперь, как МП может принять данные или передать их внешним устройствам. Способ решения этой задачи в зависимости от конструкции ЭВМ (подчеркнем: не от конструкции МП, а от конструкции всей ЭВМ!) может быть одним из двух указанных ниже:

1) устройства ввода-вывода включаются в общее адресное пространство;

2) устройства ввода-вывода имеют собственное адресное пространство.

В первом случае при обращении к определенным адресам памяти вместо обмена с ОЗУ происходит аппаратное подключение того или иного внешнего устройства. При этом для «общения» с внешними устройствами и с памятью используются одни и те же команды МП, хотя, конечно, обмен с внешним устройством протекает по более сложному протоколу, чем с памятью.

Во втором случае внешние устройства образуют отдельное адресное пространство, обычно значительно меньшее, чем у ОЗУ. Каждая ячейка этого дополнительного адресного пространства называется *портом.* Каждому внешнему устройству обычно соответствует несколько портов с последовательными адресами. Обмен процессора с организованными подобным образом устройствами осуществляется специальными командами ввода-вывода.

На самом деле, с точки зрения схемной организации оба описанных способа имеют очень много общего. При любом обмене, будь то обращение к ОЗУ, ПЗУ или внешнему устройству, процессор выставляет адрес информации на единую адресную шину, а данные передает или принимает по общей шине данных. Выбор же требуемого адресата - ячейки памяти или порта - осуществляется подачей специального управляющего сигнала.

В одном и том же компьютере могут встречаться оба способа адресации устройств ввода-вывода одновременно. Так, в ПЭВМ «Ямаха» накопитель на магнитных дисках включен в общее адресное пространство памяти, а печатающее устройство оформлено в виде нескольких портов. Некоторые порты этого компьютера служат для подключения к небольшому 64-килобайтному адресному пространству 128 кбайт ОЗУ и многочисленных ПЗУ. При этом все ресурсы памяти разбиваются на отдельные страницы по 16 кбайт каждая, и «активными» в каждый момент времени могут быть только четыре из них.

Рассказ о работе с периферийными устройствами был бы неполным без описания того диалога, который ведет с ними МП. Рассмотрим этот диалог на примере простейших устройств ввода-вывода - например, печатающего устройства.

Обмен с подключенным к ЭВМ печатающим устройством производится через два основных порта - порт состояния и порт данных. В первом хранится информация о состоянии устройства в данный момент времени, а во второй МП помещает данные для вывода на бумагу. Каждый бит порча состояния хранит ответ на вполне определенный вопрос: заправлена ли бумага, готов ли принтер принять данные от компьютера и т.д. Все это для МП - входные сигналы. Но есть и выходные, которые через порт передаются от МП к печатающему устройству. Наиболее важным из них является бит, свидетельствующий о готовности информации в порту данных к передаче. Этот управляющий сигнал часто называют *стробом.*

В наиболее простом случае обмен информацией между процессором и принтером может протекать следующим образом. Пусть МП должен вывести на печать какой-нибудь символ. Он считывает порт состояния принтера и анализирует содержимое его бита готовности. Если результат положительный, т.е. печатающее устройство готово принять информацию, обмен продолжается, в противном случае МП снова считывает порт состояния и повторяет анализ. Когда процессор получит от принтера сигнал о готовности к обмену, он заносит требуемый символ в порт данных и установкой стробирующего сигнала сообщает об этом принтеру. Затем МП снова периодически считывает порт состояния, но следит уже за другим битом (через этот бит принтер сообщит процессору о том, что данные приняты, т.е. скопированы в собственное ОЗУ принтера). После этого МП убирает стробирующий потенциал и продолжает работу по программе.

Конечно, реальный диалог современного компьютера с печатающим устройством сложнее. Вот пример. Вы, наверное, заметили, что описанный выше алгоритм «зациклится», если принтер не подключен или не готов к работе: процессор будет ждать сигнала о готовности принтера, а того не будет. Выход - ожидать сигнала готовности в течение какого-то времени, а затем выдать пользователю сообщение о неготовности принтера к работе. При этом желательно проверить содержимое других битов порта состояния (например, отвечающих за наличие бумаги и т.п.) и уточнить, если потребуется, диагностическое сообщение.

Обмен информацией между МП и устройством ввода следует рассмотреть отдельно, так как он может иметь некоторые существенные особенности. Описанная выше идеология передачи данных, конечно, может быть использована и для устройств ввода - например, для клавиатуры. В этом случае программа в тот момент, когда ей потребуются входные данные, опросит состояние клавиатуры и примет с нее данные. Однако для сложных современных программных систем такой метод неудобен из-за существенного замедления времени реакции ЭВМ. В самом деле, если некоторая сложная программа занята вычислениями, а вы хотите ее прервать, то придется подождать, пока процессор освободится и займется опросом клавиатуры (собственно говоря, тогда и прерывать-то будет нечего!). Все это сильно напоминает очередь в приемной бюрократа-начальника, когда люди вынуждены часами ждать, пока их вызовут для пятиминутного решения вопроса.

Для предотвращения подобных неприятностей во всех ЭВМ на базе МП наряду с программным опросом устройств ввода существует еще один механизм - механизм *прерывания от внешних устройств.* Такие прерывания настолько широко используются в компьютерах, что имеет смысл рассмотреть их подробнее. Для понимания сути работы прерываний снова обратимся к аналогии с руководителем. Пусть в его кабинете идет совещание по подведению итогов деятельности предприятия, и в этот момент по телефону поступает очень важная информация, требующая немедленного принятия решения. Как в таком случае обычно развиваются события? Секретарь, не дожидаясь конца совещания, сообщает шефу о звонке. Тот, прервав свое выступление, снимает трубку и выясняет суть дела. Затем он либо тут же принимает решение и сообщает его, либо предлагает пока сделать самое необходимое, а после совещания обещает перезвонить и дать дальнейшие указания. Затем (обратим внимание на эту деталь!) он произносит что-нибудь в духе «Так, на чем мы остановились?» и продолжает совещание как ни в чем не бывало. Впрочем, если ситуация экстренная, то совещание может быть прекращено или в его ход могут быть внесены определенные коррективы.

Вернемся к компьютеру. Каждое его устройство (клавиатура, мышь, дисковод и др.) способно затребовать внимание процессора, выставляя сигнал требования прерывания. Процессор проверяет наличие этого сигнала после выполнения каждой операции. «Увидев» требование прерывания, МП немедленно начинаетего обрабатывать. Прежде всего, он запоминает свое текущее состояние (счетчик команд и регистр состояния) с тем, чтобы в дальнейшем иметь возможность продолжить выполнение прерванной программы. После этого происходит переход на программу, обрабатывающую данное прерывание. Заметим, что все перечисленные действия реализуются на аппаратном уровне, т.е. фактически являются «врожденными рефлексами» МП.

Программа обработки прерывания, получив управление, проводит оперативный анализ причины прерывания. Если причина серьезная и требует немедленных действий, то эти действия выполняются. Если же с обработкой можно подождать (например, просто нажата обычная символьная клавиша), в специальную область памяти (буфер) заносится информация о происшедшем прерывании. Во всех случаях прерывания завершаются выполнением специальной команды возврата, которая восстанавливает содержимое программного счетчика и регистра состояния, давая тем самым микропроцессору возможность продолжить работу прерванной программы. Естественно, что программа обработки прерывания должна была предварительно восстановить все испорченные ей рабочие регистры МП.

Реальная картина, конечно, еще сложнее: прерывания от разных устройств имеют разные уровни приоритета, существует возможность маскировки прерываний и т.п. Однако сути дела все это существенно не меняет.

В последнее время необходимость понимания механизма работы прерываний сильно возросла в связи с возникновением идеологии программирования по событиям. Она связана с распространением среды «Windows» и лежит в основе систем типа «Visual Basic» или «Delphi». Приведем примеры нескольких событий, на которые программа может реагировать: сдвинута мышь, нажата (или отпущена) клавиша мыши, нажата клавиша «ввод», выбран тот или иной пункт меню, открыто новое окно на экране и многие-многие другие. Полный перечень событий занимает в описании несколько страниц. Интересно, что программа на таком языке уже не является единым целым: на каждое событие пишется своя собственная программа, хотя все они и могут быть связаны между собой.

Итак, мы рассмотрели наиболее общие принципы работы микропроцессоров вне зависимости от их модели. Теперь познакомимся с конкретными процессорами.

### 3.7. ПРИМЕР: СИСТЕМА КОМАНД ПРОЦЕССОРОВ СЕМЕЙСТВА PDP

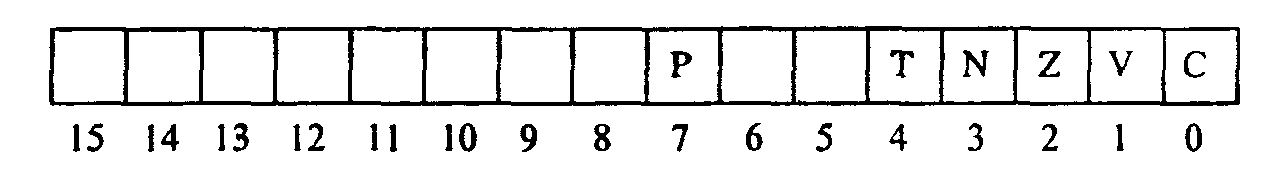
В качестве примера реально существующего процессора, удобного для более детального изучения, возьмем процессор серии машин, созданных фирмой DEC (США) и известных под названием PDP-11 (в нашей стране аналогичные процессоры использовались в семействах 16-разрядных мини- и микро-ЭВМ «Электроника», ДВК, БК, а также в школьном компьютерном классе УКНЦ; всего таких машин было выпущено в СССР до 1,3 млн. штук и примерно столько же в США). Это семейство мини- и микро-компьютеров - одно из самых долгоживущих: первая машина этой серии была выпущена в 1970 г., а прекращение выпуска было запланировано фирмой DEC на 1997 г.

Причина выбора процессора PDP для иллюстрации обсуждаемого вопроса состоит в том, что система команд этого процессора построена на простых и наглядных принципах, изучив и запомнив которые уже можно составлять несложные программы. В то же время система команд других процессоров, например, широко распространенных представителей семейства «Intel», устроена значительно сложнее, требует запоминания большого количества справочных данных. В качестве подтверждения сказанного достаточно указать, что в команде процессора PDP может быть использован любой из имеющихся внутренних регистров, тогда как многие команды процессоров «Intel» оперируют с фиксированными регистрами, не допуская альтернативного расположения данных и результатов.

Процессор машин серии PDP и его отечественные аналоги (при дальнейшем изложении будем все это обширное семейство для краткости называть «процессор PDP») с точки зрения программиста устроен довольно просто. Он состоит из восьми *регистров общего назначения* (РОН) и *особого регистра,* в котором отображается текущее состояние процессора (в иностранной литературе его принято обозначать PSW - Processor Status Word). Любой из регистров общего назначения может быть использован в командах на равных основаниях. Вместе с тем имеются два выделенных регистра, содержимое которых процессор использует для собственных нужд. Прежде всего следует назвать регистр R7, выполняющий роль счетчика команд, в котором хранится адрес следующей инструкции программы. Другим выделенным регистром является указатель стека R6, используемый при запоминании информации в момент вызова подпрограмм и при переходе к обработке прерывания.

Остальные 6 регистров, обозначаемые RO, Rl. R2, R3, R4, R5, программист может полностью использовать по своему усмотрению. Имеется только одно (непринципиальное) исключение: в обширной системе команд PDP существует лишь единственная, и то достаточно экзотическая, команда (MARK), работающая с регистром R5.

Следует отметить, что и выделенные регистры процессора R6 и R7 могут быть использованы в любой команде наряду с обычными РОН RO-R5. Например, допускается переписать информацию из R7 в R3 и тем самым сообщить программе адрес памяти, в котором находится следующая команда программы. Такой прием очень часто используется программистами для автоматической «самонастройки» на те адреса ОЗУ, в которых оказалась загруженной программа. Интересно, что такого простого доступа к содержимому программного счетчика многие процессоры не имеют.



*Рис. 4.14.* Схема регистра состояния процессора PDP

Регистр состояния процессора PSW, как и все РОН, является 16-битным. Его особенностью является то, что каждый его бит имеет самостоятельное значение и может использоваться процессором отдельно от других. В управляющих битах регистра постоянно отображается информация о выполняемых результатах (отрицательность, равенство нулю и т.п.), а также о состоянии процессора в данный момент (например, один из битов позволяет разрешать или запрещать обработку прерывании). Из всех используемых битов (не все 16 бит регистра состояния задействованы!) наиболее используемы два; их принято обозначать N и Z. Управляющий признак N (Negative) отражает знак результата операций: если число отрицательное, то N *=* 1, неотрицательное - N = 0. Бит Z (Zero) говорит о равенстве или неравенстве нулю результата: в первом случае Z = 1, во втором - Z = 0. Путем несложных рассуждений можно убедиться в том, что из шести известных математических соотношений

=, < , > , ≤ , ≥ , ≠

четыре могут быть проанализированы по одному из признаков, а два оставшихся требуют совместного анализа битов N и Z.

Другие управляющие биты, изображенные на рисунке, используются следующим образом. Бит Р - задание маскировки прерываний от внешних устройств (Р = 0 -прерывание состоится. Р = 1 - замаскировано). Бит Т - «признак ловушки»; при Т = 1 после прерывания запускается специальная системная программа, позволяющая на этапе отладки текущей программы осуществлять режим трассировки. Бит V -признак переполнения разрядной сетки при арифметической операции (если это произошло, устанавливается V = 1); бит С - аналогично при переполнении разрядной сетки при логической операции, сопровождаемой сдвигом кода. Описанные управляющие признаки широко используются для реализации разветвлений программы в зависимости от полученных результатов, анализа причин прерываний и других действий.

Процессор PDP имеет достаточно удобный и широкий набор команд. Основная их часть является двухадресными и одноадресными, т.е. они обрабатывают два или один операнд, соответственно. Для выполнения некоторых управляющих команд данные не требуются (например, команда остановки программы) - такие команды не содержат ссылок на операнды. Форматы одно- и двухадресных команд изображены ниже на рис. 4.15.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| КОП | | | КМА | N POH |
| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | | | | |
| КОП | KMA1 | N POH1 | KMA2 | N POH2 |
| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | | | | |

*Рис.4.15.* Формат одноадресной команды (вверху) и двухадресной (внизу). КОП - код операции. КМА - код метода адресации, N РОН - номер регистра общего назначения (код адреса)

Примеры некоторых наиболее важных операций приведены в табл. 4.2. Действия, выполняемые по простейшим арифметическим операциям, понятны из пояснений. В некоторых дополнительных пояснениях нуждаются только команды переходов.

Во-первых, *переходы* бывают *абсолютные* (на заданный адрес) и *относительные* (на определенное число команд относительно данной). Переходы первого типа более наглядны и могут быть реализованы на любой адрес ОЗУ. Относительные переходы требуют вычисления адреса перехода, зато они не привязаны к конкретным адресам ОЗУ: программа, использующая относительные переходы, может работать в любом месте памяти. Относительные переходы занимают меньше места в памяти, чем абсолютные; вследствие этого диапазон относительных переходов ограничен (в процессоре PDP только на 127 слов вперед и 128 - назад).

Таблица 4.2

**Наиболее важные команды процессора PDP**

|  |  |  |
| --- | --- | --- |
| Мнемоника  команды | Содержание команды | Пояснения |
| HLT | Остановка | Выполнение программы прекращается |
| CLR А | Очистить А | А:=0 (в операнд А засылается 0) |
| INC А | Увеличить А на 1 | А:=А+ 1 (значение А увеличивается на 1) |
| DEC A | Уменьшить А на 1 | А:=А-1 (значение А уменьшается на 1) |
| MOV А,В | Переписать А в В | В:=А (В принимает значение А) |
| ADD A,B | Сложить А в В | В:=А + В (сумма помещается в В) |
| SUB A,B | Вычесть А из В | В:=В-А (разность помещается в В) |
| СМР А,В | Сравнить А с В | По знаку разности А-В устанавливаются |
|  |  | Управляющие биты; А и В сохраняются |
| JMP А | Безусловный переход к А | Безусловный переход по адресу А |
| BR К | Безусловный переход на К. слов | Безусловный переход через К команд |
| BEQ К | Переход по = 0 | Переход на К слов, если результат = 0 |
| BNE К | Переход по *≠* 0 | Переход на К слов, если результат *≠* 0 |
| BPL К | Переход по ≥ 0 | Переход на К слов, если результат ≥ 0 |
| ВМ1 К | Переход по < 0 | Переход на К. слов, если результат < 0 |
| JSR А | Вызов подпрограммы по адресу А | Переход на адрес А с запоминанием адреса возврата для команды RET |
| RET | Возврат из подпрограммы | Возврат на команду, следующую за вызовом  подпрограммы |

Во-вторых, переходы делятся на *безусловные* и *условные.* Безусловные, как. следует из названия, происходят всегда. При выполнении условного перехода анализируются управляющие биты. Переход происходит только в том случае, когда признаки имеют требуемые значения; в противном случае переход игнорируется и выполняется следующая за ним команда. Чаще всего условному переходу предшествует команда сравнения, «подготавливающая» управляющие признаки для анализа. Отметим, что команды BEQ и BNE проверяют значение признака Z, a BPL и BMI -признака N.

В таблице у команд перехода для наглядности указаны адреса или величина смещения в словах. Именно таким образом и кодируются команды в памяти ЭВМ. Однако при записи текстов программ дело обычно обстоит несколько проще: в требуемых местах программы ставятся метки, а расчет конкретных адресов и смещений машина производит самостоятельно (см. приводимые ниже примеры).

В приведенных в таблице командах в качестве операндов фигурируют условные обозначения А и В. Расшифруем их. Система команд процессора PDP построена достаточно логично и закономерно, поэтому любой операнд- и первый, и второй - в любой операции задается совершенно одинаково. В PDP существует 8 различных способов задать местонахождение информации, требующейся для выполнения операции. Их принято называть *методами адресации,* и все они связаны с различным использованием РОН. Наличие развитой системы методов адресации делает программирование для процессора PDP очень гибким и мощным.

Опишем коротко организацию основной памяти, контролируемой обсуждаемым микропроцессором. Основная адресация - байтовая; каждый байт имеет уникальный адрес. Так как процессор является 16-разрядным, то объем контролируемого им номинально адресного пространства 64 кбайта; на деле программисту для размещения программы и данных доступно меньшее пространство, так как 8 кбайт памяти зарезервировано для общения с внешними устройсгвами. Реально, однако, с помощью специальных механизмов преобразования исполнительных адресов емкость ОЗУ, доступную пользователю, делают значительно большей (детали рассматривать не будем).

Байт может быть операндом для некоторых команд. Однако, большая часть команд обрабатывает операнды длиной 2 байта - «слово». Слово состоит из двух соседних байтов; адресом слова считается адрес младшего входящего в него байта.

Таблица 4.3

**Некоторые способы указания операндов в командах**

|  |  |  |
| --- | --- | --- |
| **Название** | **Мнемоника** | **Пояснение** |
| Регистровая  Косвенно-регистровая  Автоинрементная  Автодекрементная | Rn  (Rn)  (Rn)+  -(Rn) | Операнд в регистре Rn  Операнд в ячейке ОЗУ, адрес которой в Rn  То же, но после чтения Rn увеличивается на 1  То же, но перед чтением Rn уменьшается на 1 |

В качестве иллюстрации разберем выполнение команды MOV(R1)+,RO

По этой команде сначала считывается информация из памяти по адресу, находящемуся в R1. После чтения значение R1 автоматически увеличивается, так что при последующем выполнении данной команды будет обрабатываться следующая ячейка. Операция завершается записью считанной из ОЗУ информации в регистр R0.

Из разобранного примера отчетливо видно, что имеющиеся у процессора PDP способы адресации позволяют удобно работать не только с одиночными данными, но и с массивами последовательно хранящейся в памяти информации. Кроме того, некоторые приемы использования счетчика команд R7 в качестве РОН могут давать еще более интересные эффекты. Пусть, например, в некоторой программе встречается команда

MOV (R7) + , R2

и в следующем за ней слове хранится число 6. Учитывая, что после извлечения команды из ОЗУ счетчик R7 немедленно увеличивается, в момент выполнения команды его содержимое уже показывает на слово памяти, где хранится число 6. Следовательно, в качестве операнда (R7) будет считано именно оно, и, кроме того, R7 автоматически «передвинется» на следующую за константой ячейку благодаря автоинкрементному способу адресация. Таким образом, описанный прием позволяет использовать в качестве операнда константу, хранящуюся непосредственно в команде. Завершая обсуждения основных принципов программирования процессора PDP. приведем примеры линейного, разветвляющегося и циклического фрагмента программы.

*Пример I.* Программа вычисления по формулам R1:=R2+R3; R4:=R3-R2.

*Программа 131*

MOV R2, Rl ; сразу складывать нельзя, так как сумма заменяет

;второй операнд

ADD R3, Rl

MOV R3, R4

SUB R2, R4

HLT

*Пример* 2. В Rl и R2 хранятся некоторые числа. Поместить большее из них в R5, а меньшее-в RO.

*Программа 132*

СМР Rl, R2 ;сравнить Rl и R2

BPL L1 ;переход при R1≥R2, если R2>R1

MOV Rl, R0

MOV R2, R5

BR L2

;если R1>R2

LI: MOV Rl, R5

MOV R2, R0

L2: HLT

*Примечание.* В тексте программы указаны переходы на метки, но в результате трансляции они будут автоматически преобразованы в переходы через заданное число слов.

*Пример 3.* Пусть в Rl задан начальный адрес массива ОЗУ, а в R2 - число ячеек в нем. Вычислить сумму элементов массива.

*Программа 133*

CLR R0 ;очистка суммы

LI: ADD (Rl) + , R0 ;добавить к сумме очередной элемент

DEC R2 ;уменьшить на единицу число оставшихся элементов

BNE L1 ;цикл, пока не останется 0 элементов

HLT

*Примечание.* В программе используется тот факт, что команда DEC автоматически сравнивает результат с нулем, что делает ненужной специальную команду сравнения СМР.