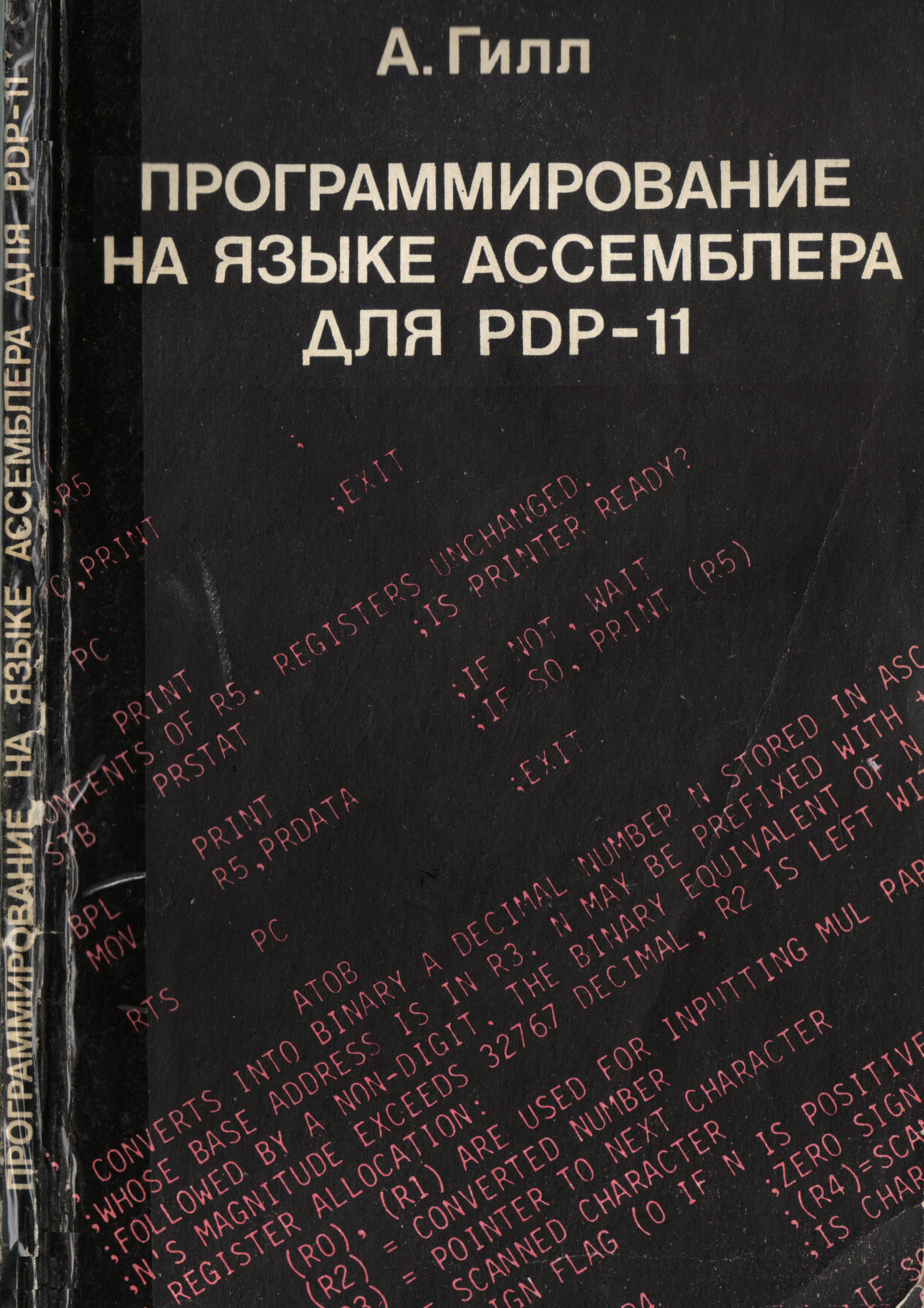
А. Гилл

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА ДЛЯ PDP-11

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА ДЛЯ PDP-11



А. Гилл

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА ДЛЯ PDP-11

Перевод с английского Н. Н. Слепова
под редакцией Б. А. Кузьмина

МОСКВА «РАДИО И СВЯЗЬ» 1983

ББК 32.973
Г47
УДК 519.68

Гилл А.

Г47 Программирование на языке ассемблера для PDP-11: Пер. с англ. — М.: Радио и связь, 1983. — 160 с., ил.

65 к.

Детально рассмотрены особенности программирования для мини-ЭВМ PDP-11. Показаны организация процесса вычислений и техника программирования на языке ассемблера. Приведено большое число упражнений и примеров программ, предназначенных для практической работы.

Для пользователей мини-ЭВМ. Может служить пособием при изучении программирования для мини-ЭВМ типа СМ-3, СМ-4, "Электроника-60", "Электроника-100/25".

Г $\frac{2405000000-196}{046(01)-83}$ 95 – 83

ББК 32.973
6Ф7.3

Редакция переводной литературы

MACHINE AND ASSEMBLY LANGUAGE PROGRAMMING OF THE PDP-11

Arthur Gill

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

PDP-11 – название одного из самых популярных семейств мини-ЭВМ американской фирмы DEC (Digital Equipment Corp.). Большие возможности и надежность оборудования, набор мощных операционных систем (можно упомянуть UNIX) оказали влияние на более поздние разработки мини-ЭВМ этого класса. Достаточно сказать, что PDP-11 является прототипом таких известных в нашей стране мини-ЭВМ, как СМ-3, СМ-4, "Электроника-60", "Электроника-100/25". Поэтому предлагаемая вниманию читателя книга может служить прекрасным пособием для изучения языка ассемблера и для перечисленных выше отечественных ЭВМ. Материал книги, содержащий довольно полные сведения о программировании на языке ассемблера, упорядочен таким образом, что ознакомления с пятью первыми главами книги достаточно, чтобы писать простые программы. Много тщательно подобранных программ, предназначенных для выполнения на ЭВМ, приведено в книге в качестве иллюстраций и упражнений. Книга может быть использована и для первого знакомства с предметом, поскольку для ее чтения не требуется какой-либо специальной подготовки.

При переводе без специальных оговорок в тексте исправлен ряд опечаток и неточностей подлинника.

Для читателей, желающих более детально ознакомиться с организацией и программным обеспечением PDP-11, а также с особенностями программирования для этих ЭВМ, в издательстве "Финансы и статистика" в 1983 г. выходит книга Экхауз Р., Моррис Л. Мини-ЭВМ: Организация и программирование.

Б. А. Кузьмин

ПРЕДИСЛОВИЕ

Данная книга является результатом обработки конспектов, написанных для курса по структуре вычислительных систем, предложенного Калифорнийским университетом в Беркли. Это второй курс в рамках курсов по ЭВМ (первым является курс по языку программирования высокого уровня), где студентам представляют основные концепции архитектуры и функционирования ЭВМ (регистры, наборы команд, режимы адресации и т. д.) и обучают технике программирования на языке ассемблера.

Этот курс прививает студентам навыки общения с вычислительной машиной как таковой, которую они могут программировать без использования операционной системы, скрывающей от программистов многие возможности оборудования. Общение с такой "голой" машиной не только устраняет многое из того таинственного ореола, которым окутаны вычислительные машины, но и позволяет экспериментировать с программированием таких процедур, как ввод-вывод, обработка прерываний и т. п. В дополнение к такой отдельно взятой машине студенты имеют доступ к большой, работающей в режиме разделения времени, машине, на которой они могут выполнять редактирование, ассемблирование, редактирование связей, организацию файлов и другие операции, которые во многом имеют формализованный характер.

Для этих целей мы выбрали одну из машин семейства PDP-11 фирмы Digital Equipment Corporation. (В частности, мы использовали PDP-11/10 в качестве отдельно взятой машины и PDP-11/70 в качестве машины, оборудованной операционной системой UNIX и работающей в режиме разделения времени.) Мини-ЭВМ PDP-11 были выбраны благодаря их широким функциональным возможностям, популярности и развитой системе программного обеспечения.

Мы нашли, что PDP-11 как нельзя лучше подходит для того, чтобы на ней можно было проиллюстрировать основные вопросы, рассматриваемые в курсе, и в то же время иметь возможность показать студентам во всей полноте мир мини-ЭВМ.

Цель этой книги, таким образом, состоит в том, чтобы познакомить читателя с основными организационными и операционными возможностями и особенностями PDP-11 и освоить технику программирования на языке ассемблера для этого класса ЭВМ. В сущности ее нельзя рассматривать как общий курс архитектуры ЭВМ или как попытку дать исчерпывающее описание возможных методов организации вычислительного процесса и ассемблирования. Однако учитывая, что концепции и методы управления вычислительным процессом и программирования для PDP-11 используются во многих

других мини-ЭВМ, можно надеяться, что материал этой книги послужит хорошим фундаментом для программирования других вычислительных машин.

В гл. 1 описаны алгоритмы преобразования чисел из одной системы счисления (двоичной, восьмеричной и десятичной) в другую. Алгоримы приведены без доказательства и могут использоваться только для справок. В гл. 2 описана организационная структура PDP-11 (память, центральный процессор и периферийные устройства). В гл. 3 рассмотрено представление чисел (целых и с плавающей точкой), символов и строк в PDP-11. В гл. 4 описаны форматы команд PDP-11 и режимы адресации, а материал гл. 5 является введением в программирование на языке ассемблера.

Первые пять глав призваны обеспечить читателю уровень знаний, достаточный для того, чтобы писать простые программы для PDP-11. Остальные главы позволяют более углубленно рассмотреть ряд особенностей в организации вычислений и технике программирования. Так, в гл. 6 рассмотрены стеки и подпрограммы (включая рекурсивные). В гл. 7 более подробно рассматриваются арифметические операции (включая операции с двойной точностью), а также операции контроля, сравнения, перехода и сдвига для PDP-11. В гл. 8 объясняется механизм захватов и прерываний, а в гл. 9 описываются работа ассемблера и редактора связей и понятие перемещения. (Хотя для иллюстрации используются ассемблер MACRO-11 и редактор связей LINKR-11, нужно отметить, что обсуждаемые концепции являются достаточно общими.) В гл. 10 введены некоторые дополнительные средства ассемблера, такие как макросы, директивы повторения и условное ассемблирование.

В конце книги помещен ряд приложений, в которых приведены справочные сведения и таблицы (коды символов, сводка режимов адресации, список кодов операций и т. д.). Кроме того, в приложении помещены заметки о стиле программирования, которые следует внимательно прочесть начинающим программистам.

Каждая глава заканчивается подборкой упражнений, предназначенных для иллюстрации, а иногда и дополнения основного материала. Это должно стимулировать читателя к решению задач и выполнению на ЭВМ программ, включенных в эти упражнения. Твердого усвоения материала этой книги можно достичь только через практику, т. е. написанием и выполнением программ.

Единственным предварительным условием, которое может быть выдвинуто, является наличие некоторого опыта в программировании на языках высокого уровня. При этом речь идет не о знании какого-то конкретного языка, а лишь о том, чтобы читатель был знаком с такими общими понятиями, как алгоритм, блок-схема и хранимая в памяти программа.

Было бы неправильным рассматривать эту книгу как единственное пособие по данному курсу. Учитывая то, что она не описывает всех нюансов и деталей команд PDP-11 и директив ассемблера, студенты должны овладеть материалом, изложенным в руководствах по процессору и ассемблеру для конкрет-

ной модели PDP-11, где эти детали могут быть найдены в случае необходимости. Меньше всего в этой книге сказано о периферийных устройствах (более или менее детально рассмотрены только телетайп и сетевой таймер), поэтому студенты, выполняя программирование операций ввода-вывода, могут воспользоваться руководствами и инструкциями по обслуживанию периферийных устройств PDP-11.

Автор признателен мистеру Р. С. Эпстейну из Калифорнийского университета в Беркли за просмотр рукописи и полезные советы, а также за участие в написании разд. 5.5 и некоторых упражнений. Следует поблагодарить также профессоров Р. С. Фэбри и М. Р. Стоунбрейкера (также из Калифорнийского университета в Беркли) за полезные комментарии и предложения.

Артур Гилл

Глава 1
СИСТЕМЫ СЧИСЛЕНИЯ

При работе с PDP-11 мы будем широко использовать двоичную и восьмеричную системы счисления, а также десятичную систему. Важно, чтобы студенты как можно скорее освоили преобразование из одной системы в другую. В этой главе мы опишем без доказательств некоторые алгоритмы для выполнения таких преобразований. Студенты, знакомые с этими алгоритмами, могут непосредственно приступить к изучению гл. 2.

Под "числом" в этой главе мы будем понимать любое неотрицательное целое число (0, 1, . . .). Число N, представленное в десятичной, восьмеричной или двоичной системе, будет обозначаться как N_{10}, N_8, N_2 соответственно. Однако соответствующий индекс может быть опущен, если из контекста понятно, о какой системе идет речь.

Число, содержащее m разрядов, будет символически записываться в виде $D_{m-1} \ldots D_1 D_0$ [D_i будет обозначать (i + 1)-й разряд, отсчитываемый справа].

1.1. Преобразование десятичного числа в двоичное

Блок-схемы, приведенные на рис. 1.1 и 1.2, описывают алгоритмы преобразования десятичного числа N в двоичный эквивалент M.

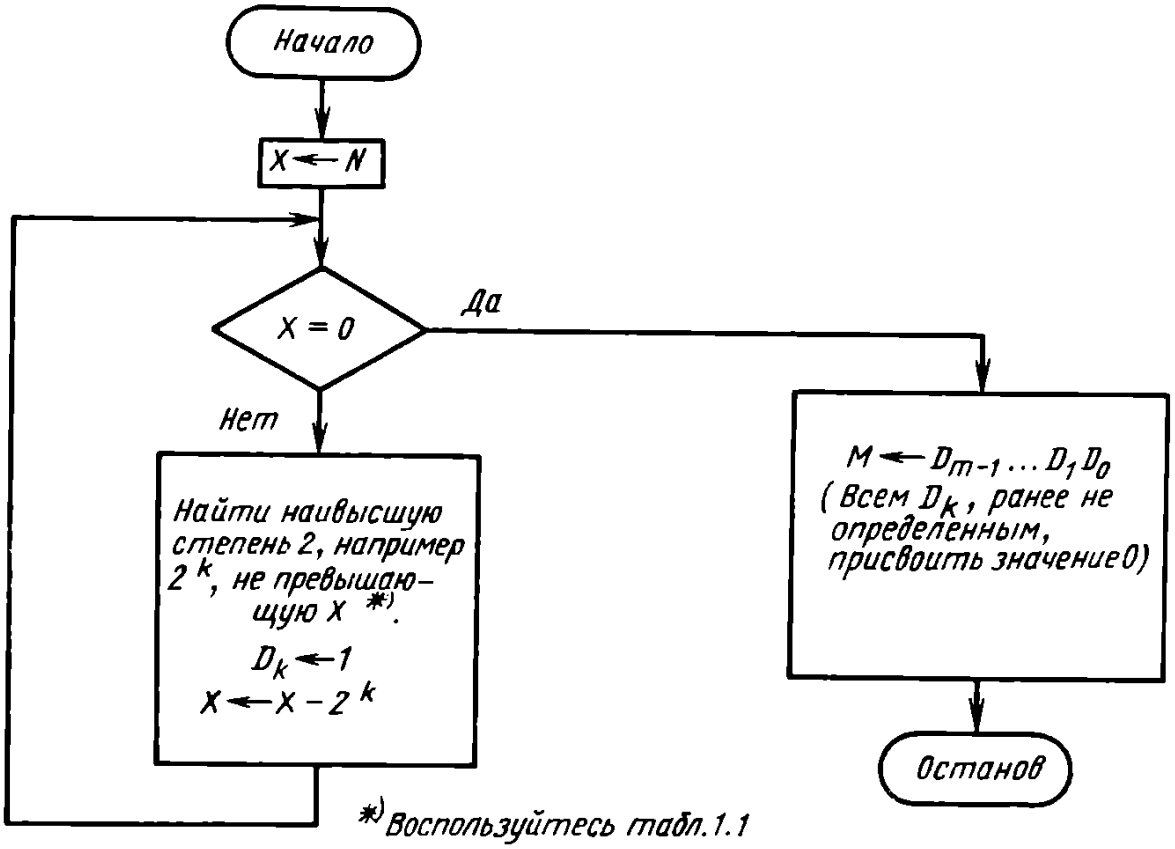


Рис. 1.1. Преобразование десятичного числа в двоичное методом вычитания степеней

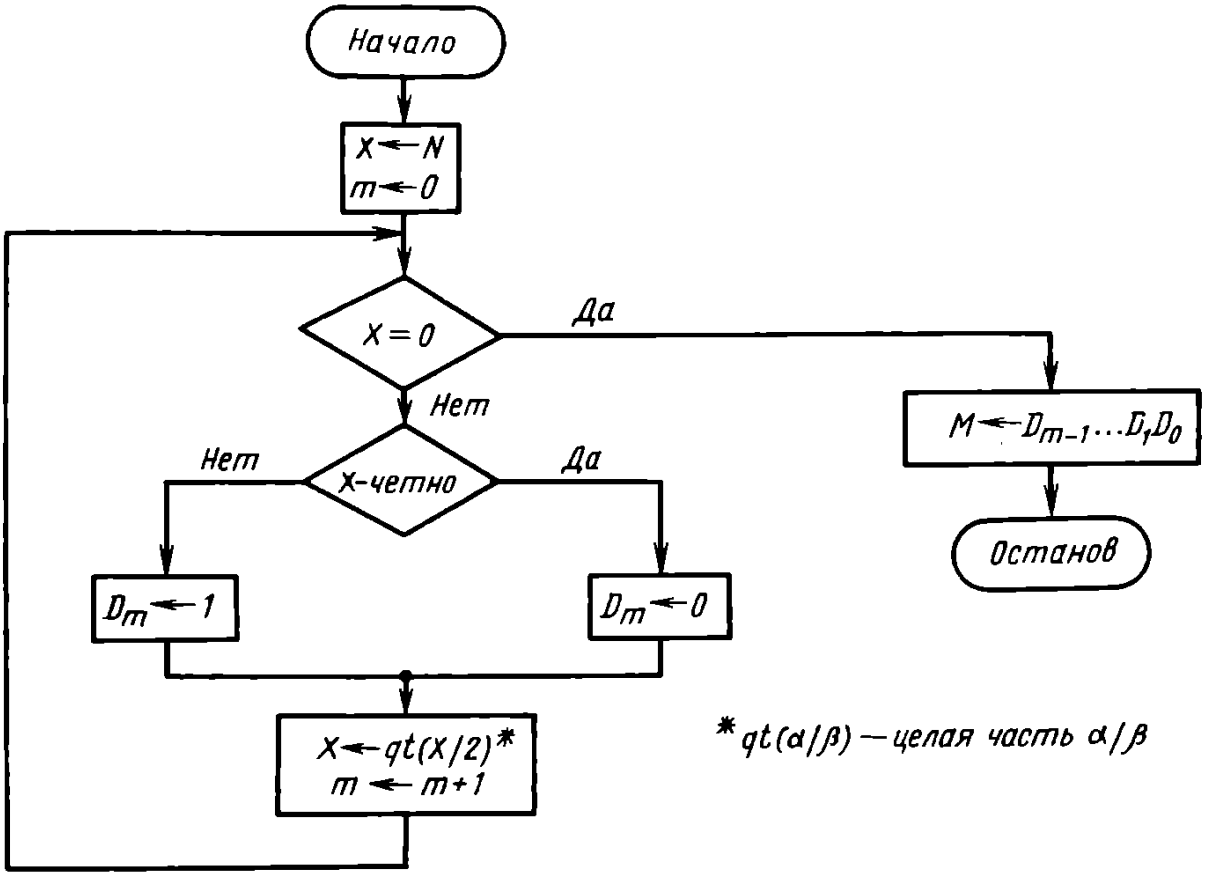


Рис. 1.2. Преобразование десятичного числа в двоичное методом деления

Пример (метод вычитания степеней)

$$N = 217_{10},$$
$$217 - 2^7 = 217 - 128 = 89 \quad (D_7 = 1),$$
$$89 - 2^6 = 89 - 64 = 25 \quad (D_6 = 1),$$
$$25 - 2^4 = 25 - 16 = 9 \quad (D_4 = 1),$$
$$9 - 2^3 = 9 - 8 = 1 \quad (D_3 = 1),$$
$$1 - 2^0 = 1 - 1 = 0 \quad (D_0 = 1),$$
$$M = 11011001_2.$$ □

Пример (метод деления)

$$N = 217_{10},$$
$$217 - \text{нечетно} \quad (D_0 = 1),$$
$$217/2 = 108 - \text{четно} \quad (D_1 = 0),$$
$$108/2 = 54 - \text{четно} \quad (D_2 = 0),$$
$$54/2 = 27 - \text{нечетно} \quad (D_3 = 1),$$
$$27/2 = 13 - \text{нечетно} \quad (D_4 = 1),$$
$$13/2 = 6 - \text{четно} \quad (D_5 = 0),$$
$$6/2 = 3 - \text{нечетно} \quad (D_6 = 1),$$
$$3/2 = 1 - \text{нечетно} \quad (D_7 = 1),$$
$$1/2 = 0 - \text{четно} \quad (D_8 = 0),$$
$$M = 11011001_2.$$ □

1.2. Преобразование десятичного числа в восьмеричное

Блок-схемы, приведенные на рис. 1.3 и 1.4, описывают алгоритмы преобразования десятичного числа N в восьмеричный эквивалент M.

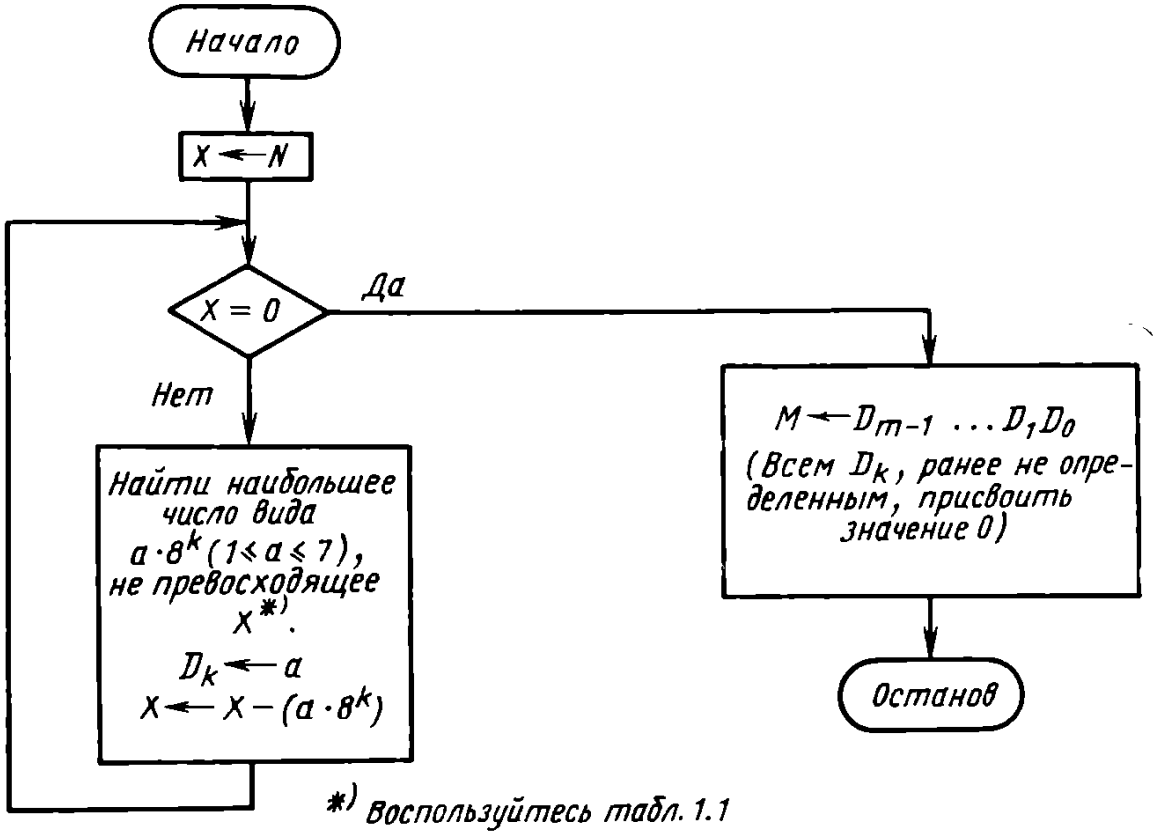


Рис. 1.3. Преобразование десятичного числа в восьмеричное методом вычитания степеней

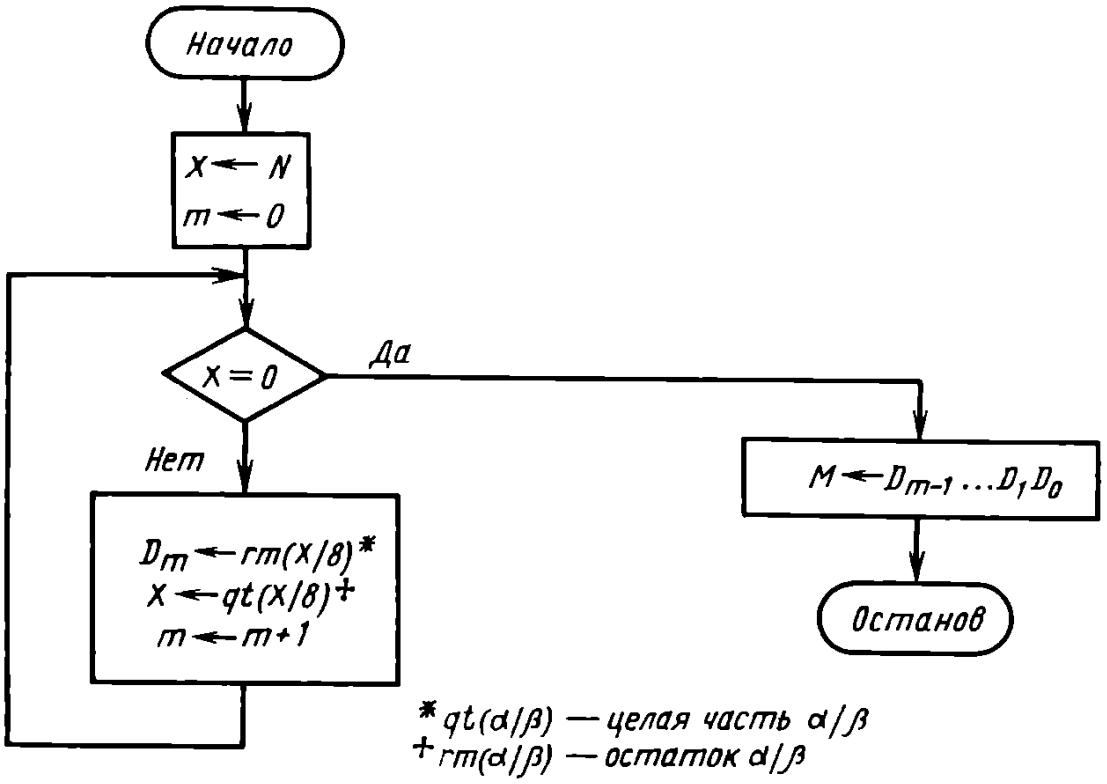


Рис. 1.4. Преобразование десятичного числа в восьмеричное методом деления

Пример (метод вычитания степеней)

$$N = 2591_{10},$$
$$2591 - 5 \cdot 8^3 = 2591 - 2560 = 31 \quad (D_3 = 5),$$
$$31 - 3 \cdot 8^1 = 31 - 24 = 7 \quad (D_1 = 3),$$
$$7 - 7 \cdot 8^0 = 7 - 7 = 0 \quad (D_0 = 7),$$
$$M = 5037_8.$$ □

Пример (метод деления)

$$N = 2591_{10},$$
$$2591/8 = 323 \text{ (остаток 7)} \quad (D_0 = 7),$$
$$323/8 = 40 \text{ (остаток 3)} \quad (D_1 = 3),$$
$$40/8 = 5 \text{ (остаток 0)} \quad (D_2 = 0),$$
$$5/8 = 0 \text{ (остаток 5)} \quad (D_3 = 5),$$
$$M = 5037_8.$$ □

Еще один метод состоит в преобразовании числа N в двоичное, как показано в разд. 1.1, и последующем преобразовании результата в восьмеричное, как показано в разд. 1.6.

1.3. Преобразование двоичного числа в десятичное

Если $N = D_{m-1} \ldots D_i D_0$ – двоичное число, то его десятичный эквивалент

$$M = \sum_{i=0}^{m-1} D_i \cdot 2^i.$$

(Степени основания 2 приведены в табл. 1.1.)

Пример

$$N = 1011100_2,$$
$$M = 2^2 + 2^3 + 2^4 + 2^6 = 4 + 8 + 16 + 64 = 92_{10}.$$ □

Таблица 1.1. Степени оснований 2 и 8

$8^0 = 2^0 = 1$	$8^2 = 2^6 = 64$	$8^4 = 2^{12} = 4096$
$2^1 = 2$	$2^7 = 128$	$2^{13} = 8192$
$2^2 = 4$	$2^8 = 256$	$2^{14} = 16384$
$8^1 = 2^3 = 8$	$8^3 = 2^9 = 512$	$8^5 = 2^{15} = 32768$
$2^4 = 16$	$2^{10} = 1024$	$2^{16} = 65536$
$2^5 = 32$	$2^{11} = 2048$	$2^{17} = 131072$

1.4. Преобразование восьмеричного числа в десятичное

Если $N = D_{m-1} \ldots D_i D_0$ – восьмеричное число, то его десятичный эквивалент

$$M = \sum_{i=0}^{m-1} D_i \cdot 8^i.$$

(Степени основания 8 приведены в табл. 1.1.)

Пример

$$N = 3107_8,$$
$$M = 7 \cdot 8^0 + 0 \cdot 8^1 + 1 \cdot 8^2 + 3 \cdot 8^3$$
$$= 7 \cdot 1 + 0 \cdot 8 + 1 \cdot 64 + 3 \cdot 512$$
$$= 7 + 64 + 1536$$
$$= 1607_{10}$$ □

1.5. Преобразование восьмеричного числа в двоичное

Двоичный эквивалент M восьмеричного числа N получается заменой каждой цифры числа N группой из трех двоичных цифр, как показано в табл. 1.2. (Старшие нулевые разряды крайней левой группы могут быть удалены.)

Пример

$$N = 160734_8,$$
$$M = 1\ 110\ 000\ 111\ 011\ 100_2. \quad \square$$

Таблица 1.2. **Преобразование восьмеричного числа в двоичное**

Восьмеричное	0	1	2	3	4	5	6	7
Двоичное	000	001	010	011	100	101	110	111

1.6. Преобразование двоичного числа в восьмеричное

Восьмеричный эквивалент M двоичного числа N получается разбиением N на группы по три двоичных числа справа налево и последующей заменой каждой группы восьмеричной цифрой в соответствии с табл. 1.2. Если длина[1] N не делится на три, следует добавить старшие нулевые разряды, чтобы полученное число разрядов делилось на три.

Пример

$$N = 11010101_2 = 011\ 010\ 101,$$
$$M = 325_8. \quad \square$$

1.7. Сложение двоичных и восьмеричных чисел

Два двоичных числа складываются, как если бы они были десятичными, с учетом следующих правил:

$$1 + 1 = 0 \quad \text{(перенос 1)},$$
$$1 + 1 + 1 = 1 \quad \text{(перенос 1)}.$$

Пример

$$\begin{array}{r} 101101_2 \\ +\ 100111_2 \\ \hline 1010100_2. \end{array} \quad \square$$

Два восьмеричных числа складываются, как если бы они были десятичными, с учетом следующих правил: если две цифры, складываемые как десятичные, дают в сумме число $D \geqslant 8$, то это число заменяется на $D - 8$ и осуществляется перенос 1 в следующий разряд (т. е. $4 + 5 = 1$ с переносом 1; $7 + 7 = 6$ с переносом 1).

Пример

$$\begin{array}{r} 67036_8 \\ +\ 52147_8 \\ \hline 141205_8. \end{array} \quad \square$$

Упражнения

1.1. Используя метод вычитания степеней и метод деления, преобразуйте следующие десятичные числа в двоичные и восьмеричные эквиваленты:

а) 2337_{10}; б) 10000_{10}; в) 16383_{10}.

---

[1] Число двоичных разрядов. – *Прим. пер.*

1.2. Преобразуйте следующие двоичные числа в десятичные и восьмеричные эквиваленты:
а) 1111111_2; б) 10000000_2; в) 1010011100101110_2.

1.3. Преобразуйте следующие восьмеричные числа в десятичные и двоичные эквиваленты;
а) 377_8; б) 77777_8; в) 123456_8.

1.4. Выполните сложение следующих двоичных чисел и проверьте результат путем преобразования их в десятичные.
а) $1011010_2 + 11010_2$; б) $1111101_2 + 1110_2$.

1.5. Выполните сложение следующих восьмеричных чисел и проверьте результат путем преобразования их в десятичные:
а) $1456_8 + 567_8$; б) $1234_8 + 7473_8$.

1.6. Вычислите произведение следующих двоичных чисел и проверьте результат путем преобразования их в десятичные:
а) $11010111_2 \times 100101_2$.

1.7. Покажите, что двоичный эквивалент $2^k - 1$ имеет вид 111 . . . 1 (k единиц).

Глава 2
ПРИНЦИПЫ ОРГАНИЗАЦИИ PDP-11

PDP-11 состоит из *центрального процессора* CP, в котором осуществляются все вычисления, *оперативной памяти* CM, в которой хранятся данные и программа, и *периферийных устройств*, таких как телетайп, печатающее устройство, ленточный перфоратор, устройство считывания перфокарт, графопостроитель, электронно-лучевой дисплей, накопитель на магнитных дисках, накопитель на магнитной ленте, таймер и пульт оператора. Ниже мы будем рассматривать самую простую конфигурацию PDP-11, которая включает только такие периферийные устройства, как *телетайп* и *сетевой таймер*. Подробное описание этих и других устройств можно найти в руководстве "PDP-11 Peripheral Handbook".

Общая структура PDP-11 представлена на рис. 2.1. В этой главе мы опишем основные особенности различных компонентов системы, показанных на этом рисунке.

Если какой-то символ, например A, записан просто как A, то он будет представлять адрес в оперативной памяти или имя регистра. Если же используется обозначение (A), то оно соответствует "содержимому A". Аналогично обозначение ((A)) используется для указания "содержимого (A)", т. е. содержимого адреса памяти, находящегося в A[1].

2.1. Оперативная память

Основной единицей памяти ЭВМ является *бит*. Элемент памяти емкостью до 1 бит содержит один двоичный разряд, т. е. 0 или 1. Последовательный набор битов, из которых состоит оперативная память, объединяется в *байты*, которые, в свою очередь, объединяются в *слова*.

---

[1] В руководстве "PDP-11 Processor Handbook" этих обозначений придерживаются только для адресов памяти. Если же A – имя регистра, то это обозначение в руководстве употребляется для "содержимого регистра A"; тогда как (A) обозначает "содержимое адреса памяти, находящегося в регистре A".

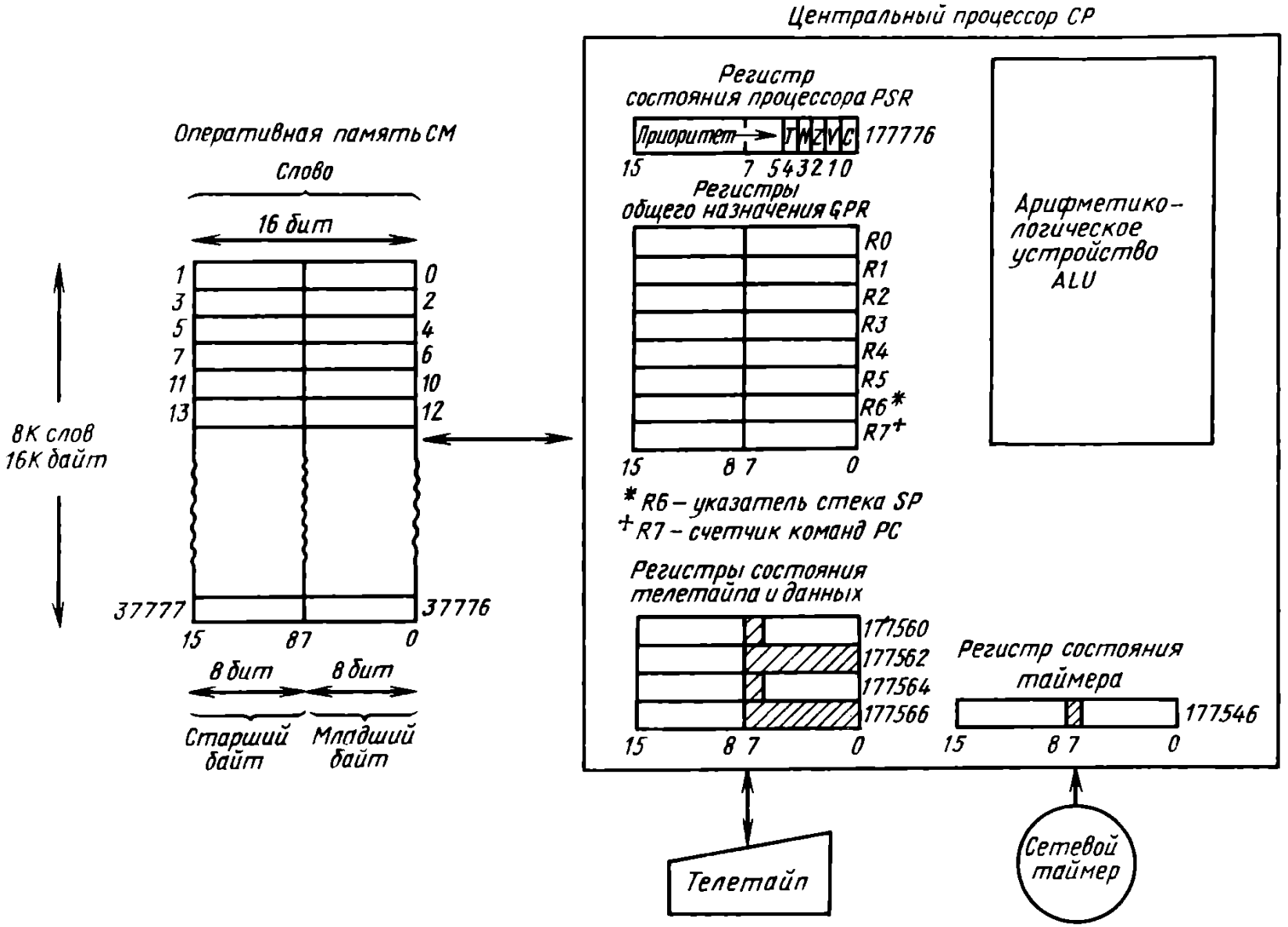
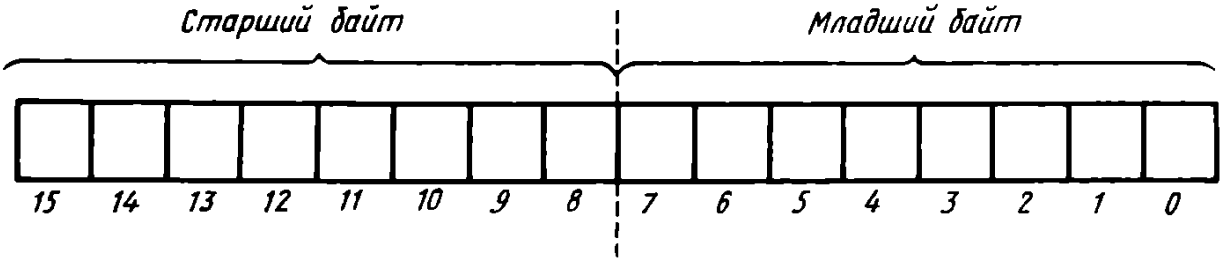


Рис. 2.1. Структура PDP-11

В PDP-11 каждое слово состоит из 16 бит, перенумерованных от 0 до 15 (справо налево); биты, имеющие номера от 0 до 7, составляют младший байт, а биты с номерами от 8 до 15 – старший байт рассматриваемого слова:



На рис. 2.1 показана память PDP-11, состоящая из $2^{13} = 8192_{10} = 8К$ слов, т. е. имеющая емкость $2^{14} = 16384_{10} = 16К$ байт (где К соответствует $2^{10} = 1024$). Возможна память и большей емкости, точный размер которой (всегда кратный 4096_{10}) изменяется от модели к модели.

Каждый байт памяти определяется соответствующим (присущим только ему) номером, называемым *адресом* (или *местоположением*) байта. Если память имеет b байтов, то последовательность байтов имеет адреса 0, 1, 2, . . . , b – 1. Адрес слова является, по соглашению, адресом его младшего байта. Следовательно, последовательность слов имеет адреса 0, 2, 4, . . . b – 2. Заметим, что *адрес слова всегда является четным числом.* Например, в ЭВМ с памятью 8К слов, показанной на рис. 2.1, байты будут иметь следующие адреса:

$$0, 1, 2, \ldots, 37777_8 (37777_8 = 16383_{10}),$$

а слова – следующие адреса:

$$0, 2, 4, \ldots, 37776_8.$$

Мы всегда будем записывать адрес памяти в *восьмеричной системе.*

Содержимые слова или байта могут быть, естественно, представлены последовательностью из 16 или 8 двоичных цифр. Когда эта последовательность, рассматриваемая как двоичное число, преобразуется в шестиразрядный или трехразрядный восьмеричный эквивалент, то мы получаем *восьмеричное содержимое* слова или байта – краткое и удобное представление, которое часто будет использоваться в тексте. Например, слово, фактическое (двоичное) содержимое которого составляет 1010011100101110, имеет восьмеричное содержимое 123456; его младший байт имеет восьмеричное содержимое 056, а старший – 247.

2.2. Центральный процессор

Центральный процессор состоит из *арифметико-логического устройства* ALU, где осуществляются все арифметические и логические операции, и набора 16-разрядных *регистров* (соответствующих длине слова), которые составляют "собственную память" центрального процессора.

Существует 8 *регистров общего назначения* GPR, обозначенных как R0, R1, . . . , R7. Регистр R6 называют также *указателем стека* SP по причине, рассмотренной ниже. Регистр R7 называется также *счетчиком команд* PC,

который всегда содержит адрес следующей команды, которая будет выполняться.

Регистр состояния процессора PSR содержит набор одноразрядных кодов условий, значение которых зависит от результата последней выполненной команды. Например:

$$\text{Z-код (2-й разряд)} = \begin{cases} 1, \text{ если результат был} = 0, \\ 0, \text{ если результат был} \neq 0, \end{cases}$$

$$\text{N-код (3-й разряд)} = \begin{cases} 1, \text{ если результат был} < 0, \\ 0, \text{ если результат был} \geqslant 0. \end{cases}$$

Другие коды условий будут рассмотрены ниже в следующих главах. Следует заметить, что различные команды по-разному влияют на коды условий (а некоторые из них вообще не влияют на эти коды). Точное указание на то, как каждая команда влияет на коды условий, дано в руководстве "PDP-11 Processor Handbook".

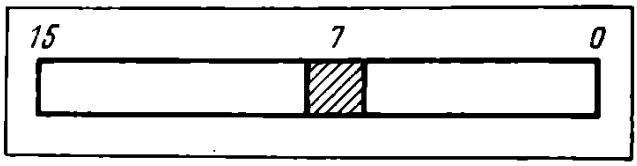
Ниже рассмотрен ряд регистров, связанных с периферийными устройствами.

2.3. Телетайп

Информация в PDP-11 может вводиться (оператором) с помощью телетайпа через *клавишный пульт*, а распечатываться с помощью PDP-11 на *печатающем устройстве.* Важно отметить, что, в отличие от обычной пишущей машинки, телетайп состоит из двух устройств: клавишного пульта и печатающего устройства, которые функционируют независимо. Например, нажатие клавиши на клавишном пульте не приводит автоматически к печати соответствующего ей символа.

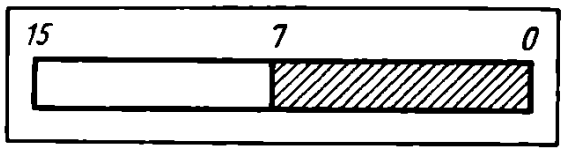
Телетайп связан с четырьмя регистрами.

Регистр состояния клавишного пульта (имеет адрес 177560):



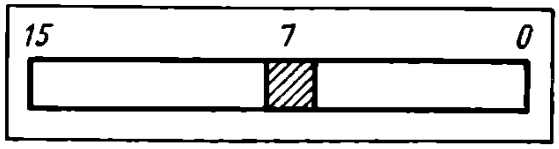
Разряд 7 этого регистра (разряд готовности) содержит 1, когда возможна печать указанного символа. В этом разряде формируется 0 только после того, как будет выполнена команда обращения к регистру данных клавишного пульта.

Регистр данных клавишного пульта (имеет адрес 177562):



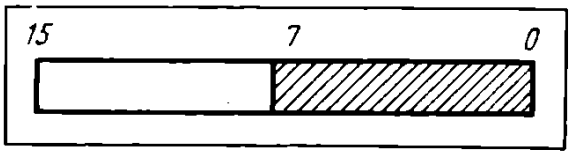
Разряды 0 – 7 содержат закодированный эквивалент символа, который нажат на пульте. Этот регистр предназначен только для чтения.

Регистр состояния печатающего устройства (имеет адрес 177564):



Разряд 7 (разряд готовности) этого регистра содержит 1, когда печатающее устройство может осуществить печать. В этом разряде формируется 0 как только код символа загружается в регистр данных печатающего устройства.

Регистр данных печатающего устройства (имеет адрес 177566):



Разряды 0 – 7 этого регистра содержат закодированный эквивалент символа, который нужно напечатать. Если разряд готовности в регистре состояния печатающего устройства содержит 1, то печать осуществляется сразу после загрузки регистра данных печатающего устройства. Этот регистр предназначен только для записи. При считывании его содержимое воспринимается так, как если бы все его биты были нулевыми.

Ниже мы увидим, что 1 в разряде 7 какого-то байта с адресом α означает, что число, хранящееся в байте α, – отрицательное. Следовательно, для того чтобы проконтролировать, записан ли новый символ, полученный с клавишного пульта, в регистр данных клавишного пульта (полагая, что предыдущий символ уже обработан), нужно просто выяснить, содержит ли байт 177560 отрицательное число. Аналогично, чтобы проконтролировать, может ли печатающее устройство телетайпа осуществить печать содержимого регистра данных печатающего устройства, достаточно просто выяснить, является ли содержимое байта 177564 отрицательным числом.

2.4. Сетевой таймер

Для целей синхронизации к PDP-11 подключается сетевой таймер с частотой отсчета 60 Гц[1]. Он связан с *регистром состояния таймера* (адрес 177546), в разряд 7 (разряд готовности) которого заносится единица каждую 1/60 с. После очередного отсчета таймера этот разряд не обнуляется, поэтому его нужно очищать программно, сразу после того, как он становится единицей, чтобы можно было принять следующий отсчет[2].

Как было показано в предыдущем разделе, факт появления отсчета сетевого таймера можно определить по моменту появления отрицательного числа в байте 177546.

Упражнения

2.1. Записать в восьмеричной форме адреса старших байта и слова для ЭВМ с объемом памяти w слов, где w (в десятичной форме) равно:
а) 28К; б) 2^{15}.

2.2. Сколько слов содержится в памяти, адрес старшего байта которой равен 17777_8?

2.3. Определить восьмеричный эквивалент содержимого слов:
а) 0110110110110110; б) 1001011111010101.
Чему равен восьмеричный эквивалент младшего и старшего байтов каждого из этих слов?

---

[1] Эта частота соответствует стандарту на частоту питающей сети в США. В соответствии с отечественным стандартом сетевой таймер имеет частоту 50 Гц . – *Прим. пер.*

[2] Однако, когда разряд 6 (разряд " прерывание возможно") содержит 1, то сброс разряда готовности после каждого отсчета осуществляется *автоматически.*

Глава 3
ПРЕДСТАВЛЕНИЕ ЧИСЕЛ И СИМВОЛОВ

В этой главе мы рассмотрим, как запоминаются и хранятся в PDP-11 числа (целые и действительные, положительные и отрицательные), символы и строки символов.

Пока не определено иначе, будем считать, что "число" означает целое число, а "положительное число" означает целое неотрицательное число (0,1,2,...).

3.1. Представление в дополнительном двоичном коде

Числа в PDP-11 представлены в форме *n-разрядного дополнительного двоичного кода,* где n равно либо 16 (для представления слов), либо 8 (для представления байтов). Эта форма образуется в соответствии со следующими правилами.

Для запоминания положительного числа N *нужно*:

1. Представить N в двоичной форме.

2. Запомнить двоичное N, представив его в соответствии с правилами в виде слова или байта, добавляя при этом столько ведущих нулей, сколько необходимо, чтобы сделать общее число разрядов равным n.

Примеры

1. $N = 1607_{10} = 3107_8$.

Представление в форме 16-разрядного дополнительного двоичного кода:
0 000 011 001 000 111 (003107_8).

2. $N = 10_{10} = 12_8$.

Представление в форме 8-разрядного дополнительного двоичного кода:
00 001 010 (012_8). □

Максимальное положительное число, которое может быть представлено в форме n-разрядного дополнительного двоичного кода, равно $\underbrace{111\ldots1}_{n-1}{}_2 = 2^{n-1} - 1$.

Если n = 16, то $N \leqslant 0\,111\,111\,111\,111\,111_2 = 077777_8 = 32767_{10}$.

Если n = 8, то $N \leqslant 01\,111\,111_2 = 177_8 = 127_{10}$.

Таким образом, старший значащий разряд (MSB) в любом положительном числе, представленном двоичным дополнительным кодом, равен 0.

Для запоминания отрицательного числа N *нужно*:

1. Запомнить положительное число N.

2. Получить ~ N (обратный код числа N) из N поразрядной заменой 0 на 1 и 1 на 0. (В восьмеричной форме ~ N можно получить из N заменой каждой цифры D числа N на 7 – D.) [1]

3. Прибавить 1 к ~ N (не учитывая перенос от MSB, если требуется).

Примеры

1. $N = 1607_{10} = 3107_8$.

---

[1] Если максимальное возможное значение крайней левой цифры равно 1 или 3 (а не 7), то она заменяется на 1 – D или 3 – D соответственно (а не на 7 – D).

Представление в форме 16-разрядного двоичного дополнительного кода:

$$\begin{array}{rl} N: & 0\,000\,011\,001\,000\,111 \\ \sim N: & 1\,111\,100\,110\,111\,000 \\ +1: & \underline{\qquad\qquad\qquad\qquad\qquad 1} \\ -N: & 1\,111\,100\,110\,111\,001 \end{array}$$

В восьмеричном коде:

$$\begin{array}{rl} N: & 0\,0\,3\,1\,0\,7 \\ \sim N: & 1\,7\,4\,6\,7\,0 \\ +1: & \underline{\qquad\qquad 1} \\ -N: & 1\,7\,4\,6\,7\,1 \end{array}$$

2. $N = -1607_{10}$.

Представление в форме 16-разрядного двоичного дополнительного кода:

$$\begin{array}{rl} N: & 1\,111\,100\,110\,111\,001 \\ \sim N: & 0\,000\,011\,001\,000\,110 \\ +1: & \underline{\qquad\qquad\qquad\qquad\qquad 1} \\ -N: & 0\,000\,011\,001\,000\,111 \end{array}$$

В восьмеричном коде:

$$\begin{array}{rl} N: & 1\,7\,4\,6\,7\,1 \\ \sim N: & 0\,0\,3\,1\,0\,6 \\ +1: & \underline{\qquad\qquad 1} \\ -N: & 0\,0\,3\,1\,0\,7 \end{array}$$

3. $N = 10_{10} = 12_8$.

Представление в форме 8-разрядного двоичного дополнительного кода:

$$\begin{array}{rl} N: & 00\,001\,010 \\ \sim N: & 11\,110\,101 \\ +1: & \underline{\qquad\qquad 1} \\ -N: & 11\,110\,110 \end{array}$$

В восьмеричном коде:

$$\begin{array}{rl} N: & 0\,1\,2 \\ \sim N: & 3\,6\,5 \\ +1: & \underline{\qquad 1} \\ -N: & 3\,6\,6 \end{array}$$

□

Заметим, что разряд MSB любого отрицательного числа в указанном представлении равен 1. Таким образом, MSB указывает, является ли число положительным или отрицательным. Поэтому MSB часто называют *знаковым разрядом.*

Если $N = 0$, то $-N$ в дополнительном коде получается так:

$$\begin{array}{rl} N: & 00\ldots00 \\ \sim N: & 11\ldots11 \\ +1: & \underline{\qquad\qquad\qquad 1} \\ -N: & 00\ldots00 \end{array}$$

Следовательно, представления +0 и −0 в двоичном дополнительном коде совпадают.

Двоичное n-разрядное число 100 . . . 0 является "отрицательным самому себе" и не может быть отрицательным эквивалентом ни одному положительному n-разрядному числу в двоичном дополнительном коде.

Таблица 3.1. **Диапазон представления чисел в форме n-разрядного двоичного числа в дополнительном коде**

	Десятичное число	Восьмеричное представление		Десятичное число	Восьмеричное представление
n = 16	32767	077777	n = 8	127	177
	32766	077776		126	176
	.	.		.	.
		.			.
	.	.			.
	2	000002		2	002
	1	000001		1	001
	0	000000		0	000
	-1	177777		-1	377
	-2	177776		-2	376
					.
	.	.		.	.
	-32766	100002		-126	202
	-32767	100001		-127	201
	-32768	100000		-128	200

$$\begin{array}{rr} N: & 100\ldots00 \\ \sim N: & 011\ldots11 \\ +1: & \qquad 1 \\ \hline -N: & 100\ldots00 \end{array}$$

Это число и используют для представления -2^{n-1}. Следовательно, получаем диапазон представления чисел при использовании n-разрядного двоичного дополнительного кода:

$$-2^{n-1} \leqslant N \leqslant 2^{n-1} - 1.$$

В табл. 3.1 показан диапазон представления чисел при использовании 16-разрядного и 8-разрядного двоичных дополнительных кодов. Заметим, что 8-разрядное число в двоичном дополнительном коде (хранящееся в младшем байте) может быть преобразовано в 16-разрядное число в двоичном дополнительном коде простым переносом содержимого знакового разряда (разряд 7) во все разряды старшего байта.

3.2. Сложение и вычитание

Сумма двух чисел N_1 и N_2 в форме n-разрядного двоичного числа в дополнительном коде вычисляется сложением N_1 и N_2 так, как если бы у них не было знака (т. е. как положительные n-разрядные числа), игнорируя при этом перенос от MSB, если потребуется. Разность $N_1 - N_2$ вычисляется сложением N_1 с двоичным дополнительным кодом N_2.

Примеры для случаев различных представлений (n = 16)

Десятичное	Двоичное	Восьмеричное
1. 11	0 000 000 000 001 011	000013
+ 21	+ 0 000 000 000 010 101	+ 000025
32	0 000 000 000 100 000	000040
2. 21	0 000 000 000 010 101	000025
- 11	+ 1 111 111 111 110 101	+ 177765
10	0 000 000 000 001 010	000012
	1	
3. 11	0 000 000 000 001 011	000013
- 21	+ 1 111 111 111 101 011	+ 177753
- 10	1 111 111 111 110 110	177766
4. - 11	1 111 111 111 110 101	177765
- 21	+ 1 111 111 111 101 011	+ 177753
- 32	1 111 111 111 100 000	177740
	1	

□

Как было упомянуто в гл. 2, арифметические операции в PDP-11 влияют на коды условий Z и N в PSR. Они также влияют на коды C ("перенос") и V ("переполнение") в PSR. Например, после операции сложения

$$\text{код C (разряд 0)} = \begin{cases} 1, \text{если сложение приводит к переносу от MSB,} \\ 0 \text{ в других случаях,} \end{cases}$$

$$\text{код V (разряд 1)} = \begin{cases} 1, \text{если складываются числа одинакового знака, а их сумма противоположного знака ("переполнение"),} \\ 0 \text{ в других случаях.} \end{cases}$$

О кодах C и V будет сказано в последней главе.

3.3. Представление символов

Рассматриваемые символы включают:

буквы: A, B, C, . . . , Z, a, b, . . . , z;

цифры: 0, 1, 2, . . . , 9;

специальные знаки: +, *, /, ↑, $, пробел, . . . ;

непечатные указатели (знаки.): звонок, подача на строку (LF), возврат каретки (CR), . . .

В PDP-11 указанные символы представляются 8-разрядным двоичным числом, занимающим 1 байт. Правые 7 разрядов этого числа соответствуют коду ASCII данного символа (см. табл. 3.2). Заметим, что представление кода ASCII для цифры i соответствует 60 + i (восьмеричное).

Крайний левый разряд байта, который содержит символ, называется *разрядом контроля четности* и иногда используется для обнаружения ошибок.

Таблица 3.2. **Некоторые символы и их коды (восьмеричные) в ASCII**

Символ	Код	Символ	Код	Символ	Код
Звонок	007	>	076	←	137
Перевод строки	012	?	077	`	140
Возврат каретки	015	@	100	a	141
Пробел	040	A	101	b	142
!	041	B	102	c	143
"	042	C	103	d	144
#	043	D	104	e	145
$	044	E	105	f	146
%	045	F	106	g	147
&	046	G	107	h	150
'	047	H	110	i	151
(	050	I	111	j	152
)	051	J	112	k	153
*	052	K	113	l	154
+	053	L	114	m	155
,	054	M	115	n	156
-	055	N	116	o	157
.	056	O	117	p	160
/	057	P	120	q	161
0	060	Q	121	r	162
1	061	R	122	s	163
2	062	S	123	t	164
3	063	T	124	u	165
4	064	U	125	v	166
5	065	V	126	w	167
6	066	W	127	x	170
7	067	X	130	y	171
8	070	Y	131	z	172
9	071	Z	132	{	173
:	072	[	133	|	174
;	073	\	134	}	175
<	074	]	135	~	176
=	075	↑	136		

Если используется *схема "контроля четности"*, то разряд контроля принимает такое значение, чтобы общее число разрядов в байте, содержащих 1, было четным. Например, буква G (в ASCII ей соответствует код $107_8 = 01000111_2$) запоминается как $107_8 = 01000111_2$, так как число разрядов с 1 в коде ASCII для G четно. Однако буква g (в ASCII ей соответствует код $147_8 = 01100111_2$) запоминается как $347_8 = 11100111_2$, так как число разрядов с 1 в коде ASCII для g нечетно.

Предположим теперь, что символ (закодированный в соответствии со схемой контроля четности) передается из периферийного устройства в оперативную память или процессор. Если в процессе передачи произошло случайное изменение значения одного из разрядов, 0 изменился на 1 или наоборот, то число единиц в принятом байте уже не будет четным. Следовательно,

ошибка в передаче одного из разрядов байта может быть определена простым контролем на четность или нечетность числа единиц в байте[1].

Тот же принцип положен в основу схемы "контроля нечетности", используемый для этих целей, за исключением того, что в этом случае разряд контроля принимает такое значение, чтобы общее число разрядов в байте, содержащих 1, было нечетным.

Символ, вводимый в PDP-11 через телетайп, может в некоторых случаях иметь ненулевой разряд контроля четности и, следовательно, может не совпадать с его эквивалентом в коде ASCII, приведенным в табл. 3.2. Следовательно, прежде чем сравнивать двоичные коды введенного символа и символа, хранящегося в памяти, было бы правильно занести нули во все разряды принятого символа, исключая семь правых разрядов.

Разряд контроля четности не влияет на работу печатающего устройства: символ будет, как и положено, напечатан независимо от того, содержится ли в его разряде контроля четности 1 или 0.

Строки символов (т. е. последовательности символов) запоминаются в последовательно расположенных байтах памяти. Например, строка символов PDP-11 запоминается в трех последовательно расположенных словах:

042120 (D = 104, P = 120),
026520 (– = 055, P = 120),
030461 (1 = 061, 1 = 061).

В оперативной памяти последовательные строки символов могут быть для удобства разделены *нулевым* символом (код в ASCII – 000), т. е. таким символом, который редко включается в запись "обычной" строки символов.

3.4. Представление чисел с плавающей точкой

Базовая ЭВМ PDP-11 не приспособлена для обработки "действительных" чисел (т. е. чисел с дробной частью). Однако для некоторых моделей имеются дополнительные аппаратные средства для выполнения арифметических операций с вещественными числами; такие операции могут быть реализованы и программно.

Читатель, вероятно, знаком с *обычным представлением* действительного десятичного числа в виде числа с плавающей точкой[2], где, например, число 12345000000_{10} записывается в виде $.12345 \times 10^{11}$, а число 0,00012345 – в виде $.12345 \times 10^{-3}$. Это обозначение, представляющее число в виде *дробной его части или мантиссы* (т. е. 12345) и *показателя степени или порядка* (т. е. 11 или –3), а также десятичной точки (которая ставится слева от дробной части), особенно удобно для выражения очень больших и очень малых чисел.

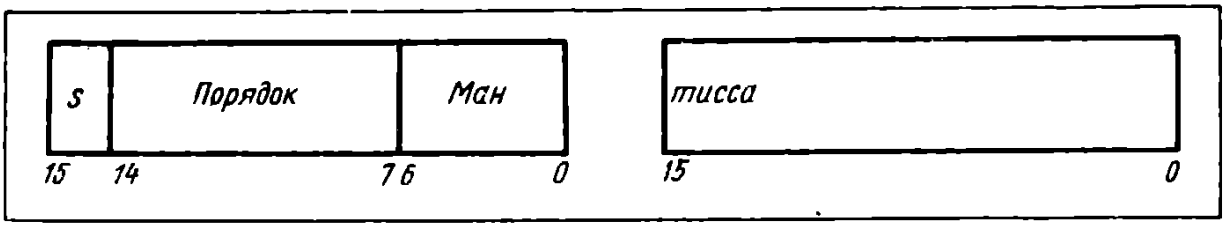
---

[1] Заметим, что любое нечетное число ошибок (для четного их числа это не так) может быть обнаружено с помощью этой схемы. Обычно вероятность получения больше одного ошибочного разряда слишком мала, чтобы заботиться об этом.

[2] В отечественной практике записи таких чисел используют понятие плавающая запятая, однако в практике записи машинных программ в качестве разделителя используется не запятая, а точка, поэтому в переводе оставлен термин плавающая точка. – *Прим. пер.*

Двоичный эквивалент представления числа в форме с плавающей точкой имеет следующее обозначение: .f × 2^e, где дробная часть и показатель степени представлены в двоичном виде. Точка, стоящая перед f, называется при этом *двоичной точкой*. Например, $-832_{10} = -1101000000_2$ записывается в виде $-.1101 \times 2^{1010}$, а $0{,}0390625_{10} = 0.0000101_2$ записывается в виде $.101 \times 2^{100}$.

Представление действительных чисел в PDP-11 (как и в большинстве других машин) базируется на форме записи с двоичной плавающей точкой. Действительное число с "одинарной точностью" занимает в памяти два слова (32 разряда) и записывается в следующем формате[1]:



Знаковый разряд s содержит 0, если число положительное, или 1, если число отрицательное. В разрядах 7 – 14 левого слова запоминается не истинное значение двоичного порядка, а его значение, увеличенное на $200_8 = 10000000_2$, что соответствует диапазону изменения порядка от -128_{10} до $+127_{10}$. Предполагается, что двоичная точка располагается сразу слева от мантиссы. Мантисса нормализуется, т. е. сдвигается влево (при этом порядок соответственно увеличивается) до тех пор, пока самый левый его разряд не станет равным 1. Эта ведущая 1 (она будет в этом разряде всегда после нормализации, а следовательно, будет избыточной) затем удаляется и мантисса сдвигается еще на один дополнительный разряд влево. Исключением является действительное число 0, которое представляется произвольной мантиссой и заполненным нулями полем порядка (разряды 7 – 14).

Примеры

1. $N = -832_{10} = -.1101 \times 2^{+1010}$.

Нормализованная мантисса равна 1101 (ведущая 1 затем удаляется), а порядок (со смещением 200_8) равен 10000000 + 1010 = 10001010. Следовательно, двоичное представление N с плавающей точкой имеет вид 1 10001010 1010000 0000000000000000.

2. $N = 0{,}0390625 = .101 \times 2^{-100}$.

Нормализованная мантисса равна 101 (ведущая 1 затем удаляется), а порядок (со смещением 200_8) равен 10000000 – 100 = 1111100. Следовательно, двоичное представление N с плавающей точкой имеет вид 0 01111100 0100000 0000000000000000. □

Ниже в тексте мы больше не встретимся с действительными числами, а будем иметь дело только с целыми числами.

Упражнения

3.1. Показать восьмеричное содержимое слов, содержащих:
а) –1; б) -1000_{10}; в) -1000_8; г) -7530_8.

[1] Числа с "двойной точностью" в формате с плавающей точкой занимают в памяти четыре слова, причем поле мантиссы состоит из 55 разрядов (вместо 23 при одинарной точности).

3.2. Найти восьмеричное число N, если:

а) 16-разрядный двоичный дополнительный код числа −N (его восьмеричная форма) имеет вид 176542;

б) 8-разрядный двоичный дополнительный код числа −N (его восьмеричная форма) имеет вид 273.

3.3. Без перевода в двоичное или десятичное представление написать восьмеричное содержимое слов со следующими числами:

а) $500_8 - 311_8$; б) $-1043_8 - 751_8$; в) $3721_8 - 6260_8$.

3.4. Пусть байты с 1000 по 1007 заполнены следующими числами и символами.

1000: 123_{10}	1001: 17_8
1002: -32_8	1003: ASCII "6"
1004: ASCII "Z"	1005: -100_{10}
1006: ASCII "LF"	1007: ASCII "CR"

Показать восьмеричное содержимое слов 1000, 1002, 1004, 1006.

3.5. Определить строку символов, хранящихся в пяти последовательно расположенных словах, восьмеричное содержимое которых равно

020061, 026117, 046103, 041517, 003513.

3.6. Показать представление в форме с плавающей точкой чисел

а) $-(1/16)_{10} = -0.0001_2$; б) $(10.25)_{10} = 1010.01_2$.

3.7. Каким десятичным числам соответствует следующее представление в форме с плавающей точкой (заданное в восьмеричном коде):

а) 041377 000000; б) 137000 000000?

Глава 4

Команды и режимы адресации

В этой главе мы опишем формат команд PDP-11 и различные методы, используемые для определения адресов операндов (режимы адресации). Команды в PDP-11 занимают одно слово (за которым следует одно или два дополнительных слова для определения операндов) и записываются в виде шестиразрядных восьмеричных чисел. Эти числа соответствуют представлению команд в программе, написанной в *машинных кодах*, т. е. программе, фактически находящейся в оперативной памяти и выполняемой процессором. Мы будем также записывать команды в *мнемонической* или *символической* форме. Это та форма, в которой команды изображаются в программе, написанной на *языке ассемблера*, т. е. в "символической" программе, которая требует трансляции (с помощью *ассемблера*) на язык машинных кодов перед ее выполнением.

С этого момента все числа предполагаются восьмеричными, если это не будет оговорено особо.

4.1. Цикл выполнения команд

Как было упомянуто в гл. 2, счетчик команд (PC) в любой момент времени содержит адрес следующей выполняемой команды. После того, как команда выбрана по этому адресу, но *перед ее выполнением*, содержимое счетчика команд увеличивается на 2. Следовательно, если счетчик команд не из-

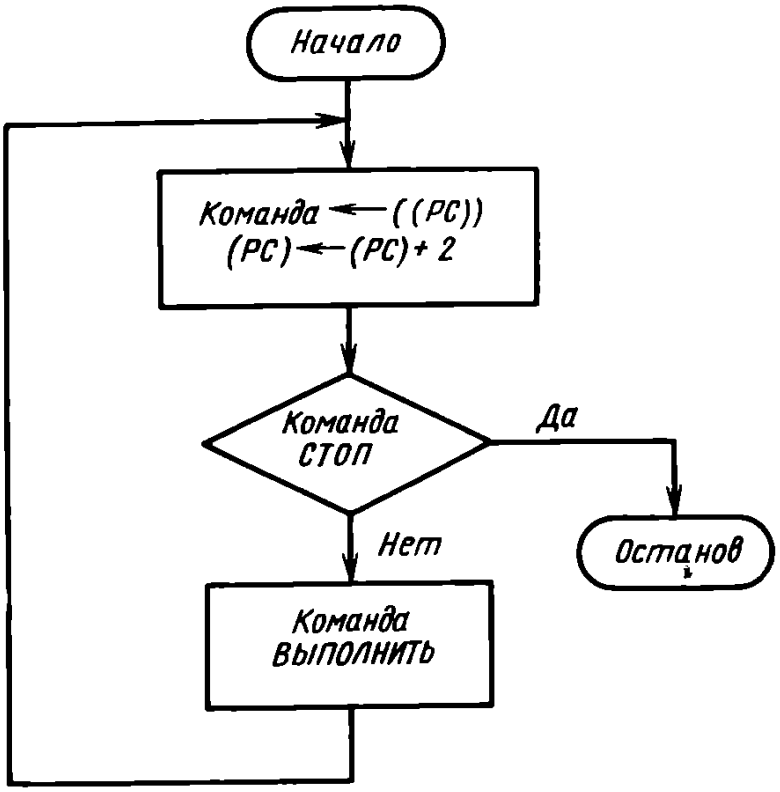
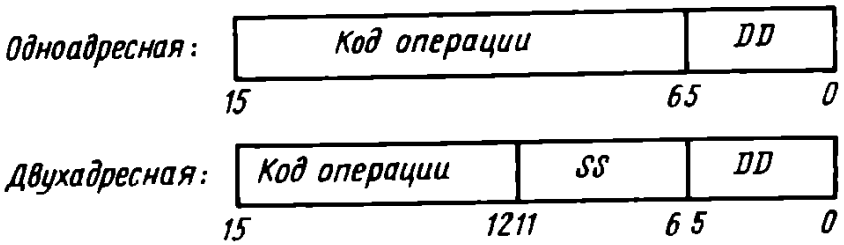


Рис. 4.1. Цикл выполнения программы на PDP-11

менился в процессе выполнения (что может произойти при командах перехода или других), следующая команда автоматически выбирается из следующего слова. Этот "цикл выполнения команд" продолжается до тех пор, пока не встретится команда HALT (ОСТАНОВ), которая вызывает останов PDP-11 (рис. 4.1).

4.2. Одноадресные и двухадресные команды

Одноадресные команды состоят из *кода операции* и *поля* DD, которое определяет местоположение *операнда-приемника*. Двухадресные команды содержат, кроме кода операции и поля DD, *поле* SS, которое определяет местоположение *операнда-источника*. Ниже приведены форматы этих команд:



Если команда оперирует байтами ("байтовая команда"), а не словами, то разряд 15 устанавливается в 1, в другом случае – в 0. (*Замечание*: адрес команды, оперирующей словами, должен быть четным, в противном случае будет ошибка в программе.)

На языке ассемблера одноадресные и двухадресные команды имеют формат

OPR d
OPR s, d

где OPR обозначает код операции (от английского operation code), а d и s – символы, соответствующие операндам DD и SS соответственно. В байтовой команде к OPR прибавляется буква B, т. е. CLR принимает вид CLRB.

В табл. 4.1 перечислены одноадресные и двухадресные команды PDP-11. Все команды могут быть выполнены в форме байтовых команд, за исключением ADD и SUB (СЛОЖЕНИЕ и ВЫЧИТАНИЕ), которые могут оперировать только 16-разрядными числами в дополнительном коде, а также SWAB и JMP (ПЕРЕСТАНОВКА БАЙТОВ и АБСОЛЮТНЫЙ БЕЗУСЛОВНЫЙ ПЕРЕХОД).

Таблица 4.1. **Одноадресные и двухадресные команды**

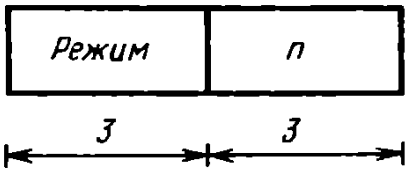
Машинный язык	Ассемблер	Название команды	Алгоритм выполнения
		Одноадресные	
0050DD	CLR d	ОЧИСТКА	(d) ← 0
0051DD	COM d	ИНВЕРТИРОВАНИЕ	(d) ← ~(d)
0052DD	INC d	ИНКРЕМЕНТ	(d) ← (d) + 1
0053DD	DEC d	ДЕКРЕМЕНТ	(d) ← (d) – 1
0054DD	NEG d	СМЕНА ЗНАКА	(d) ← –(d)
0057DD	TST d	ПРОВЕРКА	(d) ← (d)
0060DD	ROR d	ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО	(d) ← (d), сдвинутое вправо на 1 разряд
0061DD	ROL d	ЦИКЛИЧЕСКИЙ СДВИГ ВЛЕВО	(d) ← (d), сдвинутое влево на 1 разряд
0062DD	ASR d	АРИФМЕТИЧЕСКИЙ СДВИГ ВПРАВО	(d) ← (d)/2
0063DD	ASL d	АРИФМЕТИЧЕСКИЙ СДВИГ ВЛЕВО	(d) ← 2*(d)
0003DD	SWAB d	ПЕРЕСТАНОВКА БАЙТОВ	$(d)_{млад} \leftrightarrow (d)_{стар}$
0055DD	ADC d	СЛОЖЕНИЕ С ПЕРЕНОСОМ	(d) ← (d) + C
0056DD	SBC d	ВЫЧИТАНИЕ ПЕРЕНОСА	(d) ← (d) – C
0001DD	JMP d	АБСОЛЮТНЫЙ БЕЗУСЛОВНЫЙ ПЕРЕХОД	(PC) ← (d)
		Двухадресные	
01SSDD	MOV s, d	ПЕРЕСЫЛКА	(d) ← (s)
02SSDD	CMP s, d	СРАВНЕНИЕ	формирование (s) – (d)
06SSDD	ADD s, d	СЛОЖЕНИЕ	(d) ← (s) + (d)
16SSDD	SUB s, d	ВЫЧИТАНИЕ	(d) ← (d) – (s)
03SSDD	BIT s, d	ПРОВЕРКА РАЗРЯДОВ	формирование (s)∧(d)*
04SSDD	BIC s, d	ОЧИСТКА РАЗРЯДОВ	(d) ← [~ (s)] ∧(d)*
05SSDD	BIS s, d	УСТАНОВКА РАЗРЯДОВ	(d) ← (s) ∨(d)*

*Знак ∨ соответствует операции "ИЛИ"; ∧ – "И"; ~ – "НЕ".

Бо́льшая часть команд, перечисленных в табл. 4.1, не требует пояснений. Подробное описание каждой из них можно найти в руководстве "PDP-11 Processor Handbook". Некоторые из этих команд будут подробно рассмотрены в последующих главах.

4.3. Режимы адресации

Каждое из полей DD и SS делится на трехразрядное *подполе регистра* (номер регистра n) и трехразрядное *подполе режима* (адресации):



В табл. 4.2 описан способ, по которому из этих полей в процессе выполнения программы можно получить адреса операндов. В табл. 4.3 приведен пример, показывающий различные режимы адресации.

Таблица 4.2. **Режимы адресации**

Режим	Имя режима	Язык ассемблера	Местоположение операнда	Описание режима
0	Регистровый	Rn	Rn	Операнд в регистре Rn
1	Регистровый косвенный	(Rn)	(Rn)	Адрес операнда находится в регистре Rn
2	Автоинкрементный	(Rn)+	(Rn)	Адрес операнда находится в Rn; после выборки операнда (Rn) ← (Rn) + 2, т. е. содержимое Rn увеличивается на 2*
3	Автоинкрементный косвенный	@(Rn)+	((Rn))	В регистре Rn находится адрес адреса операнда; после выборки операнда (Rn) ← ← (Rn) + 2
4	Автодекрементный	–(Rn)	(Rn)	Перед вычислением адреса (Rn) ← (Rn) – 2, т. е. содержимое Rn уменьшается на 2**. Адрес операнда находится в Rn
5	Автодекрементный косвенный	@–(Rn)	((Rn))	Перед вычислением адреса (Rn) ← (Rn) – 2. В регистре Rn находится адрес адреса операнда
6	Индексный	X(Rn)	X + + (Rn)	Адрес операнда равен сумме X + (Rn). Адрес X находится в счетчике команд PC; после выборки X (PC) ← (PC) + 2
7	Индексный косвенный	@X(Rn)	(X + + (Rn))	Адрес адреса операнда равен X + (Rn). Адрес X находится в (PC); после выборки X (PC) ← (PC) + 2

*Но (Rn) ← (Rn) + 1, если команда является байтовой командой и n < 6.
**Но (Rn) ← (Rn) – 1, если команда является байтовой командой и n < 6.

Таблица 4.3. **Примеры применения режимов адресации**

Перед выполнением каждой команды предполагается, что

(R3) = 000124 (456) = 000174
(122) = 000701 (524) = 000304
(124) = 000456 (700) = 000613
(304) = 000537

Команды

Машинный язык*	Язык ассемблера	Совершаемое действие
005003	CLR R3	(R3) ← 0
005013	CLR (R3)	(124) ← 0
005023	CLR (R3) +	(124) ← 0, (R3) ← 000126
105023	CLRB (R3) +	(124) ← 000400, (R3) ← 000125
005033	CLR@ (R3) +	(456) ← 0, (R3) ← 000126
005043	CLR – (R3)	(R3) ← 000122, (122) ← 0
105043	CLRB – (R3)	(R3) ← 000123, (122) ← 000301
105053	CLRB@ – (R3)	(R3) ← 000122, (700) ← 000213
005063 000400	CLR 400(R3)	(524) ← 0
005073 000400	CLR@ 400(R3)	(304) ← 0
010300	MOV R3, R0	(R0) ← 000124
011300	MOV (R3), R0	(R0) ← 000456
012300	MOV (R3) + , R0	(R0) ← 000456, (R3) ← 000126
112300	MOVB (R3) + , R0	(R0) ← 000056, (R3) ← 000125**
013300	MOV@ (R3) + , R0	(R0) ← 000174, (R3) ← 000126
014300	MOV – (R3), R0	(R3) ← 000122, (R0) ← 000701
114300	MOVB – (R3), R0	(R3) ← 000123, (R0) ← 000001
115300	MOVB@ – (R3), R0	(R3) ← 000122, (R0) ← 000001
016300 000400	MOV 400(R3), R0	(R3) ← 000304
117303 000400	MOVB @ 400 (R3), R3	(R3) ← 000137
016363 000400 000500	MOV 400(R3), 500(R3)	(624) ← 000304
012363 000400	MOV (R3) + , 400(R3)	(526) ← 000456, (R3) ← 000126

*Поля SS и DD подчеркнуты.

**По команде MOVB s, Rn происходит запоминание байта s в младшем байте Rn; старший байт Rn заполняется затем содержимым 7-го разряда ("знака расширения").

4.4. Непосредственная адресация

Предположим, что мы имеем

Адрес	Содержимое	
α	OPR (PC) +, d	(команда с SS = 27)
α + 2	k	(константа)
α + 4	. . .	

Перед выполнением команды содержимое PC увеличивается на два (см. рис. 4.1). Следовательно, адрес операнда этой команды равен $\alpha + 2$ и, следовательно, k – операнд-источник. После того как этот операнд выбран, (PC) ← ← (PC) + 2 = $\alpha + 4$ (в силу автоинкрементного режима). В таком случае операнд может быть размещен сразу же (непосредственно) за командой и не будет ошибочно интерпретирован как следующая команда (отсюда ясен термин *непосредственная адресация*).

Последовательность OPR (PC) +, d на языке ассемблера можно сокращенно записать как

OPR # k, d

(использование непосредственной адресации для операнда-приемника лишено смысла).

Примеры

	Машинный язык	Язык ассемблера	Совершаемое действие
1.	012700 177776	MOV #177776, R0	(R0) ← 177776
2.	062716 000005	ADD #5, (SP)	((SP)) ← 5 + ((SP))

□

4.5. Абсолютная адресация

Адрес	Содержимое	
α	OPR @(PC) +, d	(команда с SS = 37)
$\alpha + 2$	A	(адрес)
$\alpha + 4$	. . .	

Перед выполнением данной команды содержимое PC увеличивается на два (см. рис. 4.1). Следовательно, адрес адреса операнда равен $\alpha + 2$, поэтому сам операнд содержится по адресу, указанному в A. После выборки операнда (PC) ← (PC) + 2 = $\alpha + 4$ (в силу автоинкрементного режима). В таком случае адрес операнда может располагаться сразу после команды и не будет ошибочно интерпретирован как следующая команда. Этот метод адресации называется *абсолютной адресацией.*

Последовательность OPR @(PC) +, d можно в этом случае записать как

OPR @ # A, d.

Аналогичный комментарий применим и когда DD = 37.

Примеры

	Машинный язык	Язык ассемблера	Совершаемое действие
1.	005037 000472	CLP @ # 472	(472) ← 0
2.	012737 000007 177566	MOV # 7, @ # 177566	(177566) ← 000007 (звенит звонок)

□

4.6. Относительная адресация

Адрес	Содержимое	
α	OPR 4 (PC), d	(команда с SS = 67)
$\alpha + 2$	X	(относительный адрес)
$\alpha + 4$	. . .	

Перед выполнением команды содержимое PC увеличивается на два (см. рис. 4.1) и, следовательно, будет равно $\alpha + 2$. Как только X будет выбрано из $\alpha + 2$, PC снова увеличивается и становится равным $\alpha + 4$. Таким образом, адрес A операнда-источника, а именно X + (PC), будет равен $A = X + \alpha + 4$.

Видно, что X является *адресом операнда относительно текущего значения* PC, отсюда понятно содержание термина *относительная адресация*.

Команда OPR X(PC), d может быть в этом случае сокращенно записана как OPR A, d ($A = X + \alpha + 4$).

Аналогичный комментарий можно привести и для DD = 67.

Примеры

	Машинный язык	Язык ассемблера	Совершаемое действие
1.	500: 005067 502: 000200	CLP 704	(704) ← [704 − 504 = 200]

Заметим, что команда, приведенная выше, эквивалентна

	500: 005037 502: 000704	CLR @ # 704	
2.	500: 162767		
	502: 000015	SUB # 15,306	(306) ← (306) − 15
	504: 177600		[306 − 506 = −200 = 177600]

Заметим, что в этом случае X запоминается в $\alpha + 4$, а не в $\alpha + 2$, и, следовательно, $A = X + \alpha + 6$. □

Мы видим, что относительный адрес X является определенным "расстоянием" между фактическим адресом и текущим значением PC. Таким образом, если вся программа сдвигается в памяти, то нет необходимости модифицировать относительный адрес, так как указанное расстояние останется тем же самым. Относительная адресация широко используется в "позиционно-независимом программировании" — написании таких программ, которые могут размещаться в любом месте в памяти. Более подробно об этом будет сказано в гл. 9.

4.7. Косвенно-относительная адресация

Адрес	Содержимое	
α	OPR @ X(PC), d	(команда с SS = 77)
$\alpha + 2$	X	(относительный адрес)
$\alpha + 4$	. . .	

Адрес A адреса операнда равен X + (PC), т. е. адрес операнда равен A = X + α + 4 (см. разд. 4.6). Следовательно, X является адресом адреса операнда относительно текущего значения PC.

Команду OPR @ X(PC), d можно в этом случае сокращенно записать в виде OPR @ A, d.

Аналогичный комментарий можно привести и для DD = 77.

Примеры

Машинный язык	Язык ассемблера	Совершаемое действие
500: 005077 502: 000200	CLP @ 704	(123) ← 0
.		
.		
.		
704: 000123		
.		□

4.8. Команды перехода

Команды перехода используются для программ, разветвляющихся тогда, когда выполняются определенные условия. Они имеют вид "базовый код + смещение", где базовый код — 16-разрядное число, младший байт которого всегда нулевой, а смещение (обозначаемое f) — 8-разрядное в дополнительном коде числа. Следовательно (см. табл. 3.1),

$$-128_{10} \leqslant f \leqslant 127_{10} \text{ или } 200_8 \leqslant f \leqslant 177_8.$$

Если в процессе выполнения условия перехода удовлетворяются, то (PC) ← ← (PC) + 2f.

Следовательно, если команда перехода находится по адресу β, то следующая команда будет взята по адресу $\beta = 2f + \alpha + 2$. И наоборот, если мы хотим запрограммировать переход с адреса α на адрес β, то должны иметь $f = \frac{1}{2}(\beta - \alpha - 2)$.

Заметим, что f — это расстояние, измеряемое числом слов между (PC) и операндом-приемником для данного перехода, следовательно, эта команда ветвления является позиционно-независимой.

В табл. 4.4 приведены команды перехода для PDP-11.

При переходе за пределы области размером 127_{10} может быть использована команда JMP.

Примеры

1. Найти эквивалент на машинном языке для команды 554: BEQ 570.

Ответ: $f = (1/2)(570 - 554 - 2) = (1/2) \cdot 12 = 5$,

001400 + 005 = 001405 ← команда.

2. Найти эквивалент на машинном языке для команды 624: BR 602.

Ответ: $f = (1/2)(602 - 624 - 2) = (1/2) \cdot (-24) = -12 = 366$,

000400 + 366 = 000766 ← команда. □

Для "коротких" переходов от α до β можно применить следующую упрощенную процедуру для вычисления смещения: если $\beta > \alpha$, сопоставить словам $\alpha + 2$, $\alpha + 4$, $\alpha + 6$, . . . числа 0, 1, 2, 3, . . . ; тогда число, сопоставленное β, и будет f. Если $\beta \leqslant \alpha$, сопоставить словам α, $\alpha - 2$, $\alpha - 4$, . . . числа 377, 376, 375, . . . ; тогда число, сопоставленное β, и будет f.

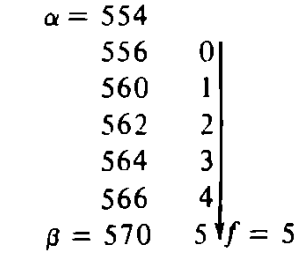
Таблица 4.4. **Команды перехода**

Базовый код	Язык ассемблера	Название перехода[1]
000400	BR a	БЕЗУСЛОВНЫЙ
001000	BNE a	ПО НЕРАВЕНСТВУ НУЛЮ (Z = 0)
001400	BEQ a	ПО РАВЕНСТВУ НУЛЮ (Z = 1)
100000	BPL a	ПО ПЛЮСУ (N = 0)
100400	BMI a	ПО МИНУСУ (N = 1)
102000	BVC a	ПО ОТСУТСТВИЮ ПЕРЕПОЛНЕНИЯ (V = 0)
102400	BVS a	ПО ПЕРЕПОЛНЕНИЮ (V = 1)
103000	BCC a	ПО ОТСУТСТВИЮ ПЕРЕНОСА (C = 0)
103400	BCS a	ПО ПЕРЕНОСУ (C = 1)
002000	BGE a	ПО "БОЛЬШЕ ИЛИ РАВНО НУЛЮ" (N ∀ V = 0)
002400	BLT a	ПО "МЕНЬШЕ НУЛЯ" (N ∀ V = 1)
003000	BGT a	ПО "БОЛЬШЕ НУЛЯ" [Z ∨ (N ∀ V) = 0]
003400	BLE a	ПО "МЕНЬШЕ ИЛИ РАВНО НУЛЮ" [Z ∨ (N ∀ V) = 1]
101000	BHI a	БЕЗЗНАКОВЫЙ ПО "БОЛЬШЕ НУЛЯ" (C∨Z = 0)
101400	BLOS a	БЕЗЗНАКОВЫЙ ПО "МЕНЬШЕ ИЛИ РАВНО НУЛЮ" (C∨Z = 1)
103000	BHIS a	БЕЗЗНАКОВЫЙ ПО "БОЛЬШЕ ИЛИ РАВНО НУЛЮ" (C = 0)
103400	BLO a	БЕЗЗНАКОВЫЙ ПО "МЕНЬШЕ НУЛЯ" (C = 1)

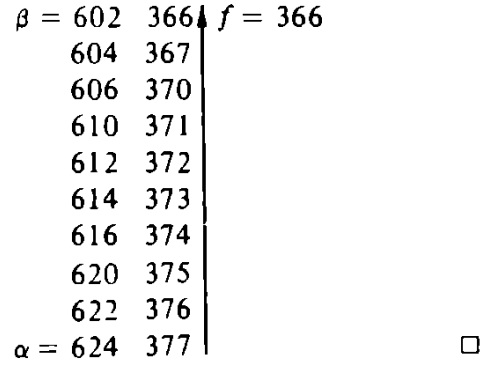
[1] **Правила выполнения операций:** ∀: 0 + 0 = 0, 0 + 1 = 1, 1 + 1 = 0; V: 0 + 0 = 0, 0 + + 1 = 1, 1 + 1 = 1.

Примеры

1. Осуществить переход от 554 к 570:



2. Осуществить переход от 624 к 602:



□

4.9. Команды без операндов

В табл. 4.5 приведены некоторые команды PDP-11 без операндов. Команда HALT (ОСТАНОВ) является одной из таких команд, которая используется почти в каждой программе. Другими такими командами являются команды, осуществляющие операции над кодами условий и используемые либо для сброса (т. е. установки 0), либо установки (т. е. установки 1) соответствующих разрядов кодов условий в PSR.

Таблица 4.5. **Команды без операндов**

Машинный язык	Язык ассемблера	Описание команды
000000	HALT	Останов
000241	CLC	Сброс C
000242	CLV	Сброс V
000244	CLZ	Сброс Z
000250	CLN	Сброс N
000257	CCC	Сброс всех разрядов кода условий (C, V, Z, N)
000261	SEC	Установка C
000262	SEV	Установка V
000264	SEZ	Установка Z
000270	SEN	Установка N
000277	SCC	Установка 1 во все разряды кода условий (C, V, Z, N)

Две или больше команд сброса/установки можно записать подряд, разделяя их знаком !. Например, для сброса разрядов C и V служит команда ⟨CLC!CLV⟩.

4.10. Примеры

Ниже приводится несколько фрагментов программ, показывающих главным образом режимы адресации PDP-11, а также некоторые команды PDP-11. Следует заметить, что некоторые команды ассемблера приводят к использованию "символической" адресации (например, с меткой NEXT (СЛЕДУЮЩИЙ), а не "абсолютной" адресации (т. е. числовой).

Место в программе, соответствующее такой адресации, указывается соответствующим символом, записанным в дополнительной колонке слева (например, NEXT:).

Предполагается, что все фрагменты программ начинаются с ячейки 600.

1. Переслать в R1 абсолютное значение 16-разрядного числа X в дополнительном коде, хранящегося в R0.

Машинный язык	Язык ассемблера	Комментарии
600: 010001	MOV R0, R1	;ПЕРЕСЛАТЬ X В R1
602: 100001	BPL NEXT	;ЕСЛИ X ⩾ 0, ТО ПЕРЕЙТИ НА МЕТКУ NEXT
604. 005401	NEG R1	;В ПРОТИВНОМ СЛУЧАЕ СМЕНИТЬ ЗНАК В R1
606: . . .	NEXT: . . .	

Замечание. Команда MOV автоматически проводит сброс или установку кодов условий Z и N в PSR в соответствии с значением пересылаемого числа. Таким образом, переход на NEXT в соответствии с командой BPL произойдет, если только X положительно.

2. Принять символ с телетайпа, переслать его в R5 и напечатать его.

Машинный язык	Язык ассемблера		Комментарии
600: 105767	WAIT1:	TSTB 177560	;СИМВОЛ ПРИНЯТ?
602: 176754			
604: 100375		BPL WAIT1	;ЕСЛИ НЕТ, ЖДАТЬ (ПЕРЕЙТИ К ;WAIT1)
606: 016705		MOV177562,R5	;ИНАЧЕ, ПЕРЕСЛАТЬ ЕГО В R5
610: 176750			
612: 105767	WAIT2:	TSTB 177564	;УСТРОЙСТВО ПЕЧАТИ СВОБОДНО?
614: 176746			
616: 100375		BPL WAIT2	;ЕСЛИ НЕТ, ЖДАТЬ (ПЕРЕЙТИ К ;WAIT2)
620: 010567		MOV R5, 177566	;ИНАЧЕ, ПЕЧАТЬ (R5)
622: 176742			

Замечание. Как было упомянуто в разд. 2.3, факт наличия или отсутствия 1 в разряде "готовности" (разряд 7) регистра состояния печатающего устройства можно установить, проверяя, является ли младший байт данного регистра (рассматриваемый как 8-разрядное целое число в дополнительном коде) отрицательным. Для этого используется пара команд TSTB/BPL. Заметим также, что все регистры состояния и регистры данных адресуются здесь в режиме относительной адресации.

3. Запомнить буквы A, B, C, . . . , в последовательности байтов, начиная с 1200-го.

Машинный язык	Язык ассемблера		Комментарии
600: 012700		MOV # 101,R0	;УСТАНОВИТЬ НАЧАЛЬНЫЙ СИМВОЛ A
602: 000101			
604: 012701		MOV #1200, R1	;УСТАНОВИТЬ АДРЕС БАЙТА РАВ- ;НЫМ 1200
610: 110021	AGAIN:	MOVB R0,(R1)+	;ЗАПОМНИТЬ СИМВОЛ, УВЕЛИЧИТЬ ;АДРЕС
612: 020027		CMP R0, # 132	;ОБРАБАТЫВАЕТСЯ ЛИ СИМВОЛ Z?
614: 000132			
616: 001402		BEQ EXIT	;ЕСЛИ ДА, ЗАКАНЧИВАЕМ ОБРА- ;БОТКУ
620: 005200		INC R0	;ИНАЧЕ, ФОРМИРУЕМ СЛЕДУЮЩИЙ ;СИМВОЛ
622: 000772		BR AGAIN	;ВОЗВРАТ ДЛЯ ОБРАБОТКИ ДРУГО- ;ГО СИМВОЛА
624: . . .	EXIT:	. . .	

Замечание. Константы, представляющие адреса или символы (такие, как 1200 или 101), пересылаются с помощью непосредственной адресации. Автоинкрементный режим, используемый в AGAIN, автоматически модифици-

рует адрес после каждого обращения. Команда INC формирует следующую букву путем добавления 1 к коду ASCII предыдущей буквы.

4. Адрес числа Y запоминается в ячейке 1000. Определить число C единиц в двоичном коде Y и переслать результат в R4.

Машинный язык	Язык ассемблера	Комментарии
600: 005004	CLR R4	; УСТАНОВИТЬ ЗНАЧЕНИЕ С РАВ- ; НЫМ 0
602: 017705	MOV @1000, R5	; ПЕРЕСЛАТЬ Y В R5
604: 000172		
606: 001404	REPT: BEQ OUT	; ЕСЛИ Y = 0, ПЕРЕЙТИ НА МЕТКУ ; OUT
610: 100001	BPL SHIFT	; ЕСЛИ MSB = 0, C НЕ ИЗМЕНЯЕТСЯ
612: 005204	INC R4	; ИНАЧЕ, C = C + 1
614:006305	SHIFT:ASL R5	; СДВИНУТЬ Y НА ОДИН РАЗРЯД ; ВЛЕВО
616: 000773	BR REPT	; ПОВТОРИТЬ ЦИКЛ
620: . . .	OUT: . . .	

Замечание. Y находится при использовании косвенно-относительного режима адресации. Число единиц в Y вычисляется подсчетом, сколько раз Y становится отрицательным (т. е. сколько раз 15-й разряд становится равным 1) при сдвиге Y влево (используя команду ASL).

5. Ячейка 160 содержит команду с одним операндом. Массив из восьми слов, начиная с 750, содержит восемь адресов. Проверить режим адресации указанной команды. Если адресация соответствует D, то в программе осуществить переход по адресу, хранящемуся в (D + 1)-м слове массива (т. е. по адресу 750 + 2 * D).

Машинный язык	Язык ассемблера	Комментарии
600: 013700	MOV @ # 160, R0	; ПЕРЕСЛАТЬ КОМАНДУ В R0
602: 000160		
604: 042700	BIC # 177707, R0	; НАЙТИ ЗНАЧЕНИЕ 8 * D
606: 177707		
610: 006000	ROR R0	; ВЫЧИСЛИТЬ
612: 006000	ROR R0	; (8 * D)/4 = 2 * D
614: 000170	JMP @ 750 (R0)	; ПЕРЕЙТИ ПО АДРЕСУ 750 + 2 * D
616: 000750		

Замечание. Команда размещается в R0 при использовании абсолютной адресации (хотя можно было бы использовать и относительный режим адресации). Команда BIC осуществляет сброс всех разрядов в R0, за исключением разрядов 3, 4 и 5 (т. е. она сбрасывает на нуль все разряды, соответствующие единичным разрядам в 177707; в результате D сдвигается на три разряда влево (т. е. формируется 8 * D). Чтобы вычислить 2 * D = (8 * D)/4,

R0 сдвигается вправо на два разряда (используется команда ROR в предположении, что C вначале равно 0).

6. Вставить новое число в нужное место массива 16-разрядных чисел в дополнительном коде, хранящихся в памяти; первое и последнее слова массива имеют адреса A и B соответственно. Считать, что все числа положительные, что все они расположены в порядке возрастания модулей, что новое число N находится в R0 и что адреса A и B хранятся в R1 и R2 соответственно.

```
Машинный язык     Язык ассемблера          Комментарии
-------------------------------------------------------------------
600: 012741                MOV #177777,-(R1); УСТАНОВИТЬ -1 ПО АДРЕСУ A-2

602: 177777
604:005722                 TST (R2) +       ; ЗАНЕСТИ B + 2 В R2
606: 024200  LOOP:         CMP - (R2),R0    ; СЛЕДУЮЩИЙ НОМЕР K ⩽ N?

610: 003403                BLE INSEPT       ; ЕСЛИ ДА, ТО ПЕРЕЙТИ К INSERT
612: 011262                MOV(R2),2(R2)    ; ИНАЧЕ,СДВИНУТЬ К НА ОДНО СЛОВО
614: 000002
616: 000773                BR LOOP          ; ВЕРНУТЬСЯ И ВЗЯТЬ СЛЕДУЮЩЕЕ К
620: 010062 INSERT:MOVR0, 2(R2)             ; ПОСТАВИТЬ N ПЕРЕД К
622: 000002
```

Замечание. Последовательные числа K, пробегающие ряд значений (в порядке уменьшения) от B до A, сравниваются с N и сдвигаются на одно слово вперед, пока не найдется такое K, которое меньше или равно N. Когда такое K найдется (а −1, хранящаяся в A − 2, гарантирует, что это в конце концов произойдет), N будет помещено перед этим K. Указанное сравнение осуществляется в LOOP с использованием автодекрементного режима, когда уменьшение адреса происходит всегда *перед* очередным обращением; таким образом, сканирование массива должно начинаться с регистра R2, содержащего B + 2, а не B. Это начальное обращение осуществляется с помощью команды TST (R2) +, единственная цель которой увеличить содержимое R2 (изменить B на B + 2), а не провести какой-то контроль. Заметим, что индексный режим адресации 2(R2), в отличие от автоинкрементной адресации (R2) +, оставляет содержимое R2 без изменения.

Упражнения

4.1. Каждая из нижеследующих команд хранится по адресу 500. Перед каждым исполнением такой команды в соответствующих ячейках хранятся следующие данные: (R0) = 100, (76) = 176, (100) = 200, (176) = 276 и (200) = 500. Транслировать каждую команду на машинный язык и определить содержимое R0 и R1, после того как она выполнится.

а) MOV R0, R1
б) MOV (R0), R1,
в) MOV (R0) +, R1
г) MOV @ (R0) +, R1
д) MOV -(R0), R1
е) MOV @ -(R0), R1

ж) MOV 100(R0), R1
и) MOV #100, R1
л) MOV 100, R1
з) MOV @100(R0), R1
к) MOV @#100, R1
м) MOV @100, R1

4.2. Как выполняется следующая команда?

```
MOV -(PC), -(PC)
```

4.3. Чему равно восьмеричное содержание R0 после окончания следующей программы?

```
MOV   #1, R0
MOV   (PC), -(PC)
DEC   R0
HALT
```

4.4. Ячейка 1000 содержит восьмеричное число 42356. Полагая, что следующая программа начинается с адреса 500, найти восьмеричное содержимое R0 после выполнения каждой программы:

```
MOVB  @#1001, R0
BIC   #770, R0
MOV   500(R0), R0
SUB   #123456, R0
```

4.5. Следующая программа начинается с адреса 500. Транслируйте ее на машинный язык.

```
CMP   (R3), #123
BLT   650
MOVB  @(R4)+, 701
BIC   R5, -(R0)
CMP   @100, -350(PC)
BEQ   400
SUB   @22(SP), (R2)+
COMB  @#755
JMP   @-(R1)
```

4.6. Следующая машинная программа начинается с адреса 500. Транслируйте ее на язык ассемблера.

```
022733
000456
003756
012146
126467
000160
000273
001104
043777
000666
177652
160750
067215
177773
```

4.7. Следующая машинная программа начинается с адреса 0. Что делает эта программа?

```
016700
000022
016701
000020
105710
100376
105711
100376
022021
111011
000765
177560
177564
```

4.8. Следующая машинная программа начинается с адреса 0. Чему равно восьмеричное содержимое ячейки 0 после окончания программы?

```
062737
040000
000000
100774
000000
```

4.9. Следующая машинная программа начинается с адреса 0. Чему равно восьмеричное содержимое PC и слов с адресами: 0, 2, 4, . . . , 32 после выполнения пяти команд?

```
066737
000000
000004
062737
000004
000000
012767
000002
177770
163777
000006
177766
000137
000000
```

4.10. Написать машинную программу (начиная с адреса 500), которая принимает восьмеричное число n с пульта и n раз включает звонок.

4.11. Написать машинную программу (начиная с адреса 500), которая распечатывает символ, принятый с пульта так, чтобы все строчные буквы стали прописными.

4.12. Байты с 100 по 355 содержат 8-разрядные двоичные числа в дополнительном коде. Написать машинную программу (начиная с адреса 500), которая поместит в регистр R0 наибольшее из них.

4.13. Слова с 2000 по 3776 хранят 1000_8 16-разрядных двоичных чисел в дополнительном коде. Написать машинную программу (начиная с адреса 500), которая поместит в R0 среднее значение этих чисел, отбросив дробную его часть.

Глава 5
ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА

В этой главе мы объясним, что такое язык ассемблера, и приведем некоторые основные сведения об ассемблере MACRO-11. Затем дадим пример полной программы на языке ассемблера и приведем соответствующие указания по кодированию.

5.1. Сравнение языка ассемблера с машинным языком

Для любой, даже самой простой задачи, программирование в двоичных или восьмеричных кодах (т. е. на *машинном языке*) является достаточно обременительным занятием. Основные недостатки этой формы программирования сводятся к следующему.

1. Команды трудно кодировать и интерпретировать. Программист должен помнить коды операций и номера режимов, прежде чем он почувствует себя свободно владеющим машинным языком. Особенно трудно кодировать команды переходов.

2. Модификация программ также чрезвычайно затруднена. Если обнаруживается ошибка, требующая вставки или удаления какой-либо команды, то могут потребоваться пересмотр всей программы в целом и коррекция многих команд, прежде чем можно будет с уверенностью сказать, что модификация выполнена.

Чтобы обойти эти трудности, пользователю, в большинстве случаев, разрешается писать программы на *языке ассемблера,* языке во многом эквивалентном машинному языку, за исключением того, что:

1) коды операций записываются мнемонически, а не численно;

2) адреса могут быть записаны символически, нет необходимости записывать их численно.

Специальная программа, которая транслирует программу на языке ассемблера (или *исходную программу*) в программу на машинном языке (или *объектную программу*), называется *ассемблером* (процесс трансляции называется *ассемблированием*). Система PDP-11 имеет несколько различных ассемблеров. Один из них, который будет использоваться в этом тексте, называется MACRO-11.

Хороший ассемблер должен быть гибким и достаточно мощным средством, чтобы осуществить задачу написания простой и эффективной программы. Ассемблер MACRO-11 обладает рядом особенностей, способствующих этому.

1. Пользователю разрешается записывать константы в форме, отличной от восьмеричной (например, в десятичной или двоичной).

2. Пользователю разрешается снабжать программу комментариями.

3. Ассемблер выдает листинг программы как на языке ассемблера, так и в кодах машины.

4. Ассемблер выдает диагностические сообщения, если программа содержит синтаксические ошибки. Ассемблер допускает условное или повторное ассемблирование и макросы (эти средства более подробно объясняются ниже).

5.2. Директивы языка ассемблера

Если не определено иначе, все числа, появляющиеся в программе на языке ассемблера, ассемблером считаются восьмеричными. Однако, используя определенные преобразования, можно также включить в программу десятичные и двоичные числа, наряду с восьмеричными числами и числами с дополнением до единицы:

n.или ↑Dn	обозначает десятичное число,
↑Bn	обозначает двоичное число,
↑On	обозначает восьмеричное число,
↑Cn	обозначает дополнение числа n до единицы.

Код ASCII для символа p может быть записан как 'p, а код ASCII для пары символов $p_1 p_2$ как $''p_1 p_2$.

Пользователь может управлять операциями ассемблера путем включения *директив* между командами языка ассемблера. Директивы являются командами, адресованными ассемблеру, они обязательны к выполнению в *процессе ассемблирования,* но не во время выполнения программы (*во время прогона*).

Вот, например, наиболее часто используемые директивы:

1. Директивы

.WORD $d_1, d_2, \ldots, d_r$
.BYTE $d_1, d_2, \ldots, d_r$

дают команду ассемблеру запомнить данные (числа или символы) $d_1, d_2, \ldots, d_r$ в последовательных словах или байтах соответственно. Например, последовательность директив

```
.WORD 35,18.,''90
.BYTE ↑B1011,'K
```

осуществляет запоминание в четырех последовательных словах следующих восьмеричных чисел:

000035
000022
030071
045413

2. Директива .ASCII/str/ дает команду ассемблеру запомнить код ASCII строки "str" в последовательных байтах (где / представляет любой символ, кроме < и =, расположенный вне "str"). Символы ASCII, не предназначенные для печати и имеющие код n, могут быть определены как ⟨ n ⟩. Например,

```
.ASCII  /THE ANSWER IS/
.ASCII  /THE BELL WILL NOW RING/<7>
.ASCII  *COMPUTATION OF X/Y* <15> <12>
```

Директива .ASCIZ/str/ осуществляет те же указания, что и .ASCII, но после строки "str" вставляется в качестве завершающего нулевой байт.

3. Директивы

.BLKW n
.BLKB n

дают команду ассемблеру зарезервировать блок памяти на n слов или байтов.

4. Директива .RADIX r дает указание ассемблеру рассматривать все последующие числа с основанием r (r = 2, 4, 8 или 10). Например, директива

.RADIX 10 приводит к тому, что все последующие числа интерпретируются как десятичные (т. е. уже нет необходимости использовать префиксы типа ↑D или суффиксы .), если нет других указаний с помощью соответствующих префиксов ↑B, ↑O или ↑C. Эта директива может быть также отмечена с помощью другой директивы .RADIX.

5. "Директива присваивания" sym = exp является указанием ассемблеру присвоить символическому имени (переменной) "sym" значение выражения "exp". Например, директива KBSTAT = 177560 присваивает значение 177560 символическому имени KBSTAT. Следовательно, каждый раз, когда ассемблер встречает имя KBSTAT, он заменяет его на 177560. Следующая директива KBSTAT = KBSTAT + 2 присваивает символическому имени KBSTAT значение 177562.

Символическому имени (переменной) в программе можно присваивать определенные значения любое число раз. Например, директива I = 1000, помещенная перед директивой I = I + 5, присвоит I значение 1000, однако после второй директивы значение I будет равно 1005.

6. Символ . ("точка") ставится в соответствие той переменной ассемблера (называемой *счетчиком адреса*), которая в любой момент в процессе ассемблирования программы содержит адрес слова, в котором содержится следующая подлежащая ассемблированию команда[1]. Его начальное значение равно 0. По причине, которая будет объяснена ниже, все программы должны начинаться с директивы LC = .

Если после этого вы хотите, чтобы секция подлежащей сборке программы начиналась с адреса α, нужно написать директиву . = α + LC, которая заставит "точку" принять значение α.

7. Следующие директивы .EVEN и .ODD заставят "точку" принять значение следующего четного адреса (если он нечетен) или следующего нечетного адреса (если он четен).

8. Программа обычно начинается с директивы .TITLE name , в соответствии с которой в заголовке листинга программы будет напечатано "имя", указанное в этой директиве.

9. Последним оператором каждой программы должен быть оператор .END sym , где "sym" – символический адрес начальной точки программы. В соответствии с этой директивой "загрузчик" (программа, которая загружает машинную программу в оперативную память) автоматически начнет программу с адреса "sym".

5.3. Формат записи программ на языке ассемблера

Исходная программа составляется в виде последовательности строк, каждая из которых содержит единственное предложение, которое может быть составлено не более чем из четырех записанных в определенном порядке полей, а именно:

метка: оператор операнд; комментарий

---

[1] Точка может быть также использована для определения адресов. Например, чтобы осуществить "переход на три слова вперед", можно написать JMP. + 6; "переслать текущий адрес в SP" – можно записать MOV # ., SP.

Например,

```
LOOP: MOV  R0, @ # 177566   ; ПЕЧАТЬ СИМВОЛА
```

Поля оператора и операнда должны быть разделены по крайней мере одним пробелом или специальным горизонтальным тэбом (контактным знаком), положение в пределах полей – произвольно. Однако так как горизонтальные тэбы телетайпа обычно расставляются в каждой восьмой колонке (начиная с первой), то удобно принять следующий стандартный формат:

Поле	Номер начальной колонки
Метка	1
Оператор	9
Операнд	17
Комментарий	33

Метка, которая присваивает символический адрес оператору, необходима только тогда, когда есть ссылка на имя этого оператора. Символическое имя может быть любой длины, но только первые шесть символов (букв и/или цифр)[1] считаются значащими. Наличие двух или более меток, имеющих одинаковые первые шесть символов, приводит к появлению сообщения об ошибке.

Поле комментария не является обязательным, однако если требуется хорошо документированная программа, то его следует широко использовать. Строка, состоящая целиком из комментариев, может начинаться в любой колонке со знака ; .

Условимся все исходные программы в этом тексте представлять в следующем виде:

```
Col. 1          9       17              33
                .TITLE  Заголовок программы
;               ---
;               ---
                (Описание программы)
;               ---
LC=.
.=4+LC
                .WORD 6,0,12,0          ; УСТАНОВКА ВЕКТОРОВ ОШИБОК
.=500+LC                                ; РЕЗЕРВИРОВАНИЕ ПАМЯТИ ДЛЯ СТЕКА
START:      MOV   PC,SP
            TST   -(SP)                 ; НАЧАЛЬНАЯ УСТАНОВКА SP
                  ---
                  ---
              (Программа)
                  ---
                .END START
```

В соответствии с этим форматом выполнение программы начнется с адреса 500 установкой SP на 500. Пояснение этого момента, а также "векторов ошибок" будет дано в следующих главах.

5.4. Пример: многократное отображение ("мультиэхо")

Пусть мы хотим написать программу, которая "обратно отображает" (распечатывает) каждый введенный с телетайпа n раз символ. Так как обратное

[1] Можно также использовать $ и . , однако начинающим не рекомендуем пользоваться этими символами.

отображение символа не может быть завершено прежде, чем следующий символ не будет введен, программа должна организовать "буфер" (т. е. блок временной памяти) для хранения символов, ожидающих распечатки. Предположим, что максимально возможное число таких символов не превышает 64_{10}.

В программе применен "указатель ввода буфера", в любой момент времени указывающий следующий свободный байт буфера (т. е. байт, в котором должен запоминаться следующий вводимый символ), и "указатель вывода буфера", в любой момент времени указывающий следующий байт буфера, который должен распечатываться. Обычно указатель вывода отстает от указателя ввода; когда первый из них окончательно сравняется с последним, "буфер" становится "пустым" (нет больше символов, которые ожидали распечатки).

Чтобы операция была эффективной, предлагается использовать "кольцевой буфер", в котором за байтом с наибольшим адресом располагается байт с наименьшим адресом. При такой организации требуется память емкостью 64_{10} байт независимо от числа введенных символов. Реализация такого буфера требует, чтобы при достижении указателем буфера значения 63_{10} (77_8) следующее значение сбрасывало бы его показание на нуль. Один из возможных путей осуществить это состоит в том, чтобы снять маску со всех разрядов указателя за исключением шести крайних справа разрядов после каждого увеличения.

На рис. 5.1 показана блок-схема программы, а на рис. 5.2 – листинг исходной и объектной программ, полученный в результате работы ассемблера. (Символические имена KBSTAT, KBDATA и т. д. используются в программе, чтобы сделать ее удобочитаемой.) В этой программе N выбрано равным пяти.

Заметим, что каждый адрес в исходной программе представляется в виде "символическое имя ± константа" (любая из двух частей, составляющих адрес, может отсутствовать) и преобразуется с помощью MACRO-11 в относительный адрес (режим 6). Например, команда MOVB KBDATA, BUFFER(R0) (хранящаяся по адресу 520) преобразуется в

```
116760
177036  (177562 – 524 = 177036).
000602
```

(см. разд. 4.6.)[1]

Помните, что каждый символический адрес, используемый программой, должен быть определен как метка или с помощью директивы присваивания (sym = exp), или с помощью директивы .GLOBL, которая будет рассмотрена в последующих главах. Символическое имя, определенное иначе или определенное более одного раза, приводит к появлению диагностического сообщения об ошибке.

Операция с маской осуществляется в программе с помощью команды BIC [bit clear (ОЧИСТКА РАЗРЯДОВ)]. Команда BIC s, d сбрасывает на нуль все те разряды в d, которые соответствуют единичным разрядам в s, оставляя все остальные разряды неизменными. Таким образом BIC # 177700, R0

[1] Значение апострофа, присутствующего в некоторых относительных адресах, будет пояснено в следующих главах.

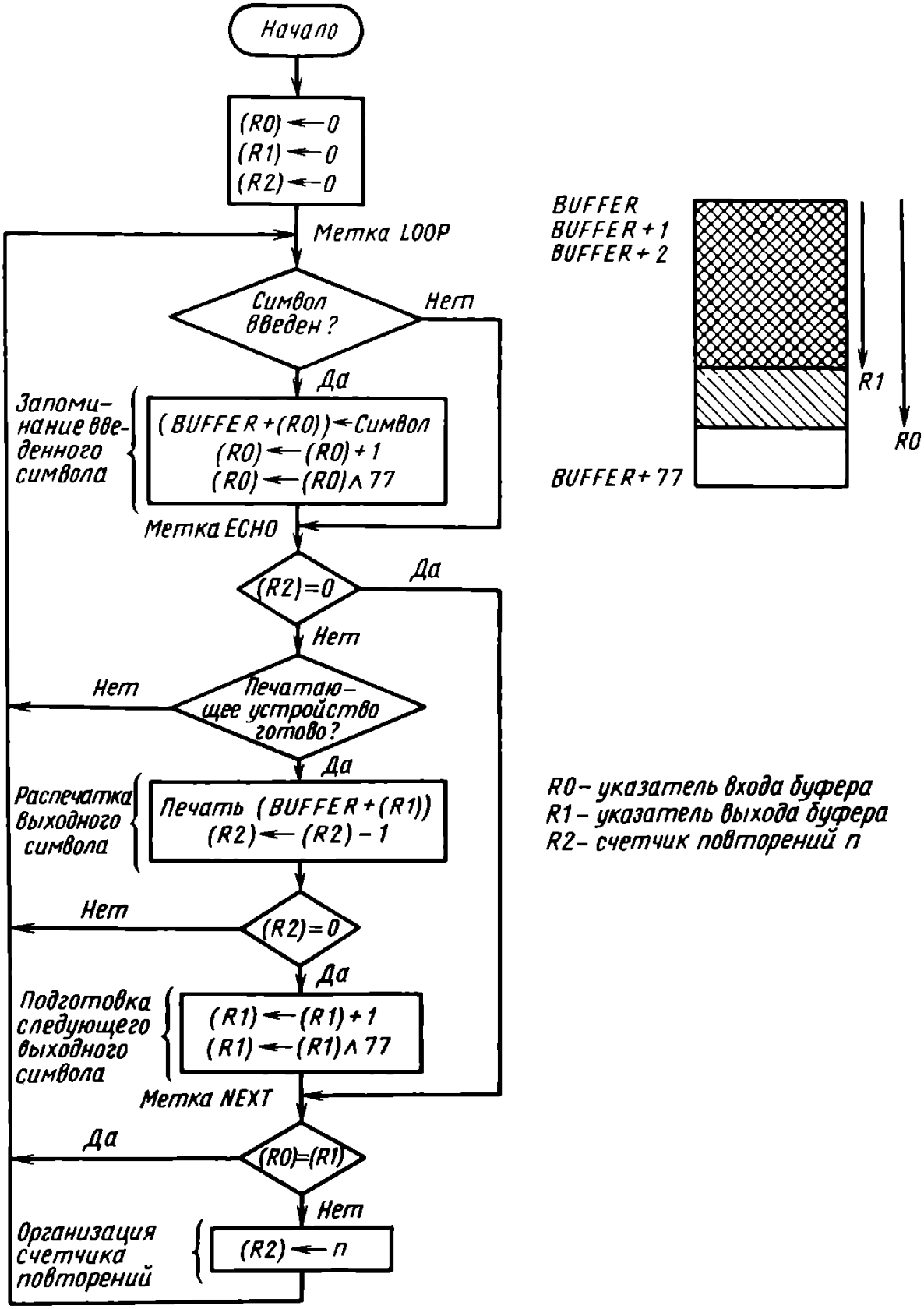


Рис. 5.1. Блок схема программы ("мультиэхо")

```
 1                      .TITLE MULTECHO
 2                      ;ПРОГРАММА МУЛЬТИЭХО РАСПЕЧА-
 3                      ;ТЫВАЕТ ВВЕДЕННЫЙ СИМВОЛ N
 4                      ;РАЗ. ИСПОЛЬЗУЕТ КОЛЬЦЕВОЙ
 5                      ;БУФЕР НА 64(ДЕСЯТИЧНЫХ) БАЙ-
 6                      ;ТА ДЛЯ ХРАНЕНИЯ СИМВОЛОВ.
 7              000000'LC=.
 8              000004'.=4+LC
 9  00004       000006 .WORD 6,0,12,0       ;УСТАНОВКА
                                            ;ВЕКТОРОВ
                                            ;ОШИБОК
10              000000
11              000012
12              000000
13              000500'.=500+LC             ;РЕЗЕРВИРО-
                                            ;ВАНИЕ ПАМЯ-
                                            ;ТИ ДЛЯ СТЕКА
14  00500       010706 START: MOV PC,SP
15  00502       005746        TST -(SP);НАЧАЛЬНАЯ
                                            ;УСТАНОВКА
                                            ;SP
16                      ;
17              177560 KBSTAT=177560
18              177562 KBDATA=177562
19              177564 PRSTAT=177564
20              177566 PRDATA=177566
21                      ;
22  00504       005000        CLR R0        ;УСТАНОВКА
                                            ;УКАЗАТЕЛЯ
                                            ;ВХОДА БУФЕРА
23  00506       005001        CLR R1        ;УСТАНОВКА
                                            ;УКАЗАТЕЛЯ
                                            ;ВЫХОДА БУФЕ-
                                            ;РА
24  00510       005002        CLR R2        ;УСТАНОВКА
                                            ;СЧЕТЧИКА
                                            ;ПОВТОРЕНИЯ
25  00512       105767 LOOP:  TSTB KBSTAT ;СИМВОЛ
                                               ;ВВЕДЕН?
                177042'
26  00516       100006        BPL ECHO    ;ЕСЛИ НЕТ,
                                            ;ПРОДОЛЖИТЬ
                                            ;РАСПЕЧАТКУ
27  00520       116760        MOVB KBDATA,BUFFER(R0)
                                            ;ЕСЛИ ДА,
                                            ;ЗАПОМНИТЬ
                                            ;СИМВОЛ В
                                            ;BUFFER+(R0)
                177036'
                000602'
28  00526       005200        INC R0      ;(R0)=(R0)+1
29  00530       042700        BIC #177700,R0
                                            ;ЗАНЕСТИ 0 В
                                            ;R0,ЕСЛИ >77
                177700
```

Рис. 5.2. Программа "мультиэхо"

```
30 00534 005702 ECHO:  TST R2     ;ЕСЛИ МУЛЬ-
                                  ;ТИЭХО
                                  ;ОКОНЧЕНО,
31 00536 001413        BEQ NEXT   ;ПОДГОТОВКА
                                  ;К СЛЕДУЮЩЕ-
                                  ;МУ ВЫХОДНО-
                                  ;МУ СИМВОЛУ
32 00540 105767        TSTB PRSTAT
                                  ;Т.Е. ЗАП-
                                  ;РОС: УСТ-
                                  ;РОЙСТВО ПЕ-
                                  ;ЧАТИ ГОТОВО
         177020'
33 00544 100362        BPL LOOP   ;ЕСЛИ НЕТ,
                                  ;ПРИНЯТЬ
                                  ;СЛЕДУЮЩИЙ
                                  ;СИМВОЛ
34 00546 116167        MOVB BUFFER(R1),PRDATA
                                  ;ЕСЛИ ДА,
                                  ;РАСПЕЧАТКА
                                  ;ВЫХОДНОГО
                                  ;СИМВОЛА
         000602'
         177012'
35 00554 005302        DEC R2     ;(R2)=(R2)-1
36 00556 001355        BNE LOOP   ;ЕСЛИ (R2) НЕ
                                  ;0, ПРИНЯТЬ
                                  ;СЛЕДУЮЩИЙ
                                  ;СИМВОЛ
37 00560 005201        INC R1     ;ИНАЧЕ (R1)=
                                  ;(R1)+1
38 00562 042701        BIC #177700,R1 ;ЗАНЕСТИ
                                  ;0 В R1, ЕС-
                                  ;ЛИ >77
         177700
39 00566 020001 NEXT:  CMP R0,R1  ;ЕСЛИ (R0)=
                                  ;(R1)(Т.Е.
                                  ;БУФЕР ПУСТ)
                                  ;,
40 00570 001750        BEQ LOOP   ;ПРИНЯТЬ
                                  ;СЛЕДУЮЩИЙ
                                  ;СИМВОЛ
41 00572 016702        MOV N,R2   ;(R2)=(N)
                                  ;(СЧЕТЧИК
                                  ;ПОВТОРЕНИЯ)
         000002
42 00576 000745        BR LOOP    ;ПРИНЯТЬ
                                  ;СЛЕДУЮЩИЙ
                                  ;СИМВОЛ
43                ;
44 00600 000005 N:     .WORD 5    ;СЧЕТЧИК
                                  ;ПОВТОРЕНИЯ
45 00602       BUFFER: .BLKB 4    ;ОБ'ЕМ БУ-
                                  ;ФЕРА: 64
                                  ;(ДЕСЯТИЧ-
                                  ;НЫХ) БАЙТА
46         000500'     .END START
```

Рис. 5.2 (окончание)

очистит все разряды в R0, кроме шести крайних правых. (Команда BIS s, d [bit set (УСТАНОВКА РАЗРЯДОВ)] установит 1 во все те разряды в d, которые соответствуют единичным разрядам в s, оставляя другие разряды неизменными.)

5.5. Указания по кодированию

Полезно указать на некоторые общие ошибки, которые допускают начинающие программисты, изучающие язык ассемблера.

Когда нет уверенности в том, правильно или нет написан тот или иной фрагмент программы, программист должен задать себе вопрос, какие команды были бы в этом случае на машинном языке. Например, рассмотрим оператор

```
MOV R1 + 4, R0
```

При этом мы хотим, чтобы 4 было прибавлено к содержимому R1, а результат запомнен в R0, т. е. (R0) ← (R1) + 4. То, что было записано,формально правильно, но фактически соответствует пересылке содержимого R5 в R0. Правильный вариант решения требует двух команд:

```
MOV  R1, R0
ADD  #4, R0
```

Предположим, что желательно использовать индексный режим с последующим увеличением содержимого регистра. Было бы заманчиво представить это в виде

```
CLRB 1000(R0) +
```

Хотя, возможно, это был бы очень удобный режим адресации, но его не существует. Решение указанной задачи вновь требует двух команд:

```
CLRB 1000(R0)
INC   R0
```

Существует большая разница между

```
MOV  #500, R0
```

и

```
MOV  500, R0
```

Первое означает, что

```
(R0) ← 500
```

тогда как во втором случае

```
(R0) ← (500)
```

Предположим, что нужно очистить ячейки 1000, 1002 и 1004. Правильной будет следующая последовательность команд:

```
MOV  #1000, R0
CLR  (R0) +
CLR  (R0) +
CLR  (R0)
```

Первая команда пересылает адрес ячейки 1000 в регистр R0. Вторая команда очищает ячейку 1000 и добавляет 2 в R0, третья и четвертая команды очищают соответственно ячейки 1002 и 1004.

Предположим, что содержимое ячейки 1000 равно 500, и рассмотрим следующую последовательность команд:

```
MOV  1000, R0
CLR  (R0) +
CLR  (R0) +
CLR  (R0)
```

Первая команда пересылает содержимое ячейки 100 в R0. R0 теперь содержит число 500. Следующие три команды будут проводить очистку ячеек 500, 502, 504. Обратите внимание на разницу между этой и предыдущей последовательностями команд.

Еще одно предостережение. *Никогда* не используйте непосредственной адресации для указания операнда-приемника. Например, едва ли имеет смысл писать

```
CLR  # 500
MOV # 500, # 600
```

(Почему?) Если вы видите знак # в указании операнда-приемника, то почти наверняка вы напишете не то, что хотите.

Все символические адреса, отличные от R0, R1, . . . , R7, SP (который может быть заменен на R6) и PC (который может быть заменен на R7), должны определяться внутри программы. Например, команда CLR PSR будет ошибочной, если ей не предшествует PSR = 177776.

Отметьте разницу между командами

```
SAM:  .WORD 0
```

и

```
CLP   SAM
```

Первая является директивой ассемблеру, предписывающей MACRO-11 установить 0 в ячейке SAM объектной программы. Вторая является командой процессора PDP-11, выполняемой во время прогона программы. В то время как команда .WORD 0 выполняется только раз (в процессе ассемблирования), команда CLR может быть размещена в цикле, и очистка SAM выполняется каждый раз при повторении команд цикла во время прогона программы. (MOV # 0, SAM эквивалентна CLR SAM, но на нее затрачивается больше времени и места.)

Команды сравнения (CMP) и вычитания (SUB) обрабатывают свои операнды в противоположной манере. По команде SUB операнд-источник вычитается из операнда-приемника, тогда как в CMP операнд-приемник вычитается из операнда-источника. Например, SUB, R0, R1 формирует разность (R1) – – (R0), запоминая ее в R1, тогда как CMP R0, R1 формирует разность (R0) – – (R1), причем содержимое регистра не меняется.

Остерегайтесь выполнения небайтовых команд над операндами, которые располагаются по нечетным адресам. Например, не используйте CMP (R0), пока не убедитесь, что R0 всегда будет содержать четный адрес.

Недостаточно написать программу, которая только правильно выполняется. Программа должна быть хорошо организована и соответствующим образом документирована. Более подробно этот вопрос рассмотрен в приложении Д "Заметки о стиле программирования".

Упражнения

5.1. Модифицировать программу "мультиэхо" на рис. 5.2, приспособив ее к буферу емкостью 128_{10} байтов.

5.2. Какими возможными способами можно заставить ассемблер запомнить восьмеричную константу 050520 в одном из слов памяти (например, .WORD 050520).

5.3. Транслировать следующую (бессмысленную) программу с языка ассемблера на машинный язык.

```
LC=.
.=500+LC
JOE=100
SAM=JOE+20
START:  MOV     #SAM,JOE
        JMP     START
        .ASCIZ  $/XYZ/$ <130>
        .BYTE   ^B1,19., 'A
        .RADIX  10
        .BLKB   11
        .EVEN
        .WORD   "12,^O13,^C^B11010,333
        .END    START
```

5.4. Чему равно восьмеричное содержимое восьми регистров общего назначения после окончания работы следующей программы?

```
LC=.
.=500+LC
START:  CLR     R0
        MOVB    SAM,R1
        MOV     #'W,R2
        MOV     PC,R3
        MOV     SAM,R4
        BIC     ANN,R4
        MOV     #JOE,SP
        MOV     ANN,(SP)+
JOE:    CLR     R5
SAM:    .WORD   123456
        HALT
ANN:    .WORD   012705
        .END    START
```

5.5. Что делает следующая программа? Чему равно восьмеричное содержимое R0, R1, R2 после окончания программы?

```
LC=.
.=500+LC
START:  MOV     #A3,R0
        CLR     R2
A1:     MOVB    (R0)+,R1
        BIC     #177600,R1
        CMP     R1,#'0
        BLT     A2
        CMP     R1,#'7
        BGT     A2
        SUB     #'0,R1
        ASL     R2
        ASL     R2
        ASL     R2
        ADD     R1,R2
        BR      A1
A2:     HALT
A3:     .ASCII  '010706'
A4:     .BYTE   15,12
        .END    START
```

5.6. Написать программу на языке ассемблера для деления целого числа, хранящегося в A, на целое число, хранящееся в B (дробная часть отбрасывается).

5.7. Написать программу на языке ассемблера, которая рассматривает слово K состоящим из восьми 2-разрядных "четверть-байтов", и подсчитать число таких "четверть-байтов", которые имеют значение, равное 3 (т. е. 11_2). Результат счета должен остаться в R0.

5.8. Написать программу на языке ассемблера, которая распечатывает восьмеричное содержание слова, адрес которого находится в ADDR. (Например, если ADDR содержит 123, а по адресу 123 находится 456, то программа должна распечатать 000456.)

5.9. Сто (100_{10}) 16-разрядных чисел в дополнительном коде (не обязательно отличающихся друг от друга) хранятся в порядке возрастания модуля в массиве, начинающегося с адреса TABLE. Поместить в R0 число, указывающее, сколько раз в массиве повторено наиболее часто встречающееся число.

5.10. Пятьдесят (50_{10}) чисел в диапазоне $0 - 100_{10}$ хранятся в массиве длиной 50 слов, начиная с адреса GRADE. Содержимое GRADE + i представляет ранг студента с номером i + 1. Написать программу на языке ассемблера, которая формирует массив размером 50 слов, начинающийся с адреса RANK, в котором содержимое ячейки RANK + i является местом студента номер i + 1 в классе указанных пятидесяти. (Место студента равно единице плюс число студентов, чей ранг превышает ранг данного студента.)

5.11. Рассмотрим шахматную доску, горизонтали и вертикали которой пронумерованы от 0 до 7. Написать программу на языке ассемблера, которая принимает входную информацию вида ij ($0 \leqslant i \leqslant 7, 0 \leqslant j \leqslant 7$) и распечатывает все возможные позиции слова, начинающиеся на пересечении i-й горизонтали и j-й вертикали. Например, если на входе принято 42, то выход должен быть таким:

```
  0 1 2 3 4 5 6 7
0             *
1           *
2 *       *
3   *   *
4     *
5   *   *
6 *       *
7           *
```

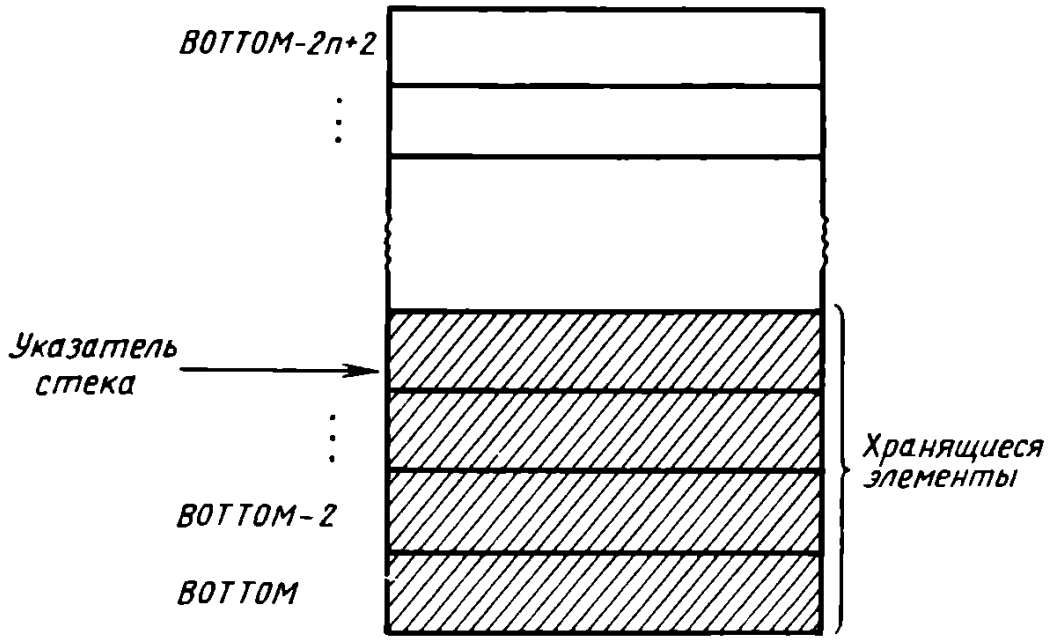
Глава 6
СТЕКИ И ПОДПРОГРАММЫ

В этой главе мы введем понятие стека и рассмотрим, как стеки организуются в PDP-11. Затем мы рассмотрим "системный стек" и его основное применение – организацию связи подпрограммы. В этой главе будут также обсуждены рекурсивные подпрограммы.

6.1. Стеки

Стек – структура данных, организованная так, что выборка данных осуществляется в порядке, обратном тому, в котором они запоминались ("последним введен, первым выведен").

В PDP-11 стек обычно организуется как блок последовательно расположенных слов:



Добавляемые элементы размещаются в стеке, начиная с адреса BOTTOM в порядке уменьшения адресов. В любой момент "*указатель стека*" содержит адрес текущей "*вершины*" стека.

Существуют три основные операции для работы со стеком:

Начальная установка:

(указатель стека) ← BOTTOM + 2

Операция "включение" (размещение содержимого операнда-источника на вершине стека):

(указатель стека) ← (указатель стека) – 2
((указатель стека)) ← (операнд-источник)

Операция "исключение" (удаление вершины стека и пересылка ее содержимого в операнд-приемник):

(операнд-приемник) ← ((указатель стека))
(указатель стека) ← (указатель стека) + 2

Автоинкрементный и автодекрементный режимы адресации делают регистры общего назначения (за исключением PC) идеальными для использования в качестве указателей стека. Если регистр Ri выбирается в качестве указателя стека, то основные операции со стеком можно кодировать очень просто:

Начальная установка: MOV # BOTTOM + 2, Ri
Включение. MOV операнд-источник, – (Ri) [обозначение (↓ (Ri)) ← (операнд-источник)]
Исключение: MOV (Ri) +, операнд-приемник [обозначение (операнд-приемник) ← ((Ri)↑)]

(Стеки с байтами организуются аналогичным способом, за исключением того, что нужно будет использовать MOVB, а не MOV в операциях записи и выборки.)

В большинстве наших программ мы будем размещать стек в заголовке программы, так что BOTTOM + 2 будет являться начальным адресом START

нашей программы. Кроме того, в качестве указателя стека мы будем использовать регистр R6 (SP). Учитывая эти замечания, имеем:

Начальная установка: START: MOV #START, SP
или START: MOV PC, SP
TST – (SP)

Включение: MOV операнд-источник, –(SP)

Исключение: MOV (SP) +, операнд-приемник

При нормальном использовании данные записываются в стек тогда, когда они должны быть сохранены для последующего использования, и выбираются из него, если хранить их нет больше необходимости. В этом случае память для хранения данных выделяется только тогда, когда она действительно необходима, в другое время она может использоваться для других целей.

Объем стека n является оценкой максимального числа элементов данных, которые необходимо поместить в стек в любой заданный момент времени. Любая попытка поместить элемент в стек, когда он полон (т. е. после того, как указатель стека достиг BOTTOM – 2n + 2), приводит к *переполнению стека.* Другая крайность – пытаться извлечь элемент из стека, когда он пуст (т. е. когда указатель стека соответствует BOTTOM + 2). Как переполнение, так и попытки извлечения элементов из пустого стека могут привести к непоправимым последствиям и должны быть предусмотрены в программе.

6.2. Пример: распечатка символов в обратном порядке ("обратное эхо")

Пусть требуется написать программу, которая воспринимает строку символов с телетайпа и затем распечатывает ее в обратном порядке. Например, если на телетайпе набрано

ABC . . . XYZ ↵

(где знак ↵ означает возврат каретки), то в результате распечатки получаем

ZYX . . . CBA ↵

Так как символы распечатываются в обратном порядке, то стек будет наиболее естественной структурой данных для такой программы. В момент поступления символы записываются в стек, пока на телетайпе не будет набрано "возврат каретки". С этого момента стек готов к выборке и символы распечатываются, пока стек не станет пустым. (Для того чтобы подготовить печатающее устройство для печати другой строки, в два нижних слова стека должны быть предварительно занесены коды перевода строки и возврата каретки.)

На рис. 6.1 показана блок-схема программы, а на рис. 6.2 приведен листинг ассемблера этой программы.

6.3. Подпрограммы

Часто оказывается, что однотипные сегменты кодов требуется использовать в различных местах программы; сегменты отличаются только тем, что некоторым ключевым переменным присваиваются различные значения. В этом случае можно значительно облегчить программирование и уменьшить размеры программы, если записать все эти сегменты в виде одной *подпрограммы,* ключевые переменные играют при этом роль *аргументов* подпрограммы (или *параметров*). Такая подпрограмма *вызывается вызыва-*

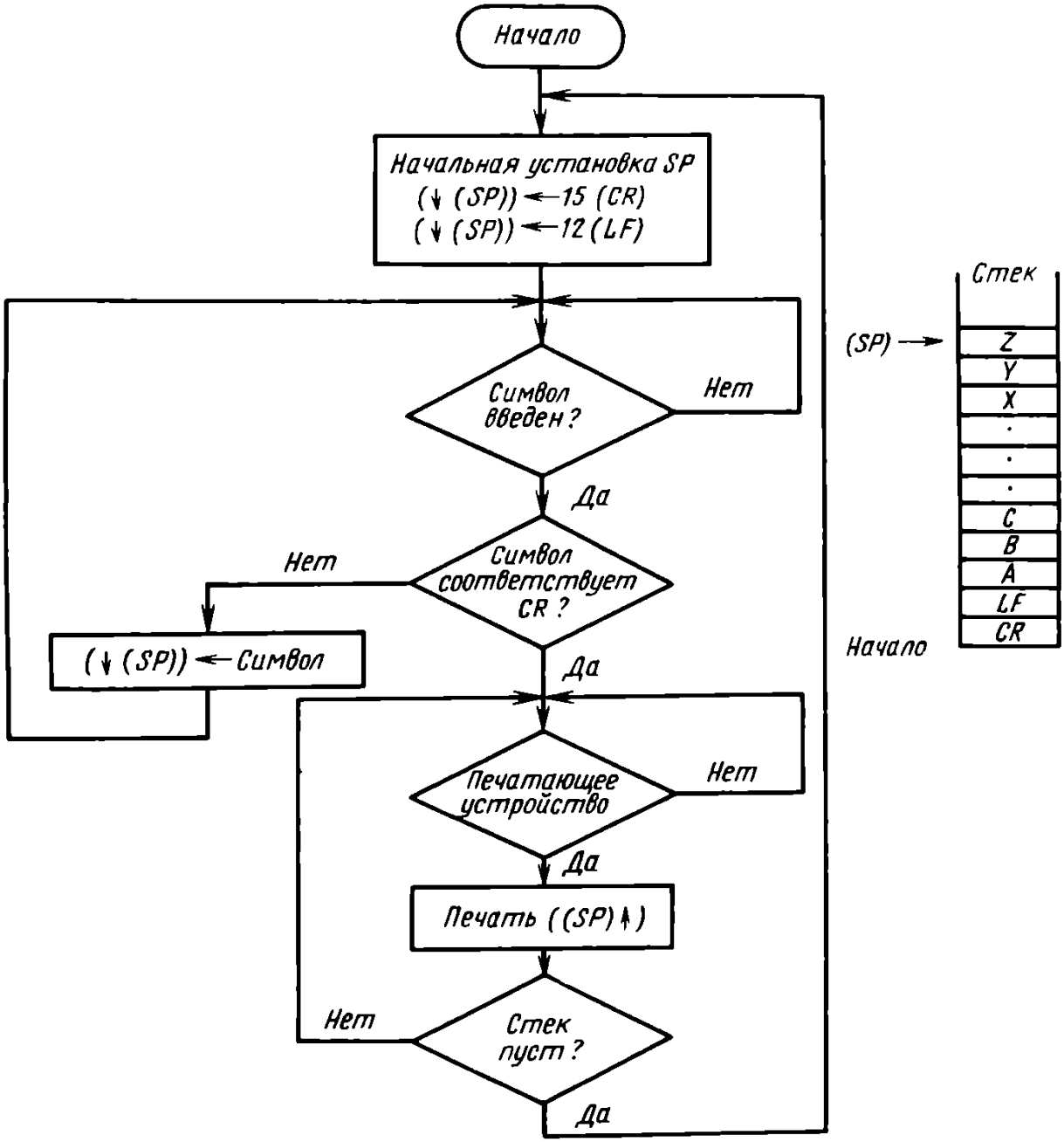


Рис. 6.1. Блок схема программы "обратное эхо"

```
                .TITLE BACKWARD
;ПРИНЯТЬ СТРОКУ СИМВОЛОВ ОТ ТЕЛЕТАЙПА И РАС-
ПЕЧАТАТЬ В ОБРАТНОМ ПОРЯДКЕ
LC=.
.=4+LC
        WORD 6,0,12,0 ;УСТАНОВКА ВЕКТОРА ОШИ-
                      ;БОК
.=500+LC              ;РЕЗЕРВИРОВАНИЕ ПАМЯТИ
                      ;ДЛЯ СТЕКА
```

Рис. 6.2. Программа "обратное эхо"

```
START:  MOV  PC,SP
        TST  -(SP)      ;НАЧАЛЬНАЯ УСТАНОВКА SP
;
KBSTAT=177560
KBDATA=177562
PRSTAT=177564
PRDATA=177566
;
        MOV #15,-(SP) ;ЗАНЕСТИ КОД CR (ВК) В
                      ;СТЕК
        MOV #12,-(SP) ;ЗАНЕСТИ КОД LF (ПС) В
                      ;СТЕК
LOOP:   TSTB KBSTAT   ;СИМВОЛ ВВЕДЕН?
        BPL  LOOP     ;ЕСЛИ НЕТ,ПРОДОЛЖИТЬ
                      ;ОПРОС
        MOV  KBDATA,R0 ;(R0)=СИМВОЛУ
        BIC  #177600,R0 ;ОЧИСТИТЬ ВСЕ,КРОМЕ
                        ;РАЗРЯДОВ КОДА
        CMP  R0,#15   ;ЯВЛЯЕТСЯ ЛИ СИМВОЛ
                      ;СИМВОЛОМ CR?
        BEQ  OUT      ;ЕСЛИ ДА,ТО ИДТИ НА ВЫ-
                      ;ХОД (OUT)
        MOV  R0,-(SP) ;ИНАЧЕ ЗАНЕСТИ СИМВОЛ
                      ;В СТЕК
        BR   LOOP     ;И ВЕРНУТЬСЯ ЗА СЛЕДУ-
                      ;ЮЩИМ СИМВОЛОМ
OUT:    TSTB PRSTAT   ;УСТРОЙСТВО ПЕЧАТИ ГО-
                      ;ТОВО?
        BPL  OUT      ;ЕСЛИ НЕТ, ПРОДОЛЖИТЬ
                      ;ОПРОС
        MOV  (SP)+,PRDATA ;ЕСЛИ ДА, ИЗВЛЕЧЬ
                          ;СЛЕДУЮЩИЙ СИМВОЛ
                          ;И РАСПЕЧАТАТЬ
        CMP  SP,#START    ;СТЕК ПУСТ?
        BEQ  START    ;ЕСЛИ ДА, ПРИНЯТЬ НО-
                      ;ВУЮ СТРОКУ
        BR   OUT      ;ЕСЛИ НЕТ, ВЫХОД НА
                      ;ПЕЧАТЬ
        .END START
```

Рис. 6.2 (окончание)

ющей программой в тех местах программы, где она нужна, и *возвращает управление* вызывающей программе после того, как она была выполнена.

Следует заметить, что даже если уменьшение объема программы само по себе не имеет большого значения, повсеместное использование подпрограммы крайне желательно. Это помогает пользователю писать свои программы методом "сверху вниз" (или в "модульной" форме),т. е. шаг за шагом добавляя необходимые детали. Такой подход к программированию не только проще, но и приводит к программам, которые вполне обозримы, легче отлаживаются и обслуживаются.

Учитывая, что такая подпрограмма может быть использована другими пользователями для их программ, ее структура должна быть тщательно продумана, хорошо документирована и аннотирована. Кроме того, подпрограм-

ма всегда должна быть записана в виде *законченной процедуры* (которая может быть многократно и одновременно используема), т. е. команды подпрограммы не могут модифицироваться во время выполнения. Это обязательное условие, если подпрограмма загружается в память, а затем последовательно вызывается различными независимыми программами (которые предполагают использовать ее в исходной форме).

Вызов подпрограммы влечет за собой два основных действия:

1. *Осуществление связи*: передача подпрограмме адреса в вызывающей программе, к которому она должна вернуться.

2. *Передача аргументов* (*параметров*): сообщение подпрограмме фактических значений ее параметров.

В следующем разделе мы увидим, как эти действия осуществляются в PDP-11.

6.4. Вызов подпрограммы и возврат из нее

Удобный механизм связи обеспечивается в PDP-11 командой JSR ("переход к подпрограмме"). Для вызова подпрограммы SUB (т. е. подпрограммы, которая имеет адрес входа SUB) достаточно записать

```
JSR Ri, SUB
```

где Ri – любой регистр общего назначения[1] (именуемый затем как *регистр связи*). Результат действия этой команды эквивалентен действию следующей последовательности команд, выполняемых в одном цикле:

```
MOV  Ri, –(SP)
MOV  PC, Ri
JMP  SUB
```

Таким образом, JSR сохраняет содержимое Ri его засылкой в стек с указателем SP, после чего Ri используется для хранения содержимого PC (т. е. адреса возврата), и, наконец, происходит переход к подпрограмме SUB. (Так как SP автоматически рассматривается системой как указатель стека, то стек, указанный в SP, называется *системным стеком*.)

Выход из подпрограммы осуществляется по команде RTS ("возврат из подпрограммы"). Если вызывающая программа использовала Ri в качестве регистра связи при вызове подпрограммы, то для возврата из этой подпрограммы в вызывающую программу достаточно написать

```
RTS Ri
```

Действие этой команды аналогично действию следующей последовательности команд, выполняемых в одном цикле:

```
MOV  Ri, PC
MOV  (SP) +, Ri
```

Таким образом, RTS пересылает содержимое Ri (т. е. адрес возврата) в PC, а затем восстанавливает исходное содержимое Ri, считывая его из системного стека.

---

[1] Однако следует избегать использования регистра R6 для этих целей.

Пример

	Машинный язык	Язык ассемблера	Результат действия
Вызывающая программа	1000: 004567	JSP R5, SAM	(↓(SP)) ← (R5), (R5) ←
	1002: 000060		← (PC) = 1004
	1004:	(Точка возврата)	
	.		
	.		
	.		
Подпрограмма	1064:	SAM: (Точка входа)	
	.		
		RTS R5	(PC) ← (R5) = 1004;
			(R5) ← ((SP)↑)

□

В большинстве случаев удобно использовать регистр R7 (PC) в качестве регистра связи. В этом случае вызов осуществляется так:

JSR PC, SUB

что эквивалентно последовательности команд

MOV PC, – (SP)
MOV PC, PC
JMP SUB

Таким образом, JSR просто записывает адрес возврата в системный стек (не изменяя содержимого ни одного из регистров R1 – R5), осуществляя затем вход в подпрограмму SUB. Возврат осуществляется командой

RTS PC

которая эквивалентна последовательности команд

MOV PC, PC
MOV (SP) +, PC

Таким образом, RTS просто извлекает адрес возврата из системного стека (на вершине которого он, предположительно, находился), пересылая его в PC.

Пример

	Машинный язык	Язык ассемблера	Результат действия
Вызывающая программа	1000: 004767	JSR PC, SAM	(↓(SP)) ← (PC) = 1004
	1002: 000060		
	1004:	(Точка возврата)	
	.		
	.		
	.		
Подпрограмма	1064:	SAM: (Точка входа)	
		RTS PC	(PC) ← ((SP)↑) = 1004

□

6.5. Передача аргументов

Существует ряд методов передачи аргументов из вызывающей программы в подпрограмму, каждый из них имеет свои преимущества и недостатки. Со-

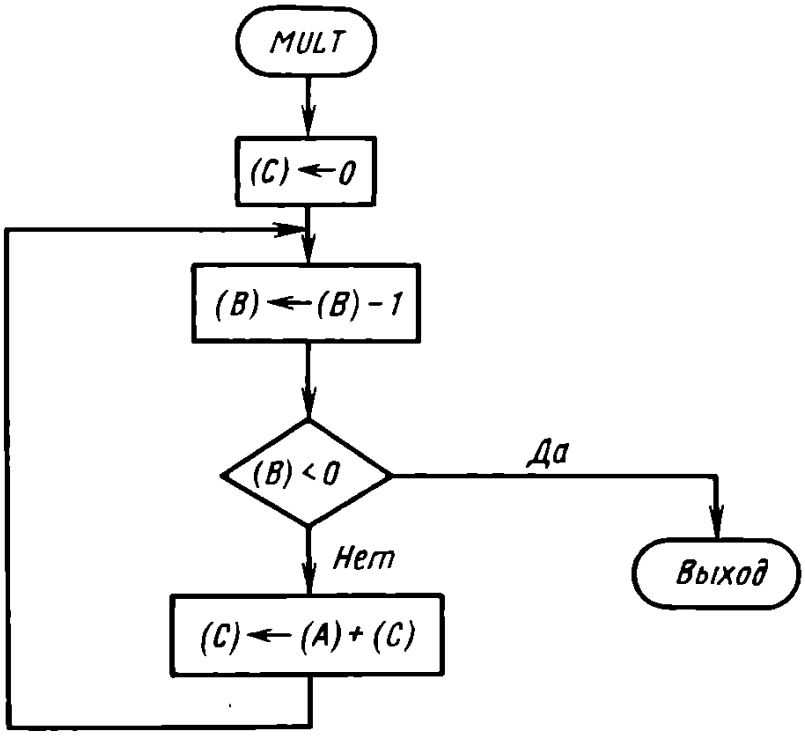


Рис. 6.3. Блок-схема подпрограммы MULT

ответствующий метод, используемый данной подпрограммой, всегда должен быть указан в комментариях к данной подпрограмме.

Фрагменты программ, приведенных на рис. 6.4 — 6.7, иллюстрируют возможные методы передачи аргументов в подпрограмму MULT, которая выполняет следующие операции: (C) ← (A) * (B), где (B) ⩾ 0. Блок-схема подпрограммы MULT показана на рис. 6.3. (Алгоритм умножения, использованный в подпрограмме MULT, нельзя считать эффективным, но он вполне пригоден для данного примера.)

```
        MOV A,R1        ;(R1)=(A)
        MOV B,R2        ;(R2)=(B)
        JSR PC,MULT     ;ВЫЗОВ MULT
        MOV R3,C        ;(C)=(R3)
.
.
.
A:      .BLKW 1
.
.
.
B:      .BLKW 1
.
.
.
C:      .BLKW 1
.
.
.
MULT:   CLR R3          }
LOOP:   DEC R2          }
        BMI EXIT        } (R3)←(R1)*(R2)
        ADD R1,R3       }
        BR LOOP         }
EXIT:   RTS PC          ; ВОЗВРАТ
```

Рис. 6.4. Передача аргументов — метод I

```
        JSR R5,MULT     ; ВЫЗОВ MULT
A:      .BLKW 1
B:      .BLKW 1
C:      .BLKW 1
.
.
.
MULT:   MOV (R5)+,R1    ;(R1)=(A)
        MOV (R5)+,R2    ;(R2)=(B)
        CLR R3          }
LOOP:   DEC R2          }
        BMI EXIT        } (R3)←(R1)*(R2)
        ADD R1,R3       }
        BR LOOP         }
EXIT:   MOV R3,(R5)+    ;(C)=(R3)
        RTS R5          ;ВОЗВРАТ
```

Рис. 6.5. Передача аргументов — метод II

```
        JSR R5,MULT         ;ВЫЗОВ MULT
        .WORD A,B,C         ;АДРЕСА A, B, C ЗАПОМИНАЮТСЯ ЗДЕСЬ
.
.
.
A:      .BLKW 1
.
.
.
B:      .BLKW 1
.
.
.
C:      .BLKW 1
.
.
.
MULT:   MOV @(R5)+,R1       ;(R1)=(A)
        MOV @(R5)+,R2       ;(R2)=(B)
        CLR R3              )
LOOP:   DEC R2              )
        BMI EXIT            )   (R3)←(R1)*(R2)
        ADD R1,R3           )
        BR  LOOP            )
EXIT:   MOV R3,@(R5)+       ;(C)=(R3)
        RTS R5              ;ВОЗВРАТ
```

Рис. 6.6. Передача аргументов – метод III

```
        MOV #ARG,R5         ;(R5)= БАЗОВЫЙ АДРЕС МАССИВА АРГУМЕНТОВ
        JSR PC,MULT         ; ВЫЗОВ MULT
.
.
.
ARG:    .BLKW 3             ; ПАМЯТЬ ДЛЯ МНОЖИМОГО, МНОЖИТЕЛЯ
.                           ;И ПРОИЗВЕДЕНИЯ
.
.
.
.
MULT:   MOV (R5),R1         ;(R1)=(ARG)
        MOV 2(R5),R2        ;(R2)=(ARG+2)
        CLR R3              )
LOOP:   DEC R2              )
        BMI EXIT            )   (R3)←(R1)*(R2)
        ADD R1,R3           )
        BR  LOOP            )
EXIT:   MOV R3,4(R5)        ;(ARG+4)←(R3)
        RTS PC              ; ВОЗВРАТ
```

Рис. 6.7. Передача аргументов – метод IV

Ниже приведены комментарии относительно применения различных методов.

Метод I (рис. 6.4). Перед вызовом аргументы пересылаются в любой из регистров R1 – R5. Возможно, это самый простой метод, но практически он может быть использован только для небольшого числа аргументов.

Метод II (рис. 6.5). Аргументы размещаются сразу за командой вызова. Подпрограмма пересылает указанные аргументы (местоположение которых передается с помощью регистра связи) в любой подходящий регистр или

ячейки оперативной памяти. Основной недостаток метода в том, что аргументы должны храниться в середине программы сразу после команды вызова.

Метод III (рис. 6.6). Аналогичен методу II за исключением того, что адреса аргументов размещаются сразу после команды вызова, тогда как сами аргументы могут находиться где-то в оперативной памяти.

Метод IV (рис. 6.7). Аргументы заносятся в массив, базовый адрес которого передается в подпрограмму через регистры R1 – R5. Метод имеет преимущества при большом списке аргументов, так как подпрограмма не должна проводить распределение памяти для их хранения. Однако вызов аргументов происходит медленнее как следствие использования индексного режима адресации.

6.6. Вложенные подпрограммы

Часто в модульных программах возникает такая ситуация, когда одна подпрограмма вызывает другую подпрограмму, которая в свою очередь вызывает еще одну подпрограмму и т. д. Такие подпрограммы будут называться "вложенными".

Так как подпрограмма может быть связана с некоторым регистром общего назначения, содержащим информацию, существенную для вызывающей ее программы, то вызывающая программа часто обязана сохранить содержимое регистра в памяти до того, как она осуществит вызов. В частности, если вызывающая программа сама является подпрограммой, она должна сохранить информацию связи, переданную ей вызывающей ее программой. Состояние всех регистров должно быть восстановлено вызывающей программой, как только она снова получит управление.

С возрастанием "глубины" вложения растет и количество информации, которая должна быть одновременно сохранена. Требуется тщательный подсчет, чтобы решить куда и когда переслать эту информацию. В PDP-11 эта операция подсчета осуществляется исключительно просто с помощью системного стека и механизма автоматической связи.

Идея состоит в том, чтобы вся информация связи и содержимое используемых регистров сохранялись (и затем восстанавливались) в системном стеке. В этом случае всякий раз, когда вызывающая программа готова восстановить содержимое регистра или же когда подпрограмма готова использовать информацию связи (для того, чтобы осуществить возврат), они могут найти необходимую информацию непосредственнно на вершине стека. В частности, при каждом вызове подпрограммы вызывающая программа должна выполнить следующие шаги:

1. Переслать содержимое необходимых регистров в системный стек (используя команды MOV Ri, –(SP)).

2. Вызвать подпрограмму, используя PC в качестве регистра связи (применяя команды JSR PC, SUB). При этом в системный стек автоматически записывается адрес возврата.

3. Восстановить содержимое необходимых регистров, последовательно считывая его из системного стека (используя команды MOV (SP) +, Ri).

Соответственно из каждой программы следует вернуться, используя команду RTS PC (которая осуществляет выборку адреса возврата из системного стека).

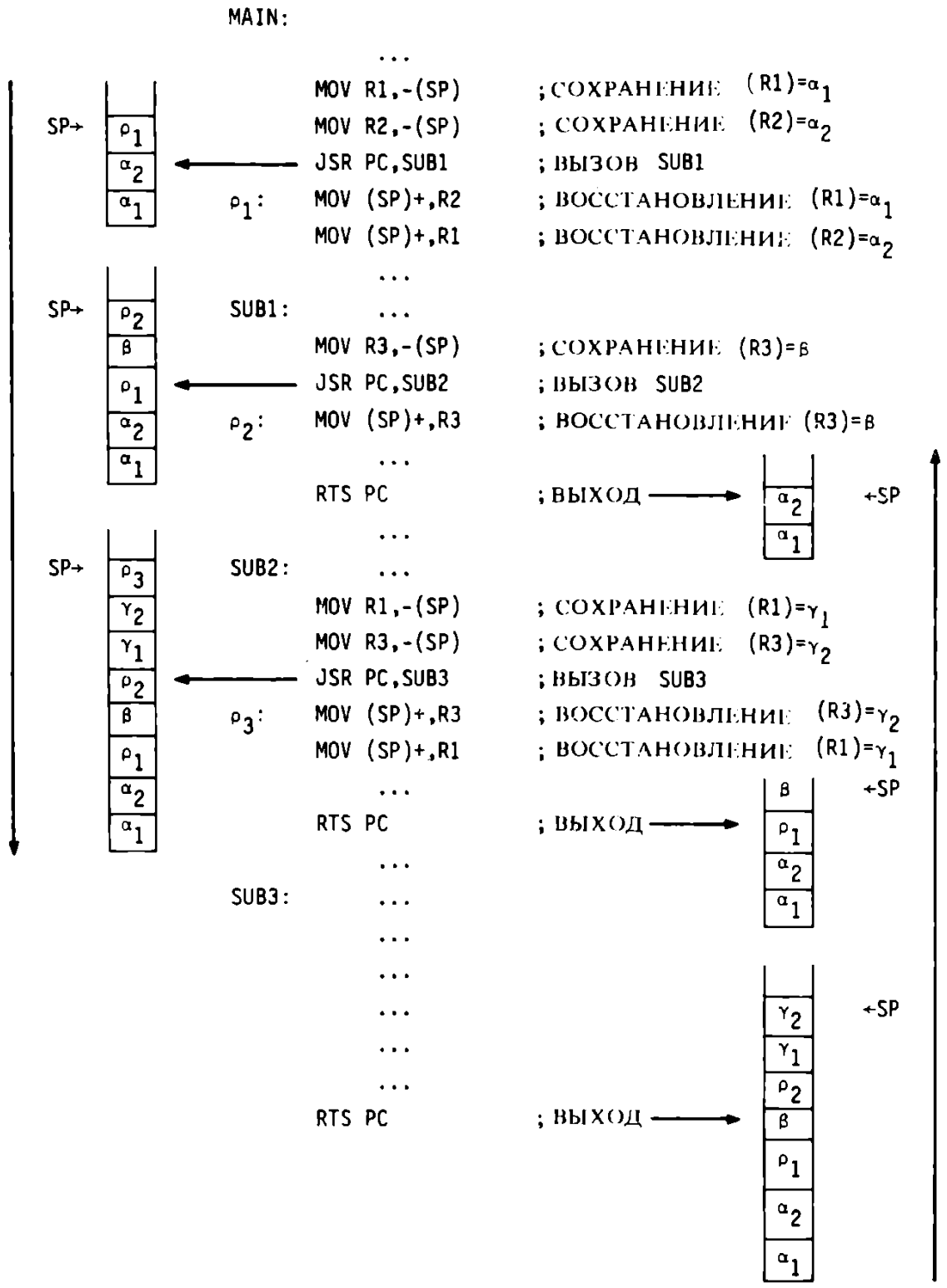


Рис. 6.8. Вложенные подпрограммы

Эта схема показана на рис. 6.8. В левой части рисунка показаны текущие состояния системного стека в моменты, когда программа MAIN вызывает подпрограмму SUB1, подпрограмма SUB1 вызывает SUB2, а SUB2 вызывает SUB3. С правой стороны показано текущее состояние стека в моменты, когда из SUB3 происходит возврат в SUB2, из SUB2 в SUB1 и, наконец, из SUB1 в программу MAIN.

В схеме, приведенной выше, сохранение содержимого используемых регистров выполняется вызывающей программой. В противном случае подпрограмма сама может организовать такую защиту, сохраняя на момент входа содержимое всех регистров, которые должны будут использоваться ею, и восстанавливая их содержимое непосредственно перед моментом выхода из подпрограммы. Удобно этот процесс сохранения осуществить, пересылая содержимое регистров в системный стек (как это и сделано выше в п. 1) и восстанавливая затем то же содержимое путем выборки его из системного стека (как в п. 2).

6.7. Рекурсивные подпрограммы

В ряде случаев проще определить математическую функцию не прямо, а рекурсивно. *Рекурсивное определение* функции $F(n)$ для всех целых $n \geqslant n_0$ состоит из следующих этапов:

1. *Определение базиса*: определить значение $F(n_0), F(n_0 + 1), \ldots, F(n_0 + k)$, используя явные выражения.

2. *Определение последующего шага*: для всех $n > n_0 + k$ определить $F(n)$ через любые из значений $F(n_0), F(n_0 + 1), \ldots, F(n - 1)$.

Примеры

1. Определение факториала $FACT(n)$ для всех $n \geqslant 0$.

   Базис: $FACT(0) = 1$.

   Последующий шаг: $FACT(n) = FACT(n - 1) \cdot n (n > 0)$.

   Например:

$$FACT(4) = FACT(3) \cdot 4 = FACT(2) \cdot 3 \cdot 4 = FACT(1) \cdot 2 \cdot 3 \cdot 4 = \\ = FACT(0) \cdot 1 \cdot 2 \cdot 3 \cdot 4 = 1 \cdot 1 \cdot 2 \cdot 3 \cdot 4 = 24.$$

2. Определение чисел Фибоначчи $FIB(n)$ для всех $n \geqslant 1$.

   Базис: $FIB(1) = 1, FIB(2) = 1$.

   Последующий шаг: $FIB(n) = FIB(n - 2) + FIB(n - 1)\ (n > 2)$.

   Например:

$$FIB(6) = FIB(4) + FIB(5) = FIB(2) + FIB(3) + FIB(3) + FIB(4) = 1 + \\ + FIB(1) + FIB(2) + FIB(1) + FIB(2) + FIB(2) + FIB(3) = 1 + \\ + 1 + 1 + 1 + 1 + 1 + FIB(1) + FIB(2) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 8.$$

3. Определение множества $PERM(n)$ всех перестановок в строке из символов $(a_1, a_2, \ldots, a_n)$ $(n \geqslant 1)$.

   Базис: $PERM(1) = \{(a_1)\}$.

   Последующий шаг: $PERM(n) = a_n \cdot (PERM(n - 1))$ = множеству всех n-кратных наборов, полученных подстановкой a_n во все возможные позиции всех элементов $PERM\ (n - 1)\ (n > 1)$.

   Например:

$$PERM(3) = a_3 \cdot (PERM(2)) = a_3 \cdot (a_2 \cdot (PERM(1))) = a_3 \cdot (a_2 \cdot (\{(a_1)\})) = \\ = a_3 \cdot (\{(a_2, a_1), (a_1, a_2)\}) = \{(a_3, a_2, a_1), (a_2, a_3, a_1), \\ (a_2, a_1, a_3), (a_3, a_1, a_2), (a_1, a_3, a_2), (a_1, a_2, a_3)\}.$$

□

Рекурсивная подпрограмма является подпрограммой, вычисляющей рекурсивную функцию. Она, соответственно, состоит из базиса, который вычисляет непосредственно $F(n)$, и последующего шага, который состоит из подпрограммы, *вызывающей саму себя* с аргументом n, замененным на

"меньший" аргумент, обычно n – 1. В качестве примера рассмотрим не требующую пояснений рекурсивную процедуру (написанную на языке Паскаль), которая вычисляет FACT(n)[1]:

```
FUNCTION FACT(N: INTEGER): INTEGER;
   BEGIN IF N=0 THEN FACT := 1
         ELSE FACT := FACT(N-1)*N
   END;
```

Схема выполнения этой подпрограммы типична для всех рекурсивных подпрограмм: если аргумент n больше 0, она вызывает саму себя с последовательно уменьшающимся значением n, пока n не станет равным 0. В этой точке она обращается сама к себе (умножая каждый раз значение, полученное при возврате, на текущее значение n) с обязательным возвратом к вызывающей программе.

Рекурсия является просто частным случаем вложения подпрограмм, реализация которых на PDP-11 описана в предыдущем разделе.

6.8. Пример: "Ханойская башня"

Классическим примером рекурсивной подпрограммы является подпрограмма, которая решает загадку "Ханойская башня". В этой загадке ряд дисков, уменьшающихся по размеру, нанизывается на стержень A. Они должны быть переставлены на стержень C (причем нанизаны в исходном порядке) при использовании в случае необходимости стержная B для временного хранения (рис. 6.9). В процессе перестановки дисков обязательно должны соблюдаться следующие правила: (1) одновременно может быть переставлен только один диск (с одного из стержней на другой); (2) ни в какой момент времени диск не может находиться на другом диске меньшего размера.

Подпрограмма решения этой загадки носит имя HANOI (N, X, Y, Z). Аргумент N является номером дисков, а аргументы X, Y и Z являются соот-

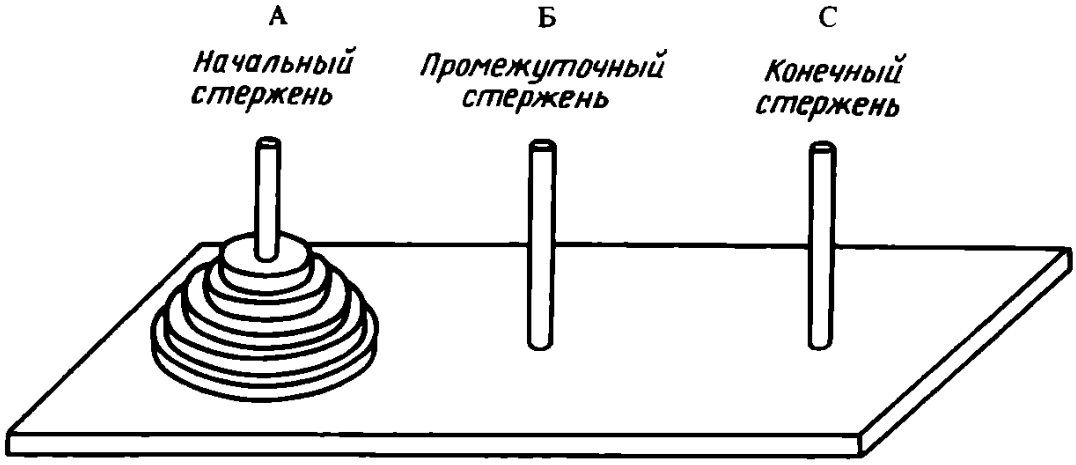


Рис. 6.9. Загадка "Ханойская башня"

[1] Подпрограммы на языке Паскаль или Фортран, результатом работы которых является одно числовое значение, называются функциями (или подпрограммами-функциями).

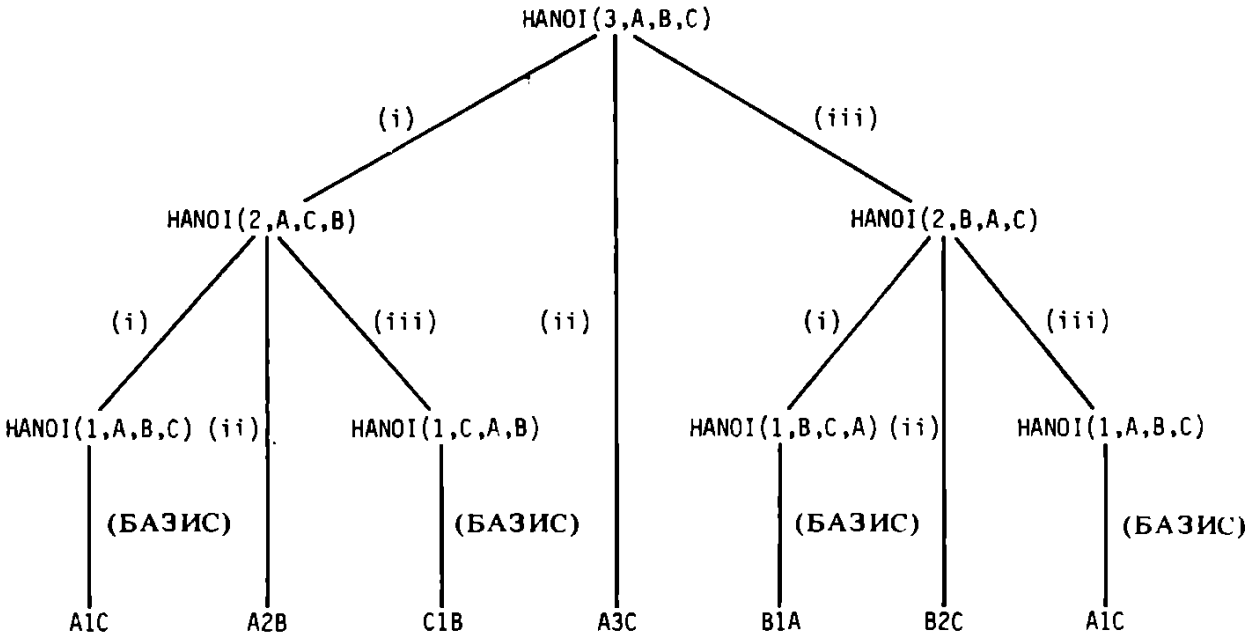


Рис. 6.10. Схема выполнения подпрограммы HANOI (3, A, B, C)

ветственно именами стержней, используемых в качестве начального, промежуточного и конечного. Таким образом, для ситуации, изображенной на рис. 6.9, решение указанной загадки получается вызовом подпрограммы HANOI (5, A, B, C).

Присвоим N дискам номера 1, 2, . . . , N (упорядоченно по возрастанию номеров) и обозначим перемещение диска i со стержня u на стержень v как uiv. Подпрограмма HANOI (N, X, Y, Z), которая распечатывает последовательность перемещений, обозначенных так, как указано выше, может быть описана рекурсивно:

Базис: HANOI (1, X, Y, Z) распечатывает X1Z.

Последующий шаг: HANOI (N, X, Y, Z) (N > 1) делает следующее:

1. Выполняет HANOI (N – 1, X, Z, Y).
2. Распечатывает XNZ.
3. Выполняет HANOI (N – 1, Y, X, Z).

В качестве примера на рис. 6.10 схематически показано выполнение подпрограммы HANOI (3, A, B, C).

На рис. 6.11 приведен листинг программы решения загадки "Ханойская башня". (Для простоты предположено, что N ⩽ 7.) Кроме рекурсивной под-

```
        .TITLE HANOI
;РЕШЕНИЕ ЗАГАДКИ "ХАНОЙСКАЯ БАШНЯ". РАСПЕЧАТКА
;ПОСЛЕДОВАТЕЛЬНОСТИ ПЕРЕМЕЩЕНИЙ N<7 ДИСКОВ.
;A -НАЧАЛЬНЫЙ СТЕРЖЕНЬ, C -КОНЕЧНЫЙ СТЕРЖЕНЬ,
;B -СТЕРЖЕНЬ ДЛЯ ВРЕМЕННОГО ХРАНЕНИЯ.
LC=.
.=4+LC
```

Рис. 6.11. Программа "Ханойская башня"

```
        .WORD   6,0,12,0        ;УСТАНОВКА ВЕКТО-
                                ;РОВ ОШИБОК
.=500+LC                        ;РЕЗЕРВИРОВАНИЕ
                                ;ПАМЯТИ ДЛЯ СТЕКА
START:  MOV     PC,SP
        TST     -(SP)           ;НАЧАЛЬНАЯ УСТА-
                                ;НОВКА SP
;
KBSTAT=177560
KBDATA=177562
PRSTAT=177564
PRDATA=177566
;
;       MAIN PROGRAM
        JSR     PC,INPUT        ;СЧИТЫВАНИЕ N В R5
                                ;( В ФОРМЕ ASCII)
                                ;И ЕГО РАСПЕЧАТКА
        MOV     R5,R0
        BIC     #177770,R0      ;(R0)=N
        JSR     PC,NULINE
        JSR     PC,NULINE       ;ОСТАВИТЬ ДВОЙНОЙ
                                ;ПРОМЕЖУТОК
        TST     R0
        BEQ     OUT             ;ЕСЛИ N=0, ТО ОСТА-
                                ;НОВ
        MOV     #'A,R1          ;X='A
        MOV     #'B,R2          ;Y='B
        MOV     #'C,R3          ;Z='C
        JSR     PC,HANOI        ;ВЫЗОВ ПОДПРОГРАМ-
                                ;МЫ HANOI(N,'A,'B,
                                ;'C)
OUT:&1  HALT
;
;       HANOI(N,X,Y,Z)
;РЕШЕНИЕ ЗАГАДКИ "ХАНОЙСКАЯ БАШНЯ". РАСПЕЧАТКА
;ПОСЛЕДОВАТЕЛЬНОСТИ ПЕРЕМЕЩЕНИЙ N ДИСКОВ С
;НАЧАЛЬНОГО СТЕРЖНЯ X НА КОНЕЧНЫЙ СТЕРЖЕНЬ Z,
;С ПРОМЕЖУТОЧНЫМ СТЕРЖНЕМ Y ДЛЯ ВРЕМЕННОГО
;ХРАНЕНИЯ. АРГУМЕНТЫ: (R0)=N,(R1)=X,(R2)=Y,
;(R3)=Z. ИСПОЛЬЗОВАТЬ R4,R5.
HANOI:  CMP     R0,#1
        BEQ     BASIS           ;БАЗИС, ЕСЛИ N=1
        JSR     PC,SAVE         ;СОХРАНИТЬ N,X,Y,Z
        DEC     R0              ;(R0)=N-1
        MOV     R2,R4
        MOV     R3,R2           ;(R2)=Z
        MOV     R4,R3           ;(R3)=Y
        JSR     PC,HANOI        ;ВЫЗОВ ПОДПРОГРАМ-
                                ;МЫ HANOI(N-1,X,Y,
                                ;Z)
      &2 JSR    PC,RESTOR       ;ВОССТАНОВЛЕНИЕ
                                ;N,X,Y,Z
        JSR     PC,OUTPUT       ;ПЕЧАТЬ XNZ
        JSR     PC,SAVE         ;СОХРАНИТЬ N,X,Y,Z
        DEC     R0              ;(R0)=N-1
```

Рис. 6.11 (продолжение)

```
        MOV     R1,R4
        MOV     R2,R1           ;(R1)=Y
        MOV     R4,R2           ;(R2)=X
        JSR     PC,HANOI        ;ВЫЗОВ ПОДПРОГРАМ-
                                ;МЫ HANOI (N-1,Y,
                                ;X,Z)
     &3 JSR     PC,RESTOR       ;ВОССТАНОВЛЕНИЕ
                                ;N,X,Y,Z
        RTS     PC              ;ВЫХОД
BASIS:  JSR     PC,OUTPUT       ;ПЕЧАТЬ X1Z
        RTS     PC              ;ВЫХОД
;
;       OUTPUT
; РАСПЕЧАТКА X,N,Z (ОЧЕРЕДНОЕ ПЕРЕМЕЩЕНИЕ
; ДИСКА). РЕГИСТРЫ НЕ МЕНЯЮТСЯ.
OUTPUT: MOV     R1,R5
        JSR     PC,PRINT        ;ПЕЧАТЬ X
        MOV     R0,R5
        ADD     #60,R5          ;ПРЕОБРАЗОВАНИЕ
                                ;N В КОД ASCII
        JSR     PC,PRINT        ;ПЕЧАТЬ N
        MOV     R3,R5
        JSR     PC,PRINT        ;ПЕЧАТЬ Z
        JSR     PC,NULINE       ;ПРОПУСК СТРОКИ
        RTS     PC              ;ВЫХОД
;
;       SAVE
;ЗАНЕСТИ СОДЕРЖИМОЕ R0,R1,R2,R3 В СИСТЕМ-
;НЫЙ СТЕК. РЕГИСТРЫ НЕ МЕНЯЮТСЯ.
SAVE:   MOV   (SP)+,R4          ;СОХРАНЕНИЕ АД-
                                ;РЕСА ВОЗВРАТА
        MOV     R0,-(SP)        ;ЗАНЕСТИ (R0) В
                                ;СТЕК
        MOV     R1,-(SP)        ;ЗАНЕСТИ (R1) В
                                ;СТЕК
        MOV     R2,-(SP)        ;ЗАНЕСТИ (R2) В
                                ;СТЕК
        MOV     R3,-(SP)        ;ЗАНЕСТИ (R3) В
                                ;СТЕК
        MOV     R4,-(SP)        ;ВОССТАНОВЛЕНИЕ
                                ;АДРЕСА ВОЗВРАТА
        RTS     PC              ;ВЫХОД
;
;       RESTOR
;ВЫБРАТЬ СОДЕРЖИМОЕ R3,R2,R1,R0 ИЗ СИСТЕМНО-
;ГО СТЕКА. R4 И R5 НЕ МЕНЯЮТСЯ.
RESTOR: MOV     (SP)+,R4        ;СОХРАНЕНИЕ АДРЕ-
                                ;СА ВОЗВРАТА
        MOV     (SP)+,R3        ;ВЫБОРКА (R3) ИЗ
                                ;СТЕКА
        MOV     (SP)+,R2        ;ВЫБОРКА (R2) ИЗ
                                ;СТЕКА
        MOV     (SP)+,R1        ;ВЫБОРКА (R1) ИЗ
                                ;СТЕКА
```

Рис. 6.11 (продолжение)

```
        MOV     (SP)+,R0        ;ВЫБОРКА (R0) ИЗ
                                ;СТЕКА
        MOV     R4,-(SP)        ;ВОССТАНОВЛЕНИЕ
                                ;АДРЕСА ВОЗВРАТА
        RTS     PC              ;ВЫХОД
;
;       PRINT
;ПЕЧАТЬ СОДЕРЖИМОГО R5. РЕГИСТРЫ НЕ МЕНЯЮТСЯ.
PRINT:  TSTB    PRSTAT          ;УСТРОЙСТВО ПЕЧАТИ
                                ;ГОТОВО?
        BPL     PRINT           ;ЕСЛИ НЕТ, ЖДАТЬ
        MOV     R5,PRDATA       ;ЕСЛИ ДА, ПЕ-
                                ;ЧАТЬ (R5)
        RTS     PC              ;ВЫХОД
;       NULINE
;ПРОПУСК СТРОКИ. ИСПОЛЬЗОВАТЬ R5.
NULINE: MOV     #12,R5
        JSR     PC,PRINT        ;ПЕЧАТЬ LF
        MOV     #15,R5
        JSR     PC,PRINT        ;ПЕЧАТЬ CR
        RTS     PC              ;ВЫХОД
;
;       INPUT
;ЗАПОМИНАНИЕ ВВЕДЕННОГО СИМВОЛА В R5 И РАСПЕ-
;ЧАТКА ЕГО. ДРУГИЕ РЕГИСТРЫ НЕИЗМЕННЫ.
INPUT:  TSTB    KBSTAT          ;СИМВОЛ ВВЕДЕН?
        BPL     INPUT           ;ЕСЛИ НЕТ, ЖДАТЬ
        MOV     KBDATA,R5       ;(R5)=СИМВОЛУ
        JSR     PC,PRINT        ;ПЕЧАТЬ СИМВОЛА
        RTS     PC              ;ВЫХОД

        .END    START
```

Рис. 6.11 (окончание)

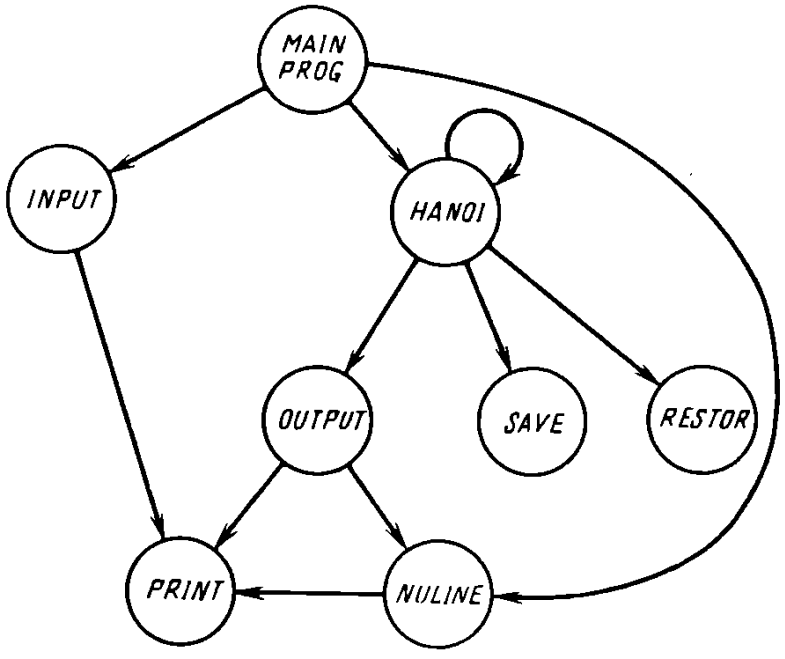


Рис. 6.12. Структура связи подпрограмм в программе "Ханойская башня"

```
α1
α1 3 A B C α2
α1 3 A B C α2 2 A C B α2
α1 3 A B C α2 2 A C B α2 1 A B C α2          (Печать A1C)
α1 3 A B C α2 2 A C B α2                     (Печать A2B)
α1 3 A B C α2 2 A C B α2 1 C A B α3          (Печать C1B)
α1 3 A B C α2 2 A C B α2
α1 3 A B C α2                                (Печать A3C)
α1 3 A B C α2 2 B A C α3
α1 3 A B C α2 2 B A C α3 1 B C A α2          (Печать B1A)
α1 3 A B C α2 2 B A C α3                     (Печать B2C)
α1 3 A B C α2                                (Печать A1C)
α1
```

Рис. 6.13. Состояния стека для подпрограммы HANOI (3, A, B, C)

программы HANOI, описанной выше, программа содержит подпрограммы INPUT, OUTPUT, PRINT, NULINE, SAVE и RESTOR. На рис. 6.12 приведена структура связи указанных подпрограмм. Эту структуру, которая на первый взгляд кажется достаточно сложной, можно легко осуществить в PDP-11, используя схему, описанную в разд. 6.6.

На рис. 6.13 показаны состояния системного стека (основание стека расположено слева, вершина стека – справа) так, как они выглядят в процессе выполнения подпрограммы HANOI (3, A, B, C). (Соответствующие изменения стека, происходящие в процессе выполнения подпрограммы OUTPUT, не показаны.) (Сравните рис. 6.13 и рис. 6.10.)

Следует отметить, что подпрограммы SAVE и RESTOR осуществляют извлечение адреса возврата из системного стека в момент входа в них и пересылают его обратно в системный стек непосредственно перед выходом из них. Эти операции необходимы потому, что процедуры "сохранения" и "восстановления", осуществляемые подпрограммами SAVE и RESTOR, могли бы в противном случае удалить адрес возврата из вершины стека.

6.9. Сопрограммы

Иногда возникает такая ситуация, что две подпрограммы вызывают друг друга попеременно так, что в соответствующий момент переход осуществляется не в начало другой подпрограммы, а в точку, где другая подпрограмма была последний раз прервана (рис. 6.14). В этом случае (когда каждая подпрограмма играет роль как вызывающей программы, так и подпрограммы) такие подпрограммы называются *сопрограммами*.

Связи между сопрограммами можно очень просто осуществить в PDP-11. Предположим, что β_1 – вершина системного стека в тот момент, когда начинает выполняться подпрограмма # 1. Когда # 1 достигнет $\alpha_1 - 2$ и готова передать управление подпрограмме # 2 (при β_1), осуществляется команда

```
JSR PC, @ (SP) +
```

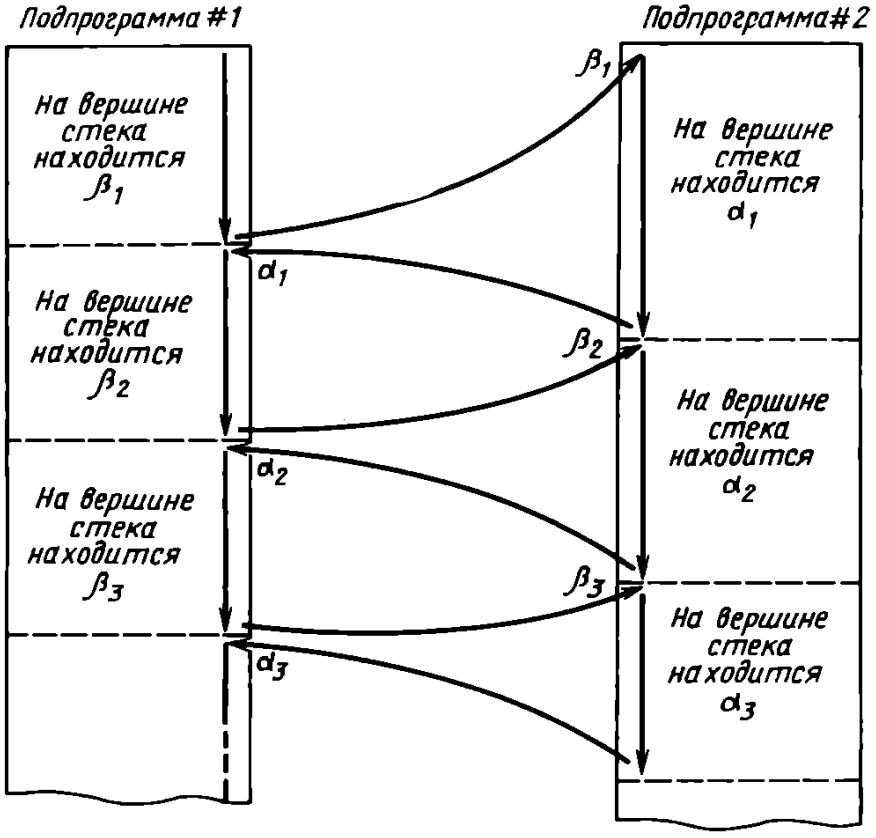


Рис. 6.14. Сопрограммы

которая делает следующее:

1. Осуществляет выборку вершины системного стека (т. е. β_1) и пересылку содержимого в ячейку вре́менной памяти.
2. Осуществляет запись PC (т. е. α_1) в системный стек.
3. Копирует содержимое вре́менной ячейки в PC.

Таким образом, α_1 теперь находится на вершине системного стека, и подпрограмма # 2 начинает выполнение с адреса β_1. Когда подпрограмма # 2 достигнет $\beta_2 - 2$ и готова передать управление подпрограмме # 1 (по адресу α_1), выдается та же команда JSR PC, @ (SP) +. Адрес возврата β_2 заменит α_1 на вершине стека, и выполнение начнется с α_1. И так далее.

Упражнения

6.1. Что вычисляет следующий сегмент программы?

```
        CLR   R0
        JSR   PC,SUB
        HALT
SUB:    DEC   R2
        BMI   EXIT
        ADD   R1,R0
        JSR   PC,SUB
EXIT:   RTS   PC
```

6.2. Проследить выполнение следующей программы, указывая после каждого выполнения команды восьмеричное содержимое R3,R4, SP и системного стека:

```
LC=.
.=500+LC
START:  MOV     PC,SP
        TST     -(SP)
        MOV     #123,R3
        JSR     R3,GAME
        .WORD   5
        HALT
GAME:   MOV     (R3)+,R4
        DEC     R4
        BEQ     EXIT
        JSR     R3,GAME
        .WORD   1
EXIT:   RTS     R3
        .END
```

6.3. Рассмотреть следующую программу:

```
LC=.
.=500+LC
START:  MOV     #4567,R0
        JSR     PC,BTOA
        HALT
;       ПОДПРОГРАММА BTOA
;BTOA ПРЕОБРАЗУЕТ ВОСЬМЕРИЧНОЕ СОДЕРЖИМОЕ
;R0 В КОД ASCII И РАСПЕЧАТЫВАЕТ ЕГО.
BTOA:   CMP     #7,R0           ;N МЕНЬШЕ ЛИБО
                                ;РАВНО 7?
        BHIS    B1              ;ЕСЛИ ДА, ТО ПРЕОБ-
                                ;РАЗОВАТЬ И ПЕЧАТАТЬ
        MOV     R0,-(SP)        ;ЕСЛИ НЕТ,   СОХРА-
                                ;НИТЬ N
        BIC     #177770,(SP);ОЧИСТКА ВСЕХ, КРО-
                                ;МЕ ТРЕХ ПОСЛЕДНИХ,
                                ;РАЗРЯДОВ СОХРАНЕН-
                                ;НОГО N
        CLC                     ;ПОДГОТОВИТЬ К СДВИ-
                                ;ГУ N ВПРАВО
        ROR     R0              ;СДВИГ N
        ASR     R0              ;  ВПРАВО
        ASR     R0              ;    НА ТРИ РАЗРЯДА
        JSR     PC,BTOA         ;ПРЕОБРАЗОВАТЬ ТО,
                                ;ЧТО СЛЕВА
        MOV     (SP)+,R0        ;ВЕРНУТЬСЯ К ЗАПОМ-
                                ;НЕННОМУ N
B1:     ADD     #'0,R0          ;ПРЕОБРАЗОВАТЬ В
                                ;КОД ASCII
LOOP:   TSTB    177564          ;УСТРОЙСТВО ПЕЧАТИ
                                ;ГОТОВО?
        BPL     LOOP            ;ЕСЛИ НЕТ, ПОВТО-
                                ;РИТЬ ЗАПРОС
        MOV     R0,177566       ;ЕСЛИ ДА, НАПЕЧАТАТЬ
                                ;СИМВОЛ
        RTS     PC              ;И ВОЗВРАТ
        .END    START
```

Показать содержимое системного стека в момент его максимального заполнения.

6.4. Рассмотреть следующую программу:

```
LC=.
.=500+LC
START:      MOV     PC,SP
            TST     -(SP)
            MOV     SIZE,R2
            MOV     #ARRAY,R1
            MOV     (R1)+,R0
            JSR     PC,COMPR
            HALT
SIZE:       .WORD   6
ARRAY:      .WORD   137,10,205,647,53,122
;           ПОДПРОГРАММА COMPR
COMPR:      DEC     R2
            BEQ     RETURN
            CMP     (R1)+,R0
            BLT     REPEAT
            MOV     -2(R1),R0
REPEAT:     JSR     PC,COMPR
RETURN:     RTS     PC
            .END    START
```

(а) Чему равно окончательное восьмеричное содержимое R0?

(б) Показать восьмеричное содержимое системного стека, когда он максимально заполнен.

(в) Кратко описать, что делает указанная программа (при произвольно взятых числах в ARRAY).

6.5. Написать подпрограмму DUMPREG, которая распечатывает восьмеричное содержимое всех регистров общего назначения и PSR в порядке их появления перед входом в подпрограмму. Перед выходом из подпрограммы DUMPREG исходное содержимое всех этих регистров должно быть восстановлено. DUMPREG может вызывать другие подпрограммы, такие как CONVERT (для преобразования восьмеричного содержимого в коды ASCII) и PRINT (для распечатки). (Вы можете использовать DUMPREG для отладки.)

6.6. Написать подпрограмму, которая имеет следующие параметры: первый адрес ADDR N-байтного массива, номер N и символ CHAR. Подпрограмма заносит 1 в R0, если CHAR находится в любом из N байтов, и 0 в другом случае. Написать основную программу (используя вышеприведенную подпрограмму), которая принимает строку символов с телетайпа и распечатывает (восьмеричные) коды этих символов, найденных в массиве.

6.7. Написать рекурсивную подпрограмму, которая складывает все 16-разрядные двоичные числа в дополнительном коде, хранящиеся в массиве из N слов, начинающемся с ячейки X. Предположите, что X хранится в R0, N – в R1, а сумма – в R2. (Это можно легко осуществить и без рекурсии, однако постарайтесь использовать ее для этих целей.)

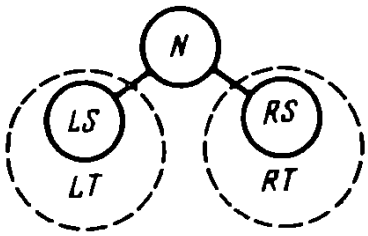
6.8. Написать рекурсивную подпрограмму, которая находит положительное число N в R0 и оставляет результат FACT(N) в R1.

6.9. Написать рекурсивную подпрограмму, которая находит целое $N \geqslant 1$ в R0 и оставляет результат FIB(N) в R1.

6.10. *Двоичное дерево* T с *корнем* N является графом, который рекурсивно можно определить так:

Базис: если T имеет единственный узел N, то T – двоичное дерево с корнем N.

Последующий шаг: (См. рис. справа.) Если T состоит из узла N, соединенного с узлами LS (*левый элемент-приемник*) и RS (*правый элемент-приемник*), и если LS и RS – корни двоичных деревьев (обозначенных LT и RT соответственно), то T – двоичное дерево с корнем N.



Как пример на рис. 6.15, *а* показано двоичное дерево, корень которого помечен цифрой 0, а узлы помечены восьмеричными числами 0, 1, 2, . . . , 14.

В PDP-11 дерево с k узлами может быть представлено массивом из k слов с базовым адресом TREE. Номер узла N хранится в ячейке TREE + 2. Старший байт TREE + 2N содержит номер LS левого узла преемника N, а младший байт – номер RS правого узла преемника N.

На рис. 6.15, *б* показано такое представление для дерева на рис. 6.15, *а*.

(а) Прохождение двоичного дерева T *во внутреннем порядке* определяется рекурсивно следующим образом:

Базис: Если T состоит из единственного узла N, посетить узел N.

Последующий шаг: Если T состоит из узла N, соединенного с двоичными деревьями LT и RT, то: а) осуществить прохождение LT во внутреннем порядке; б) посетить узел N; в) осуществить прохождение RT во внутреннем порядке.

Написать рекурсивную подпрограмму, которая распечатывает номера узлов данного двоичного дерева в порядке их появления при прохождении во внутреннем порядке.

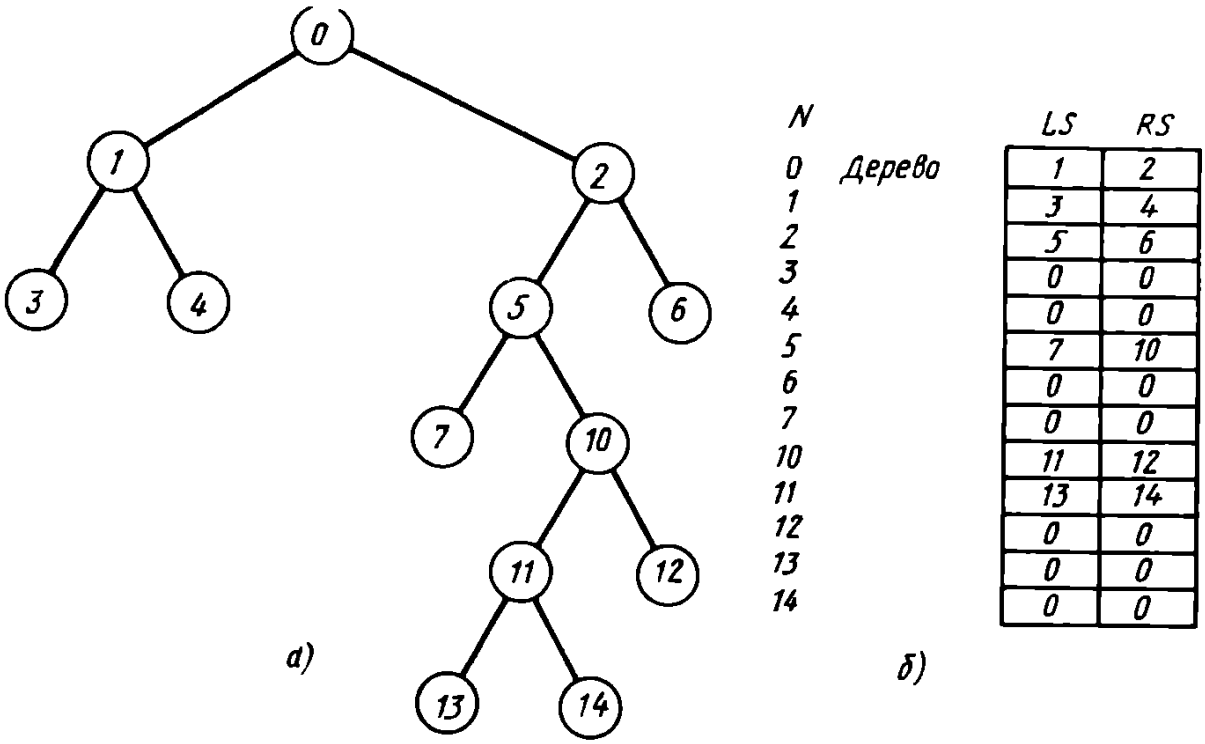


Рис. 6.15. Двоичное дерево и его представление

(Например, для дерева, изображенного на рис. 6.15, распечатка должна иметь следующий вид: 3, 1, 4, 0, 7, 5, 13, 11, 14, 10, 12, 2, 6.)

(б) *Высота* двоичного дерева T, обозначаемая как H(T), может быть рекурсивно определена следующим образом:

Базис: Если T состоит из единственного узла N, то H(T) = 0.

Последующий шаг: Если T состоит из узла N, соединенного с двоичным деревом LT и деревом RT, то

$$H(T) = 1 + \text{maximum}(H(LT), H(RT)).$$

Написать рекурсивную процедуру, которая распечатывает высоту данного двоичного дерева. (Например, для дерева на рис. 6.15 результат должен быть равен 5.)

Глава 7
АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

В этой главе мы подробно опишем, как арифметические операции влияют на коды условий в PSR. В частности, мы рассмотрим, как ведут себя коды V и C и как их можно использовать для индикации переполнения и в арифметике с двойной точностью. Мы познакомимся более подробно с командами TST и CMP, используемыми в командах перехода и сдвига. В качестве примера в этой главе будет рассмотрена программа преобразования кодов ASCII в двоичные коды, как иллюстрация некоторых моментов, рассмотренных выше.

7.1. Перенос и переполнение при сложении

Как уже отмечалось в разд. 3.2, результат выполнения команды сложения оказывает влияние на коды условий C (*Carry*) и V (*oVerflow*) в PSR, а именно:

$$C = \begin{cases} 1, \text{ если возникает перенос от MSB,} \\ 0 \text{ в остальных случаях,} \end{cases}$$

$$V = \begin{cases} 1, \text{ если операнды одного знака, а их сумма имеет противоположный знак,} \\ 0 \text{ в остальных случаях.} \end{cases}$$

Примеры

Предположим для простоты, что PDP-11 имеет четырехразрядные слова, позволяющие оперировать целыми числами в диапазоне от −8 до +7.

Пример (десятичное представление)	Вычитание двоичных чисел	Значения, принимаемые C	V	Результат
1	0001			
+ (+2)	+ 0010	0	0	Верный
3	0011			
5	0101			
+ (+6)	+ 0110	0	1	Неверный
11	1011			

Продолжение

Пример (десятичное представление)	Вычитание двоичных чисел	Значения, принимаемые C	V	Результат
-6	1010			
+ (+7)	+ 0111	1	0	**Верный**
1	↙0001			
	1			
-6	1010			
+ (+3)	+ 0011	0	0	**Верный**
-3	1101			
-5	1011			
+ (-2)	+ 1110	1	0	**Верный**
-7	↙1001			
	1			
-6	1010			
+ (-6)	+ 1010	1	1	**Неверный**
-12	↙0100			
	1			

□

Отсюда, в общем случае, можно сделать следующие выводы.

1. Если оба числа положительные, C всегда равно 0 (так как оба MSB равны 0). Сумма неверна, если только она отрицательна (V = 1).

2. Если оба числа отрицательны, C всегда равно 1 (так как оба MSB равны 1). Сумма неверна, если только она положительна (V = 1).

3. Если числа противоположных знаков, V всегда равно 0, а сумма всегда верна. C может быть либо 0, либо 1.

В заключение можно сделать вывод, что результат сложения двух чисел неверен только тогда, когда V = 1.

7.2. Перенос и переполнение при вычитании

Результат выполнения операции вычитания (осуществляемой при выполнении команд SUB и CMP) влияет на коды условий C и V, а именно:

$$C = \begin{cases} 1, \text{ если в результате вычитания нет переноса от MSB,} \\ 0 \text{ в других случаях,} \end{cases}$$

$$V = \begin{cases} 1, \text{ если операнды противоположного знака, а разность} \\ \quad \text{имеет тот же знак, что и вычитаемое}^1, \\ 0 \text{ в остальных случаях.} \end{cases}$$

Примеры

Вновь предположим, что используются четырехразрядные числа.

---

[1] В SUB вычитаемое число является *операндом-источником*; в CMP оно является *операндом-приемником*.

Пример (десятичное представление)	Вычитание двоичных чисел	Значения, принимаемые C	V	Результат
6	0110			
-(+3)	+1101	0	0	Верный
3	0011			
	1			
4	0100			
-(-2)	+0010	1	0	Верный
6	0110			
4	0100			
-(-5)	+0101	1	1	Неверный
9	1001			
5	0101			
-(+7)	+1001	1	0	Верный
-2	1110			
-3	1101			
-(+2)	+1110	0	0	Верный
-5	1011			
	1			
-2	1110			
-(-5)	+0101	0	0	Верный
3	0011			
	1			
-4	1100			
-(+6)	+1010	0	1	Неверный
-10	0110			
	1			

□

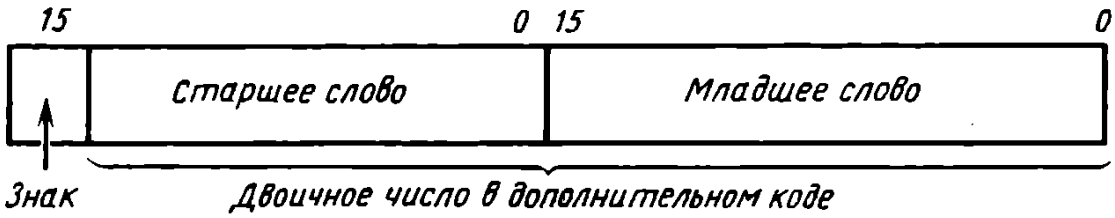
Анализируя эти примеры, можно сделать вывод, что результат выполнения операции вычитания будет неверным, если только V = 1.

Операция смены знака (NEG), если она применяется к ненулевому числу, приводит к C = 1, если же она применяется к нулевому числу, то получаем C = 0. Заметим, что операция смены знака числа, которая состоит в формировании обратного кода и прибавлении 1, формирует перенос только тогда, когда число равно 0. Таким образом, операция смены знака приводит к C = = 1, если только нет переноса, и приводит к C = 0, если перенос есть.

7.3. Арифметика вычислений с двойной точностью

Для многих применений диапазон представления чисел, соответствующих длине одного машинного слова, недостаточен, поэтому для запоминания таких чисел используются два последовательных слова. В PDP-11 двойное слово, или число с *двойной точностью,* хранится в виде двоичного 32-разрядного

числа в дополнительном коде. *Младшее* слово содержит самые младшие разряды числа, а *старшее* слово содержит самые старшие разряды числа. Старший значащий разряд MSB старшего слова является знаковым разрядом числа в целом:



При использовании представления с двойной точностью PDP-11 может обрабатывать числа вплоть до 2147483647_{10}, тогда как при одинарной точности – только 32767_{10}.

Как же сложить два числа, представленных с двойной точностью, например α и β? Предположим, что α хранится в AL и AH, а β хранится в BL и BH (где L обозначает младшие слова, а H – старшие слова). Чтобы сложить α и β, недостаточно написать команду ADD AL, BL, затем ADD AH, BH, так как при этом игнорируется возможный перенос из младших слов. Этот перенос, который соответствует значению C, после выполнения сложения младших слов должен быть прибавлен к AH (или BH) непосредственно перед выполнением операции сложения старших слов. Эта операция осуществляется с помощью команды ADC d, которая делает следующее: (d) ← (d) + C. Таким образом, чтобы вычислить $\alpha + \beta$, необходимо записать следующую последовательность команд:

```
ADD AL, BL
ADC BH
ADD AH, BH
```

Примеры

Во всех примерах этого раздела предполагается, что длина слов – четыре разряда.

		Пример 1		Пример 2
Исходные значения	AH AL	0001 1001	$\leftarrow\alpha\rightarrow$	0001 1001
	BH BL	0011 0101	$\leftarrow\beta\rightarrow$	0011 0111
После ADD AL, BL	AH AL	0001 1001		0001 1001
	BH BL	0011 1110 (C = 0)		0011 0000 (C = 1)
После ADC AH	AH AL	0001 1001		0010 1001
	BH BL	0011 1110		0011 0000
После ADD AH, BH	AH AL	0001 1001		0010 1001
	BH BL	0100 1110	$\leftarrow\alpha+\beta\rightarrow$	0101 0000

□

Чтобы изменить знак числа с двойной точностью α, хранящегося в AL и AH, мы должны сформировать обратные коды AH и AL, прибавить 1 к AL и прибавить полученный в результате перенос к AH. Обратный код (дополнение до 1) AH можно сформировать путем смены его знака и вычитания 1; обратный код и прибавление 1 для AL можно осуществить простой сменой знака. Если смена знака AL не формирует переноса (т. е. C = 1), то смена

знака у α на этом заканчивается. Если при смене знака AL формируется перенос (т. е. C = 0), то прибавление этого переноса к AH можно осуществить второй сменой знака у результата (AH) — 1 после первой смены знака AH. В итоге α будет менять знак в результате смены знаков AH и AL и последующего вычитания C из AH. Последняя операция осуществляется командой SBC d, которая делает следующее: (d) ← (d) — C. Таким образом, для вычисления $-\alpha$ мы должны написать последовательность команд вида

```
NEG AH
NEG AL
SBC AH
```

Примеры

		Пример 1		Пример 2
Исходные значения	AH FL	0101 1011	$\leftarrow\alpha\rightarrow$	0101 0000
После NEG AH	AH AL	1011 1011		1011 0000
После NEG AL	AH AL	1011 0101 (C = 1)		1011 0000 (C = 0)
После SBC AH	AH AL	1010 0101	$\leftarrow\alpha\rightarrow$	1011 0000

□

Предположим, что α и β хранятся как и прежде и мы хотим вычислить $\alpha - \beta$ и запомнить результат в AL и AH. Это можно сделать, сменив сначала знак у β и сложив затем полученный результат с α (используя для этого соответствующие последовательности команд для выполнения операций с двойной точностью, рассмотренные выше). Эти операции эквивалентны вычитанию BL из AL, прибавлению переноса к AH, вычитанию BH из AH и вычитанию 1 из полученного результата (так как второе вычитание равносильно прибавлению к AH отрицательного числа, а не дополнению до единицы, или BH). Прибавление переноса к AH и последующее вычитание 1 из полученного результата можно объединить (как это было сделано в случае смены знака числа с двойной точностью), вычитая из AH значение C, сформированное при вычитании младшего слова. Таким образом, чтобы вычислить $\alpha - \beta$, достаточно написать следующую последовательность команд:

```
SUB BL, AL
SBC AH
SUB BH, AH
```

Примеры

		Пример 1		Пример 2
Исходные значения	AH AL	0101 1001	$\leftarrow\alpha\rightarrow$	0101 0111
	BH BL	0011 0101		0011 1010
После SUB BL, AL	AH AL	0101 0100 (C = 0)		0101 1101 (C = 1)
	BH BL	0011 0101		0011 1010
После SBC AH	AH AL	0101 0100		0100 1101
	BH BL	0011 0101		0011 1010
После SUB BH, AH	AH AL	0010 0100	$\leftarrow\alpha-\beta\rightarrow$	0001 1101
	BH BL	0011 0101		0011 1010

□

Из разд. 7.1 и 7.2 известно, что результат сложения или вычитания является неверным только тогда, когда возникает переполнение (т. е. V = 1). При использовании операндов с двойной точностью это означает, что в результате сложения или вычитания *старших слов* возникает переполнение, т. е. V = 1. Переполнение, возникающее при операциях с младшими словами, не играет роли.

7.4. Команды TST и CMP

Команда CMP s, d (или CMPB s, d) формирует разность (s) − (d) в ALU (но не в d!). Она часто используется для того, чтобы установить коды условий Z, N, C и V так, чтобы облегчить сравнение (s) и (d). Важно отметить, что коды C и V, полученные в результате выполнения CMP, устанавливаются в соответствии с правилами *вычитания* (установленными в разд. 7.2). Применение CMP будет подробно рассмотрено в разд. 7.5.

Команда TST d (или TSTB d) выполняет операцию (d) ← (d). В самом общем случае следует установить коды условий Z и N так, чтобы облегчить определение того, каким является (d): нулевым или ненулевым, положительным или отрицательным.

Примеры

Содержимое СМ для слова с адресом 1000	Результат выполнения команды					
	TST1000		TSTB1000		TSTB1001	
	Z	N	Z	N	Z	N
000000	1	0	1	0	1	0
000001	0	0	0	0	1	0
000400	0	0	1	0	0	0
000401	0	0	0	0	0	0
000200	0	0	0	1	1	0
100000	0	1	1	0	0	1
100001	0	1	0	0	0	1
177500	0	1	0	0	0	1
100200	0	1	0	1	0	1
000600	0	0	0	1	0	0

□

Хотя CMP и TST используются главным образом для контроля чисел, их можно иногда использовать и "более грамотно"–для увеличения или уменьшения содержимого регистров. Например,

```
TST − (SP)          ; (SP) ← (SP) − 2
TST (R5) +          ; (R5) ← (R5) + 2
CMP (R0) +, (R0) +  ; (R0) ← (R0) + 4
```

В любом случае, когда команды TST или CMP не используются как команды условного перехода, их целесообразно применять для формирования автоинкрементного или автодекрементного режима работы регистров.

Заметим, что команда TST(R5) + недопустима, когда R5 содержит нечетный адрес (так как TST осуществляет операции над словами). Следовательно, нужно быть особенно внимательным при использовании TST и CMP в инкрементном и декрементном режимах.

В то время как команды TST (или TSTB) предназначены для контроля полных слов (или байтов), команды BIT (или BITB) могут быть использованы для контроля определенных частей слов (или байтов). В частности, команда BIT d (или BITB d) реализует функцию AND ("И") для операндов (s) и (d) внутри ALU (но не в d!), что дает возможность контролировать только те разряды в d, которые соответствуют разрядам в s, содержащим 1. Например, для того чтобы осуществить переход к L, если 7-й или 15-й разряд регистра R0 (или оба разряда одновременно) содержит 1, достаточно написать команды

```
BIT  #100200, R0
BNE  L
```

7.5. Команды перехода (дополнительные сведения)

В разд. 4.8 мы перечисляли 16 команд условного перехода. Первые восемь из них

BEQ, BNE, BPL, BMI, BVC, BVS, BCC, BCS

лучше всего подходят для того, чтобы проконтролировать, будет ли результат арифметической операции равен нулю, не равен нулю, положителен или отрицателен, произойдет или нет переполнение или перенос.

Чтобы сравнить два числа, например число α в A и β в B, лучше всего написать команду

CMP A, B

(которая формирует $(\alpha - \beta)$), а затем любую из следующих команд:

	Команды	Перейти к q, если [1]	
Знаковые условные переходы	BGE q	$N \forall V = 0$	$(\alpha \geqslant \beta)$
	BLT q	$N \forall V = 1$	$(\alpha < \beta)$
	BGT q	$Z \vee (N \forall V) = 0$	$(\alpha > \beta)$
	BLE q	$Z \vee (N \forall V) = 1$	$(\alpha \leqslant \beta)$
Беззнаковые условные переходы	BHI q	$C \vee Z = 0$	$(\alpha > \beta)$
	BLOS q	$C \vee Z = 1$	$(\alpha \leqslant \beta)$
	BHIS q	$C = 0$	$(\alpha \geqslant \beta)$
	BLO q	$C = 1$	$(\alpha < \beta)$

Знаковые условные переходы воспринимают α и β как 16-разрядные числа в дополнительном коде и соответственно сравнивают их. Беззнаковые условные переходы рассматривают α и β как 16-разрядные положительные числа (причем MSB рассматривается больше не как знаковый разряд, а как коэффициент при 2^{15} в двоичном представлении числа) и соответственно сравнивают их.

Как правило, команды BGE, BLT, BGT, BLE, BEQ и BNE следует использовать для сравнения данных, а BHI, BLOS, BHIS, BLO, BEQ и BNE – для сравнения адресов памяти.

---

[1] Правила выполнения операции $\forall$: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 1 = 0$. Правила выполнения операции $\vee$: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 1 = 1$.

Рассмотрим, например, следующий код, который очищает ячейки памяти, начиная с ячейки A и до ячейки B включительно (где B – старший адрес):

```
A = 1000
B = 2000
        MOV  #A, R0
        MOV  #B, R1
LOOP:   CLP  (R0) +
        CMP R1, R0
        BGE LOOP
```

Команда BGE обрабатывает операнды, как числа в дополнительном коде. Выход из цикла произойдет, когда (R0) = 2002, при условии нормальной работы. Предположим, однако, что первые два присваивания имеют вид

```
A = 070000
B = 170002
```

(они могут соответствовать правильным адресам в некоторых моделях PDP-11). После того как команда CLP(R0) + будет выполнена в первый раз, имеем (R1) = 170002 и (R0) = 070002. Следовательно, (R1) – (R0) = = 100000, что является отрицательным числом, вызывающим преждевременное окончание цикла.

Правильным решением будет использовать команду перехода, которая обрабатывает операнды как беззнаковые числа. Правильной является следующая последовательность команд:

```
LOOP: CLP  (R0) +
      CMP R1, R0
      BHIS LOOP
```

Команда BHIS будет работать правильно безотносительно к значениям A и B.

В табл. 7.1 приведен ряд примеров (использующих 4-разрядные слова), которые показывают, что различные комбинации кодов условий оказывают влияние на знаковые и беззнаковые команды переходов. В каждом примере в первой строке записано α, во второй строке $-\beta$ (перед которым стоит знак "минус"), а в третьей строке указан результат $\alpha - \beta$ (т. е. число, формируемое командой CMP A, B). Галочкой отмечены команды условных переходов, которые, будучи поставленными после команды CMP A, B, вызовут переход.

Рекомендуется всегда использовать команды BGE и BLT, а не BPL и BMI, если необходимо сравнить два знаковых числа. Например, требуется написать программу, которая переходит на L1, если (A) < (B), и на L2 в противном случае. Можно написать

```
      CMP  A, B
      BMI  L1
L2:  - - -
```

Если оказалось, что A содержит 177772 (т. е. –6), а B содержит 077777, то CMP A, B формирует (A) – (B) = 077773, причем Z = 0, N = 0, C = 0 и V = 1. Следовательно, программа будет переходить на L2, хотя (A) < (B).

Таблица 7.1. **Примеры**

Пример		Результирующие коды условий						
Десятичное представление	Двоичное представление	Z	N	C	V	N∀V	Z∨(N∀V)	C∨Z
7	0111							
-(+3)	-0011							
4	0100	0	0	0	0	0	0	0
	1							
5	0101							
-(+5)	-0101							
0	0000	1	0	0	0	0	1	1
	1							
3	0011							
-(+6)	-0110							
-3	1101	0	1	1	0	1	1	1
-2	1110							
-(+2)	-0010							
-4	1100	0	1	0	0	1	1	0
	1							
-4	1100							
-(+5)	-0101							
-9	0111	0	0	**0**	1	1	1	0
	1							
-3	1101							
-(-2)	-1110							
-1	1111	0	1	1	0	1	1	1
-6	1010							
-(-6)	-1010							
0	0000	1	0	0	0	0	1	1
	1							
-2	1110							
-(-5)	-1011							
3	0011	0	0	0	0	0	0	0
	1							

переходов

Знаковые условия				Беззнаковые условия			
BGE	BLT	BGT	BLE	BHI	BLOS	BHIS	BLO
√		√		√		√	
√			√		√	√	
	√		√		√		√
	√		√	√		√	
	√		√	√		√	
	√		√		√		√
√			√		√	√	
√		√		√		√	

Однако если записать

```
        CMP A, B
        BLT L1
L2: - - -
```

то программа будет переходить на L1. (Что произойдет, когда за командой CMP A, B будет стоять команда BLO L1?)

Беззнаковые условные переходы часто используются для сравнения чисел с двойной точностью). Например, предположим, что α и β — положительные числа с двойной точностью, хранящиеся так, как описано в разд. 7.3. Тогда $\alpha > \beta$, если (AH) > (BH) или если AH = BH и (AL) > (BL), где (AL) и (BL) беззнаковые! Таким образом, чтобы перейти на X, если $\alpha > \beta$, и на Y в противном случае, необходимо написать

```
        CMP AH, BH
        BGT X
        BLT Y
        CMP AL, BL
        BHI
Y: - - -

        .
        .
X: - - -
```

Другой вариант использования команд беззнакового перехода — контроль одновременно двух условий. Предположим, что нужно знать, будет ли (R0) меньше 0 или больше, чем 7, и если это так, то хотелось бы перейти к ячейке X. Можно написать

```
        CMP  R0, #7
        BGT  X
        TST  R0
        BLT  X
```

Другой вариант:

```
        CMP R0, # 7
        BHI  X
```

Он работает, потому что отрицательное число, когда оно обрабатывается как беззнаковое число, всегда "больше", чем любое положительное число. Следует упомянуть, что другим решением было бы следующее:

```
BIT  # 177770, R0
BNE X
```

Почти каждая одноадресная и двухадресная команда оказывает влияние на коды условий Z и N и большинство из них оказывает также влияние на коды C и V*. Это означает, что команды сравнения и контроля, которые, казалось, нужно было бы включить после соответствующей операции и перед командой перехода, фактически оказываются избыточными. Например, в последовательности команд

---

* Более подробно см. инструкцию "PDP-11. Processor Handbook".

```
DEC  R4
CMP  R4, #0
BNE  LOOP
```

команда CMP является излишней; в блоке

```
SUB  A, B
TST  B
BEQ  NEXT
```

излишне использовать команду TST.

7.6. Команды сдвига

В PDP-11 существует ряд команд, которые сдвигают содержимое слова или байта на один разряд вправо или на один разряд влево (рис. 7.1):

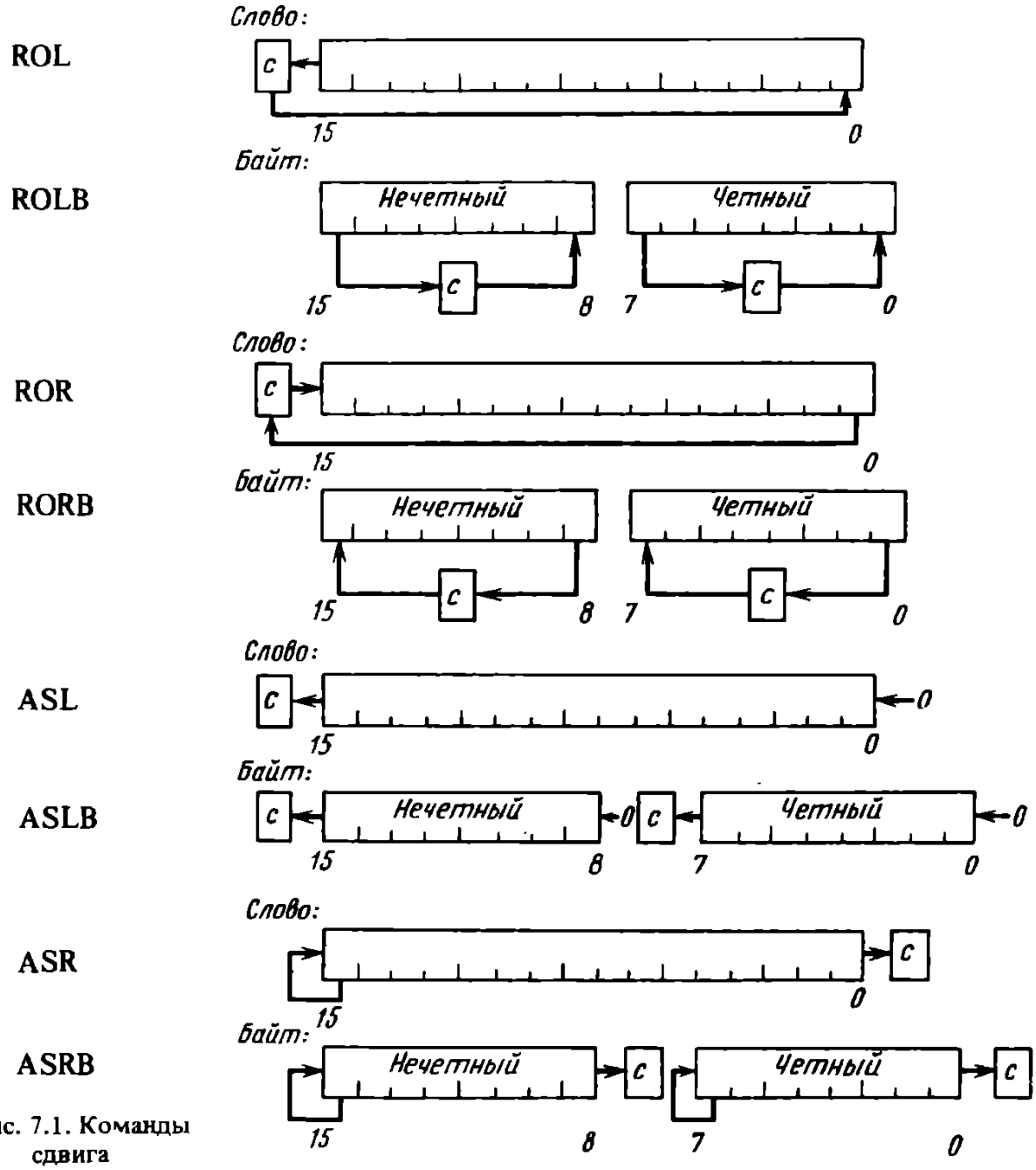


Рис. 7.1. Команды сдвига

ROL, ROLB (циклический сдвиг влево),
ROR, RORB (циклический сдвиг вправо),
ASL, ASLB (арифметический сдвиг влево),
ASR, ASRB (арифметический сдвиг вправо).

Чтобы понять, как выполняются команды ROL и ROR (или ROLB и RORB соответственно), образуем "кольцо", соединив концы операнда-приемника (слова или байты), с помощью (фиктивного) одноразрядного регистра, содержащего код условий C. Действие команды ROL будет состоять в сдвиге содержимого этого кольца на один разряд в направлении часовой стрелки, тогда как действие команды ROR будет состоять в сдвиге содержимого против часовой стрелки.

Действие команды ASL (или ASLB) состоит в сдвиге операнда-приемника (слова или байта) на один разряд влево так, чтобы крайний правый разряд был замещен 0, а старое значение C было замещено значением крайнего левого разряда. Эта операция эквивалентна умножению содержимого операнда-приемника (рассматриваемого как знаковое число в дополнительном коде) на 2. Код V при этом устанавливается равным 1, если результат этого умножения выходит за пределы разрядной сетки.

Действие команды ASR (или ASRB) состоит в сдвиге содержимого операнда-приемника (слова или байта) на один разряд вправо, причем крайний левый разряд остается неизменным, а крайний правый разряд заменяет старое значение кода C. Эта операция эквивалентна делению операнда-приемника на 2 (и вычитания 1/2, если содержимое нечетно). Существует единственное исключение: -1 остается -1.

Примеры

(A)	После ASL A	После ASR A
000032	000064 (= 32 × 2)	000015 (= 32/2)
177746 (= −32)	177714 (= −64)	177763 (= −15)

□

7.7. Пример: преобразование кода ASCII в двоичный код

Чтобы проиллюстрировать использование некоторых команд, введенных в предыдущих разделах, и их характерные особенности, рассмотрим программу, которая принимает десятичное число N с телетайпа и преобразует его в двоичное 16-разрядное число в дополнительном коде; N может иметь знаки + или − и всегда сопровождается возвратом каретки. Его значение не должно превышать 32767_{10}.

Указанная программа (блок-схема которой приведена на рис. 7.2, а распечатка — на рис. 7.3) запоминает входные символы в массиве байтов с базовым адресом STRING. Двоичный эквивалент N остается в R2. (Если N выходит за пределы разрядной сетки, в R2 остается код 100000_8.) Программа использует следующие подпрограммы:

INPUT (запоминает входные символы в массиве STRING; использует PRINT),
PRINT (распечатывает содержимое R5),

ATOB (вычисляет двоичный эквивалент N и пересылает его в R2; использует MUL),

MUL (вычисляет (R0) * (R1) и пересылает результат в R2).

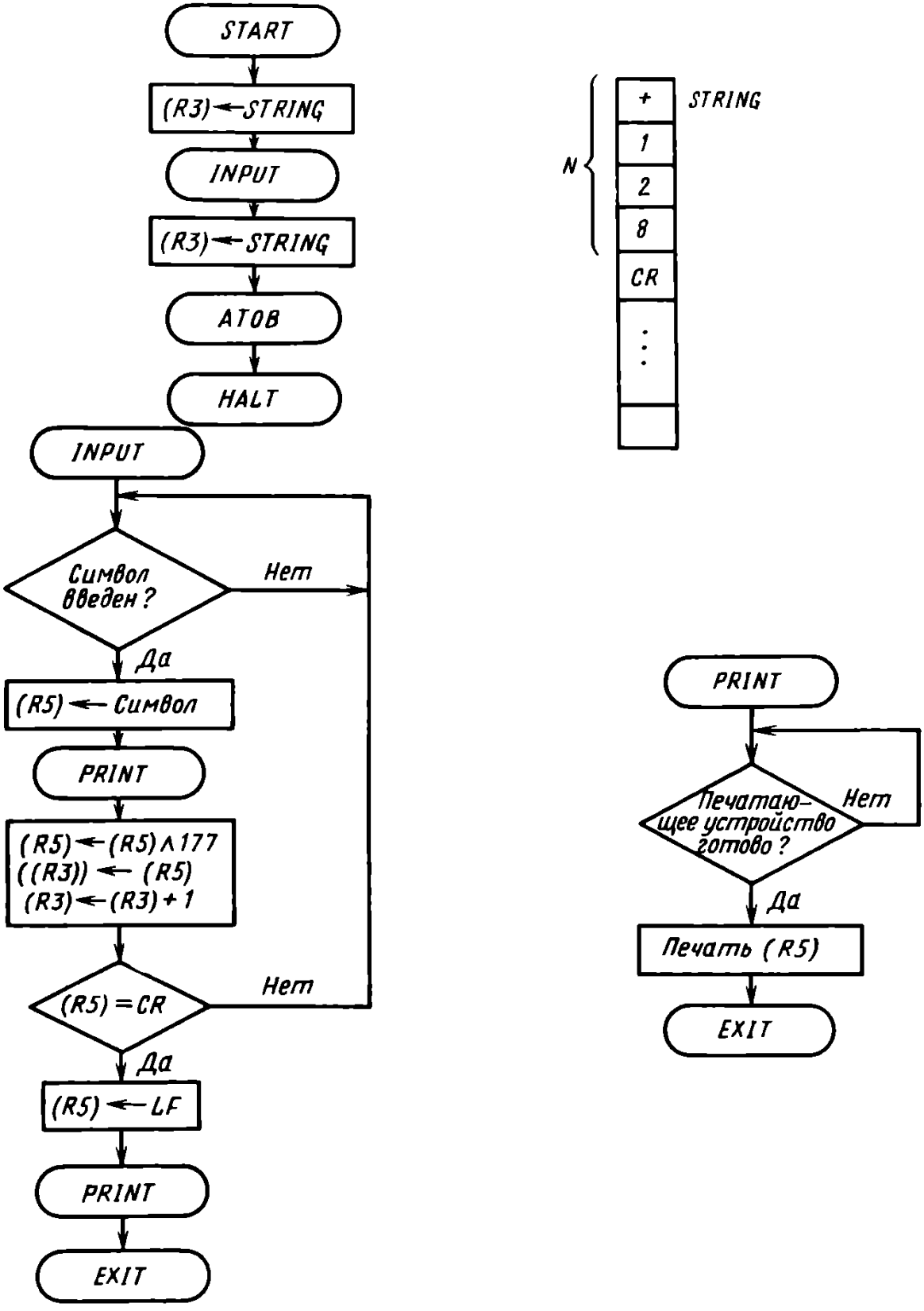


Рис. 7.2. Блок-схема программы преобразования кода ASCII в двоичный

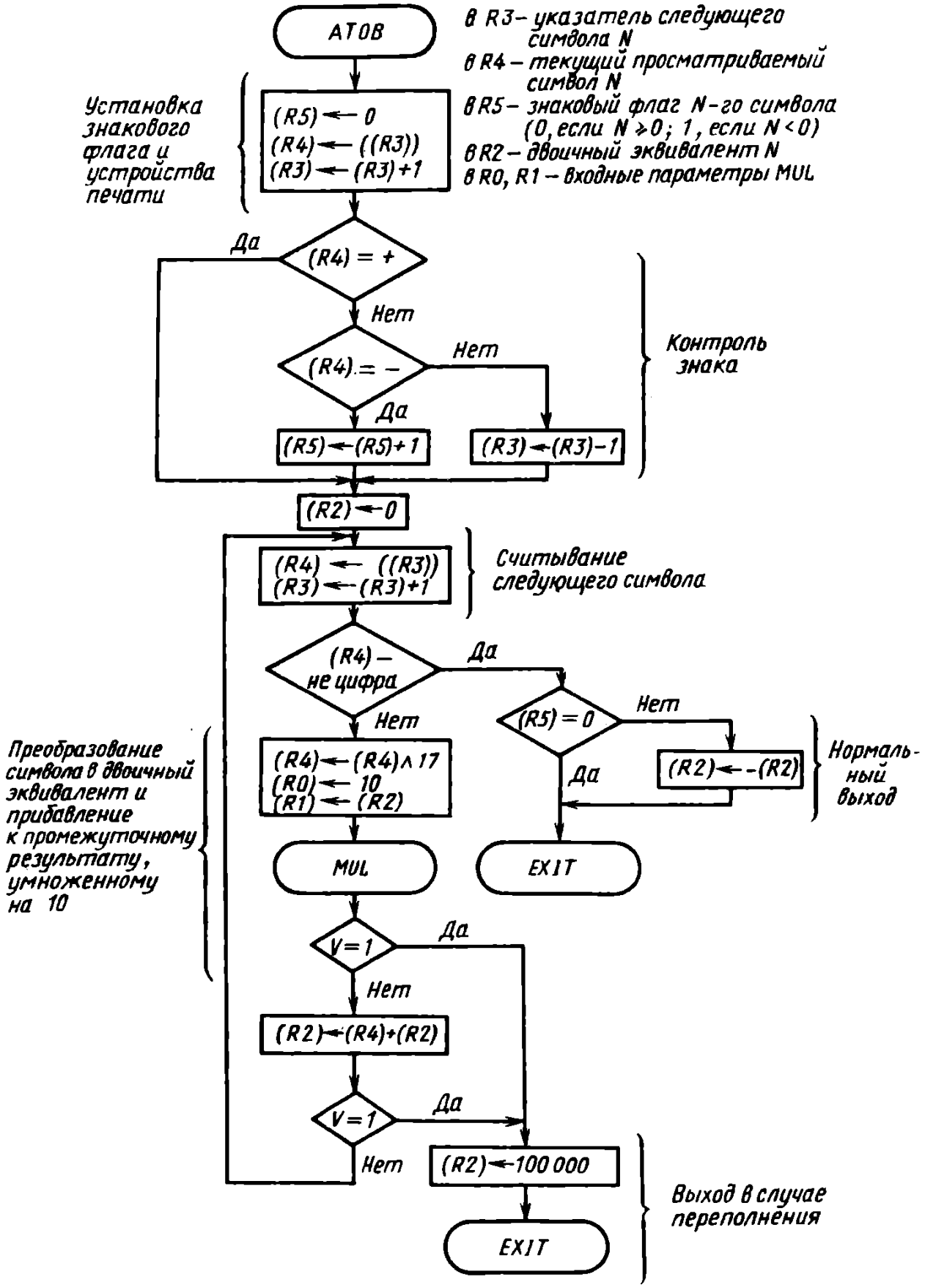


Рис. 7.2 (продолжение)

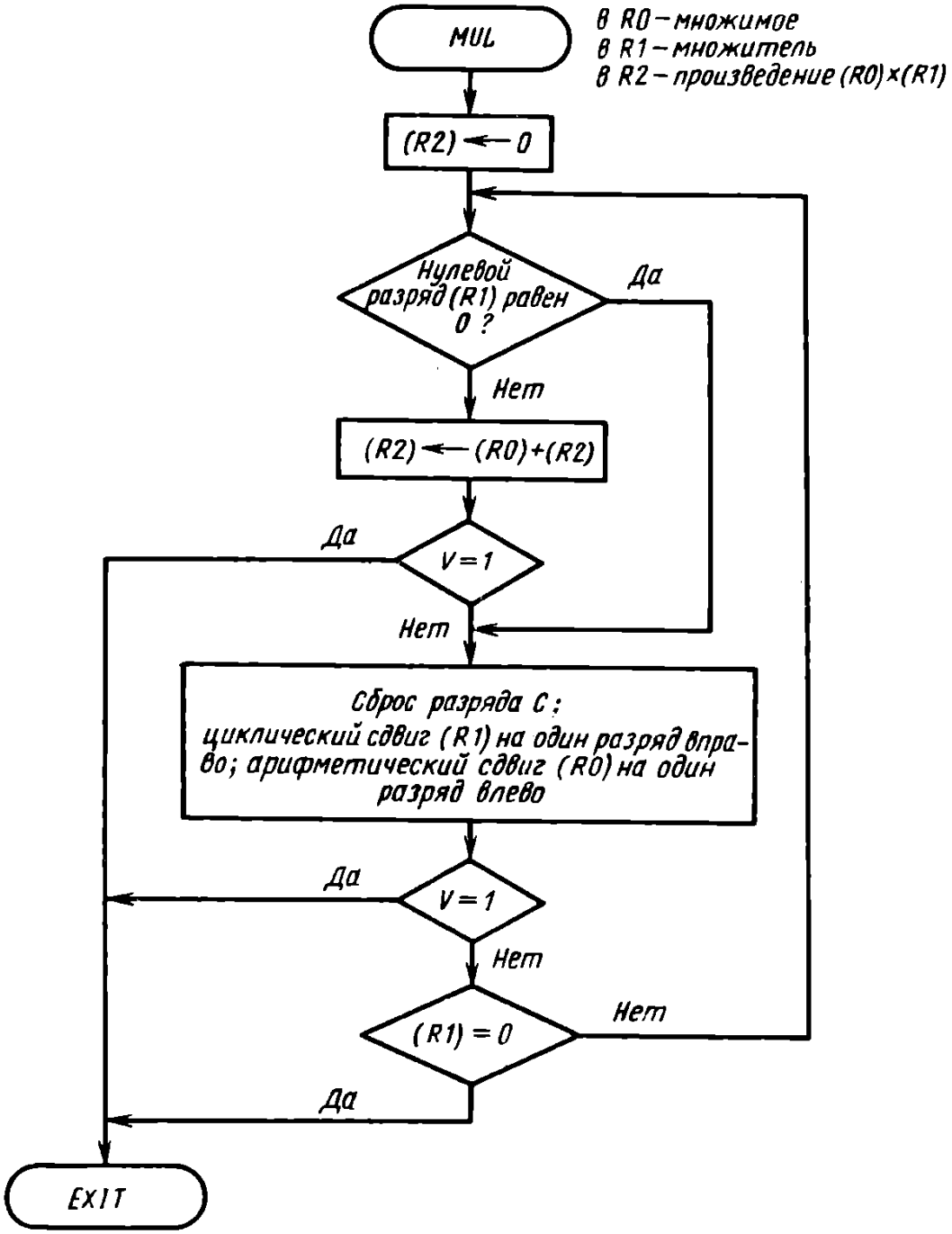


Рис. 7.2 (окончание)

```
        .TITLE ASCTOBIN
        .ENABLE LC
;ПРЕОБРАЗОВАНИЕ ВВЕДЕННОГО ДЕСЯТИЧНОГО ЧИСЛА
;N В ДВОИЧНЫЙ ЭКВИВАЛЕНТ. ПЕРЕД N МОЖЕТ СТО-
;ЯТЬ ЗНАК + ИЛИ -, ПОСЛЕ N ДОЛЖЕН БЫТЬ ВВЕ-
;ДЕН СИМВОЛ ВОЗВРАТА КАРЕТКИ.
```

Рис. 7.3. Программа преобразования кода ASCII в двоичный

```
;ДВОИЧНЫЙ ЭКВИВАЛЕНТ N ОСТАЕТСЯ В R2. ЕСЛИ
;МОДУЛЬ N ПРЕВЫШАЕТ ЧИСЛО 32767 (ДЕСЯТИЧ-
;НОЕ), ТО В (R2) ОСТАНЕТСЯ ВОСЬМЕРИЧНОЕ
;ЧИСЛО 100000.
LC=.
.=4+LC
        .WORD   6,0,12,0        ;УСТАНОВКА ВЕКТО-
                                ;РОВ ОШИБОК
.=500+LC                        ;РЕЗЕРВИРОВАНИЕ
                                ;ПАМЯТИ ДЛЯ СТЕКА
START:  MOV     PC,SP
        TST     -(SP)           ;НАЧАЛЬНАЯ УСТА-
                                ;НОВКА SP
;
KBSTAT=177560
KBDATA=177562
PRSTAT=177564
PRDATA=177566
LF=12
CR=15
;
;       MAIN PROGRAM
        MOV     #STRING,R3      ;(R3)=STRING
        JSR     PC,INPUT        ;ЗАНЕСТИ ВХОДНУЮ
                                ;СТРОКУ (STRING)
                                ;В МАССИВ
        MOV     #STRING,R3      ;(R3)=STRING
        JSR     PC,ATOB         ;ПРЕОБРАЗОВАТЬ
                                ;СТРОКУ В ДВОИЧНЫЙ
                                ;ЭКВИВАЛЕНТ
        HALT
;
STRING: .BLKB   20.             ;ПАМЯТЬ ПОД ВВЕ-
                                ;ДЕННУЮ СТРОКУ
;
;       INPUT
;РАСПЕЧАТКА ВВЕДЕННЫХ СИМВОЛОВ И ЗАПОМИНАНИЕ
;ИХ В МАССИВЕ БАЙТОВ, БАЗОВЫЙ АДРЕС КОТОРОГО
;В R3. ВЫХОД ПОСЛЕ ТОГО, КАК НАПЕЧАТАНО CR.
;ИЗМЕНЕНИЕ R3 И R5.
INPUT:  TSTB    KBSTAT          ;СИМВОЛ ВВЕДЕН?
        BPL     INPUT           ;ЕСЛИ НЕТ, ЖДАТЬ
        MOV     KBDATA,R5       ;(R5)=СИМВОЛУ
        JSR     PC,PRINT        ;ПЕЧАТЬ СИМВОЛА
        BIC     #177600,R5      ;УДАЛИТЬ КОНТРОЛЬ-
                                ;НЫЙ РАЗРЯД
        MOVB    R5,(R3)+        ;ЗАНЕСТИ СИМВОЛ В
                                ;МАССИВ,СКОРРЕКТ.
                                ;ИНДЕКС
        CMP     #CR,R5          ;СИМВОЛ СООТВЕТ-
                                ;СТВУЕТ CR?
        BNE     INPUT           ;ЕСЛИ НЕТ, ПРИНЯТЬ
                                ;СЛЕДУЮЩИЙ СИМВОЛ
        MOV     #LF,R5          ;ЕСЛИ ДА,
```

Рис. 7.3 (продолжение)

```
        JSR     PC,PRINT        ;ПЕРЕЙТИ НА СЛЕ-
                                ;ДУЮЩУЮ СТРОКУ
        RTS     PC              ;ВЫХОД
;
;       PRINT
;ПЕЧАТЬ СОДЕРЖИМОГО R5. РЕГИСТРЫ НЕ МЕНЯЮТСЯ.
PRINT:  TSTB    PRSTAT          ;УСТРОЙСТО ПЕЧАТИ
                                ;ГОТОВО?
        BPL     PRINT           ;ЕСЛИ НЕТ, ЖДАТЬ
        MOV     R5,PRDATA       ;ЕСЛИ ДА, ПЕЧАТЬ
                                ;(R5)
        RTS     PC              ;ВЫХОД
;
;       ATOB
;ПРЕОБРАЗОВАНИЕ В ДВОИЧНОЕ ДЕСЯТИЧНОГО ЧИСЛА
;N, ХРАНЯЩЕГОСЯ В КОДЕ ASCII В МАССИВЕ БАЙ-
;ТОВ, БАЗОВЫЙ АДРЕС КОТОРОГО В R3. N МОЖЕТ
;ИМЕТЬ ЗНАК + ИЛИ -, А ЗА N ДОЛЖЕН СЛЕДОВАТЬ
;НЕЦИФРОВОЙ СИМВОЛ. ДВОИЧНЫЙ ЭКВИВАЛЕНТ N
;ОСТАЕТСЯ В R2.
;ЕСЛИ МОДУЛЬ N ПРЕВЫШАЕТ ЧИСЛО (ДЕСЯТИЧ-
;НОЕ) 32767, ТО В R2 ОСТАЕТСЯ ВОСЬМЕРИЧНОЕ
;100000. РАСПРЕДЕЛЕНИЕ РЕГИСТРОВ:
;       (R0),(R1) ИСПОЛЬЗУЮТСЯ ДЛЯ ПАРАМЕТРОВ
;                 ПОДПРОГРАММЫ MUL
;       (R2)=ПРЕОБРАЗОВАННОМУ ЧИСЛУ
;       (R3)=УКАЗАТЕЛЮ К СЛЕДУЮЩЕМУ СИМВОЛУ
;       (R4)=ПРОСМОТРЕННОМУ СИМВОЛУ
;       (R5)=ЗНАКОВОМУ ФЛАГУ (0, ЕСЛИ N ИМЕ-
;            ЕТ +, 1, ЕСЛИ - )
 ATOB:  CLR     R5              ;ЗАНУЛИТЬ ЗНАКОВЫЙ
                                ;ФЛАГ (СЧИТАЕМ N
                                ;ПОЛОЖИТ.)
        MOVB    (R3)+,R4        ;(R4)=ПРОСМОТР.
                                ;СИМВОЛУ, СКОРРЕК-
                                ;ТИР. ИНДЕКС
        CMPB    #'+,R4          ;СИМВОЛ ИМЕЕТ
                                ;ЗНАК +?
        BEQ     ATOB2           ;ЕСЛИ ДА, НАЧАТЬ
                                ;ПРЕОБРАЗОВАНИЕ
        CMPB    #'-,R4          ;СИМВОЛ ИМЕЕТ
                                ;ЗНАК -?
        BNE     ATOB1           ;ЕСЛИ НЕТ, ТО N
                                ;НЕ ИМЕЕТ ЗНАКА
        INC     R5              ;ЕСЛИ ДА, УСТАНО-
                                ;ВИТЬ ЗНАКОВЫЙ
                                ;ФЛАГ НА 1
        BR      ATOB2           ;НАЧАЛО ПРЕОБРА-
                                ;ЗОВАНИЯ
ATOB1:  DEC     R3              ;СИМВОЛ ЯВЛ. ЦИФ-
                                ;РОЙ. ВОЗВРАТ
                                ;УПРАВЛЕНИЯ
ATOB2:  CLR     R2              ;УСТАНОВИТЬ РЕ-
                                ;ЗУЛЬТАТ РАВНЫМ 0
ATOB3:  MOVB    (R3)+,R4        ;(R4)=ПРОСМОТР.
```

Рис. 7.3 (продолжение)

```
                                    ;СИМВОЛУ, СКОРРЕК-
                                    ;ТИР. ИНДЕКС
        CMPB    #'0,R4              ;ЕСЛИ '0>(R4)
                                    ;(Т.Е. НЕ ЦИФРА),
        BHI     ATOB4               ;ТО ПОДГОТОВКА К
                                    ;ВЫХОДУ
        CMPB    #'9,R4              ;ЕСЛИ '9<(R4)
                                    ;(Т.Е. НЕ ЦИФРА),
        BLO     ATOB4               ;ТО ПОДГОТОВКА К
                                    ;ВЫХОДУ
        BIC     #177760,R4          ;ПРЕОБРАЗ. ЦИФРЫ
                                    ;В ДВОИЧН. ЭКВИ-
                                    ;ВАЛЕНТ
        MOV     #10.,R0             ;(R0)=10. (ПАРА-
                                    ;МЕТР ПОДПРОГР.
                                    ;MUL)
        MOV     R2,R1               ;(R2)=(R1) (ПАРА-
                                    ;МЕТР ПОДПРОГР.
                                    ;MUL)
        JSR     PC,MUL              ;(R2)=(R0)*(R1)=
                                    ;10.*(R2)
        BVS     ATOB6               ;ЕСЛИ ПЕРЕПОЛНЕ-
                                    ;НИЕ, ПОДГОТОВКА
                                    ;К ВЫХОДУ
        ADD     R4,R2               ;(R2)=(R4)+(R2)
        BVS     ATOB6               ;ЕСЛИ ПЕРЕПОЛНЕ-
                                    ;НИЕ, ПОДГОТОВКА
                                    ;К ВЫХОДУ
        BR      ATOB3               ;ПРОСМОТРЕТЬ СЛЕ-
                                    ;ДУЮЩИЙ СИМВОЛ
;NORMAL EXIT(НОРМАЛЬНЫЙ ВЫХОД)
ATOB4:  TST     R5                  ;ПРОВЕРКА ЗНАКО-
                                    ;ВОГО ФЛАГА
        BEQ     ATOB5               ;ЕСЛИ ЧИСЛО ПОЛО-
                                    ;ЖИТЕЛЬНОЕ, ТО
                                    ;ВЫХОД
        NEG     R2                  ;ЕСЛИ НЕТ, ТО
                                    ;(R2)=-(R2)
ATOB5:  RTS     PC                  ;ВЫХОД
;OVERLOW EXIT(ВЫХОД ПО ПЕРЕПОЛНЕНИЮ)
ATOB6:  MOV     #100000,R2          ;(R2)=100000
        RTS     PC                  ;ВЫХОД
;
;       MUL
;ВЫЧИСЛЕНИЕ (R0)*(R1) И ЗАПИСЬ РЕЗУЛЬТАТОВ
;В R2. ЕСЛИ МОДУЛЬ РЕЗУЛЬТАТА ПРЕВЫШАЕТ
;32767 (ДЕСЯТИЧНОЕ), ТО В РАЗРЯДЕ V УСТАНО-
;ВИТЬ 0. R3,R4,R5 НЕ ИСПОЛЬЗУЮТСЯ.
;
MUL:    CLR     R2                  ;(R2)=0
MUL1:   BIT     #1,R1               ;ПРОВЕРКА НУЛЕВО-
                                    ;ГО РАЗРЯДА R1
        BEQ     MUL2                ;ЕСЛИ 0, НЕ ПРО-
                                    ;ВОДИТЬ СЛОЖЕНИЯ
```

Рис. 7.3 (продолжение)

```
        ADD     R0,R2           ;ИНАЧЕ, (R0)=(R0)
                                ;+(R2)
        BVS     MUL3            ;ВЫХОД ПО ПЕРЕ-
                                ;ПОЛНЕНИЮ
MUL2:   CLC                     ;ОЧИСТКА РАЗ-
                                ;РЯДА С
        ROR     R1              ;ЦИКЛ. СДВИГ R1
                                ;НА 1 РАЗРЯД
                                ;ВПРАВО
        ASL     R0              ;АРИФМ. СДВИГ (R0)
                                ;НА 1 РАЗРЯД
                                ;ВЛЕВО
        BVS     MUL3            ;ВЫХОД ПО ПЕРЕ-
                                ;ПОЛНЕНИЮ
        TST     R1              ;ПРОВЕРКА (R1)
        BNE     MUL1            ;ЕСЛИ НЕ 0, ПРО-
                                ;ДОЛЖИТЬ ПЕРЕМНО-
                                ;ЖЕНИЕ
MUL3:   RTS     PC              ;ВЫХОД
;
        .END    START
```

Рис. 7.3 (окончание)

Алгоритм, используемый в подпрограмме MUL для умножения, значительно эффективнее, чем тот, что использовался в MULT (см. разд. 6.5). В соответствии с этим алгоритмом требуется:

1. Установить 0 в (R2).
2. Если 0-й разряд в (R1) содержит 1, прибавить (R0) к (R2). В противном случае пропустить этот шаг.
3. Сдвинуть (R0) влево на один разряд, а (R1) вправо на один разряд.
4. Если (R1) = 0, R2 содержит (R0) * (R1), то осуществить выход. В противном случае вернуться к шагу 2.

Тот факт, что этот алгоритм работает для положительных чисел, ясен из следующего примера (где все числа двоичные):

```
0 . . .. 00011010 ← исходное (R0)
0 . . . 01001011 ← исходное (R1)
-------------------------------------
        11010   |
       11010    | сдвинутые (R0)
      11010     |
     11010      |
-------------------------------------
11110011110 ← окончательное (R2) (сумма сдвинутых (R0))
```

Чтобы убедиться, что алгоритм работает и для отрицательных чисел, следует осуществить арифметический сдвиг (R0) влево (т. е. используя ASL, а не ROL) и циклический сдвиг (R1) вправо (т. е. используя ROR, а не ASP), очистив предварительно C.

Если произведение (R0) * (R1) выходит за пределы разрядной сетки, то сложение на шаге 2 или левый сдвиг на шаге 3 приведут к формированию V = 1, что может быть использовано вызывающей программой в качестве индикатора ошибки.

Подпрограмма ATOB начинает с установки "флага" (хранящегося в R5) на 0, если N положительно, и на 1 в противном случае. Затем она преобразует | N | в двоичный код, используя следующий алгоритм:

1. Установить (R2) на 0.

2. Принять следующий символ. Если это не цифра, R2 содержит двоичный эквивалент | N |; если (R5) = 1, то сменить знак (R2) и осуществить выход. Если указанный символ является цифрой, перейти к шагу 3.

3. Умножить (R2) на 10_{10} (или 12_8) и прибавить двоичный эквивалент цифры, принятой на шаге 2. Вернуться к шагу 2.

Например, если N = 2591, то R2 будет последовательно содержать в десятичном коде:

$$\begin{aligned} &0 \\ (0 \times 10) + 2 &= 2 \\ (2 \times 10) + 5 &= 25 \\ (25 \times 10) + 9 &= 259 \\ (259 \times 10) + 1 &= 2591 \end{aligned}$$

или в восьмеричном коде

$$\begin{aligned} &0 \\ (0 \times 12) + 2 &= 2 \\ (2 \times 12) + 5 &= 31 \\ (31 \times 12) + 11 &= 403 \\ (403 \times 12) + 1 &= 5037 \end{aligned}$$

Если умножение или сложение на шаге 3 происходит с переполнением, то 100000_8 хранится в R2 в качестве индикатора ошибки.

Заметим, что цифры в коде ASCII можно преобразовать в двоичный эквивалент, если просто взять четыре крайних правых разряда [или осуществлением операции "AND" (т. е. логическое умножение) над кодом 17_8]. Например, 5 в ASCII соответствует 065; проводя указанную операцию, получаем 000065 ∧ 17 = 000005; 9 в ASCII соответствует 071, следовательно, 000071 ∧ 17 = 000011.

Упражнения

7.1. Чему равно восьмеричное содержимое R0 и PSR после выполнения каждой команды в следующем программном блоке? (В случае необходимости можно воспользоваться руководством "PDP-11 Processor Handbook".)

```
CLR     R0
DEC     R0
ADD     #77777,R0
ADD     #2,R0
COM     R0
SUB     #177777,R0
ADD     #100000,R0
SUB     #1,R0
BIC     #54321,R0
ASL     R0
ASR     R0
NEG     R0
ROR     R0
```

7.2. Рассмотрите следующий программный сегмент:

```
ADD  A1, B1
BVC  L1
BCC  L2
SUB  A2, B2
BCC  L3
BVC  L4
BR   L5
```

После выполнения указанного сегмента определить, куда должна идти программа, если содержимые A1, B1, A2 и B2 равны:

	(A1) и (A2)	(B1) и (B2)
(а)	012306	100334
(б)	132550	052267
(в)	041316	060215
(г)	150043	117720
(д)	117720	150043

7.3. Дано: (1000) = 177465, (1002) = 000313. Определить значения кодов Z, N, C, V после выполнения каждой из следующих команд. (Позиции, помеченные крестиками, заполнять не обязательно.)

Команды	Z	N	C	V
TST 1000			X	X
TSTB 1000			X	X
TSTB 1001			X	X
TST 1002			X	X
TSTB 1002			X	X
TSTB 1003			X	X
CMP 1000, 1002				
ADD 1000, 1002				

7.4. Следующей за командой CMP A, B должна быть команда перехода вида B . . . . . L1, где . . . может быть любой командой из набора GE, LT, GT, LE, HI, LOS, HIS, LO. В результате выполнения какой из этих восьми команд произойдет переход к ячейке L1, если содержимое A и B следующее:

	(A)	(B)
(а)	017522	017522
(б)	104311	104311
(в)	177602	000176
(г)	054272	001016
(д)	177705	177613
(е)	004640	017227
(ж)	105352	176051
(з)	151547	031246

7.5. Следующий программный сегмент предназначен для очистки содержимого слов с адресами с 000000 по 100000. Дописать четвертую команду сегмента.

```
        CLR     R0
LOOP    CLR     (R0)+
        CMP     #100000,R0
        B...    LOOP
        HALT
```

7.6. Пусть ячейки X и X + 2 содержат число с двойной точностью p, а Y и Y + 2 содержат число с двойной точностью q. (X и Y соответственно младшие слова.) Описать работу следующего программного сегмента:

```
        ADD     X+2,X+2
        ADD     X,X
        ADC     X+2
        CMP     X+2,Y+2
        BLT     L2
        BGT     L1
        CMP     X,Y
        BLOS    L2
L1:     MOV     #1,R0
        HALT
L2:     MOV     #2,R0
        HALT
```

7.7. Рассмотрите следующую программу:

```
START   MOV     X,R0
        CLR     R1
LOOP:   TST     R0
        BEQ     EXIT
        BPL     HERE
        ASL     R1
        INC     R1
HERE:   ASL     R0
        BR      LOOP
EXIT:   MOV     R1,Y
        HALT
X:      .WORD   123456
Y:      .BLKW   1
```

(а) Чему равно окончательное содержимое Y?

(б) Что осуществляется с помощью указанной программы в общем случае (при произвольном содержимом X)?

7.8. Написать на языке ассемблера программу, которая вычисляет с двойной точностью произведение двух чисел, представленных с одинарной точностью. Считайте, что множитель и множимое хранятся по адресам A и B; результат остается в C и C + 2.

7.9. Написать подпрограмму, которая подсчитывает число разрядов, содержащих 1, в R0 и результат помещает в R1. Например, если (R0) = 123456, то (R1) = 000011.

7.10. Написать на языке ассемблера программу, которая принимает с телетайпа шестиразрядные восьмеричные числа и распечатывает их двоичный эквивалент.

7.11. Написать и отладить программу, которая моделирует стековый калькулятор, выполняющий четыре действия. На вход калькулятора поступают десятичные знаковые и беззнаковые числа, не превышающие 32767_{10}, которые обрабатываются следующими операторами: + (сложить), – (вычесть), X (умножить), / (разделить нацело), S (изменить знак), X (очистить вершину стека), C (очистить стек). Числа и операторы вводятся с телетайпа, ввод каждого элемента заканчивается возвратом каретки. Числа запоминаются с помощью указанной программы в некотором стеке, который не является системным стеком. Любой из операторов +, –, * или / осуществляет соответствующие операции над двумя верхними элементами стека, которые затем заменяются в стеке полученным результатом. Оператор S проводит замену знака у элемента, расположенного на вершине стека. Оператор X используется для считывания (исключения) вершины стека в случае ошибки набора на телетайпе. Оператор C проводит установку в исходное состояние (инициализацию) стека; этот оператор должен использоваться каждый раз, когда начинаются новые вычисления. Таблица 7.2 содержит более точное описание действий, осуществляемых данной программой, а в табл. 7.3 приведен пример, иллюстрирующий указанную программу.

Таблица 7.2. **Операции стекового калькулятора**

Вход[1]	Операция		Распечатка
Десятичное число, не превышающее 32767_{10} (может быть со знаком + или –)	Число преобразуется в двоичную форму и включается в стек		Вход распечатывается
+	Вершина стека пересылается в A; Выполняются следующие операции:	(B) + (A)	Вход распечатывается
–		(B) – (A)	Новая вершина стека распечатывается в виде 5-значного десятичного числа (возможно со знаком –), далее следуют CR и LF
*		(B) * (A)	
/		(B) / (A)	
S	Изменяется знак вершины стека		
X	Вершина стека исключается и игнорируется		Вход распечатывается
C	Стек очищается		

[1] За которым всегда следует CR.

Программу написать в модульной форме, используя такие подпрограммы, как ATOB (которая преобразует десятичные ASCII числа в двоичные), BTOAS (которая преобразует двоичные числа в десятичные ASCII числа), MUL (программа умножения), DIV (программа деления) и PRINT (программа распечатки).

Таблица 7.3. **Пример вычислений с помощью стекового калькулятора. Арифметическое выражение, которое предстоит вычислить:**

-((-5)*((6+13)*(37-(125/7))+(143/(-15))))

Вход	Стек*	Распечатка
C		C
-5	-5	-5
6	6,-5	6
13	13,6,-5	13
+	19,-5	+
		00019
37	37,19,-5	37
125	125,37,19,-5	125
7	7,125,37,19,-5	7
/	17,37,19,-5	/
		00017
-	20,19,-5	-
		00020
*	380,-5	*
		00380
143	143,380,-5	143
-15	-15,143,380,-5	-15
/	-9,380,-5	/
		-00009
+	371,-5	+
		00371
*	-1855	*
		-01855
S	1855	S
		01855

* Направление сверху вниз (в стеке) соответствует направлению слева направо (в таблице)

Глава 8
ЗАХВАТЫ И ПРЕРЫВАНИЯ

В этой главе мы изучим механизм осуществления и применение захватов и прерываний. Мы рассмотрим захваты, вызванные неверной адресацией и неверной командой, разряд захвата, команду BPT и приоритет прерываний. Глава заканчивается программой, иллюстрирующей организацию вложенных подпрограмм и прерываний и манипуляцию приоритетами прерываний.

8.1. Захваты

Чтобы оградить пользователя от различных неприятностей, которые могут возникнуть в процессе вычислений, в PDP-11 предусмотрено осуществление *процессорных захватов* (непредусмотренных программой прерываний) при

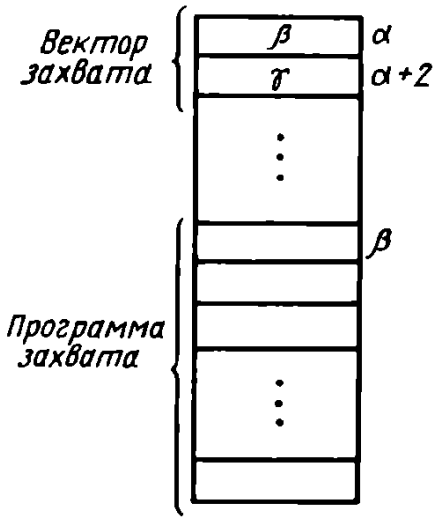


Рис. 8.1. Вектор и программа захватов

возникновении определенных наиболее общих программных ошибок (захват состоит в *автоматическом* переходе к фиксированным ячейкам в оперативной памяти). В частности, каждая такая ошибка ассоциируется с некоторой ячейкой α в памяти (которая фиксируется изготовителем и не может быть изменена); когда происходит указанная ошибка, последующий ход событий предопределен и осуществляется автоматически (в едином цикле):

```
MOV PSR, –(SP)      ; Включить регистр состояния процессора в
                      системный стек
MOV PC, –(SP)       ; Включить счетчик программ в системный стек
MOV @ #α PC         ; (PC) ← (α)
MOV @ #α + 2, PSR   ; (PSR) ← (α + 2)
```

Таким образом, PSR и PC сохраняются в системном стеке, а новое содержимое PSR выбирается из $\alpha + 2$ и осуществляется переход по адресу, находящемуся в α. Если α содержит адрес β, а $\alpha + 2$ содержит константу γ, то процессор приступает к выполнению программы, которая начинается с β при использовании γ в качестве PSR (рис. 8.1). В обязанность программиста входят загрузка в α и $\alpha + 2$ соответственно β и γ (мы будем в дальнейшем ссылаться на них как на *вектор захвата*) и написание программы (называемой программой захвата), которая начинается с β и делает все, что необходимо, если произошла ошибка (например, она осуществляет выдачу сообщения об ошибке и останов).

8.2. Захват при неверной адресации и неверной команде

Одна из наиболее распространенных ошибок при программировании состоит в неверной адресации, попытке выполнить операцию над словами при нечетном адресе или попытке перейти по адресу несуществующей ячейки памяти. Эта ошибка приводит к осуществлению захвата через ячейку 4, который состоит в следующей последовательности действий:

```
MOV  PSP, –(SP)
MOV  PC, –(SP)
MOV  @ # 4, PC
MOV  @ # 6, PSR
```

Другой распространенной ошибкой является попытка выполнить неправильную команду (например, с несуществующим кодом операции). Эта ошибка приводит к осуществлению захвата через ячейку 10, который состоит из следующей последовательности действий:

```
MOV  PSR, -(SP)
MOV  PC, - (SP)
MOV  @ # 10, PC
MOV  @ # 12, PSR
```

Простейшим решением в обоих указанных случаях является останов машины. Это можно осуществить (как было предложено в разд. 5.3) запуском программы

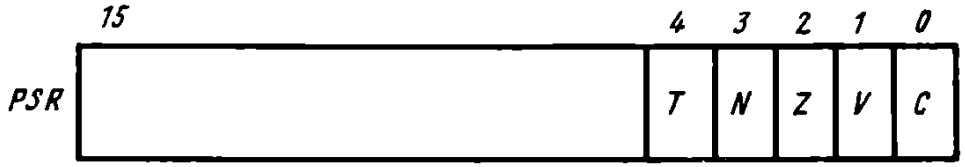
```
LC = .
. = 4 + LC
        .WORD 6, 0, 12, 0
```

которая заполнит ячейки (слова, начинающиеся с соответствующих ячеек) 4, 6, 10 и 12 константами соответственно 6, 0, 12 и 0. Когда в программе ссылаются на неверный адрес, процессор осуществляет через ячейку 4 захват ячейки 6, которая содержит команду 000000, т. е. команду HALT (ОСТАНОВ). Аналогично, когда пытаются выполнить неверную команду, процессор через ячейку 10 осуществляет захват ячейки 12, которая также содержит 000000.

Заметим, что после останова центрального процессора, вызванного любой из этих двух программ захвата, вершина системного стека содержит значение PC в момент возникновения ошибки. Таким образом, PC содержит либо 10, либо 14 в зависимости от того, какая программа захвата была выполнена: для неверной адресации или для неверной команды. Следовательно, анализируя содержимое (SP) и PC, можно определить указанную ячейку и тип возникшей ошибки.

8.3. Разряд захвата и команда BPT

Из рис. 2.1 видно, что PSR содержит кроме кодов условий N, Z, V, C еще и *разряд захвата* T, который соответствует разряду 4:



Как и другие разряды, разряд захвата может быть установлен (на 1) или очищен (сброс на 0) с помощью программных средств. Когда он установлен, процессорный захват осуществляется через ячейку 14.

Разряд захвата особенно удобен, когда в текст основной программы встраиваются специальные программы-мониторы (такие, как программы трассировки и отладки), функция которых состоит в покомандном исполнении программы пользователя при наличии контроля после каждого исполнения. Программа-монитор может осуществить это, устанавливая T на 1 перед тем, как передать управление программе пользователя. После того как первая команда пользователя будет выполнена, осуществляется захват

центральным процессором через ячейку 14, возвращающий управление программе-монитору, которая и выполняет положенные ей действия. Используя информацию, хранящуюся в системном стеке в момент захвата программы пользователя, программа-монитор может теперь передать управление программе пользователя, возвращая PC и PSR те значения, которые они имели в момент последнего прерывания. После этого выполняется вторая команда пользователя, осуществляется захват программы и процесс повторяется.

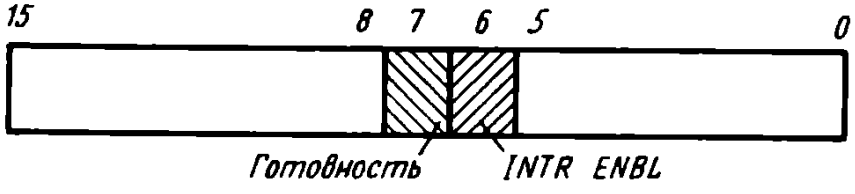
Захват может генерироваться самой программой (а не только осуществляется автоматически аппаратными средствами), если использовать команду BPT (breakpoint (ТОЧКА ПРЕРЫВАНИЯ)). При выполнении этой команды захват осуществляется через ячейку 14.

8.4. Прерывания

Другим видом процессорного захвата является прерывание. В то время как процессорный захват, как мы убедились, может запускаться самой ошибкой программирования, прерывание может запускаться при наличии определенного условия, которое оказывается выполненным для одного из периферийных устройств. Когда наступает такое условие, программа автоматически переходит по адресу β, запомненному в некоторой фиксированной ячейке α центрального процессора, в то время как происходит замена содержимого PSR на содержимое γ ячейки $\alpha + 2$. Содержимое ячеек α и $\alpha + 2$ интерпретируется далее как *вектор прерываний* (аналогично вектору захвата), а программа, которая начинается с адреса β, называется *программой прерывания* или *программой обработки прерываний* (аналогично программе захвата).

Основное различие между системным захватом и системным прерыванием состоит в том, что захват имеет место *всегда*, когда возникают ошибки определенного рода – они находятся за пределами возможности управления со стороны программиста. С другой стороны, прерываниями программист может управлять, прерывания могут быть "запрещены" либо временно, либо постоянно, если этого пожелает программист.

Мы покажем действие механизма прерывания на примере трех периферийных устройств: клавишного пульта телетайпа, печатающего устройства телетайпа и сетевого таймера. Как известно из разд. 2.3 и 2.4, каждое из этих устройств связано с определенным регистром состояния, 7-й разряд которого является разрядом готовности устройства, указывающим, что символ набран на клавишном пульте телетайпа или что печатающее устройство телетайпа свободно, или что таймер выдал отсчет. Теперь мы рассмотрим еще один разряд в указанном регистре состояния – разряд 6, который указывает на возможность прерывания и называется разрядом "возможности прерывания" (или INTR ENBL):



Каждое из этих трех устройств вызывает прерывание, когда разряд готовности в регистре состояния меняется с 0 на 1, при условии, что его разряд INTR ENBL установлен на 1. Таким образом, для того чтобы возникло прерывание, один из двух указанных разрядов должен содержать 1, а другой должен при этом изменяться с 0 на 1.

Векторы прерываний телетайпа и сетевого таймера располагаются следующим образом:

Устройство	Адрес вектора прерываний
Клавишный пульт телетайпа	60
Печатающее устройство телетайпа	64
Сетевой таймер	100

Программист должен позаботиться о том, чтобы установить разряд INTR ENBL на 1, если предполагается осуществить прерывание со стороны соответствующего устройства (этот разряд автоматически очищается, когда нажимается переключатель "начало" (start) в PDP-11 или когда осуществляется команда RESET (СБРОС). Конечно, как и в случае захватов, программист должен сформировать вектор прерываний и программу прерываний для каждого устройства.

В то время как программа захвата должна закончиться с командой HALT (ОСТАНОВ), программа прерывания почти всегда заканчивается возвратом управления прерванной программе. Этот возврат управления осуществляется по команде RTI (ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ), которая выполняет следующие действия (в одном цикле):

```
MOV (SP) +, PC   ; Переслать указатель системного стека в счетчик команд
MOV (SP) +, PSR  ; Переслать указатель системного стека в регистр состо-
                   яния процессора
```

Таким образом, RTI просто аннулирует действия, предпринятые механизмом прерывания непосредственно перед тем, как программа прерывания взяла на себя все функции (см. разд. 8.1).

Способ принятия и возврата управления программой прерывания напоминает способ принятия и возврата управления подпрограммой (при использовании команд JSP и RTS). Как программа прерывания, так и подпрограмма используют системный стек для хранения информации связи. Однако, наряду с адресом возврата, механизм прерывания обеспечивает хранение в стеке PSR прерванной программы.

Когда механизм прерывания обслуживает в программе больше одного устройства, всегда существует возможность того, что одна программа прерывания прерывается другой. Указанное "вложение" одних программ прерывания в другие в этом случае очень похоже на вложение подпрограмм. Фактически можно вкладывать друг в друга в произвольном порядке программы прерывания и подпрограммы без риска какой-либо путаницы.

8.5. Для чего используются прерывания?

Основное преимущество использования механизма прерывания состоит в том, что он избавляет от необходимости неоднократно проверять состояние (или "опрашивать") устройства, чтобы определить, достигнуты или нет опре-

деленные условия. Если устройство обслуживается программой прерываний, то факт наступления определенных условий будет обнаружен автоматически! Так как контроль периферийных устройств относительно долгая процедура, то механизм прерывания является важным средством экономии времени для программы. Например, в рассмотренных ранее программах мы писали

```
LOOP: TSTB @ # 177564
      BPL  LOOP
```

для того, чтобы убедиться, что печатающее устройство свободно. Если печатающее устройство обслуживается программой прерываний, то этот цикл становится излишним, а время, требуемое для проверки состояния печатающего устройства (порядка 1/10 с), может быть использовано для выполнения десятков тысяч полезных команд процессора.

Ниже приведены более подробные примеры. Причины выбора компонент γ вектора прерываний (200 в примере 1 и 300 в примере 2) для этих примеров будут объяснены в следующем разделе.

Примеры

1. Допустим, что мы хотели бы использовать телетайп как обычную пишущую машинку (т. е. иметь распечатку каждого набираемого на нем символа) в процессе выполнения основной программы. На рис. 8.2 приведена программа, показывающая, как это может быть сделано. Основная программа работает до тех пор, пока не будет набран символ, после чего она прерывается механизмом прерывания клавишного пульта и управление передается INTHND. Программа прерывания, которая начинается с INTHND, просто осуществляет распечатку символа. (Предполагается, что символы на клавиатуре набираются медленно и, следовательно, нет необходимости использовать буферизацию.)

Заметим, что основная программа возобновляет свою работу с тем же самым PSR, который она имела перед последним прерыванием. Это важный момент, так как прерывание могло наступить сразу после команд TST или

```
        .TITLE  INTYPE
;ПРОГРАММА РАСПЕЧАТКИ ВВОДИМОГО СИМВОЛА С
;ИСПОЛЬЗОВАНИЕМ ПРЕРЫВАНИЙ
LC=.
.=4+LC
        .WORD 6,0,12,0      ;УСТАНОВКА ВЕКТО-
                            ;РОВ ОШИБОК
.=60+LC
        .WORD INTHND,200    ;УСТАНОВКА ВЕКТОРА
                            ;ПРЕРЫВАНИЙ
.=500+LC                    ;РЕЗЕРВИРОВАНИЕ
                            ;ПАМЯТИ ДЛЯ СТЕКА
START:  MOV   PC,SP
        TST   -(SP)         ;НАЧАЛЬНАЯ УСТАНОВ-
                            ;КА SP
        MOV #100,@#177560;УСТАНОВКА 1 В РАЗ-
                            ;РЯДЕ INTR ENBL
```

Рис. 8.2. Программа распечатки вводимого с телетайпа символа INTYPE

```
;  ОСНОВНАЯ ПРОГРАММА
;                    \
;                     !
;                     !     ТЕКСТ
;                      >  ОСНОВНОЙ
;                     !   ПРОГРАММЫ
;                     !
;                    /
;
;ПРОГРАММА ОБРАБОТКИ ПРЕРЫВАНИЙ
INTHND: MOV @#177562,@#177566;ПЕЧАТЬ ВХОДНО-
                             ;ГО СИМВОЛА
        RTI                  ;ВОЗВРАТ ПОСЛЕ ПРЕ-
                             ;РЫВАНИЯ

        .END    START
```

Рис. 8.2 (окончание)

CMP, для которых сохранение кодов условий (а следовательно, и PSR существенно).

2. Допустим, что мы хотели бы, чтобы звонок звонил каждые 10 с во время работы основной программы. На рис. 8.3 приведена программа, которая показывает, как это может быть сделано. Основная программа прерывается сетевым таймером каждые 1/60 с. Программа прерывания уменьшает значение счетчика на 1 каждый раз, когда ей передается управление[1]; когда это значение становится равным нулю, оно снова восстанавливается до 600 (600/60 = 10) и звонок звенит. Таким образом, звонок звенит сразу же, как только программа начинает работу (так как начальное значение счетчика устанавливается равным 1) и далее через каждые 10 с. □

```
        .TITLE  INBELL
;ВО ВРЕМЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ КАЖДЫЕ
;10 С ЗВЕНИТ ЗВОНОК.
LC=.
.=4+LC
        .WORD   6,0,12,0        ;УСТАНОВКА ВЕК-
                                ;ТОРОВ ОШИБОК
.=100+LC
        .WORD   INTHND,300      ;УСТАНОВКА ВЕК-
                                ;ТОРА ПРЕРЫВАНИЙ
.=500+LC                        ;РЕЗЕРВИРОВАНИЕ
                                ;ПАМЯТИ ДЛЯ
                                ;СТЕКА
START:  MOV   PC,SP
        TST   -(SP)             ;НАЧАЛЬНАЯ УСТА-
                                ;НОВКА SP
```

Рис. 8.3. Программа INBELL

[1] Когда разряд INTR ENBL таймера содержит 1, то разряд готовности регистра состояния таймера сбрасывается автоматически на 0 после каждого отсчета таймера.

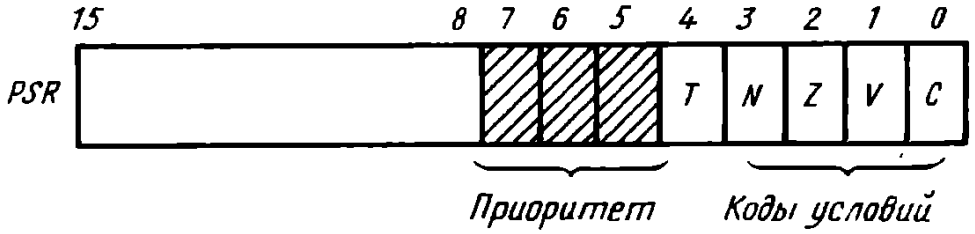
```
        MOV   #100,@#177546     ;УСТАНОВКА 1 В
                                ;РАЗРЯДЕ
                                ;       INTR ENBL
        MOV   #1,COUNT          ;УСТАНОВКА СЧЕТ-
                                ;ЧИКА ОТСЧЕТОВ
;
; ОСНОВНАЯ ПРОГРАММА
                \
                 !
                 !   ТЕКСТ ОСНОВНОЙ
                  >  ПРОГРАММЫ
                 !
                 !
                /
;
;  ПРОГРАММА ОБРАБОТКИ ПРЕРЫВАНИЙ
INTHND: DEC   COUNT             ;(COUNT)=(COUNT)
                                ;-1
        BEQ   RING              ;ЕСЛИ (COUNT)=0.,
                                ;ЗВЕНИТ ЗВОНОК
        RTI                     ;ВОЗВРАТ ПОСЛЕ
                                ;ПРЕРЫВАНИЯ
RING:   MOV   #7,@#177566       ;ЗВЕНИТ ЗВОНОК
                                ;(ASCII КОД 007)
        MOV   #600.,COUNT       ;УСТАНОВКА СЧЕТ-
                                ;ЧИКА НА 10 С
        RTI                     ;ВОЗВРАТ ПОСЛЕ
                                ;ПРЕРЫВАНИЯ
;
COUNT:  .BLKW  1                ;ОТСЧЕТ СЧЕТЧИКА
        .END   START
```

Рис. 8.3 (окончание)

Хотя описанные выше примеры скорее надуманные, они достаточно хорошо иллюстрируют тот факт, что с помощью прерываний ЭВМ может исполнить две программы "в параллель".

8.6. Приоритет прерываний

Для более глубокого понимания механизма прерываний следует более подробно познакомиться с *полем приоритета* регистра PSR, которое состоит из разрядов 5, 6 и 7:



Содержимое поля приоритета PSR называют *процессорным приоритетом.* Он может быть установлен самим программистом на любой уровень в диапа-

зоне от 0 до 7. Например, чтобы установить уровень процессорного приоритета равным 7, достаточно написать команду

```
·MOV  # 340, @ # 177776
```

Предполагается, что когда программа начинает работать, процессорный приоритет автоматически устанавливается равным 0.

Наряду с процессорным приоритетом обычно рассматривают приоритет периферийного устройства. Например, приоритеты клавишного пульта и печатающего устройства телетайпа равны 4, а приоритет сетевого таймера равен 6 (эти приоритеты зафиксированы изготовителем). Основное правило выполнения прерываний следующее: *устройство может прерывать программу только тогда, когда приоритет устройства строго больше, чем процессорный приоритет.* Например, телетайп и таймер могут прерывать любую программу, которая работает с процессорным приоритетом 0; с другой стороны, программа, выполняемая с процессорным приоритетом 7, не может прерываться ни одним из устройств!

Предположим, что два устройства D1 и D2, обслуживаемые программой прерываний и имеющие соответственно приоритеты p_1 и p_2, используются программой, которая работает с процессорным приоритетом p_0. Предположим, что $p_0 < p_1 < p_2$. Некоторое время спустя после начала работы программы D1 пытается осуществить прерывание. Так как $p_1 > p_0$, прерывание "подтверждается", и программа прерывания, инициированного D1, выполняется. Процессорный приоритет, при котором эта программа работает, определяется новым содержимым PSR, т. е. вторым компонентом (который мы назовем γ) вектора прерываний устройства D1. Обозначим через p_1' новый процессорный приоритет, формируемый из 5, 6 и 7 разрядов γ. Предположим теперь, что в то время как прерывание от устройства D1 еще "обслуживается", устройство D2 инициирует прерывание. Будет это прерывание подтверждено или нет, зависит от соотношения p_2 и p_1'. Если $p_2 > p_1'$, то программа прерываний, инициируемая D2, выполняется до того, как управление будет возвращено (вместе с RTI) программе прерывания D1. Если $p_2 \leqslant p_1'$, то прерывание от D2 должно быть оставлено в очереди на обслуживание до тех пор, пока не завершится программа прерываний от D1, управление не вернется (вместе с RTI) к основной программе и рассматриваемый приоритет вновь не понизится до p_0.

Таким образом, присваивая соответствующее значение для γ в векторах прерываний, программист получает возможность определять заранее порядок обслуживания программных прерываний. Конечно, процессорный приоритет можно изменить не только с помощью изменения содержимого векторов прерываний; его можно всегда изменить, написав команду MOV # CONST, @ # 177776, где разряды 5, 6 и 7 операнда CONST соответствуют желаемому приоритету.

В большинстве случаев процессорный приоритет, при котором мы хотим начать программу прерываний устройства, одинаков с приоритетом указанного устройства. Например, предположим, что мы хотим написать программу, которая позволяет использовать телетайп как пишущую машинку во время выполнения основной программы, а также заставляет звонок звонить каждые 10 с (комбинация заданий в примерах 1 и 2 разд. 8.5). Мы, конечно, хотим, чтобы основная программа имела более низкий приоритет, чем кла-

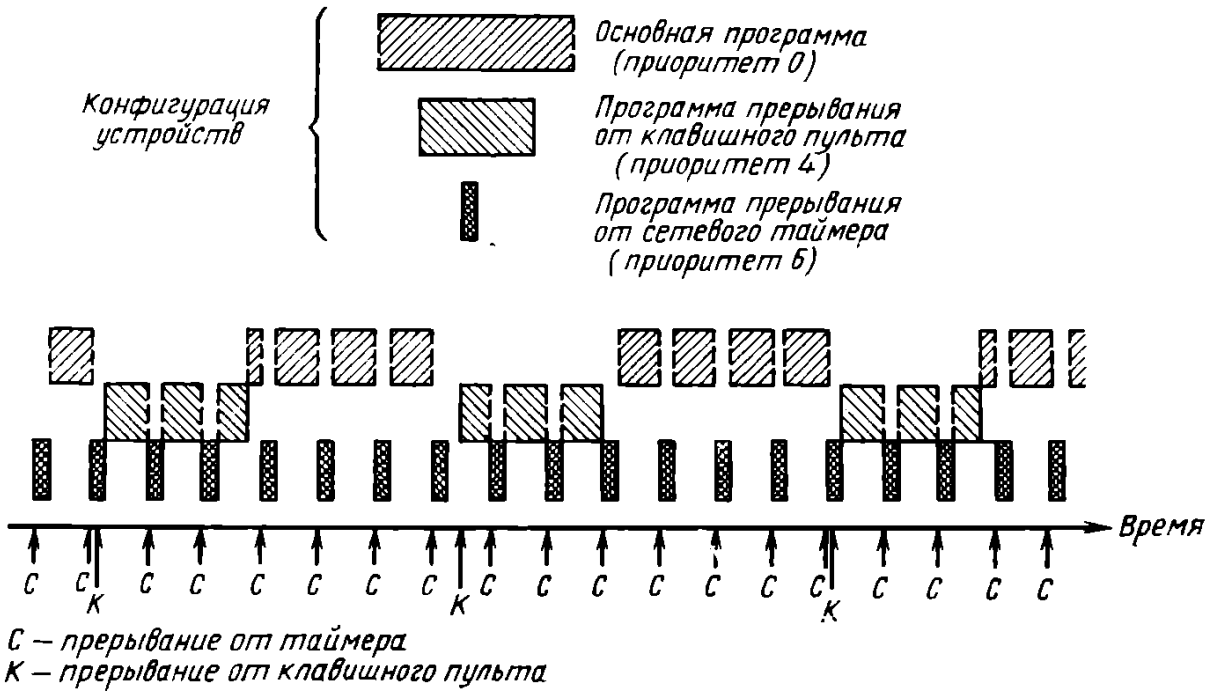


Рис. 8.4. Приоритет прерываний

вишный пульт телетайпа или таймер. Мы также хотим, чтобы таймер имел более высокий приоритет по сравнению с клавишным пультом, так как прерывания таймера каждые 1/60 с являются существенными, если все отсчеты должны быть сохранены. Таким образом, получаем следующее распределение приоритетов:

Программа	Процессорный приоритет
Основная программа	0
Программа обработки прерываний от клавишного пульта	4
Программа обработки прерываний от таймера	6

Нулевой процессорный приоритет основной программы, как правило, имеет место в начале работы программы; процессорный приоритет 4 для программы обработки прерываний от клавишного пульта можно задать, занеся 200 по адресу 62; процессорный приоритет 6 для программы обработки прерываний от таймера можно установить, занеся 300 по адресу 102. На рис. 8.4 показано, как время процессора распределяется между основной программой, клавишным пультом и таймером.

8.7. Пример: временной запрос

В заключение рассмотрим программу, которая иллюстрирует обработку прерывания от двух устройств, манипуляцию процессорными приоритетами и пример вложения подпрограмм (включающий рекурсивную подпрограмму) и программ прерывания.

Программа (рис. 8.5) начинается вопросом к пользователю: "Который час?" – на что пользователь отвечает указанием точного времени в форме XXYY (например, 1143, если точное время равно 11:43). После этого, всякий раз, когда пользователь набирает символ, программа распечатывает сообщение: В МОМЕНТ ЗВОНКА ВРЕМЯ БУДЕТ:, после чего указывается точное время в форме ЧЧ:ММ:СС (часы:минуты:секунды) и дается звонок.

```
        .TITLE TIME
;В ОТВЕТ НА ПРОГРАММНЫЙ ЗАПРОС "КОТОРЫЙ ЧАС?"
;ПОЛЬЗОВАТЕЛЬ УСТАНАВЛИВАЕТ ПОКАЗАНИЕ ВНУТ-
;РЕННЕГО ТАЙМЕРА, ПЕЧАТАЯ ВРЕМЯ В ВИДЕ
;4-РАЗРЯДНОГО ЧИСЛА XXYY. ПОСЛЕ ЭТОГО, КАК
;ТОЛЬКО С КЛАВИАТУРЫ ВВОДИТСЯ СИМВОЛ, ПРОГ-
;РАММА РАСПЕЧАТЫВАЕТ СООБЩЕНИЕ "В МОМЕНТ
;ЗВОНКА ВРЕМЯ БУДЕТ:", ЗА КОТОРЫМ СЛЕДУЮТ ОТ-
;СЧЕТ ВРЕМЕНИ В ФОРМАТЕ HH:MM:SS И ЗВОНОК.
LC=.
.=4+LC
        .WORD  6,0,12,0     ;УСТАНОВКА ВЕКТО-
                            ;РОВ ОШИБОК
.=60+LC
        .WORD  KBINT,340    ;УСТАН. ВНУТ. ВЕК-
                            ;ТОРА КЛАВИШ. ПУЛЬ-
                            ;ТА (ПРИОРИТЕТ 7)
.=100+LC
0       .WORD  CLINT,300    ;УСТАНОВКА ВНУТ.ВЕ-
                            ;КТОРА ТАЙМЕРА
                            ;(ПРИОРИТЕТ 6)
.=500+LC                    ;РЕЗЕРВИРОВАНИЕ
                            ;ПАМЯТИ ДЛЯ СТЕКА
START:  MOV  PC,SP
        TST  -(SP)          ;НАЧАЛЬНАЯ УСТАНОВ-
                            ;КА УКАЗАТЕЛЯ СТЕ-
                            ;КА (SP)
;
KBSTAT=177560
KBDATA=177562
PRSTAT=177564
PRDATA=177566
CLSTAT=177546
;
;ПЕЧАТЬ ЗАПРОСА (QUERY)
        MOV  #QUERY,R0      ;УСТАНОВКА ПАРАМЕТ-
                            ;РОВ
        MOV  #ENDQ,R1       ;ДЛЯ ПЕЧАТИ ПОДПРО-
                            ;ГРАММЫ
        JSR  PC,PRINT       ;ПЕЧАТЬ LF,CR И
                            ;ТЕКСТА ЗАПРОСА
;ВРЕМЯ ПРИЕМА И НАЧАЛА РАСПЕЧАТКИ XXYY
        MOV  #4,R2          ;(R2)=ЦИФРОВОЙ
                            ;ОТСЧЕТ
        MOV  #ITIME,R0      ;УСТАНОВКА ПАРАМЕТ-
```

Рис. 8.5. Программа временного запроса

```
                                ;РОВ
NEXTD:  MOV   R0,R1             ;ДЛЯ ПЕЧАТИ ПОДПРО-
                                ;ГРАММЫ
        TSTB KBSTAT             ;СИМВОЛ ВВЕДЕН?
        BPL  .-4                ;ЕСЛИ НЕТ, ПРОДОЛ-
                                ;ЖАТЬ ПРОВЕРКУ
        MOVB KBDATA,(R0)        ;ЕСЛИ ДА,ЗАПОМНИТЬ
                                ;ЦИФРОВОЙ ОТСЧЕТ В
                                ;МАССИВЕ ITIME
        BICB #200,(R0)          ;УДАЛЕНИЕ РАЗРЯДА
                                ;КОНТРОЛЯ ИЗ ЦИФРО-
                                ;ВОГО ОТСЧЕТА
        JSR  PC,PRINT           ;ПЕЧАТЬ ОТСЧЕТА
        INC  R0                 ;(R0)=(R0)+1
        DEC  R2                 ;(R2)=(R2)-1
        BNE  NEXTD              ;ЕСЛИ (R2) НЕ НУЛЕ-
                                ;ВОЕ,ПРИНЯТЬ СЛЕДУ-
                                ;ЮЩИЙ ОТСЧЕТ
;ПРЕОБРАЗОВАНИЕ НАЧАЛЬНОГО ЗНАЧЕНИЯ ЧАСОВ
;(XX) В ДВОИЧНУЮ ФОРМУ
        MOVB ITIME+1,R0         ;УСТАНОВКА ПАРАМЕТ-
                                ;РОВ
        MOVB ITIME,R1           ;ДЛЯ ПОДПРОГРАММЫ
                                ;INCON
        JSR  PC,INCON           ;ПРЕОБРАЗОВАНИЕ XX
                                ;В ДВОИЧНУЮ ФОРМУ
        MOV  R2,HOUR            ;И ЗАПОМИНАНИЕ В
                                ;ЯЧЕЙКЕ HOUR (ЧАСЫ)
;ПРЕОБРАЗОВАНИЕ НАЧАЛЬНОГО ЗНАЧЕНИЯ МИНУТ
;(YY) В ДВОИЧНУЮ ФОРМУ
        MOVB ITIME+3,R0         ;УСТАНОВКА ПАРАМЕТ-
                                ;РОВ
        MOVB ITIME+2,R1         ;ДЛЯ ПОДПРОГРАММЫ
                                ;INCON
        JSR  PC,INCON           ;ПРЕОБРАЗОВАНИЕ YY
                                ;В ДВОИЧНУЮ ФОРМУ
        MOV  R2,MIN             ;И ЗАПОМИНАНИЕ В
                                ;ЯЧЕЙКЕ MIN (МИНУ-
                                ;ТЫ)
;УСТАНОВКА РАЗРЯДОВ ВОЗМОЖНОСТИ ПРЕРЫВАНИЯ
;НА 1 И ОЖИДАНИЕ
        MOV  #100,KBSTAT        ;УСТАН. РАЗР. ВОЗ-
                                ;МОЖН. ПРЕРЫВАНИЯ
                                ;ОТ КЛАВИШН. ПУЛЬТА
                                ;НА 1
        MOV  #100,CLSTAT        ;УСТАН. РАЗР. ВОЗ-
                                ;МОЖН. ПРЕРЫВАНИЯ
                                ;ОТ ТАЙМЕРА НА 1
LOOP:   BR   LOOP               ;ОЖИДАНИЕ ПРЕРЫВА-
                                ;НИЙ
;       ОБРАБОТКА ПРЕРЫВАНИЙ ОТ ТАЙМЕРА
;ОБНОВЛЕНИЕ ВРЕМЕНИ КАЖДЫЕ 1/60 СЕК
CLINT:  MOV  #TICK,R4           ;УСТАНОВКА ПАРАМЕТ-
                                ;РОВ ДЛЯ ОБНОВЛЕ-
                                ;НИЯ S.R.
```

Рис. 8.5 (продолжение)

```
        JSR   PC,UPDATE     ;ОБНОВЛЕНИЕ ПОКАЗА-
                            ;НИЯ ТАЙМЕРА
        CMP   HOUR,#12.     ;(HOUR)=12. ИЛИ
                            ;МЕНЬШЕ?
        BLE   EXIT3         ;ЕСЛИ ДА, ОБНОВЛЕ-
                            ;НИЕ ВРЕМЕНИ ЗАКАН-
                            ;ЧИВАЕТСЯ
        SUB   #12.,HOUR     ;ЕСЛИ НЕТ, СКОРРЕК-
                            ;ТИРОВАТЬ ПОКАЗАНИЯ
                            ;ТАЙМЕРА С ПЕРИОДОМ
                            ;12 ЧАСОВ
EXIT3:  RTI                 ;ВОЗВРАТ ИЗ ПРЕРЫ-
                            ;ВАНИЯ
;     ОБНОВЛЕНИЕ (РЕКУРСИВНАЯ ПОДПРОГРАММА)
;ОБНОВЛЕНИЕ ОТСЧЕТОВ, СЕКУНД, МИНУТ И ЧАСОВ.
;АДРЕС ОБНОВЛЕННОГО ПОЛЯ ХРАНИТСЯ В R4.
UPDATE: INC   (R4)          ;((R4))=((R4))+1
        CMP   (R4),#60.     ;((R4))=60.?
        BNE   EXIT4         ;ЕСЛИ НЕТ, ОБНОВЛЕ-
                            ;НИЕ ЗАВЕРШАЕТСЯ
        CLR   (R4)          ;ЕСЛИ ДА, ((R4))=0
                            ;(СБРОС СЧЕТЧИКА)
        TST   -(R4)         ;(R4)=(R4)-2 (ПЕРЕ-
                            ;ХОД К СЛЕДУЮЩЕМУ
                            ;ПОЛЮ)
        JSR   PC,UPDATE     ;ОБНОВЛЕНИЕ СЛЕДУЮ-
                            ;ЩЕГО ПОЛЯ
EXIT4:  RTS   PC            ;ВЫХОД
;   ОБРАБОТКА ПРЕРЫВАНИЙ ОТ КЛАВИШНОГО ПУЛЬТА
;РАСПЕЧАТКА ВРЕМЕНИ, КАК ТОЛЬКО СИМВОЛ ВВЕДЕН
KBINT:  MOV   #TEMP,R0      ;СОХРАНЕНИЕ ПОСЛЕД-
                            ;НИХ ПОКАЗАНИЙ:
        MOV   HOUR,(R0)+    ;ЧАС (HOUR), МИН
                            ;(MIN) И СЕК (SEK)
        MOV   MIN,(R0)+     ;В МАССИВЕ TEMP
        MOV   SEK,(R0)      ;ДЛЯ ЗАЩИТЫ ОТ
                            ;CLINT.
        CLR   @#177776      ;ПОНИЖЕНИЕ ПРИОРИ-
                            ;ТЕТА ДЛЯ ПРИНЯТИЯ
                            ;CLINT
;ПЕЧАТЬ СООБЩЕНИЯ
        MOV   #MESSG,R0     ;УСТАНОВКА ПАРАМЕТ-
                            ;РОВ
        MOV   #ENDM,R1      ;ДЛЯ ПОДПРОГРАММЫ
                            ;OUTCON
        JSR   PC,PRINT      ;ПЕЧАТЬ LF,CR И
                            ;ТЕКСТА СООБЩЕНИЯ
;ПРЕОБРАЗОВАНИЕ ЧАС (HOUR), МИН (MIN) И
;СЕК (SEK) В КОД ASCII
        MOV   #TEMP,R2      ;УСТАНОВКА ПАРАМЕТ-
                            ;РОВ
        MOV   #OUTPUT,R3    ;ПОДПРОГРАММЫ
                            ;OUTCON
        JSR   PC,OUTCON     ;ПРЕОБРАЗОВАНИЕ ЧАС
```

Рис. 8.5 (продолжение)

```
                                ;(HOUR) В КОД ASCII
                                ;(HH)
        JSR   PC,OUTCON         ;ПРЕОБРАЗОВАНИЕ МИН
                                ;(MIN) В КОД ASCII
                                ;(MM)
        JSR   PC,OUTCON         ;ПРЕОБРАЗОВАНИЕ СЕК
                                ;(SEC) В КОД ASCII
                                ;(SS)
;РАСПЕЧАТКА HH:MM:SS: И ЗВОНОК
        MOV   #OUTPUT,R0        ;УСТАНОВКА ПАРАМЕТ-
                                ;РОВ
        MOV   #ENDO,R1          ;ДЛЯ ПЕЧАТИ ПОДПРО-
                                ;ГРАММЫ
        JSR   PC,PRINT          ;ПЕЧАТЬ ВЫХОДНОГО
                                ;МАССИВА
        TST   KBDATA            ;ОЧИСТКА РАЗРЯДА
                                ;ГОТОВНОСТИ В
                                ;KBSTAT
        RTI                     ;ВОЗВРАТ ИЗ ПОДПРО-
                                ;ГРАММЫ
;       ПЕЧАТЬ
;ПЕЧАТЬ СТРОКИ СИМВОЛОВ, НАЧИНАЯ С (R0) И
;КОНЧАЯ (R1). ИЗМЕНЯЕМ ТОЛЬКО R5
PRINT:  MOV   R0,R5             ;(R5)=ИНДЕКСУ МАС-
                                ;СИВА СИМВОЛОВ
AGAIN:  CPM   R5,R1             ;СТРОКА КОНЧИЛАСЬ?
        BHI   EXIT1             ;ЕСЛИ ДА, ВЫХОД
        TSTB  PRSTAT            ;ПЕЧАТАЮЩЕЕ УСТРОЙ-
                                ;СТВО ГОТОВО?
        BPL   .-4               ;ЕСЛИ НЕТ, ПРОДОЛ-
                                ;ЖИТЬ КОНТРОЛЬ
        MOVB  (R5)+,PRDATA      ;ЕСЛИ ДА, ПЕЧАТЬ
                                ;((R5)). (R5)=(R5)
                                ;+1
        BR    AGAIN             ;СЧИТЫВАНИЕ СЛЕДУЮ-
                                ;ЩЕГО СИМВОЛА
EXIT1:  RTS   PC                ;ВЫХОД
;
;       INCON
;ПРЕОБРАЗОВАНИЕ ДВУХРАЗРЯДНОГО ДЕСЯТИЧНОГО
;ЧИСЛА ИЗ КОДА ASCII (В R0- ЕДИНИЦЫ, В R1-
;ДЕСЯТКИ) В ДВОИЧНЫЙ КОД. РЕЗУЛЬТАТ
;РАЗМЕЩАЕТСЯ В R2,R3,R4 И R5.
INCON:  BIC   #177760,R0        ;ПРЕОБРАЗОВАНИЕ
                                ;(R0) В ДВОИЧНЫЙ
                                ;КОД
        MOV   R0,R2             ;И ЗАПОМИНАНИЕ В R2
TENS:   CMPB  R1,#'0            ;(R1)='0? (ЕСТЬ ЛИ
                                ;ДЕСЯТКИ?)
        BEQ   EXIT2             ;ЕСЛИ НЕТ, ВЫХОД
        ADD   #10.,R2           ;ЕСЛИ ДА, (R2)=(R2)
                                ;+10 ДЕСЯТИЧНОЕ
        DEC   R1                ;(R1)=(R1)-1 (НА
                                ;ОДИН ДЕСЯТОК МЕНЬ-
                                ;ШЕ)
```

Рис. 8.5 (продолжение)

```
        BR    TENS              ;СНОВА КОНТРОЛЬ ДЕ-
                                ;СЯТКОВ
EXIT2:  RTS   PC                ;ВЫХОД
;
;       OUTCON
;ПРЕОБРАЗОВАНИЕ ДВОИЧНОГО ЧИСЛА N (ОТ 0 ДО 60
;ДЕСЯТИЧНОЕ) В ДВУХРАЗРЯДНЫЙ НОМЕР РО В КОДЕ
;ASCII. АДРЕС N РАВЕН (R2). АДРЕСА Р И О РАВ-
;НЫ (R3) И (R3)+1. ПЕРЕД ВЫХОДОМ СОДЕРЖИМОЕ
;R1 УВЕЛИЧИВАЕТСЯ НА 2, А R3 НА 3. R4 И R5
;ОСТАЮТСЯ НЕИЗМЕННЫМИ.
OUTCON: MOV   (R2)+,R0          ;(R0)=ДВОИЧНОМУ
                                ;ЧИСЛУ (ЧАС,МИН,СЕК)
        CLR   R1                ;УСТАНОВКА ДЕСЯТКОВ
MORE:   CMP   R0,#10.           ;ЕСТЬ ЛИ ДЕСЯТКИ В
                                ;R0?
        BLT   UNITS             ;ЕСЛИ НЕТ, ОБРАБО-
                                ;ТАТЬ ЕДИНИЦЫ
        INC   R1                ;ЕСЛИ ДА, (R1)+1
                                ;(ЕЩЕ ОДИН ДЕСЯТОК)
        SUB   #10.,R0           ;(R0)=(R0)-10 ДЕСЯ-
                                ;ТИЧНОЕ
        BR    MORE              ;КОНТРОЛЬ СЛЕДУЮЩИХ
                                ;ДЕСЯТКОВ
UNITS:  ADD   #'0,R1            ;ПРЕОБРАЗОВАНИЕ ДЕ-
                                ;СЯТКОВ В КОД ASCII
        ADD   #'0,R0            ;ПРЕОБРАЗОВАНИЕ
                                ;ЕДИНИЦ В КОД ASCII
        MOVB  R1,(R3)+          ;ЗАПОМИНАНИЕ ДЕСЯТ-
                                ;КОВ В ВЫХОДНОМ
                                ;МАССИВЕ
        MOVB  R0,(R3)+          ;ЗАПОМИНАНИЕ ЕДИНИЦ
                                ;В ВЫХОДНОМ МАССИВЕ
        INC   R3                ;ПРОПУСК БАЙТА
                                ;С ДВОЕТОЧИЕМ
        RTS   PC                ;ВЫХОД
;
;ПАМЯТЬ ДЛЯ КОНСТАНТ И ПРОМЕЖУТОЧНЫХ РЕЗУЛЬ-
;ТАТОВ
;
QUERY:  .BYTE 15,12             ;CR, LF
        .ASCII /WHAT TIME IS IT?/;ТЕКСТ ЗАП-
                                ;РОСА (КОТОРЫЙ
                                ;ЧАС?)
ENDQ:   .ASCII / /              ;КОНЕЦ ЗАПРОСА
                                ;(ПРОБЕЛ)
;
MESSG:  .BYTE 15,12             ;CR, LF
        .ASCII /AT THE BELL TIME WILL BE:/
                                ;ТЕКСТ СООБЩЕНИЯ
                                ;("В МОМЕНТ ЗВОНКА
                                ;ВРЕМЯ БУДЕТ:")
ENDM:   .ASCII / /              ;КОНЕЦ СООБЩЕНИЯ
                                ;(ПРОБЕЛ)
;
```

Рис. 8.5 (продолжение)

```
OUTPUT: .ASCII /HH:MM:SS/ ;ЗАПОМИНАНИЕ
                          ;HH:MM:SS
ENDO:   .BYTE 7           ;КОНЕЦ OUTPUT (ЗВО-
                          ;НОК)
;
ITIME:  .BLKB 4           ;ПАМЯТЬ ДЛЯ НА-
                          ;ЧАЛЬНОГО ВРЕМЕНИ
                          ;(XXYY)
;
        .EVEN             ;КОНТРОЛЬ ГРАНИЦЫ
                          ;СЛОВА
HOUR:   .BLKW 1           ;ПАМЯТЬ ДЛЯ ЧАСОВ
                          ;(ДВОИЧНОЕ)
MIN:    .BLKW 1           ;ПАМЯТЬ ДЛЯ МИНУТ
                          ;(ДВОИЧНОЕ)
SEK:    .WORD 0           ;ПАМЯТЬ ДЛЯ СЕКУНД
                          ;(ДВОИЧНОЕ)
TICK:   .WORD 0           ;ПАМЯТЬ ДЛЯ СЧЕТ-
                          ;ЧИКА
                          ;(ДВОИЧНОЕ)
TEMP:   .BLKW 3           ;ПАМЯТЬ ДЛЯ ВРЕМ.
                          ;ХРАНЕНИЯ ЧАСОВ,
                          ;МИНУТ, СЕКУНД
;
        .END  START
```

Рис. 8.5 (окончание)

Основная программа (функционирующая с процессорным приоритетом 0) состоит из начального запроса и последующей распечатки с преобразованием в двоичный код ответа пользователя XXYY. Затем выполняется бесконечный цикл (LOOP:BR LOOP), который фактически можно заменить любой полезной для пользователя программой (при условии, что ее прогон осуществляется с процессорным приоритетом 0). Программа прерываний от таймера CLINT прерывает основную программу каждые 1/60 с, чтобы изменить показания таймера, хранящиеся в ячейках HOUR (ЧАСЫ), MIN (МИН), SEC (СЕК) и TICK (ОТСЧЕТ). Программа CLINT увеличивает процессорный приоритет с 0 до 6 (т. е. изменяет компоненты γ вектора прерываний от таймера) и, следовательно, не может прерываться программой прерываний от клавишного пульта (приоритет которой равен 4). Когда вводится символ, то управление программе прерываний от каждого пульта KBINT может быть передано либо сразу, если в этот момент выполняется основная программа, либо сразу после CLINT. Подпрограмма KBINT выполняет следующие функции: распечатывает сообщение В МОМЕНТ ЗВОНКА и т. д.; преобразует из двоичного в код ASCII, распечатывает содержимое ячеек HOUR, MIN, SEC и дает звонок — все это осуществляется в то время, когда подпрограмма CLINT осуществляет обслуживание временного канала модификацией каждые 1/60 с содержимого ячеек HOUR, MIN, SEC и TICK. Так как HOUR, MIN и SEC могут изменяться в то время, когда подпрограмма KBINT осуществляет обработку их содержимого, то первое, что должна сделать подпрограмма KBINT, это записать их в ячейки промежуточного массива для времен-

ного хранения (массив TEMP). Это должно быть сделано без прерывания программы CLINT, а следовательно, при наличии процессорного приоритета, равного по крайней мере 7 (7 выбирается и присваивается компонентам γ вектора прерываний клавишного пульта). Как только перезапись в TEMP закончена, KBINT понижает процессорный приоритет до 0 (с помощью команды CLR @ # 177776), переходя в режим готовности быть прерванной подпрограммой CLINT каждые 1/60 с, как и требуется.

Ядром подпрограммы CLINT является рекурсивная подпрограмма UPDATE, аргумент которой представляет собой адрес Z ячейки памяти, содержащей положительное число, меньшее 60_{10}. Подпрограмма UPDATE вызывается с Z = TICK, SEC, MIN и HOUR в указанном порядке. Она увеличивает (Z) на 1 и проверяет, стало ли увеличенное (Z) равным 60; если нет, то UPDATE осуществляет выход; в противном случае UPDATE сбрасывает (Z) на 0 и вызывает саму себя со следующим адресом Z. Например,

	HOUR	MIN	SEC	TICK
Начальные значения	12	59	59	59
После 1-го вызова UPDATE	12	59	59	00
После 2-го вызова UPDATE	12	59	00	00
После 3-го вызова UPDATE	12	00	00	00
После 4-го вызова UPDATE	13	00	00	00
После модификации с помощью CLINT	01	00	00	00

Для преобразования входов XX и YY из кода ASCII в двоичный код основная программа использует подпрограмму INCON. Для двухразрядных чисел эта подпрограмма функционирует так же, как это имеет место в подпрограмме ASCTOBIN (см. разд. 7.7).

Заметим, что перед возвратом KBINT должна сбросить на 0 разряд готовности в KBSTAT, так чтобы можно было подтвердить последующее прерывание от клавишного пульта. Это можно сделать с помощью ссылки на программу KBDATA (т. е. используя TST KBDATA).

Следует помнить, что директивы .WORD и .BLKW (аналогичные тем, что появляются в конце программы) должны соответствовать четным адресам. Следовательно, чтобы не было сомнений (например, после использования директив .BYTE, .BLKB или .ASCII), нужно предварительно использовать директиву .EVEN.

Упражнения

8.1. Приведенная ниже программа выполняется на модели PDP-11 с памятью 8K слов. Чему равно содержимое R0, SP, PC и системного стека после останова программы?

```
LC=.
. = 4+LC
        .WORD  X,0
. = 500+LC
START:  MOV    PC,SP
        TST    -(SP)
        CLR    R0
LOOP:   TST    (R0)+
        BR     LOOP
X:      SUB    #2,R0
        HALT
        .END   START
```

8.2. Следующая программа состоит из основной программы и программы прерывания от таймера (начинающейся с CLINT).

```
LC=.
.=100+LC
        .WORD CLINT,300 ;УСТАНОВКА ВЕКТОРА
                        ;ПРЕРЫВАНИЙ
.=500+LC                ;РЕЗЕРВИРОВАНИЕ ПА-
                        ;МЯТИ ДЛЯ СТЕКА
START:  MOV  PC,SP
        TST  -(SP)      ;УСТАНОВКА SP В НА-
                        ;ЧАЛЬНОЕ СОСТОЯНИЕ
        MOV #100,@#177546 ;УСТАНОВКА 1 В
                        ;РАЗРЯД INTR ENBL
LOOP:   BR   LOOP       ;ОЖИДАНИЕ ПРЕРЫВАНИЯ
                        ;ОТ ТАЙМЕРА
NOW:    MOV  R0,R1      ;СОХРАНЕНИЕ СОДЕРЖИ-
                        ;МОГО R0 В R1 ПОСЛЕ
                        ;ПРЕРЫВАНИЯ
        HALT

;
CLINT: <--------------- ПОМЕСТИТЬ ЗДЕСЬ
                        ОДНУ КОМАНДУ
        RTI             ;ВОЗВРАТ ПОСЛЕ ПРЕРЫ-
                        ;ВАНИЯ
        .END
```

Вставьте пропущенную команду так, чтобы выполнить следующее: за первым возвратом после прерывания выполнение основной команды возобновляется с метки NOW, а не LOOP, причем содержимое SP устанавливается равным 500.

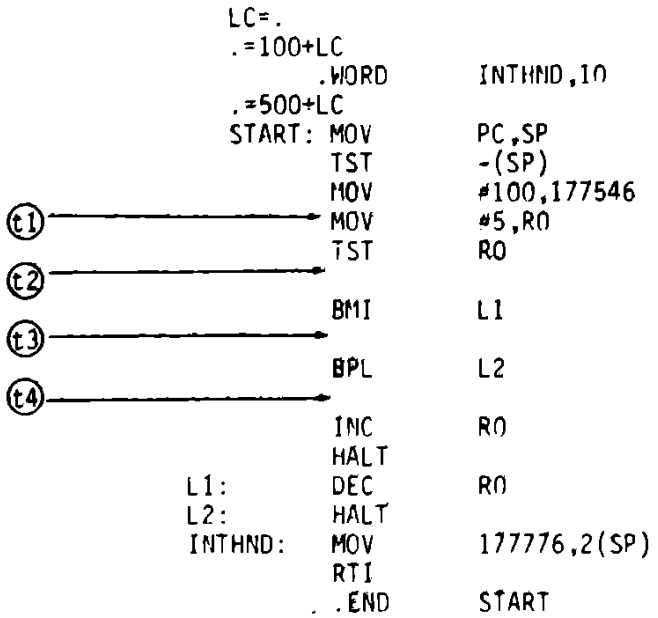
8.3. В следующей программе определить содержимое PC, SP и системного стека. (*Замечание: 177777 является недопустимой командой!*)

```
LC=.
.=4+LC
        .WORD  6,0,12,0
.=500+LC
START:  MOV    PC,SP
        TST    -(SP)
        CLR    177776
        JSR    PC,SUB
        HALT
SUB:    MOV    #123123,-(SP)
        ASR    (SP)
        .WORD  177777
        RTS    PC
        .END   START
```

8.4. Показать, что следующая программа в конце концов остановится. Чему будет равно содержимое PC, PSR и системного стека?

```
LC=.
.=10+LC
      .WORD      514,340
.=100+LC
      .WORD      100,300
.=500+LC
START: MOV       PC,SP
       TST       -(SP)
       MOV       #100,@#177546
LOOP:  BR        LOOP
       HALT
       .END      START
```

8.5. Рассмотрите следующую программу, содержащую программу прерывания от таймера INTHND:




```
            LC=.
            .=100+LC
                  .WORD     INTHND,10
            .=500+LC
            START: MOV      PC,SP
                   TST      -(SP)
                   MOV      #100,177546
                   MOV      #5,R0
                   TST      R0
                   BMI      L1
                   BPL      L2
                   INC      R0
                   HALT
            L1:    DEC      R0
            L2:    HALT
            INTHND: MOV     177776,2(SP)
                   RTI
                  .END      START
```

Чему равно содержимое R0, когда программа остановится, если первое прерывание от таймера произойдет в следующих точках:

а) точке t1, б) точке t2, в) точке t3, г) точке t4?

8.6. Устройства D1, D2, D3 и D4 имеют приоритеты устройств: 4, 5, 6 и 7 соответственно; адреса ячеек соответствующих векторов прерываний равны: 60, 100, 170, 174.

Рассмотрите следующую программу:

```
LC=.
.=4+LC
      .WORD    6,0
.=60+LC
      .WORD    INT1,200
.=100+LC
      .WORD    INT2,240
.=170+LC
      .WORD    INT3,0
.=174+LC
      .WORD    INT4,340
.=500+LC
START:          -
                -          } В SP первоначально устанавливается 500, а разряд
                -            INTR ENBL во всех устройствах устанавливается в 1
ENTER1: MOV  #100,177776
                -
                -          } Подпрограмма ENTER1
                -
ENTER2: MOV  #300,177776
                -
                -          } Подпрограмма ENTER2
                -
INT1:
                -
                -          } Подпрограмма INT1
                -
```

```
INT2:
                -
                -          } Подпрограмма  INT2
                -
INT3:
                -
                -          } Подпрограмма  INT3
                -
INT4:
                -
                -          } Подпрограмма  INT4
                -
          .END START
```

Для каждой из шести подпрограмм (ENTER1, ENTER2, INT1, INT2, INT3, INT4) определить, какое из шести устройств может вызвать прерывание.

8.7. Используя те же устройства D1, D2, D3 и D4, определенные в упр. 8.6, модифицировать программу следующим образом:

ENTER1 может быть прервана от D3 и D4;
ENTER2 может быть прервана от D2, D3 и D4;
INT1 может быть прервана от D3 и D4;
INT2 может быть прервана от D1, D2, D3 и D4;
INT3 не может быть прервана ни одним из указанных устройств;
INT4 может быть прервана от D1, D2, D3 и D4.

8.8. Написать программы захвата для случаев ошибок из-за неправильной адресации и недопустимой команды, которые осуществляют распечатку типа совершенной ошибки и содержимое PC в момент возникновения ошибки.

8.9. Программа INTYPE на рис. 8.2 была написана в предположении, что набор символов на телетайпе осуществляется достаточно медленно и не требуется входного буфера. Пересмотреть программу для случая, когда это предположение не выполняется.

8.10. Написать и отладить программу, выполняющую следующее: во время выполнения основной программы (моделируемой с помощью LOOP: BR LOOP) каждые 10 с звенит звонок; при нажатии клавиши клавишного пульта выполнение основной программы и звонок прекращаются и происходит распечатка последовательных байтов памяти в кодах ASCII, начиная с адреса, хранящегося в R0; при новом нажатии на клавишу распечатка прекращается и выполнение основной программы и звонок возобновляются. Этот процесс может быть повторен любое число раз.

Включить в указанную программу программу прерывания от таймера CLINT для подачи звонков и программу прерывания от клавишного пульта KBINT для печати байтов в (R0), (R0) + 1, (R0) + 2, . . . Как CLINT, так и KBINT должны иметь возможность прерывать основную программу; CLINT не может прерывать KBINT.

8.11. Выбрав произвольную программу, предложите метод измерения времени ее исполнения. Насколько точен этот метод?

Глава 9

АССЕМБЛЕР И РЕДАКТОР СВЯЗЕЙ

Языки высокого уровня, такие как Фортран, Бейсик, Паскаль и другие, имеют то преимущество, что позволяют пользователю облегчить написание сложных программ, не загромождая их деталями обслуживающих программ

[например, можно легко включить в программу цикл, используя команду (оператор цикла) DO 10 I = 1, 100, 3]. Сравнение показывает, что программы на языках высокого уровня легче составлять, воспринимать, отлаживать и обслуживать, чем программы на языке ассемблера. Чем же тогда объяснить необходимость использования ассемблера? Причина заключается в том, что, в отличие от программ на языке ассемблера, программы, написанные на языках высокого уровня, редко позволяют использовать все преимущества организации (памяти, процессора и процедур ввода-вывода) конкретных ЭВМ, на которых их предполагается выполнять. Следовательно, такие программы – даже если они транслируются высококачественными компиляторами – не всегда достаточно эффективны. Таким образом, в случаях, когда скорость выполнения программы особенно важна (как это обычно требуется от обслуживающих программ), приходится возвращаться к языку ассемблера. Программы придется писать на языке ассемблера естественно и тогда, когда ЭВМ, доступная пользователю, просто не имеет (или не может быть обеспечена) трансляторов с языков высокого уровня.

В этой главе мы опишем две программы, которые дают возможность программировать на языке ассемблера: ассемблер и редактор связей. Хотя ниже будут рассматриваться программы MACRO-11 и LINKR-11 мини-ЭВМ PDP-11, общие выводы, сделанные при этом, применимы к другим ассемблерам и редакторам связей, используемым в настоящее время.

9.1. Двухпроходный процесс ассемблирования

Как мы уже знаем, основное различие между языком ассемблера и машинным языком состоит в том, что в языке ассемблера используются символические коды операций и адреса вместо числовых кодов и адресов. Соответствие между символическими именами и их численными значениями записывается ассемблером в две таблицы: *таблицу кодов операций и таблицу символических имен.*

В таблице кодов операций (которая для всех ассемблеров заранее предопределена) перечислены все коды операций, каждый со своим численным эквивалентом. Таблица символических имен (которая изменяется от программы к программе) содержит все определенные в программе символические адреса и операнды вместе с их численными значениями.

Другим важным звеном по отношению к операциям ассемблера является *счетчик адреса* [символически обозначаемый в языке ассемблера знаком (точка)], который на протяжении всех операций ассемблера содержит адрес памяти, по которому хранится выполняемая в текущий момент команда. Первоначально содержание счетчика адреса равно 0.

MACRO-11, как и большинство ассемблеров, является *двухпроходным ассемблером,* т. е. он осуществляет просмотр исходной программы *дважды* до того, как будет создана программа в объектных кодах. Эти два просмотра, или *прохода*, имеют следующие назначения:

Проход I: отыскать все символические описания в исходной программе и внести их в таблицу символических имен.

Проход II: используя таблицу имен, составленную при первом проходе, получить машинный эквивалент каждой инструкции исходной программы.

Блок-схемы первого и второго проходов показаны на рис. 9.1 и 9.2. Эти блок-схемы значительно упрощены и лишь в общих чертах передают процесс ассемблирования. В частности, на них не показаны результаты выполнения команд ассемблера. Ниже рассмотрены некоторые примеры:

1. .WORD $d_1, d_2, \ldots, d_r$ (или .BYTE $d_1, d_2, \ldots, d_r$) заставляет ассемблер заполнить последовательные слова (или байты) двоичными эквивалентами $d_1, d_2, \ldots, d_r$, увеличивая содержимое счетчика адреса на 2 (или на 1) после каждого такого заполнения.

2. .ASCII (str) заставляет ассемблер заполнить последовательные байты двоичным эквивалентом ASCII кода символов "str", увеличивая содержимое счетчика ячеек на 1 после каждого заполнения.

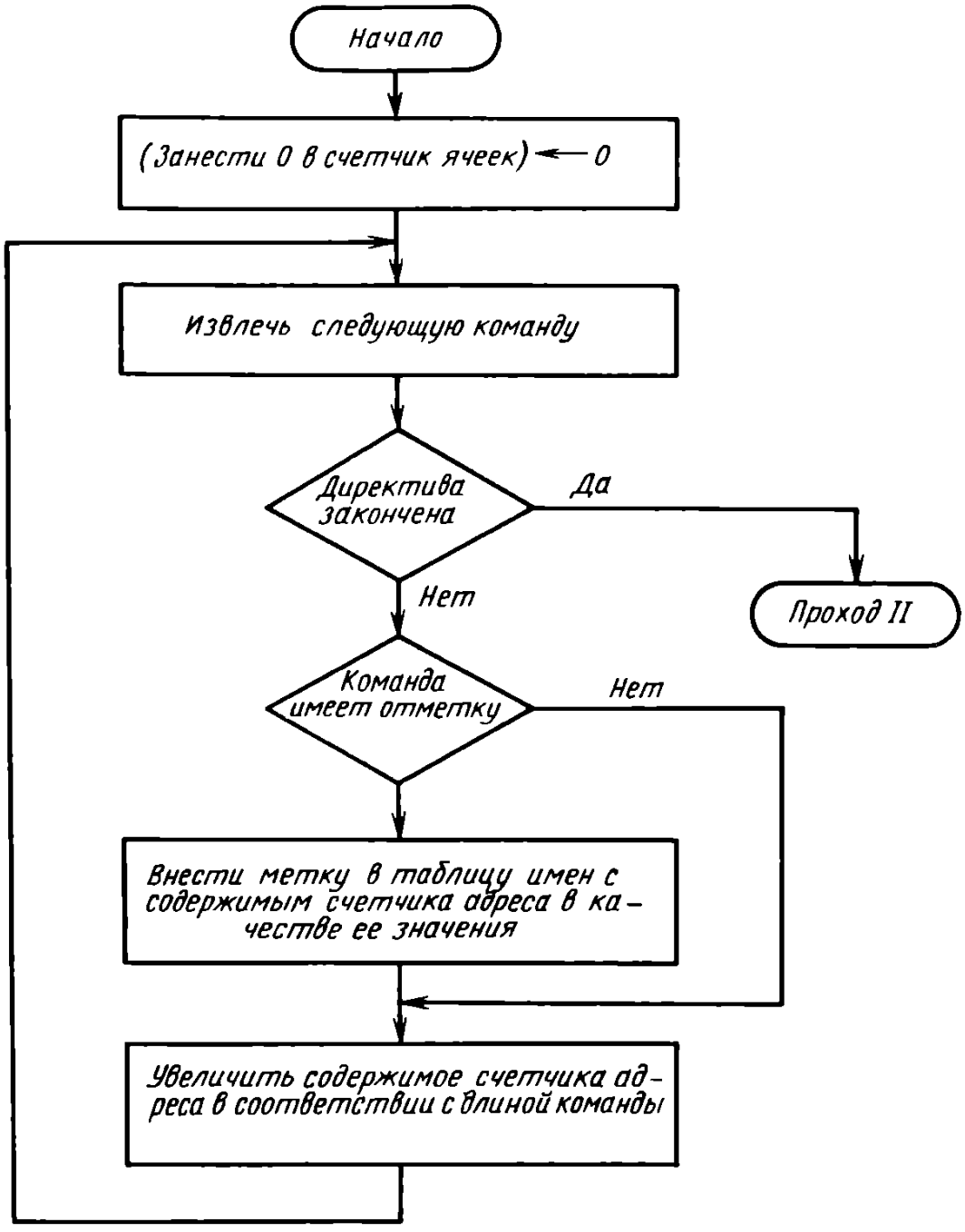


Рис. 9.1. Первый проход ассемблера

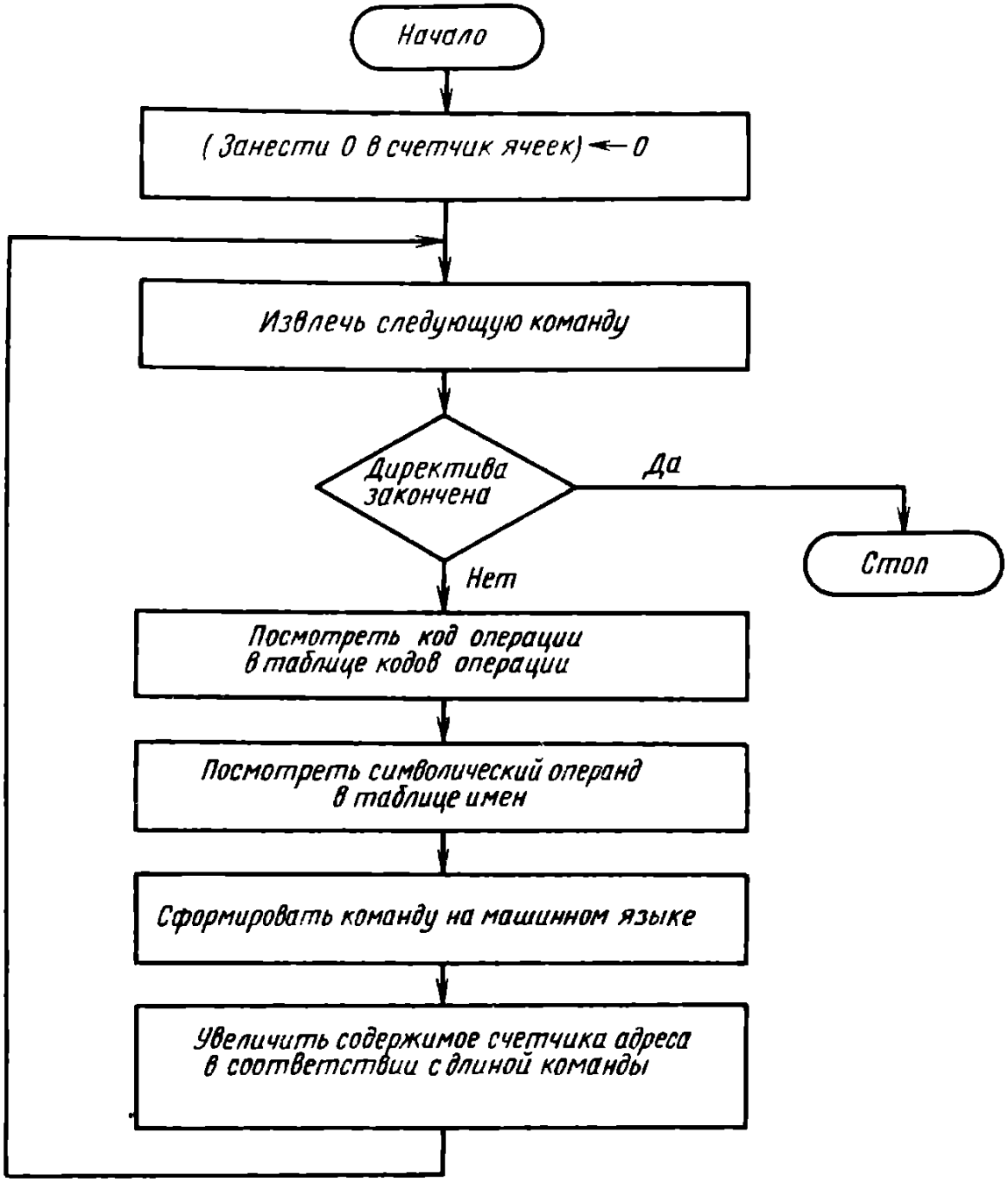


Рис. 9.2. Второй проход ассемблера

3. .BLKW n (или .BLKB n) заставляет ассемблер увеличить содержимое счетчика на 2n (или n), т. е. пропустить n слов (или байтов).

4. Директива присваивания sym = exp заставляет ассемблер (при первом проходе) внести имя "sym" и его значение "exp" в таблицу имен[1].

На блок-схеме невозможно также показать различные диагностические средства, имеющиеся в распоряжении ассемблера. Например, ассемблер сообщает, что какая-то метка "многократно определена", если (при первом

[1] Если "exp" содержит еще не определенный символ ("ссылка вперед"), то требуется дополнительный проход для определения его значения.

проходе) она уже найдена в таблице имен. Он указывает, что какая-то команда ошибочна, если (при втором проходе) ее код операции не найден в таблице кодов операций или не найден ее символический операнд в таблице имен.

9.2. Пример листинга, выдаваемого ассемблером

В разд. 7.7 была описана программа ASCTOBIN, которая преобразует числа из десятичного представления в двоичное. Обработка этой программы с помощью MACRO-11 дает в результате объектную программу ASCTOBIN, а также листинг исходной программы, объектной программы и таблицу символических имен. На рис. 9.3 приведена распечатка этого листинга.

На листинге ассемблера каждая исходная команда распечатывается вместе с соответствующим ей объектным кодом. Восьмеричные числа, помеченные

```
 1                                              .TITLE ASCTOBIN
 2                                              .ENABLE LC
 3                                      ;ПРЕОБРАЗОВАНИЕ ВВЕДЕННОГО ДЕСЯТИЧНОГО ЧИСЛА
 4                                      ;N В ДВОИЧНЫЙ ЭКВИВАЛЕНТ. ПЕРЕД N МОЖЕТ СТО-
 5                                      ;ЯТЬ ЗНАК + ИЛИ -, ПОСЛЕ N ДОЛЖЕН БЫТЬ ВВЕ-
 6                                      ;ДЕН СИМВОЛ ВОЗВРАТА КАРЕТКИ.
 7                                      ;ДВОИЧНЫЙ ЭКВИВАЛЕНТ N ОСТАЕТСЯ В R2. ЕСЛИ
 8                                      ;МОДУЛЬ N ПРЕВЫШАЕТ ЧИСЛО 32767 (ДЕСЯТИЧ-
 9                                      ;НОЕ), ТО В (R2) ОСТАНЕТСЯ ВОСЬМЕРИЧНОЕ
10                                      ;ЧИСЛО 100000.
11          000000'                     LC=.
12          000004'                     .=4+LC
13 000004   000006  000000  000012              .WORD  6,0,12,0     ;УСТАНОВКА ВЕКТО-
   000012   000000
14                                                                  ;РОВ ОШИБОК
15          000500'                     .=500+LC                    ;РЕЗЕРВИРОВАНИЕ
16                                                                  ;ПАМЯТИ ДЛЯ СТЕКА
17 000500   010706                      START: MOV    PC,SP
18 000502   005746                             TST    -(SP)        ;НАЧАЛЬНАЯ УСТА-
19                                                                  ;НОВКА SP
20                                      ;
21          177560                      KBSTAT=177560
22          177562                      KBDATA=177562
23          177564                      PRSTAT=177564
24          177566                      PRDATA=177566
25          000012                      LF=12
26          000015                      CR=15
27                                      ;
28                                      ;       MAIN PROGRAM
29 000504   012703  000526'                    MOV    #STRING,R3   ;(R3)=STRING
30 000510   004767  000036                    JSR    PC,INPUT     ;ЗАНЕСТИ ВХОДНУЮ
31                                                                  ;СТРОКУ (STRING)
32                                                                  ;В МАССИВ
33 000514   012703  000526'                    MOV    #STRING,R3   ;(R3)=STRING
34 000520   004767  000106                    JSR    PC,ATOB      ;ПРЕОБРАЗОВАТЬ
35                                                                  ;СТРОКУ В ДВОИЧНЫЙ
36                                                                  ;ЭКВИВАЛЕНТ
37 000524   000000                            HALT
38                                      ;
39 000526                               STRING: .BLKB  20.          ;ПАМЯТЬ ПОД ВВЕ-
40                                                                  ;ДЕННУЮ СТРОКУ
41                                      ;
42                                      ;       INPUT
43                                      ;РАСПЕЧАТКА ВВЕДЕННЫХ СИМВОЛОВ И ЗАПОМИНАНИЕ
44                                      ;ИХ В МАССИВЕ БАЙТОВ, БАЗОВЫЙ АДРЕС КОТОРОГО
45                                      ;В R3. ВЫХОД  ПОСЛЕ ТОГО, КАК НАПЕЧАТАНО CR.
46                                      ;ИЗМЕНЕНИЕ R3 И R5.
47 000532   105767  177560              INPUT: TSTB   KBSTAT        ;СИМВОЛ ВВЕДЕН?
48 000536   100375                             BPL    INPUT        ;ЕСЛИ НЕТ, ЖДАТЬ
49 000540   016705  177562                    MOV    KBDATA,R5    ;(R5)=СИМВОЛУ
50 000544   004767  000026                    JSR    PC,PRINT     ;ПЕЧАТЬ СИМВОЛА
51 000550   042705  177600                    BIC    #177600,R5   ;УДАЛИТЬ КОНТРОЛЬ-
52                                                                  ;НЫЙ РАЗРЯД
53 000554   110523                            MOVB   R5,(R3)+     ;ЗАНЕСТИ СИМВОЛ В
54                                                                  ;МАССИВ,СКОРРЕКТ.
55                                                                  ;ИНДЕКС
56 000556   022705  000015                    CMP    #CR,R5       ;СИМВОЛ СООТВЕТ-
```

Рис. 9.3. Листинг программы ASCTOBIN

```
57                                                      ;СТВУЕТ CR?
58 000602  001363                    BNE     INPUT      ;ЕСЛИ НЕТ, ПРИНЯТЬ
59                                                      ;СЛЕДУЮЩИЙ СИМВОЛ
60 000604  012705  000012            MOV     #LF,R5     ;ЕСЛИ ДА,
61 000610  004767  000002            JSR     PC,PRINT   ;ПЕРЕЙТИ НА СЛЕ-
62                                                      ;ДУЮЩУЮ СТРОКУ
63 000614  000207                    RTS     PC         ;ВЫХОД
64                                  ;
65                                  ;       PRINT
66                                  ;ПЕЧАТЬ СОДЕРЖИМОГО R5. РЕГИСТРЫ НЕ МЕНЯЮТСЯ.
67 000616  105767  177564           PRINT:  TSTB    PRSTAT     ;УСТРОЙСТО ПЕЧАТИ
68                                                      ;ГОТОВО?
69 000622  100375                    BPL     PRINT      ;ЕСЛИ НЕТ, ЖДАТЬ
70 000624  010567  177566            MOV     R5,PRDATA  ;ЕСЛИ ДА, ПЕЧАТЬ
71                                                      ;(R5)
72 000630  000207                    RTS     PC         ;ВЫХОД
73                                  ;
74                                  ;       ATOB
75                                  ;ПРЕОБРАЗОВАНИЕ В ДВОИЧНОЕ ДЕСЯТИЧНОГО ЧИСЛА
76                                  ;N, ХРАНЯЩЕГОСЯ В КОДЕ ASCII В МАССИВЕ БАЙ-
77                                  ;ТОВ, БАЗОВЫЙ АДРЕС КОТОРОГО В R3. N МОЖЕТ
78                                  ;ИМЕТЬ ЗНАК + ИЛИ -, А ЗА N ДОЛЖЕН СЛЕДОВАТЬ
79                                  ;НЕЦИФРОВОЙ СИМВОЛ.  ДВОИЧНЫЙ ЭКВИВАЛЕНТ N
80                                  ;ОСТАЕТСЯ В R2.
81                                  ;ЕСЛИ МОДУЛЬ N ПРЕВЫШАЕТ ЧИСЛО (ДЕСЯТИЧ-
82                                  ;НОЕ) 32767, ТО В R2 ОСТАЕТСЯ ВОСЬМЕРИЧНОЕ
83                                  ;100000. РАСПРЕДЕЛЕНИЕ РЕГИСТРОВ:
84                                  ;       (R0),(R1) ИСПОЛЬЗУЮТСЯ ДЛЯ ПАРАМЕТРОВ
85                                  ;                 ПОДПРОГРАММЫ MUL
86                                  ;       (R2)=ПРЕОБРАЗОВАННОМУ ЧИСЛУ
87                                  ;       (R3)=УКАЗАТЕЛЮ К СЛЕДУЮЩЕМУ СИМВОЛУ
88                                  ;       (R4)=ПРОСМОТРЕННОМУ СИМВОЛУ
89                                  ;       (R5)=ЗНАКОВОМУ ФЛАГУ (0, ЕСЛИ N ИМЕ-
90                                  ;            ЕТ +, 1, ЕСЛИ - )
91 000632  005005                   ATOB:   CLR     R5         ;ЗАНУЛИТЬ ЗНАКОВЫЙ
92                                                      ;ФЛАГ (СЧИТАЕМ N
93                                                      ;ПОЛОЖИТ.)
94 000634  112304                    MOVB    (R3)+,R4   ;(R4)=ПРОСМОТР.
95                                                      ;СИМВОЛУ, СКОРРЕК-
96                                                      ;ТИР. ИНДЕКС
97 000636  122704  000053            CMPB    #'+,R4     ;СИМВОЛ ИМЕЕТ
98                                                      ;ЗНАК +?
99 000642  001406                    BEQ     ATOB2      ;ЕСЛИ ДА, НАЧАТЬ
100                                                     ;ПРЕОБРАЗОВАНИЕ
101 000644  122704  000055           CMPB    #'-,R4     ;СИМВОЛ ИМЕЕТ
102                                                     ;ЗНАК -?
103 000650  001002                   BNE     ATOB1      ;ЕСЛИ НЕТ, ТО N
104                                                     ;НЕ ИМЕЕТ ЗНАКА
105 000652  005205                   INC     R5         ;ЕСЛИ ДА, УСТАНО-
106                                                     ;ВИТЬ ЗНАКОВЫЙ
107                                                     ;ФЛАГ НА 1
108 000654  000401                   BR      ATOB2      ;НАЧАЛО ПРЕОБРА-
109                                                     ;ЗОВАНИЯ
110 000656  005303                  ATOB1:  DEC     R3         ;СИМВОЛ ЯВЛ. ЦИФ-
111                                                     ;РОЙ. ВОЗВРАТ
112                                                     ;УПРАВЛЕНИЯ
113 000660  005002                  ATOB2:  CLR     R2         ;УСТАНОВИТЬ РЕ-
114                                                     ;ЗУЛЬТАТ РАВНЫМ 0
115 000662  112304                  ATOB3:  MOVB    (R3)+,R4   ;(R4)=ПРОСМОТР.
116                                                     ;СИМВОЛУ, СКОРРЕК-
117                                                     ;ТИР. ИНДЕКС
118 000664  122704  000060           CMPB    #'0,R4     ;ЕСЛИ '0>(R4)
119                                                     ;(Т.Е. НЕ ЦИФРА),
120 000670  101016                   BHI     ATOB4      ;ТО ПОДГОТОВКА К
121                                                     ;ВЫХОДУ
122 000672  122704  000071           CMPB    #'9,R4     ;ЕСЛИ '9<(R4)
123                                                     ;(Т.Е. НЕ ЦИФРА),
124 000676  103413                   BLO     ATOB4      ;ТО ПОДГОТОВКА К
125                                                     ;ВЫХОДУ
126 000700  042704  177760           BIC     #177760,R4 ;ПРЕОБРАЗ. ЦИФРЫ
127                                                     ;В ДВОИЧН. ЭКВИ-
128                                                     ;ВАЛЕНТ
129 000704  012700  000012           MOV     #10.,R0    ;(R0)=10. (ПАРА-
130                                                     ;МЕТР ПОДПРОГР.
131                                                     ;MUL)
132 000710  010201                   MOV     R2,R1      ;(R2)=(R1) (ПАРА-
133                                                     ;МЕТР ПОДПРОГР.
134                                                     ;MUL)
135 000712  004767  000026           JSR     PC,MUL     ;(R2)=(R0)*(R1)=
136                                                     ;10.*(R2)
137 000716  102407                   BVS     ATOB6      ;ЕСЛИ ПЕРЕПОЛНЕ-
138                                                     ;НИЕ, ПОДГОТОВКА
139                                                     ;К ВЫХОДУ
```

Рис. 9.3 (продолжение)

```
140 000720  060402                         ADD     R4,R2           ;(R2)=(R4)+(R2)
141 000722  102405                         BVS     ATOB6           ;ЕСЛИ ПЕРЕПОЛНЕ-
142                                                                 ;НИЕ, ПОДГОТОВКА
143                                                                 ;К ВЫХОДУ
144 000724  000756                         BR      ATOB3           ;ПРОСМОТРЕТЬ СЛЕ-
145                                                                 ;ДУЮЩИЙ СИМВОЛ
146                                 ;NORMAL EXIT(НОРМАЛЬНЫЙ ВЫХОД)
147 000726  005705                  ATOB4: TST     R5              ;ПРОВЕРКА ЗНАКО-
148                                                                 ;ВОГО ФЛАГА
149 000730  001401                         BEQ     ATOB5           ;ЕСЛИ ЧИСЛО ПОЛО-
150                                                                 ;ЖИТЕЛЬНОЕ, ТО
151                                                                 ;ВЫХОД
152 000732  005402                         NEG     R2              ;ЕСЛИ НЕТ, ТО
153                                                                 ;(R2)=-(R2)
154 000734  000207                  ATOB5: RTS     PC              ;ВЫХОД
155                                 ;OVERLOW EXIT(ВЫХОД ПО ПЕРЕПОЛНЕНИЮ)
156 000736  012702  100000          ATOB6: MOV     #100000,R2      ;(R2)=100000
157 000742  000207                         RTS     PC              ;ВЫХОД
158                                 ;
159                                 ;       MUL
160                                 ;ВЫЧИСЛЕНИЕ (R0)*(R1) И ЗАПИСЬ РЕЗУЛЬТАТОВ
161                                 ;В R2. ЕСЛИ МОДУЛЬ РЕЗУЛЬТАТА ПРЕВЫШАЕТ
162                                 ;32767 (ДЕСЯТИЧНОЕ), ТО В РАЗРЯДЕ V УСТАНО-
163                                 ;ВИТЬ 0. R3,R4,R5 НЕ ИСПОЛЬЗУЮТСЯ.
164                                 ;
165 000744  005002                  MUL:   CLR     R2              ;(R2)=0
166 000746  032701  000001          MUL1:  BIT     #1,R1           ;ПРОВЕРКА НУЛЕВО-
167                                                                 ;ГО РАЗРЯДА R1
168 000752  001402                         BEQ     MUL2            ;ЕСЛИ 0, НЕ ПРО-
169                                                                 ;ВОДИТЬ СЛОЖЕНИЯ
170 000754  060002                         ADD     R0,R2           ;ИНАЧЕ, (R0)=(R0)
171                                                                 ;+(R2)
172 000756  102406                         BVS     MUL3            ;ВЫХОД ПО ПЕРЕ-
173                                                                 ;ПОЛНЕНИЮ
174 000760  000241                  MUL2:  CLC                     ;ОЧИСТКА РАЗ-
175                                                                 ;РЯДА С
176 000762  006001                         ROR     R1              ;ЦИКЛ. СДВИГ R1
177                                                                 ;НА 1 РАЗРЯД
178                                                                 ;ВПРАВО
179 000764  006300                         ASL     R0              ;АРИФМ. СДВИГ (R0)
180                                                                 ;НА 1 РАЗРЯД
181                                                                 ;ВЛЕВО
182 000766  102402                         BVS     MUL3            ;ВЫХОД ПО ПЕРЕ-
183                                                                 ;ПОЛНЕНИЮ
184 000770  005701                         TST     R1              ;ПРОВЕРКА (R1)
185 000772  001365                         BNE     MUL1            ;ЕСЛИ НЕ 0, ПРО-
186                                                                 ;ДОЛЖИТЬ ПЕРЕМНО-
187                                                                 ;ЖЕНИЕ
188 000774  000207                  MUL3:  RTS     PC              ;ВЫХОД
189
190         000500'                         .END    START
```

```
SYMBOL TABLE

ATOB    000632R   ATOB5   000734R   KBSTAT= 177560     MUL1    000746R   PRINT   000616P
ATOB1   000656R   ATOB6   000736R   LC    = 000000R    MUL2    000760R   PRSTAT= 177564
ATOB2   000660R   CR    = 000015    LF    = 000012     MUL3    000774R   START   000500R
ATOB3   000662R   INPUT   000552R   MUL     000744R    PRDATA= 177566    STRING  000526R
ATOB4   000726R   KBDATA= 177562

. ABS.  000000     000
        000776     001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  8192 WORDS  ( 32 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  51 PAGES
DRAW73,DRAW73=DRAW7.BAK
```

Рис. 9.3 (окончание)

справа апострофом (например, 000526′), являются адресами, которые требуют модификации, в случае если программа "перемещаема". Более подробно об этом будет сказано в следующем разделе.

Таблица символических имен содержит (в алфавитном порядке) все определенные в программе имена [те, которые определены с помощью директивы присваивания, указываются символом (знаком) =]. Значение суффикса R будет объяснено ниже.

9.3. Абсолютные и перемещаемые адреса

Как мы видели в разд. 9.1, счетчик адреса ассемблера перед началом работы всегда устанавливается на 0. Следовательно, объектная программа будет загружаться, начиная с адреса 0. Однако в ряде случаев было бы желательно "переместить" программу, т. е. изменить *начало загрузки* (адрес, с которого начинается загрузка) с 0 на некоторую другую величину. Например, если большая программа формируется путем "связывания" ряда независимо скомпилированных программ, ясно, что только одна из них может иметь начало загрузки, равное 0, все остальные должны быть перемещаемыми. Точно так же, если несколько программ, принадлежащих различным пользователям, должны размещаться в памяти одновременно, что имеет место в режиме обслуживания с разделением времени, то только одна из них может быть загружена с начальным адресом 0, а остальные должны быть перемещены.

При этом возникает вопрос: будет ли объектная программа выполняться надлежащим образом, и если нет — как ее модифицировать, чтобы сделать такое выполнение программы возможным? Прежде чем ответить на эти вопросы, нужно установить различие между двумя типами адресов, определенными программой: *абсолютными адресами* и *перемещаемыми адресами*. В своей основе абсолютные адреса представляют собой просто числа или знаки (или имена, сопоставленные этим числам или знакам с помощью директив присваивания); перемещаемые адреса — это просто метки (или имена, сопоставленные меткам с помощью директив присваивания). Например, в командах MOV, ADD и BR, содержащихся в следующем программном сегменте, адреса 300 и X являются абсолютными, а адреса A, B и Y — перемещаемыми:

```
X = 100
Y = B
        ...
    MOV A,300
    ADD X,B
    BR Y
      ...
A:    ...
      ...
B:    ...
```

Адреса могут быть также определены с помощью адресных выражений; которые могут быть представлены суммами и разностями адресов. В зависимости от того, являются ли определяемые таким способом адреса абсолютными или перемещаемыми, применяются следующие правила:

абсолютный ± абсолютный = абсолютный,
перемещаемый — перемещаемый = абсолютный [1],
абсолютный ± перемещаемый = перемещаемый.

---

[1] Перемещаемый + перемещаемый → неопределен.

Например, в командах MOV и ADD, содержащихся в нижеследующем программном сегменте, значения P + Q – 100, D – C и C – D + Q являются абсолютными, а адрес C – P является перемещаемым:

```
P = 200
Q = 'Z
      ...
      MOV P+Q-100,C-P
      ADD D-C,C-D+Q
      ...
C:    ...
      ...
D:    ...
```

Предполагается (и в этом состоит одна из задач программиста), что *все абсолютные адреса остаются неизменными независимо от адреса начала загрузки программы, тогда как все относительные адреса, подлежащие перемещению, будут изменяться на ту же величину, что и начало загрузки.* Например, мы хотим, чтобы в программе ASCTOBIN адреса KBSTAT и CR, равные 177560 и 15 соответственно, оставались неизменными независимо от положения программы (так как регистр состояния клавишного пульта всегда находится по адресу 177560, код ASCII для возврата каретки всегда равен 15). С другой стороны, STPING соответствует 526 только тогда, когда адрес начала загрузки равен 0; он должен измениться на 1526, когда адрес начала загрузки изменяется на 1000[1].

В таблице имен листинга ассемблера перемещаемые имена помечены знаком (суффиксом) R.

Знак . (представляющий счетчик адреса) рассматривается в MACRO-11 как перемещаемый. Следовательно, такая директива MACRO-11, как . = 500, недопустима. Вместо этого можно было бы сделать (и это мы неизменно осуществляли раньше) так, чтобы программа начиналась с команды LC = . (которая определяет LC как "перемещаемый 0"), а затем указать команду . = 500 + LC, которая присваивает счетчику ячеек перемещаемое значение 500 + LC, где 500 отсчитывается от начала загрузки.

9.4. Редактор связей

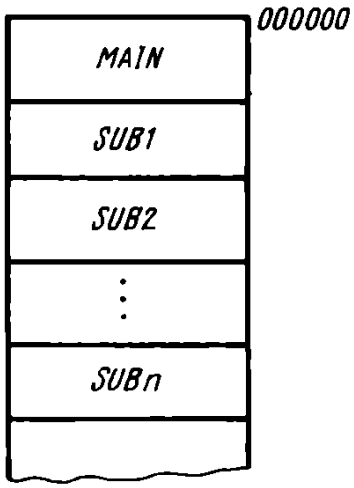
Часто подпрограммы, используемые в программе пользователя, уже существуют (например, они уже были ранее написаны пользователем для использования в других программах или же они включены в системную библиотеку в виде "стандартных" подпрограмм). В этом случае единственное, что нужно сделать – это объединить объектные коды основной программы и подпрограмм (так, чтобы можно было ссылаться на каждую из них, как на *объектный модуль*) в одну большую программу на машинном языке (называемую *загрузочным модулем*), готовую к загрузке и выполнению. Работа по такому объединению или "связи" осуществляется *редактором связей*[2].

---

[1] Адреса, соответствующие командам .BLKW и .BLKB должны быть всегда абсолютными.

[2] Редактор связей, описанный в этой главе, называется LINKR-11.

Предположим, что ассемблирование основной программы MAIN и подпрограмм SUB1, SUB2, . . . , SUBn проведено независимо. Передача соответствующих n + 1 объектных модулей (и описания, как они должны быть упорядочены) редактору связей приводит к созданию загрузочного модуля, который выглядит так, как показано ниже. Этот объектный модуль может быть загружен в память PDP-11 (начиная с адреса 0) и выполнен (начиная с адреса, первоначально заданного директивой .END).



Вообще говоря, действие редактора связей аналогично тому, как если бы программа MAIN и подпрограммы SUB1, SUB2, . . . , SUBn были ассемблированы в одно и то же время. Возникает вопрос: почему мы не можем запомнить все исходные коды (а именно, коды программы MAIN и подпрограмм SUB1, SUB2, . . . , SUBn) в одном файле, а затем ассемблировать их вместе; в этом случае необходимость в такой связи отпала бы? Однако дело в том, что иногда исходные коды для SUB1, SUB2, . . . , SUBn просто недоступны программисту (например тогда, когда они являются библиотечными подпрограммами). В других случаях язык, на котором написаны подпрограммы SUB1, SUB2, . . . , SUBn, не является языком ассемблера (это может быть, например, язык Фортран) и трансляция на машинный язык не может быть выполнена ассемблером; для этой цели необходим специальный компилятор. Во всех этих случаях необходимо наличие редактора связей.

При преобразовании объектных модулей в загрузочный редактор связей должен осуществить следующие две функции:

1. Модифицировать адреса, которые предполагается перемещать, когда это необходимо.

2. Предоставить "внешние" адреса, т. е. адреса, на которые ссылаются в одном объектном модуле, но которые определяются в другом модуле.

Подробности выполнения этих двух функций описаны в следующих двух разделах.

9.5. Модификация адреса

Учитывая сказанное в разд. 9.3, хотелось бы, чтобы редактор связей осуществлял модификацию адресов в соответствии со следующими критериями:

1. Изменения перемещаемого адреса и начала загрузки должны быть одинаковыми.

2. Абсолютный адрес должен оставаться неизменным независимо от того, изменяется или нет начало загрузки.

Таблица 9.1. **Примеры перемещений**

Пример	Исходная программа	Объектная программа	После загрузки с адреса 1000	Модификация
1	MOV A,R0	600: 016700	1600: 016700	
		602: 000100	1602: 000100	← Без изменения
	...	...	...	
	A: ...	704: ...	1704: ...	
2	MOV #A,R0	600: 012700	1600: 012700	
		602: 000704'	1602: 001704	← Добавляется 1000
	...	...	...	
	A: ...	704: ...	1704: ...	
3	MOV @#A,R0	600: 013700	1600: 013700	
		602: 000704'	1602: 001704	← Добавляется 1000
	...	...	...	
	A: ...	704: ...	1704: ...	
4	MOV A(R0),R1	600: 016001	1600: 016001	
		602: 000704'	1602: 001704	← Добавляется 1000
	...	...	...	
	A: ...	704: ...	1704: ...	
5	MOV @A(R0),R1	600: 017001	1600: 017001	
		602: 000704'	1602: 001704	← Добавляется 1000
	...	...	...	
	A: ...	704: ...	1704: ...	
6	MOV 177776,R0	600: 016700	1600: 016700	
		602: 177172'	1602: 176172	← Вычитается 1000
7	MOV #15,R0	600: 012700	1600: 012700	
		602: 000015	1602: 000015	← Без изменения
8	MOV @#60,R0	600: 013700	1600: 013700	
		602: 000060	1602: 000060	← Без изменения
9	MOV 20(R0),R1	600: 016001	1600: 016001	
		602: 000020	1602: 000020	← Без изменения
10	MOV @20(R0),R1	600: 017001	1600: 017001	
		602: 000020	1602: 000020	← Без изменения

В табл. 9.1 показаны примеры команд до и после перемещения. Чтобы облегчить работу редактора связей, каждый адрес, подлежащий модификации, помечается ассемблером с помощью апострофа.

Пользуясь табл. 9.1, можно сформулировать следующие правила модификации адресов:

1. Все перемещаемые адреса должны *увеличиваться* на ту же величину, что и адрес начала загрузки, исключая случай использования относительной адресации[1], при котором не требуется модификация.

[1] В этой главе понятие "относительная адресация" одновременно относится как к относительному, так и косвенно-относительному режимам адресации (см. разд. 4.6 и 4.7).

2. Все абсолютные адреса должны оставаться неизменными, исключая случай использования относительной адресации, при котором они должны быть *уменьшены* на ту же величину, что и адрес начала загрузки.

Заметим, что команды перехода определяют не адреса, а *смещения* и, следовательно, для них никогда не требуется модификация.

9.6. Глобальные имена

Предположим, что требуется осуществить связь программы MAIN и подпрограмм SUB1, SUB2, . . . , SUBn, ассемблированных независимо. Практически неизбежно, что некоторые из них будут обращаться к именам, определенным как внешние (т. е. как определенные с помощью других подпрограмм). Например, MAIN может обращаться к меткам, определенным в SUB1 (при использовании команд JSR PC, SUB1), а SUB1, в свою очередь, может обращаться к массиву, определенному в SUB2 (при использовании, например, команды MOV TABLE (R0), R1, где TABLE описывается в SUB2 с помощью директивы .BLKW).

Символическое имя в программе называется *глобальным*, если оно: 1) определяется в другой программе или 2) к нему обращаются из другой программы. Все глобальные имена sym_1, sym_2, . . . , sym_r в подпрограмме должны быть специально описаны (где-либо внутри программы) с помощью директивы .GLOBL

.GLOBL sym_1, sym_2, . . . , sym_r .

Неглобальные имена называют *локальными*. Как мы уже упоминали, на втором проходе процесса ассемблирования все локальные имена в программе заменяются числовыми адресами (с апострофами или без них). Следовательно, к тому моменту, когда объектные модули передаются редактору связей, локальные имена уже не существуют. По этой причине локальные имена одной программы могут быть повторены в другой программе без риска, что это приведет к путанице. Однако глобальные имена должны быть определены единственным образом во всех объединенных редактором связей программах, так как они еще "не разрешены" (т. е. не существует доступного числового эквивалента), когда редактор связей приступает к работе.

В качестве примеров на рис. 9.4, 9.5 и 9.6 показаны листинги ассемблера (включая списки символических имен) программы MAIN, SUB1 и SUB2 (программы эти служат только для иллюстрации работы редактора связей). Каждая из этих программ содержит глобальные имена. Например, MAIN обращается к глобальным именам SUB1, D и E, которые появляются в подпрограмме SUB1, и к G, которое появляется в подпрограмме SUB2. Кроме того, MAIN определяет имена A и B, на которые ссылаются подпрограммы SUB1 и SUB2. Следовательно, программа MAIN должна содержать директиву

.GLOBL SUB1, A, B, D, E, G

В листинге объектных кодов глобальные имена, определенные как внешние, показываются в виде 000000G; однако в объектном модуле они фактически появляются в символьном виде (как строка символов). В таблице имен все глобальные имена имеют в кодовом обозначении суффикс G; глобальные имена, определенные как внешние, появляются в списке в виде ******G.

```
1                       .TITLE   MAIN
2         000000'LC=.
3         000004'.=4+LC
4  000004 000006        .WORD    6,0,12,0        ;УСТАНОВКА ВЕКТОРОВ ОШИБОК
          000000
          000012
          000000                                 ;РЕЗЕРВИРОВАНИЕ ПАМЯТИ
5         000500'.=500+LC                        ;ДЛЯ СТЕКА
6  000500 010706 START:  MOV      PC,SP           ;НАЧАЛЬНАЯ УСТАНОВКА SP
7  000502 005746         TST      -(SP)
8                        .GLOBL   SUB1,A,B,D,E,G
9  000504 004767         JSR      PC,SUB1
          000000G
10 00510  005767         TST      A
          000026
11 00514  005767         TST      B
          000142
12 00520  005767         TST      C
          000256
13 00524  005767         TST      D
          000000G
14 00530  005767         TST      E
          000000G
15 00534  005767         TST      G
          000000G
16 00540  000000         HALT
17 00542         A:      .BLKW    50
18 00662         B:      .BLKW    50
19 01002         C:      .BLKW    50
20        000500'        .END     START
```

ТАБЛИЦА ИМЕН

```
A       000542RG        B       000662RG        C       001002R
D    =  ****** G        E    =  ****** G        G    =  ****** G
LC   =  000000R         START   000500R         SUB1 =  ****** G
```

Рис. 9.4. Листинг программы MAIN

```
1                       .TITLE   SUB1
2                       .GLOBL   SUB1,SUB2,A,B,D,E
3  000000 004767 SUB1:  JSR      PC,SUB2
          000000G
4  000004 005767        TST      A
          000000G
5  000010 005767        TST      B
          000000G
6  000014 005767        TST      D
          000006
7  000020 005767        TST      F
          000402
8  000024 000207        RTS      PC
9  000026         D:    .BLKW    100
10 00226          E:    .BLKW    100
11 00426          F:    .BLKW    100
12        000001        .END
```

ТАБЛИЦА ИМЕН

```
A    =  ****** G        B    =  ****** G        D       000026RG
E       000226RG        F       000426R         SUB1    000000RG
SUB2 =  ****** G
```

Рис. 9.5. Листинг подпрограммы SUB1

```
1                       .TITLE   SUB2
2                       .GLOBL   SUB2,A,D,G
3 000000 005767 SUB2:   TST      A
         000000G
4 000004 005767         TST      D
         000000G
5 000010 005767         TST      G
         000006
6 000014 005767         TST      H
         000202
7 000020 000207         RTS      PC
8 000022         G:     .BLKW    100
9 000222         H:     .BLKW    100
10       000001         .END

ТАБЛИЦА ИМЕН

A     =  ****** G      D     =  ****** G      G       000022RG
H        000222R       SUB2     000000RG
```

Рис. 9.6. Листинг подпрограммы SUB2

В вышеприведенном примере имя C локально по отношению к MAIN, F – по отношению к SUB1, а H – по отношению к SUB2. (Имена F и H могут быть заменены на C без риска путаницы.)

9.7. Двухпроходный процесс редактирования связей

Каждый из объектных модулей, передаваемых редактору связей, является результатом трансляции на машинный язык соответствующей исходной программы. Эта трансляция, однако, является недостаточной в двух отношениях: 1) некоторые из адресов, которые снабжены апострофами, возможно требуют модификации, необходимой при перемещении программы; 2) некоторые из адресов (те, которые еще представлены в символической форме) являются глобальными и требуют окончательного разрешения.

В дополнение к кодам, каждый объектный модуль передает редактору связей *таблицу глобальных имен*, которая состоит из всех описаний глобальных имен, найденных внутри модуля. Они соответствуют всем тем именам в таблице исходных имен, которые содержат суффикс G и не определены как ******.

Например, для трех программ, указанных в разд. 9.6:

Таблица глобальных имен для MAIN
- A 000542R
- B 000662R

Таблица глобальных имен для SUB1
- D 000026R
- E 000226R
- SUB1 000000R

Таблица глобальных имен для SUB2
- G 000022R
- SUB2 000000R

Каждый объектный модуль также сообщает редактору связей длину модуля.

```
ПРЕОБРАЗУЕМЫЙ АДРЕС: 000500
НИЖНЯЯ ГРАНИЦА:      000000
ВЕРХНЯЯ ГРАНИЦА:     002372
          **********
МОДУЛЬ  MAIN
СЕКЦИЯ                    ДИАПАЗОН АДРЕСОВ
          <. ABS.>        000000  000000
          <       >       000000  001122
                  A       000542'         B       000662'
          **********
МОДУЛЬ  SUB1
СЕКЦИЯ                    ДИАПАЗОН АДРЕСОВ
          <. ABS.>        000000  000000
          <       >       001122  000626
                  D       001150'         E       001350'         SUB1    001122
          **********
МОДУЛЬ  SUB2
СЕКЦИЯ                    ДИАПАЗОН АДРЕСОВ
          <. ABS.>        000000  000000
          <       >       001750  000422
                  G       001772'         SUB2    001750'
```

Рис. 9.7. Карта загрузки

Работа редактора связей по преобразованию объектного модуля в загрузочный модуль выполняется в два "прохода", примерно так:

Проход 1. Используя информацию о длине объектных модулей и о порядке, в котором эти объектные модули должны появляться в памяти, редактор связей составляет *карту загрузки*, которая показывает, как указанные модули должны размещаться в памяти. Используя карту загрузки, редактор связей осуществляет просмотр каждого из объектных модулей и модифицирует (если это необходимо) каждый помеченный апострофом адрес в соответствии с правилами, приведенными в разд. 9.5. Он также модифицирует (если это необходимо) адреса в каждой из таблиц глобальных имен так, чтобы поставить их в соответствие с начальными адресами загрузки соответствующих модулей. Затем он объединяет все таблицы глобальных имен в одну *объединенную таблицу глобальных имен*, в которой перечислены все глобальные имена, используемые в указанных программах, и их точные значения.

На рис. 9.7 показана распечатка карты загрузки, сформированная редактором связей для программ, указанных в разд. 9.6. Эта распечатка показывает начальный адрес программы (500) и ее общую длину (2372), а также начальные адреса загрузки (0, 1122, 1750) и длины (1122, 626, 422) трех модулей. На рис. 9.8 та же информация представлена в виде диа-

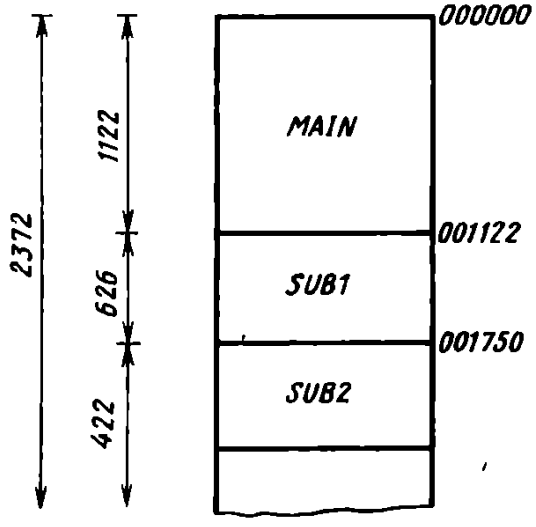


Рис. 9.8. Загрузочный модуль

граммы. Информация из этой карты загрузки используется редактором связей для модификации всех адресов, помеченных апострофами, появляющихся в объектных модулях MAIN, SUB1, SUB2, и для формирования объединенной таблицы глобальных имен. Таблица (которая отмечена звездочками между строками карты загрузки на рис. 9.7) выглядит следующим образом:

Объединенная таблица глобальных имен

A	000542
B	000662
D	001150
E	001350
SUB1	001122
G	001772
SUB2	001750

Например, значение адреса, которое в таблице глобальных имен подпрограммы SUB1 было равно 26, стало теперь 1122 + 26 = 1150; значение адреса G, которое в таблице глобальных имен подпрограммы SUB2 было равно 22, стало теперь 1750 + 22 = 1772. (Помните, однако, что значения адресов *абсолютных* глобальных имен (т. е. тех, у которых нет суффикса R) остаются неизменными.)

Проход II. Во время первого прохода загрузочный модуль был сформирован почти полностью, за исключением тех глобальных имен, которые были включены в коды как строки символов. Теперь редактор связей просматривает объектные модули и, используя объединенную таблицу глобальных имен, заменяет все глобальные имена их значениями так, как они перечислены в таблице. В этом случае объединенная таблица глобальных имен используется точно так же, как аналогичная таблица имен использовалась ассемблером на втором проходе процесса ассемблирования. Однако больше нет адресов, которые нужно было бы помечать апострофом, так как нет необходимости в дальнейших перемещениях.

9.8. Позиционно-независимый код

Из предыдущих примеров можно было видеть, что в обычной программе большинство адресов не нуждается в модификации редактором связей после ассемблирования, вне зависимости от того, где находится начало загрузки. Исходный код, адреса которого (после ассемблирования) нет необходимости модифицировать при изменении начала загрузки, называется *позиционно-независимым кодом* (*ПНК*). Иногда более предпочтительно написать специальные программы-утилиты, предназначенные для частого использования многими пользователями в ПНК, так как это приводит к объектным кодам, которые могут быть загружены в любое место в памяти без дополнительной обработки.

Чтобы решить, что можно, а что нельзя в рамках ПНК, мы можем сослаться на разд. 9.5, где мы разбирали условия, при которых адреса могут потребовать модификации, осуществляемой редактором связей (т. е. условия, при которых MACRO-11 помечает адреса с помощью апострофа). Используя эти условия, мы можем получить, что в рамках ПНК:

1) перемещаемые адреса должны быть использованы только в режиме относительной адресации;

2) абсолютные адреса могут быть использованы в любом режиме, *за исключением* режима относительной адресации.

Эти правила требуют определенного искусства при написании программ в рамках ПНК. Например, мы уже видели, что, если не использовать ПНК, то имеем

```
START: MOV  # START, SP
```

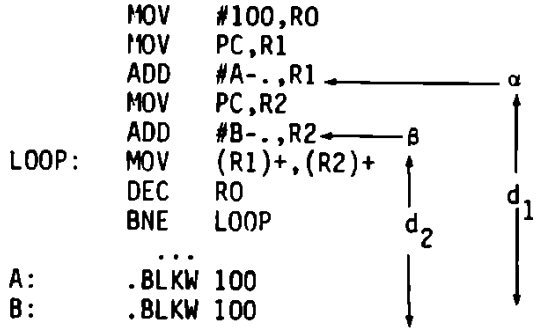
В рамках ПНК это может быть преобразовано так:

```
START: MOV PC, SP
       TST – (SP)
```

В качестве другого примера рассмотрим перезапись массива из 100 слов с базовым адресом A в массив из 100 слов с базовым адресом B:

```
        MOV   #100,R0
        MOV   #A,R1
        MOV   #B,R2
LOOP:   MOV   (R1)+,(R2)+
        DEC   R0
        BNE   LOOP
          ...
A:      .BLKW 100
B:      .BLKW 100
```

Вышеприведенная обычная запись может быть преобразована с использованием ПНК в следующий фрагмент:



В процессе выполнения MOV PC, R1 поместит в R1 абсолютный адрес α следующей команды; следующая команда ADD # A–., R1 прибавит к R1 расстояние d_1 от α до A, в результате в R1 будет содержаться адрес A. Аналогично, выполнение команд MOV PC, R2 и ADD # B–., R2 приведет к тому, что в R2 будет содержаться абсолютный адрес B. (Заметим, что A–. и B–., будучи по форме разностью (перемещаемый – перемещаемый), дают абсолютные адреса.)

Упражнения

9.1. Указать, какие из следующих команд ошибочны? Какие из них относятся к ошибкам периода ассемблирования и какие к ошибкам периода выполнения программы?

```
NEG   @#37775
MOVE  R0,R5
RTI   ;SUB R4,R1
JMP   R3
BIC   (PC),SP
ADD   R2+5,R3
CLR   TABLE (R0)+
```

9.2. Программы PROG1 и PROG2, показанные ниже, ассемблируются независимо. Записать: а) объектный модуль PROG1; б) загрузочный модуль, полученный в результате объединения PROG1 и PROG2 (в том же порядке).

```
        .TITLE PROG1
LC=.
.=476+LC
START:  HALT
        .END START
```

```
        .TITLE PROG2
X:      MOV  #X,Y-X
        MOV  #X-Y,Y
        MOV  @#X-5,Y-X(R5)
Y:      HALT
        .WORD X,X-Y+3
        .END
```

9.3. а) Записать объектный модуль и таблицу имен следующей программы. Указать в таблице имен, какие адреса являются абсолютными, а какие относительными.

```
LC=.
.=500+LC
A = 12
B = F+'h
C = E-F
D = A+C-2
START:      MOV A,B
E:          MOV #C, E(R0)
F:          MOV #A+B,@D(R1)
            HALT
G:          .BLKW  D-C
            .WORD G+C,G-B
            .END  START
```

б) Полагая, что программа загружается с адресом начала загрузки 2200, написать программу, модифицированную редактором связей.

9.4. Приведенные ниже программы MAIN, SUB1 и SUB2 ассемблируются отдельно, а затем в том же порядке объединяются.

а) Завершить описание .GLOBL во всех трех программах.

б) Показать окончательный листинг ассемблера (включая таблицу имен, таблицу глобальных имен и все адреса с апострофами и суффиксами) для каждой программы.

в) Сформировать карту загрузки и объединенную таблицу глобальных имен для всех трех программ.

г) Привести загрузочный модуль объединенной программы.

```
            .TITLE   MAIN
            .GLOBL   ---
LC=.
.=500+LC
START:      MOV      #400,100(R1)
            MOV      A,B(R0)
            MOV      #L,150
            MOV      X,@#60
            MOV      K,@B
            JSR      PC,SUB1
            HALT
A:          .WORD    C
B:          .BLKB    30
            .END     START
```

```
            .TITLE          SUB1
            .GLOBL          ---
```

```
K=12
SUB1:   MOV         @45(R1),L
        MOV         #K,B(R3)
        MOV         #B,76
        MOV         X, @#M
        JSR         PC,SUB2
        MOV         177700,B
        RTS         PC
        .WORD       X
L:      .BLKW       K
M:      .WORD       K-20
        .END


        .TITLE      SUB2
        .GLOBL      ---
SUB2:   MOV         #12,L
        MOV         X,@A(R5)
        MOV         300,@#Y
        BPL         Z
        JMP         L(R1)
Z:      MOV         44(R1),1234
        RTS         PC
X:      .WORD       Y
        .BLKW       X-Z
Y:      .BLKB       16
        .END
```

9.5. Заменить нижеследующий программный сегмент точно таким же, но записанным в рамках ПНК. (Не используйте других регистров, кроме R3, R4 и PC!)

```
CLR 100
MOV #K, R3
MOV L(R3), R4
```

9.6. Написать в рамках ПНК программу, которая очищает массив из 100_{10} слов, предшествующий начальной ячейке программы.

Глава 10

БОЛЕЕ СЛОЖНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА АССЕМБЛЕРА

В этой главе будут описаны некоторые возможности ассемблера, которые существенно облегчают написание программ на языке ассемблера. В частности, будут рассмотрены макрокоманды, повторение директив и условное ассемблирование.

10.1. Макрокоманды

В гл. 6 перечислены преимущества модульного принципа записи программ и показано, как можно применить подпрограммы для осуществления такого принципа. Следует, однако, помнить, что использование подпрограмм точно так же приводит и к издержкам – требуется время для передачи аргументов, для хранения и восстановления содержимого регистров и для выполнения команд JSR и RTS. Нетрудно представить себе подпрограмму, в которой время, затрачиваемое на эти дополнительные действия, сравнимо с временем выполнения собственно подпрограммы или превышает его.

Рассмотрим, например, подпрограмму DIV8, которая делит содержимое R0 на 8:

```
DIV8: ASR R0   ;АРИФМЕТИЧЕСКИЙ СДВИГ
      ASR R0   ; (R0) НА ТРИ РАЗРЯДА
      ASR R0   ;ВПРАВО
      RTS PC   ;ВЫХОД
```

Чтобы разделить содержимое R3 на 8, можно вызвать DIV8 так:

```
MOV  R0, -(SP)   ;СОХРАНЕНИЕ (R0)
MOV  R3, R0      ;ПЕРЕДАЧА АРГУМЕНТОВ DIV8
JSR  PC, DIV8    ;ВЫЗОВ DIV8
MOV  R0, R3      ; (R3) = (R3)/8
MOV  (SP)+, R0   ;ВОССТАНОВЛЕНИЕ (R0)
```

Отсюда видно, что требуется шесть дополнительных команд для выполнения подпрограммы, которая состоит только из трех команд. Если рассматривать время выполнения в качестве определяющей характеристики, то лучше не использовать такую подпрограмму, а просто написать

```
ASR R3   ;АРИФМЕТИЧЕСКИЙ СДВИГ (R3)
ASR R3   ;НА ТРИ РАЗРЯДА
ASR R3   ;ВПРАВО
```

Однако предположим, что имеется большая программа, в которой деление различных регистров (не только R3, но и других) на 8 происходит часто и в различных местах. Запись трех команд ASR (плюс комментарии) во всех этих местах может показаться слишком монотонной и отнимающей много времени операцией. Это тот случай, когда могут помочь макрооперации. Можно определить указанную последовательность из трех команд как "макрокоманду", названную DIV8, в которой операнд с номером регистра включается как аргумент REG:

```
.MACRO  DIV8 REG
ASR     REG      ;АРИФМЕТИЧЕСКИЙ СДВИГ
ASR     REG      ; (REG) НА ТРИ РАЗРЯДА
ASR     REG      ;ВПРАВО
.ENDM
```

Сделав это один раз, можно в дальнейшем использовать "макровызовы", записывая

```
DIV8 R3
....
DIV8 R0
....
DIV8 R2
```

Во время первого прохода в процессе ассемблирования ассемблер "расширяет" каждый из этих трех вызовов, вставляя три команды, содержащиеся в описании макрокоманды DIV8, при замене REG соответствующим операндом, появляющимся на месте аргумента при вызове:

```
ASR R3   ; АРИФМЕТИЧЕСКИЙ СДВИГ
ASR R3   ; (R3) НА ТРИ РАЗРЯДА
ASR R3   ; ВПРАВО
. . . . .
ARS R0   ; АРИФМЕТИЧЕСКИЙ СДВИГ
ASR R0   ; (R0) НА ТРИ РАЗРЯДА
ASR R0   ; ВЛЕВО
. . . . .
ASR R2   ; АРИФМЕТИЧЕСКИЙ СДВИГ
ASR R2   ; (R2) НА ТРИ РАЗРЯДА
ASR R2   ; ВЛЕВО
```

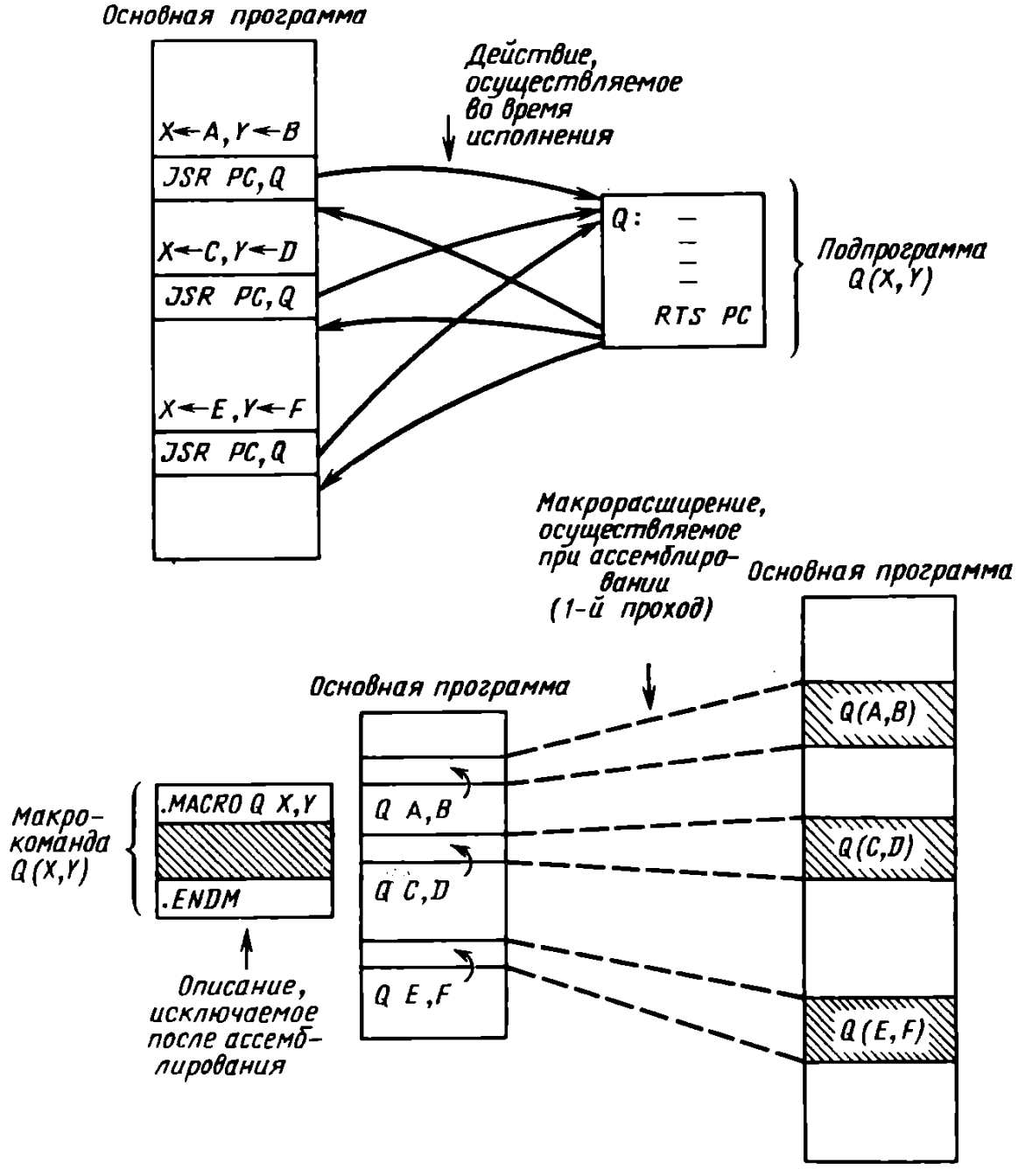


Рис. 10.1. Сравнение подпрограмм и макрокоманд

Как только первый проход ассемблера завершился, макроопределения для DIV8 больше не нужно и его можно исключить.

Отсюда можно видеть, что, кроме экономии времени и усилий программиста, макросредства делают программу более читабельной: "команда" DIV8 R3 значительно яснее передает суть, чем три команды ASR. Очевидно, что с помощью макрокоманд программист может сформировать новый набор команд так, что программа будет выглядеть, как если бы она была написана на языке высокого уровня, со всеми вытекающими из этого преимуществами.

На рис. 10.1 схематически показано использование макросредств и подпрограмм. Видно, что в то время как подпрограмма появляется в памяти только один раз, соответствующая макрокоманда дублируется столько раз, сколько раз она вызывается. Таким образом, нужно иметь в виду, что использование макрокоманд может настолько увеличить исходную программу, что потребуется существенно больше места, чем при использовании подпрограмм. Это важный момент, который нужно помнить, если предъявляются жесткие требования к объему занимаемой памяти[1].

(В некоторых ассемблерах, включая MACRO-11, макрорасширения фактически выполняются в объектных кодах во время второго прохода, а не в исходных кодах во время первого прохода. Однако макроопределения следует учесть во время первого прохода, чтобы сформировать таблицу имен. Эта альтернативная схема приводит к более экономному использованию памяти.)

10.2. Макроопределения и макровызовы

Общий формат *макроопределения* имеет вид

```
.MACRO имя d1, d2, . . . , dr   ← макрозаголовок[2]
                                }← макротело
.ENDM                           ← последний оператор
```

где $d_1, d_2, \ldots, d_r$ – *формальные аргументы*. Общий формат макровызова имеет вид

Метка: имя $a_1, a_2, \ldots, a_r$

где $a_1, a_2, \ldots, a_r$ – *фактические аргументы*. Макрокоманды должны быть определены до того, как они будут первый раз вызваны. Вызов распространяется (путем соответствующих подстановок) на макротело, в котором каждое d_1 заменяется на a_1, d_2 – на a_2 и т. д. После того как такое расширение выполнено, необходимость в использовании макроопределения отпадает и оно может быть исключено.

Если в макровызове появляется больше аргументов, чем в макроопределении, то избыточные аргументы игнорируются. Если в макровызове появля-

---

[1] Макрокоманды иногда называются "открытыми подпрограммами". Однако это название не следует использовать читателю, чтобы не путать их с обычными (или "замкнутыми") подпрограммами.

[2] За "именем" должна быть запятая; вместо запятых аргументы d_1 могут быть отделены пробелами.

ется меньше аргументов, чем в макроопределении, то пропущенные аргументы должны быть пустыми (не состоящими из символов). Число фактических аргументов, переданных макрокоманде с помощью макровызова, может быть установлено с помощью директивы

.NARG symb

которая, будучи помещенной внутри макроопределения, присваивает "symb" значение, равное числу фактических аргументов, появляющихся в данном вызове.

Примеры

1. Макрокоманда DPADD осуществляет сложение (A) и (B) с двойной точностью (см. разд 7.3):

```
Определение     .MACRO  DPADD A,B
                ADD     A,B
                ADC     A+2
                ADD     A+2,B+2
                 ENDM
Вызов           DPADD   ZZ,SUM
Расширение      ADD     ZZ,SUM
                ADC     ZZ+2
                ADD     ZZ+2,SUM+2
```

Замечание. Вызов предполагает, что операнды находятся в ZZ, ZZ + 2 и SUM, SUM + 2.

2. Макрокоманда SWAP взаимозаменяет (P) и (Q):

```
Определение     .MACRO  SWAP P,Q
                MOV     P,TEMP
                MOV     Q,P
                MOV     TEMP,Q
                .ENDM
Вызов       NOW: SWAP   F+6,(R3)
Расширение  NOW: MOV    F+6,TEMP
                 MOV    (R3),F+6
                 MOV    TEMP,(R3)
```

Замечание. Метка макровызова используется как метка первой команды в указанном расширении.

3. Макрокоманда FLIP располагает в обратном порядке содержимое в P1, P2, P3 и P4 (предполагается, что SWAP определяется как в примере 2):

```
Определение     .MACRO  FLIP P1,P2,P3,P4
                SWAP    P1,P4
                SWAP    P2,P3
                .ENDM
Вызов           FLIP    B,B+2,B+4,B+6
Расширение  (а) SWAP    B,B+6
                SWAP    B+2,B+4
            (б) MOV     B,TEMP
                MOV     B+6,B
                MOV     TEMP,B+6
                MOV     B+2,TEMP
                MOV     B+4,B+2
                MOV     TEMP,B+4
```

Замечание. Макрокоманды могут быть "вложены" друг в друга, причем число таких вложений любое, в этом случае расширение осуществляется в несколько последовательных стадий. Вложенные макрокоманды могут определяться в любом порядке до тех пор, пока вызов внешней макрокоманды происходит после того, как определена внутренняя.

4. Макрокоманда PRINT печатает один символ:

Определение

```
.MACRO  PRINT CHAR
TSTB    177564
BPL     .-4
MOV     #CHAR,177566
.ENDM
```

Вызов

```
PRINT   'Y
```

Расширение

```
TSTB    177564
BPL     .-4
MOV     #'Y,177566
```

5. Макрокоманда NULINE переводит телетайп на новую строку (PRINT считается определенным, как в примере 4):

Определение

```
.MACRO  NULINE
PRINT   15
PRINT   12
.ENDM
```

Вызов

```
NULINE
```

Расширение

(*а*)
```
PRINT   15
PRINT   12
```

(*б*)
```
TSTB    177564
BPL     .-4
MOV     #15,177566
TSTB    177564
BPL     .-4
MOV     #12,177566
```

Замечание. Существуют макрокоманды, вообще не имеющие аргументов.

6. Макрокоманда WAIT выполняет команду и пропускает данные:

Определение

```
        .MACRO  WAIT INSTR,SIZE,DATA,LOC
        INSTR
        JMP     LOC+SIZE+2
        .BYTE   DATA
LOC:    .BLKB   SIZE
        .ENDM
```

Вызов

```
        WAIT    <JSR PC,SUB3>,100.,<5,22,'H,'L>,MATRIX
```

Расширение

```
        JSR     PC,SUB3
        JMP     MATRIX+100.+2
        .BYTE   5,22,'H,'L
MATRIX: .BLKB   100.
```

Замечание. Фактический аргумент заключается в скобки ( ), если он содержит запятые и/или пробелы. Можно также использовать команду WAIT, но макроопределение предпочтительнее.

7. Макрокоманда DATA запоминает до 10 элементов данных в массиве LIST, за которым следует столько зарезервированных слов, сколько запоминается данных:

Определение

```
        .MACRO  DATA LIST,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10
        .NARG   N
LIST:   .WORD   P1,P2,P3,P4,P5,P6,P7,P8,P9,P10
        .BLKW   N-1
        .ENDM
```

Вызов

```
        DATA    KIT,17,1098.,'*
```

Расширение

```
        .NARG   N
KIT:    .WORD   17,1098.,'*,,,,,,,
        .BLKW   3
```

Замечание. Переменной N присваивается значение 4, которое является числом фактических аргументов, переданных макрокомандой DATA. Пропуск элементов данных в .WORD приводит к запоминанию нулей.

8. Макрокоманда ROTATE циклически сдвигает регистр, пересылает его содержимое в память и выполняет переходы:

Определение

```
        .MACRO  ROTATE D,REG,BASE,K,COND,DEST
        RO'D    REG
        MOV     REG,BASE'K
        B'COND  DEST
        .ENDM
```

Вызов

```
        ROTATE  L,R2,SWITCH,+30,EQ,OUT
        ...
        ROTATE  R,R5,MAT,-16,R,.+12
```

Расширение

```
        ROL     R2
        MOV     R2,SWITCH+30
        BEQ     OUT
        ...
        ROR     R5
        MOV     R5,MAT-16
        BR      .+12
```

Замечание. Апостроф (') используется для разделения соседних имен для того, чтобы избежать неопределенности. (Например, без апострофа определение ROTATE будет включать имена ROD, BASEK и BCOND.) В процессе расширения апостроф удаляется и смежные имена становятся "сцепленными".

9. Макрокоманда JOE запоминает строку "MESSAGE NO. k" по адресу Xk:

Определение

```
        .MACRO  JOE A,J
        MARY    A,\J
J=J+1
        .ENDM

        .MACRO  MARY X,K
X'K:    .ASCII  /MESSAGE NO. K/
        .ENDM
```

Вызов

```
I=0
        JOE     TEXT,I
        ...
        JOE     TEXT,I
        ...
        JOE     TEXT,I
```

Расширение

(*а*)
```
I=0
     MARY    TEXT,\I
I=I+1
     ...
     MARY    TEXT,\I
I=I+1
     ...
     MARY    TEXT,\I
I=I+1
```

(*б*)
```
I=0
TEXT0: .ASCII /MESSAGE NO. 0/
I=I+1
     ...
TEXT1: .ASCII /MESSAGE NO. 1/
I=I+1
     ...
TEXT2: .ASCII /MESSAGE NO. 2/
I=I+2
```

Замечание. Знак\I указывает, что при расширении целое число I должно быть подставлено в символьной форме. (Например, 2 в TEXT2 является *символом* 2.)

10.3. Локальные имена

Одно из основных правил написания программ на языке ассемблера состоит в том, чтобы ни одна из меток не была определена несколько раз, т. е. чтобы ни одна из меток не появилась более одного раза в поле метки. Однако существует исключение из этого правила: метки вида n$,где n десятичное целое между 1 и 65535, могут быть повторены столько раз, сколько нужно, если они будут отделены при этом по крайней мере одной "обычной" меткой (т. е. меткой, не имеющей вида n$). Например, метки

```
LABEL1: ...
3$      ...
15$     ...
18$     ...
LABEL2: ...
15$     ...
18$     ...
22$     ...
LABEL3: ...
3$      ...
22$     ...
```

записаны абсолютно верно. Ассемблер рассматривает метки n$ как *локальные* по отношению к области, ограниченной обычными метками, и формирует единственное определение для них в таблице имен.

Возвращаясь к макрокомандам, рассмотрим следующую макрокоманду MULT, которая вычисляет (C) ← (A) * (B) (см. разд. 6.5):

```
        .MACRO  MULT A,B,C
        CLR     C
LOOP:   DEC     B
        BMI     EXIT
        ADD     A,C
        BR      LOOP
EXIT:
        .ENDM
```

Предположим, что мы вызываем MULT дважды:

```
MULT  R1, R2, R3
...
MULT  P, Q, R                                    (1)
```

После расширения имеем[1]

```
        CLR   R3
LOOP:   DEC   R2
        BMI   EXIT
        ADD   R1,R3
        BR    LOOP
EXIT:
        ...
        CLR   R
LOOP:   DEC   Q
        BMI   EXIT
        ADD   P,R
        BR    LOOP
EXIT:
```

что, конечно, является неправильным, так как LOOP и EXIT определяются несколько раз. Чтобы исключить подобную ситуацию, нам следовало бы перечислить LOOP и EXIT в качестве параметров MULT:

```
.MACRO MULT A, B, C, LOOP, EXIT
...
.ENDM
```

и вместо (1) написать такой вызов:

```
MULT R1, R2, R3, LOOP1, EXIT1
...
MULT P, Q, R, LOOP2, EXIT2                       (2)
```

Однако если ни LOOP, ни EXIT не обращаются к внешним макрокомандам, то определение их в вызове не обязательно. Можно определить LOOP и EXIT как *локальные имена* (локальные по отношению к данной макрокоманде), перечисляя их в списке параметров макрокоманды следующим образом:

```
.MACRO MULT A, B, C, ? LOOP, ?EXIT
...
.ENDM                                            (3)
```

Если в вызове не указывается LOOP или EXIT (т. е. соответствующие фактические параметры равны нулю или отсутствуют), ассемблер заменяет их на 64$ и 65$ при первом расширении и на 66$ и 67$ при втором расширении и так далее. Итак, если MULT определяется как в (1) и вызывается как в (3), то расширение имеет вид

---

[1] Такая метка, как EXIT:, может появиться в строке, предшествующей той, которую она помечает. Фактически допускается появление любого числа отличающихся друг от друга меток (расположенных одна под другой), помечающих те же самые слова (и, следовательно, имеющих те же самые значения в таблице имен).

```
        CLR  R3
64$:    DEC  R2
        BMI  65$
        ADD  R1,R3
        BR   64$
65$:
        ...
        CLR  R
66$:    DEC  Q
        BMI  67$
        ADD  P,R
        BR   66$
67$:
```

Если два вызова разделяются "обычной" меткой, то метки, формируемые при втором расширении, те же, что и при первом, а именно: 64$ и 65$. Как было объяснено выше, это вполне корректно и не приводит к случаю многократного определения 64$ и 65$.

В общем случае локальные имена обозначаются в списке параметров макроопределения с помощью префикса ? . В случае отсутствия в вызове, они заменяются при расширении метками 64$, 65$, . . . , 127$. Если локальные имена определяются в вызове [как, например, в (2)], то расширение осуществляется так же, как и в обычном случае. Метки n$ будут повторяться, если вызовы разделяются обычными метками.

10.4. Директивы повторения

Иногда программа на языке ассемблера содержит последовательные повторения одинаковых или почти одинаковых кодовых последовательностей. В этом случае можно затратить значительно меньше усилий, если использовать ("повторяющую") директиву .REPT:

```
.REPT exp  ← заголовок
           } повторяющийся блок
.ENDM      ← последняя директива[1]
```

Во время первого прохода при ассемблировании ассемблер дублирует повторяющийся блок столько раз, сколько задано параметром "exp".

Примеры

1. Пропустить четыре пустые строки (используем макрокоманду MULINE примера 5 в разд. 10.2):

```
.REPT 5
MULINE
.ENDM
```

2. Сформировать массив A в 100_{10} слов, каждое слово которого содержит адрес следующего слова, за исключением последнего слова, в котором содержится адрес первого слова. (Этот массив называется "кольцевой список".)

```
A:
   .REPT 99
   .WORD . + 2
   .ENDM
   .WORD A
```

[1] Здесь можно использовать также директиву .ENDR

3. Заполнить массив TABLE кодами ASCII символов от A до Z:

```
CHAR='A
TABLE:
        .REPT  26.
        ..BYTE  CHAR
CHAR=CHAR+1
        .ENDM
```

4. Занести содержимое TAB, TAB + 1, TAB + 2, . . . , TAB + 16 в системный стек, используя макрокоманду PUSH (см. пример 9 в разд. 10.2 для объяснения знака \ ):

```
        .MACRO  PUSH K
        MOVB    TAB+K,-(SP)
        .ENDM
I=0
      .REPT     17
      PUSH      \I
I=I+1.
      .ENDM
```

5. Макрокоманда SAVE заносит содержимое TAB + I, TAB + I + 1, TAB + I + 2, ... , TAB + J (J ≥ I) в системный стек, используя макрокоманду PUSH примера 4:

Определение

```
      .MACRO  SAVE I,J
COUNT=I
      .REPT   J-I+1
      PUSH    \COUNT
COUNT=COUNT+1
      .ENDM
      .ENDM
```

Вызов

```
      SAVE    12,22
```

Расширение

(*а*)
```
COUNT=12
      PUSH    \12
COUNT=13
      PUSH    \13
      .
      .
      .
COUNT=22
      PUSH    \22
```

} 22 - 12 + 1 = 11 Макровызовы

(*б*)
```
COUNT=12
        MOVB TAB+12,-(SP)
COUNT=13
        MOVB TAB+13,-(SP)
        .
        .
        .
COUNT=22
        MOVB TAB+22,-(SP)
```

Другой полезной директивой повторения является директива .IRP ("неопределенное повторение")

.IRP d < $a_1, a_2, \ldots, a_r$ > ← заголовок

} повторяющийся блок

.ENDM ← последняя директива

где d – формальный аргумент; $a_1, a_2, \ldots, a_r$ – фактические аргументы. Ассемблер (во время первого прохода) осуществляет дублирование повторяющегося блока r раз, причем сначала d заменяется на a_1, затем на a_2 и т. д. Например, для того чтобы восстановить значения R0, R3, R4 и R5, хранящиеся в системном стеке, можно написать:

```
.IRP REG, < R0, R3, R4, R5 >
MOV (SP) +, REG
.ENDM
```

Аналогичная директива:

```
.IRPC d, str ← заголовок
             } повторяющийся блок
.ENDM        ← последняя директива
```

где d – формальный аргумент; "str" – строка символов. Ассемблер дублирует повторяющийся блок: сначала d заменяется первым символом строки "str", затем вторым символом строки "str" и т. д. Например, для восстановления R0, R3, R4 и R5 из системного стека можно написать

```
.IRPC N, 0345
MOV (SP) +, R'N
.ENDM
```

(Как и в макрокомандах, знак ' служит для разделения смежных имен, но он удаляется при дублировании.)

10.5. Условное ассемблирование

Директивы условного ассемблирования дают возможность программисту включать или исключать сегмент исходного текста в зависимости от определенных условий. Эта возможность часто используется в макроопределениях, когда значения параметров (определенные во время первого прохода в процессе ассемблирования) определяют, какая версия макроопределения должна быть расширена.

Общая форма условной директивы такова:

```
.IF усл, s ← заголовок
           } условный блок
.ENDC      ← последняя директива
```

где "усл" определяет условие, которому s может или не может удовлетворять. Если s удовлетворяет указанному условию, то условный блок ассемблируется; в противном случае он игнорируется. В табл. 10.1 перечислены некоторые допустимые условные директивы.

Если директива .IFF появляется внутри условного блока, то часть блока, лежащая ниже .IFF, ассемблируется только в том случае, если удовлетворяется предшествующее условие IF. Таким образом, директива .IFF делит указанный блок на два подблока, из которых фактически ассемблируется только один.

Таблица 10.1. **Условные директивы**

Директива	Условие ассемблирования блока
.IF EQ, s	s = 0
.IF NE, s	s ≠ 0
.IF GT, s	s > 0
.IF LE, s	s ≤ 0
.IF LT, s	s < 0
.IF GE, s	s ≥ 0
.IF DF, s	s определено
.IF NDF, s	s не определено
.IF B,<s>	макроаргумент s опущен (отсутствует)
.IF NB,<s>	макроаргумент s не опущен (присутствует)

Примеры

1. Макрокоманда BRANCH генерирует команду BR X, если относительное расстояние до X меньше, чем 255_{10} байтов, в противном случае генерируется команда JMP X. Имя X должно быть определено при вызове этой макрокоманды (т. е. указанный переход должен быть переходом назад):

Определение

```
.MACRO  BRANCH X
.IF     LT,.-X-255.
BR      X
.IFF
JMP     X
.ENDC
.ENDM
```

Замечание. Здесь директива IF вложена в макрокоманду. Директива .IFF делит блок IF на два взаимно исключающих подблока (BR X и JMP X).

2. Макрокоманда GOTO, L, X, REL, Y (где REL может быть EQ, NE, GT и т. д.) генерирует команды CMP, X, Y и B'REL L (которые во время прогона вызывают переход на метку L, если отношение между X и Y соответствует операции REL). Вызов вида GOTO L генерирует безусловный переход BR L (это попытка ввести в язык ассемблера команды языка Фортран):

Определение

```
.MACRO  GOTO L,X,REL,Y
.IF     B,<REL>
BR      L
.IFF
CMP     X,Y
B'REL   L
.ENDC
.ENDM
```

Вызов

```
GOTO    LOOP,SUM,NE,#15
...
GOTO    EXIT
```

Расширение

```
CMP     SUM,#15
BNE     LOOP
...
BR      EXIT
```

3. Макрокоманда MAX заносит в R0 максимальное значение одного из трех сравниваемых аргументов. Генерируемый код зависит от числа аргументов, которое определяется с помощью директивы NARG:

Определение

```
        .MACRO  MAX A,B,C,?NEXT,?OUT
        .NARG   K
        MOV     A,R0
        .IF     NE,K-1
        .IF     NE,K-2
        CMP     C,R0
        BLE     NEXT
        MOV     C,R0
        .ENDC
NEXT:   CMP     B,R0
        BLE     OUT
        MOV     B,R0
        .ENDC
OUT:
        .ENDM
```

Вызов

```
        MAX     P
        ...
        MAX     P,Q
        ...
        MAX     P,Q,R
```

Расширение

```
(K=1)
        MOV     P,R0
 65$:
        ...
(K=2)
        MOV     P,R0
        CMP     Q,R0
        BLE     67$
        MOV     Q,R0
 67$:
        ...
(K=3)
        MOV     P,R0
        CMP     R,R0
        BLE     68$
        MOV     R,R0
 68$:   CMP     Q,R0
        BLE     69$
        MOV     Q,R0
 69$:
```

4. Условное ассемблирование может быть использовано для осуществления "макрорекурсии". Макрокоманда POWER, показанная ниже, умножает содержимое X на 2^N путем сдвига X влево (арифметически) N раз. Фактически макрокоманда рекурсивно вызывает сама себя, причем указанная рекурсия заканчивается директивой .IF, которая сравнивает значение счетчика сдвигов COUNT с N:

Определение

```
        .MACRO  POWER X,N
        ASL     X
COUNT=COUNT+1
        .IF     NE,COUNT-N
        POWER   X,N
        .ENDC
        .ENDM
```

Вызов

```
COUNT=0
        POWER   R5,6
```

Расширение

```
        ASL     R5
        ASL     R5
        ASL     R5
        ASL     R5
        ASL     R5
        ASL     R5
```

Упражнения

10.1. Макрокоманда FUN определяется следующим образом:

```
        .MACRO  FUN A,B,C,X,Y,N,M,?LOOP
LOOP:   X'Y
        A'B     (PC)
        BV'C    LOOP
        .WORD   N,"B'M
        .ENDM
```

Расширить следующие четыре вызова, которые появляются последовательно в программе. Показать содержимое, полученное в результате выполнения директивы .WORD после каждого вызова.

```
FUN     AS,L,C,CL,C,15.,C
FUN     RO,R,S,SE,V,'X,X
FUN     RO,L,S,CL,V,13,< >,L
FUN     AS,R,C,SE,N,<1,2,3>,<,>
```

10.2. Макрокоманды MAC1 и MAC2 определяются следующим образом:

```
                .MACRO  MAC1 K,X
                MOVB    R'K,X+K
                .ENDM
                .MACRO  MAC2 N,Z
I=0
                .REPT   N
                MAC1    \I,Z
I=I+1
                .ENDM
                .ENDM
```

Объясните действия MAC2 и расширьте вызов

MAC2 5, TEMP

10.3. а) Макрокоманда STORE определяется следующим образом:

```
                .MACRO  STORE X,N
                MOVB    #I,X+I
        I=I+1
                .IF     NE,I-N
                STORE   X,N
                ENDC
                ENDM
```

Расширьте вызов

I = 0

STORE TAB, 7

б) Напишите *нерекурсивную* макрокоманду, которая осуществляет те же операции, что и STORE.

10.4. Макрокоманда FOO определяется следующим образом:

```
.MACRO    FOO R5,ITEMS,LENGTH,BETA
CMP'BETA  R0, R1
R5
.WORD     ITEMS
.IF       B,<BETA>
.BLKW     LENGTH
.IFF
.BLKB     LENGTH
.ENDC
.ENDM
```

Расширить вызовы

```
L5:  FOO ( BPL L5) , ( 5, 25, 3 ) , 100, B
     FOO ( BNE L5 ) , , 52
```

10.5. Написать подпрограмму POWER, которая умножает содержимое X на 2^N (где X находится в R0, а N – в R1). Сравнить полученную подпрограмму с макрокомандой POWER в разд. 10.5, указав их преимущества и недостатки.

10.6. Написать макрокоманду XOR, которая, будучи вызвана в виде XOR, A, B, помещает в R0 результат операции "ИСКЛЮЧАЮЩЕЕ ИЛИ" над A и B (т. е. R0 будет содержать единицы только в тех разрядах, где A и B различны).

10.7. Написать макрокоманду JMPSR с заголовком вида

.MACRO JMPSR SUB, REGS, ARRAY

которая запоминает список регистров, указанных в REGS, в массиве слов, начинающихся с ARRAY, исполняет команду JSR PC, SUB и затем восстанавливает содержимое всех указанных регистров. Например, при вызове

JMPSR DPADD, 024, TEMP

происходит запоминание R0, R2 и R4 в TEMP, TEMP + 2 и TEMP + 4, исполняется команда JSR PC, DPADD, а затем восстанавливается содержимое R0, R2 и R4.

10.8. Некая (вымышленная) ЭВМ, названная SIMCOM (SIMple COMputer), имеет один регистр общего назначения, названный *аккумулятором* (ACCumulator, или ACC). Его память и представление чисел аналогичны тому, что имеет место в PDP-11. Ниже представлен набор команд, написанных на языке ассемблера SIMCOM (где d – адрес в памяти):

Команда	Действие
LDA d	*Загрузка*: (ACC) ← (d)
STO d	*Запоминание*: (d) ← (ACC)
ADD d	*Сложение*: (ACC) ← (ACC) + (d)
SUB d	*Вычитание*: (ACC) ← (ACC) – (d)
TRA d	*Переход*: переход к d (БЕЗУСЛОВНЫЙ)
TRZ d	*Переход* ПО РАВЕНСТВУ НУЛЮ: переход к d, если (ACC = 0)
TRN d	*Переход* ПО МИНУСУ: переход к d, если (ACC < 0)
HALT	*Останов*

(Команды ввода-вывода опущены.) Директивы ассемблера в SIMCOM следующие:

Директива	Действие
OCT n	Запоминание восьмеричной константы
BSS n	Резервирование n (восьмеричных) слов
END	Конец ассемблирования

Используя R0 для моделирования ACC, напишите те макрокоманды PDP-11, чьи вызовы совпадают с одиннадцатью указанными выше командами и директивами SIMCOM и которые осуществляют те же действия. Например, макрокомандой для команды загрузки будет

```
.MACRO  LDA X
MOV     X, R0
.ENDM
```

Используя эти макрокоманды, можно выполнить SIMCOM-программы на PDP-11. Каждая SIMCOM-команда или директива будет фактически соответствовать макровызову PDP-11 (например, команда LDA SUM будет фактически командой MOV SUM, R0); в результате будет осуществлено моделирование команд и директив SIMCOM.

Используйте ваши макрокоманды в SIMCOM-программе, которая делит (A) на (B) и помещает результат (с усеченной дробной частью) в C.

ПРИЛОЖЕНИЕ А. РЕЖИМЫ АДРЕСАЦИИ PDP-11

Режим	Название режима или адресации	Ассемблер	Местоположение операнда	Пояснение
1	2	3	4	5
0	**Регистровый**	Rn	Rn	Операнд в регистре Rn
1	Регистровый косвенный	(Rn)	(Rn)	Адрес операнда в регистре Rn
2	Автоинкрементный	(Rn)+	(Rn)	Адрес операнда в регистре Rn; (Rn) ← (Rn) + 2 после выборки операнда [1]
3	Автоинкрементный косвенный	@(Rn)+	((Rn))	Адрес адреса операнда в регистре Rn; (Rn) ← ← (Rn) + 2 после выборки операнда
4	Автодекрементный	–(Rn)	(Rn)	Перед вычислением адреса [2] (Rn) ← (Rn) – 2; адрес операнда в регистре Rn
5	Автодекрементный косвенный	@–(Rn)	((Rn))	Перед вычислением адреса (Rn) ← (Rn) – 2; адрес адреса операнда в регистре Rn
6	Индексный	X(Rn)	X + (Rn)	Адрес операнда X + (Rn). Адрес X в PC; (PC) ← (PC) + 2 после выборки X
7	Индексный косвенный	@ X(Rn)	(X + + (Rn))	Адрес адреса операнда равен X + (Rn). Адрес X в PC; (PC) ← (PC) + 2 после выборки X
	PC-адресация			
2	Непосредственная	# k		Операндом является k. (k следует за командой)
3	Абсолютная	@ # A	A	A является адресом операнда. (A следует за командой)
6	Относительная	A	A	A является адресом операнда. [A – (PC) следует за командой]
7	Относительная косвенная	@ A	(A)	A является адресом адреса операнда. [A – – (PC) следует за командой]

[1] Но (Rn) ← (Rn) + 1, если команда является байтовой и n < 6.

[2] Но (Rn) ← (Rn) – 1, если команда является байтовой и n < 6.

Машинный язык	Ассемблер	Название команды	Алгоритм выполнения
1	2	3	4
		Одноадресные	
0050 DD	CLR d	ОЧИСТКА (clear)	(d) ← 0
0051 DD	COM d	ИНВЕРТИРОВАНИЕ (complement)	(d) ← ∿ (d)
0052 DD	INC d	ИНКРЕМЕНТ (increment)	(d) ← (d) + 1
0053 DD	DEC d	ДЕКРЕМЕНТ (decrement)	(d) ← (d) – 1
0054 DD	NEG d	СМЕНА ЗНАКА (negate)	(d) ← –(d)
0057 DD	TST d	ПРОВЕРКА (test)	(d) ← (d)
0060 DD	ROR d	ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО (rotate right)	(d) ← (d), сдвинутое вправо на 1 разряд
0061 DD	ROL d	ЦИКЛИЧЕСКИЙ СДВИГ ВЛЕВО (rotate left)	(d) ← (d), сдвинутое влево на 1 разряд
0062 DD	ASR d	АРИФМЕТИЧЕСКИЙ СДВИГ ВПРАВО (arith. shift right)	(d) ← (d)/2
0063 DD	ASL d	АРИФМЕТИЧЕСКИЙ СДВИГ ВЛЕВО (arith. shift left)	(d) ← 2 * (d)
0003 DD	SWAB d	ПЕРЕСТАНОВКА БАЙТОВ (swap bytes)	$(d)_{млад} \leftarrow (d)_{стар}$
0055 DD	ADC d	СЛОЖЕНИЕ С ПЕРЕНОСОМ (add carry)	(d) ← (d) + C
0056 DD	SBC d	ВЫЧИТАНИЕ ПЕРЕНОСА (subtract carry)	(d) ← (d) – C
0001 DD	JMP d	БЕЗУСЛОВНЫЙ ПЕРЕХОД (jump)	(PC) ← d
		Двухадресные	
01SS DD	MOV s,d	ПЕРЕСЫЛКА (move)	(d) ← (s)
02SS DD	CMP s,d	СРАВНЕНИЕ (compare)	формирование (s – d)
06SS DD	ADD s,d	СЛОЖЕНИЕ (add)	(d) ← (s) + (d)
16SS DD	SUB s,d	ВЫЧИТАНИЕ (subtract)	(d) ← (d) – (s)
03SS DD	BIT s,d	ПРОВЕРКА РАЗРЯДОВ (bit test)	формирование (s) ∧ (d)[1)]
04SS DD	BIC s,d	ОЧИСТКА РАЗРЯДОВ (bit clear)	(d) ← [∿ (s)] ∧ (d)[1)]
05SS DD	BIS s,d	УСТАНОВКА РАЗРЯДОВ (bit set)	(d) ← (s) ∨ (d)[1)]

[1)] Знак ∨ соответствует операции "ИЛИ", ∧ – "И", ∿ – "НЕ".

Продолжение приложения Б

Базовый код	Ассемблер	Название перехода[1]
1	2	3
000400	BR a	БЕЗУСЛОВНЫЙ (unconditionally)
001000	BNE a	ПО НЕРАВЕНСТВУ НУЛЮ (Z = 0) (not equal to 0)

[1] Правила осуществления операций: ∀ : 0 + 0 = 0, 0 + 1 = 1, 1 + 1 = 0; ∨: 0 + 0 = 0, 0 + 1 = 1, 1 + 1 = 1.

Базовый код	Ассемблер	Название перехода
1	2	3
001400	BEQ a	ПО РАВЕНСТВУ НУЛЮ (Z = 1) (equal to 0)
100000	BPL a	ПО ПЛЮСУ (N = 0) (plus)
100400	BMI a	ПО МИНУСУ (N = 1) (minus)
102000	BVC a	ПО ОТСУТСТВИЮ ПЕРЕПОЛНЕНИЯ (V = 0) (overflow clear)
102400	BVS a	ПО ПЕРЕПОЛНЕНИЮ (V = 1) (overflow set)
103000	BCC a	ПО ОТСУТСТВИЮ ПЕРЕНОСА (C = 0) (carry clear)
103400	BCS a	ПО ПЕРЕНОСУ (C = 1) (carry set)
002000	BGE a	ПО "БОЛЬШЕ ИЛИ РАВНО НУЛЮ" (N ∀ V = 0) (greater than or equal to 0)
002400	BLT a	ПО "МЕНЬШЕ НУЛЯ" (N ∀ V = 1) (less than 0)
003000	BGT a	ПО "БОЛЬШЕ НУЛЯ" [Z ∨ (N ∀ V) = 0] (greater than 0)
003400	BLE a	ПО "МЕНЬШЕ ИЛИ РАВНО НУЛЮ" [Z ∨ (N ∀ V) = 1] (less than or equal to 0)
101000	BHI a	БЕЗЗНАКОВЫЙ ПО "БОЛЬШЕ НУЛЯ" (C ∨ Z = 0) (higher)
101400	BLOS a	БЕЗЗНАКОВЫЙ "ПО МЕНЬШЕ ИЛИ РАВНО НУЛЮ" (C ∨ Z = 1) (lower or same)
103000	BHIS a	БЕЗЗНАКОВЫЙ ПО "БОЛЬШЕ ИЛИ РАВНО НУЛЮ" (C = 0) (higher or same)
103400	BLO a	БЕЗЗНАКОВЫЙ ПО "МЕНЬШЕ НУЛЯ" (C = 1) (lower)

Окончание приложения Б

Машинный язык	Язык ассемблера	Результат операции
		Операции с кодами условий
000241	CLC	Очистка (сброс) C
000242	CLV	Очистка (сброс) V
000244	CLZ	Очистка (сброс) Z
000250	CLN	Очистка (сброс) N
000257	CCC	Очистка (сброс) всех кодов условий (C, V, Z, N)
000261	SEC	Установка C
000262	SEV	Установка V
000264	SEZ	Установка Z
000270	SEN	Установка N
000277	SCC	Установка 1 во все разряды кода условий (C, V, Z, N)
		Разнородные операции
004nDD	JSR Rn, d	Переход к подпрограмме на метку d } Rn – регистр связи
00020n	RTS Rn	Возврат из подпрограммы } Rn – регистр связи
000001	WAIT	Ожидание прерывания
000002	RTI	Возврат после прерывания

Машинный язык	Язык ассемблера	Результат операции
000003	BPT	Захват точки прерывания (командное прерывание)
000240	NOP	(нет операции)
000000	HALT	Останов

ПРИЛОЖЕНИЕ В. ОСНОВНЫЕ ДИРЕКТИВЫ MACRO-11

Директивы	Объяснение
1	2
.TITLE title	Имя, используемое в листинге ассемблера
.END	Конец исходной программы
Хранения данных	
.BYTE $exp_1, \ldots, exp_n$	Запись двоичных значений $exp_1, \ldots, exp_n$ в последовательных байтах
.WORD $exp_1, \ldots, exp_n$	Запись двоичных значений $exp_1, \ldots, exp_n$ в последовательных словах
.ASCII /string/	Запись значений кода ASCII указанной строки (string) в последовательных байтах (/соответствует любому символу, не содержащемуся в строке)
.ASCIZ /string/	Директива аналогична ASCII, но вставляет нулевой байт после последнего символа строки
Управления счетчиком команд	
. = exp	Установить в счетчике значение exp (перемещаемое)
.EVEN	Если текущее значение счетчика нечетное, то добавляется 1
.ODD	Если текущее значение счетчика четное, то добавляется 1
.BLKB exp	Резервирование блока памяти размером в exp байтов (абсолютное)
.BLKW exp	Резервирование блока памяти размером в exp слов (абсолютное)
Присваивания	
sym-exp	Присваивает sym значение exp
Глобальности	
.GLOBL $sym_1, \ldots, sym_n$	Определяет sym_i как глобальные имена
Указания типа чисел и символов	
.RADIX r	Считать впредь все числа имеющими основание r (2, 4, 8 или 10)
↑ Dn или n	Считать n десятичным числом
↑ On	Считать n восьмеричным числом
↑ Bn	Считать n двоичным числом
↑ Cn	Взять дополнение n до 1 (обратный код n)
'p	Использовать код ASCII символа p
"$p_1 p_2$	Использовать код ASCII двухсимвольной строки $p_1 p_2$

ПРИЛОЖЕНИЕ Г.
СТЕПЕНИ ОСНОВАНИЯ 2

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1 024
11	2 048
12	4 096
13	8 192
14	16 384
15	32 768
16	65 536
17	131 072
18	262 144
19	524 288
20	1 048 576
21	2 097 152
22	4 194 304
23	8 388 608
24	16 777 216
25	33 554 432
26	67 108 864
27	134 217 728
28	268 435 456
29	536 870 912
30	1 073 741 824
31	2 147 483 648
32	4 294 967 296
33	8 589 934 592
34	17 179 869 184
35	34 359 738 368
36	68 719 476 736
37	137 438 953 472
38	274 877 906 944
39	549 755 813 888
40	1 099 511 627 776
41	2 199 023 255 552
42	4 398 046 511 104
43	8 796 093 022 208
44	17 592 186 044 416
45	35 184 372 088 832
46	70 368 744 177 644
47	140 737 488 355 328

ПРИЛОЖЕНИЕ Д. ЗАМЕТКИ О СТИЛЕ ПРОГРАММИРОВАНИЯ

Хорошая программа на языке ассемблера должна быть корректной, иметь хорошую структуру и быть надлежащим образом документированной. Окончательная версия программы должна гарантировать получение правильного ответа. Нижеприведенные критерии можно классифицировать как "стиль программирования". Хороший стиль программирования включает подробную аннотацию и комментарии, четкую организацию и удачное кодирование. Помните, что человек, читающий листинг вашей программы, имеет лишь общее представление о задаче, которую вы хотите решить, и почти ничего не знает о ваших приемах программирования. Ниже приведены некоторые советы, которые могут помочь при разработке программ.

1. Программы должны быть модульными. Четко определенные части алгоритма следует представлять в виде подпрограмм.

2. Каждая программа или подпрограмма должна иметь краткий вступительный комментарий, описывающий цели программы или подпрограммы, используемый алгоритм, смысл наиболее важных внутренних переменных, вид и форматы входных и выходных данных; для подпрограмм – описание значений и форматов параметров и форму их вызова (например, JSR PC, SUB), а также список регистров, используемых этой подпрограммой.

3. Используйте стандартизованную форму записи команд. Например:

поле метки: столбец 1;

поле операции: столбец 9;

поле операнда: столбец 17;

поле комментария: столбец 33.

Если поле операнда выходит за рамки, ограниченные столбцом 32, то оставьте пробел и начинайте комментарий.

4. Комментарий должен сопровождать почти каждую команду. Такой комментарий на полях должен быть не просто словесной трансляцией данной команды, а объяснением того, что она делает в программе. Например, комментарием для команды BR LOOP должен быть не просто "переход к LOOP", а "возврат для обработки следующего символа".

5. Если комментарий на полях не сделал команды достаточно ясными, не стесняйтесь включить полный комментарий между командами.

6. Комментарий на полях, занимающий несколько строк, должен записываться последовательно с отступом строки относительно предыдущей. Например,

```
ASR R0  ; ДЕЛЕНИЕ
ASR R0  ;    ОПЕРАНДА
ASR R0  ;         НА ВОСЕМЬ
```

7. Разделяйте отдельные части кода строками из пробелов (или строкой, состоящей только из знака, в первой колонке).

8. Избегайте сложных и заумных командных последовательностей. Если требуется включить такую последовательность, обеспечьте ее соответствующим полным комментарием.

9. Уделяйте внимание всем особенностям. Например, при написании программы деления включите контроль равенства нулю делителя. Если найдено, что делитель равен нулю, должны быть осуществлены некоторые специальные действия (должно быть напечатано сообщение об ошибке, установлен определенный код условий и т. д.).

10. Любыми способами избегайте написания самомодифицирующихся программ (в которых одна или больше команд изменяются в процессе исполнения). Такие программы крайне трудно отлаживать.

11. Подпрограммы, которые могут быть использованы в других программах (например, программы ввода-вывода, умножения и деления), должны быть ясны и очевидны настолько, насколько возможно для вызывающей программы. Содержимое всех ре-

гистров, используемых в таких подпрограммах, должно быть сохранено в момент входа и восстановлено перед выходом.

12. Области данных должны быть сгруппированы в одном месте в программе и не должны смешиваться с исполняемыми кодами.

13. По мере возможности используемые имена должны быть мнемоническими. Например, точки в программе должны быть поименованы LOOP ("цикл"), NEXTCH ["следующий символ" (NEXT CHaracter)], EXIT ("выход"), а не X1, X2, X3. Аналогично запоминаемые ячейки должны быть поименованы TEMP, TABLE, CONST и т. д., а не P, Q, R.

14.Для лучшей читабельности предварительно определите имена для абсолютных адресов и констант, используемых в программе: например PSR = 177776, BELL = 7, MASK = 177700. Для обозначения регистров R6 и R7 используйте соответственно SP и PC.

15. Не используйте команд, приводящих к дополнительной затрате времени, если их неэффективность очевидна. Например, пишите

```
CLR R0        а не MOV # 0, R0
DEC R3        а не SUB # 1, R3
TST (SP) +    а не ADD # 2, SP
JMP LOOP      а не MOV # LOOP, PC
```

16. Не вставляйте команд контроля и сравнения там, где в них нет необходимости. Например, в блоке

```
DEC  R4
CMP  R4, # 0
BNE  LOOP
```

команда CMP избыточна, а в блоке

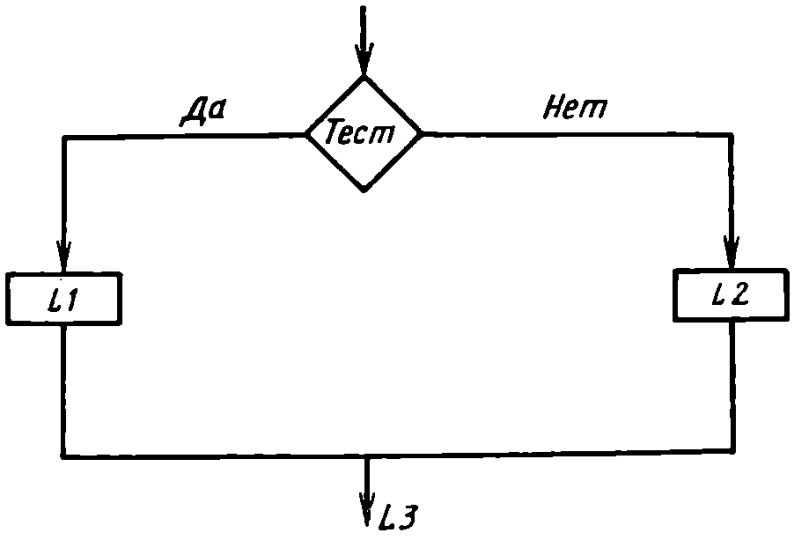
```
SUB  A, B
TST  B
BEQ  NEXT
```

избыточна команда TST.

17. Избегайте использования команд в качестве данных, даже если это может сохранить место. Например, не пишите

```
MOV  @PC, R1
BIC  P1, R5
```

18. Блок-схема, приведенная ниже,



должна быть закодирована следующим образом:

```
        TST . . .
        BNE L2
L1:     . . .
        . . .
        BR L3
L2:     . . .
        . . .
L3:     . . .
```

а не так:

```
        TST . .
        BNE L2
L1:     . . .
        . . .
L3:     . . .
        . . .
L2:     . . .
        . . .
        BR L3
```

19. Для резервирования слова памяти пишите .BLKW1, а не .WORD 0. (Последняя директива указывает, что начальное содержимое слова является важным.)

Оглавление

Стр.

Артур Гилл

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА ДЛЯ PDP-11

Редактор *О. В. Толкачева*
Обложка художника *Н. М. Ковалевой*
Художественный редактор *Л. Н. Сильянов*
Технический редактор *Л. А. Горшкова*
Корректор *Л. А. Буданцева*

ИБ № 364

Подписано в печать 17.10.83 г. Формат 60x90/16 Бумага кн.-ж. Гарнитура "Пресс-роман" Печать офсетная Усл. печ. л. 10,0 Усл. кр.-отт. 10,25 Уч.-изд. л. 10,10 Тираж 10 000 экз. Изд. № 20282 Зак. № 1867 Цена 65 к.
Издательство "Радио и связь". 101000, Москва, Почтамт, а/я 693

Московская типография № 4 Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 129041, Москва, Б. Переяславская ул., д. 46