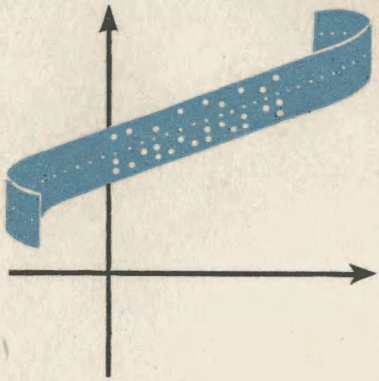


Р.С.ГУТЕР
П.Т.РЕЗНИКОВСКИЙ
С.М.РЕЗНИК

ПРОГРАММИРОВАНИЕ И ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

ВЫПУСК 1

Р. С. ГУТЕР, П. Т. РЕЗНИКОВСКИЙ, С. М. РЕЗНИК

ПРОГРАММИРОВАНИЕ И ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

ВЫПУСК ПЕРВЫЙ

ОСНОВЫ ПРОГРАММИРОВАНИЯ. АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

Допущено Министерством приборостроения, средств автоматизации и систем управления в качестве учебника для средних специальных учебных заведений по специальности «Прикладная математика»

ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА 1971

518
Г 97
УДК 518.5

Программирование и вычислительная математика, вып. 1. Основы программирования. Алгоритмические языки. Гутер Р. С., Резниковский П. Т., Резник С. М.

Книга является учебником для специальности «Прикладная математика» в средних специальных учебных заведениях (техникумах) и соответствует утвержденной программе. Выпуск 1 рассчитан на первый курс. Книга может быть также использована студентами вузов, инженерами и научными работниками нематематических специальностей для изучения программирования на трехадресных вычислительных машинах и алгоритмического языка Алгол.

Учебник состоит из четырех частей и приложения. Первая часть, «Приближенные вычисления», носит вводный характер. Вторая часть, «Основы программирования», посвящена основам программирования для трехадресных машин типа М-20. В третьей части рассматривается машинно-ориентированный язык Автокод 1 : 1 и международный алгоритмический язык Алгол. Четвертая часть посвящена организации современных вычислительных систем.

Рисунков 53, таблиц 62.

Гутер Рафаил Самойлович, Резниковский Павел Тувьевич, Резник Семен Моисеевич

ПРОГРАММИРОВАНИЕ И ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

вып. 1

Основы программирования. Алгоритмические языки

М., 1971 г., 432 стр. с илл.

Редактор *И. А. Румянцев*

Техн. редактор *Л. А. Пыжова* Корректор *И. Б. Мамулова*

Сдано в набор 11/III 1971 г. Подписано к печати 4/VIII 1971 г. Бумага 84×108$^1/_{32}$. Физ. печ. л. 13,5. Условн. печ. л. 22,68. Уч.-изд. л. 21,87. Тираж 71 000 экз. Т-12374. Цена книги 82 коп. Заказ № 1653.

Издательство «Наука»
Главная редакция физико-математической литературы.
Москва, В-71, Ленинский проспект, 15.

Ордена Трудового Красного Знамени Ленинградская типография № 1 «Печатный Двор» им. А. М. Горького Главполиграфпрома Комитета по печати при Совете Министров СССР, г. Ленинград, Гатчинская ул., 26.

2-2-4
23-71

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Книги «Программирование и вычислительная математика» — первые из серии согласованных между собою учебников для специальности «Прикладная математика».

Обучение этой специальности было начато в 1967 году в Московском математическом техникуме Министерства приборостроения, средств автоматизации и систем управления СССР и в настоящее время распространяется на другие средние специальные учебные заведения.

Содержание учебника соответствует программе дисциплины «Алгоритмические языки, программирование и вычислительная математика» учебного плана специальности «Прикладная математика» и разработано авторами в большей части для этих специальных учебных заведений впервые.

Предмет «Алгоритмические языки, программирование и вычислительная математика» занимает центральное место в цикле специальных предметов. Его изучение сопровождается практическими занятиями, на которых учащиеся знакомятся с различными устройствами электронных вычислительных машин и приемами решения задач с их помощью. Теоретическая и практическая подготовка по предмету служит основой для изучения остальных учебных предметов.

Изучение предмета опирается на знание смежных дисциплин, согласовано по содержанию и во времени с расширенной и углубленной подготовкой учащихся по таким общематематическим предметам, как математический анализ, алгебра, линейная алгебра и математическая логика.

С. И. Шварцбурд

ОТ АВТОРОВ

Широкое распространение и использование современных вычислительных машин потребовало для их эксплуатации большого числа специалистов как с высшим, так и со средним образованием. Для подготовки последних созданы средние специальные учебные заведения (математические техникумы).

Наша книга предназначена в качестве учебника по курсу «Программирование и вычислительная математика» для математических техникумов. Она написана по инициативе лаборатории прикладной математики Института содержания и методов обучения Академии педагогических наук СССР (зав. лабораторией член-корреспондент Академии Педагогических Наук СССР С. И. Шварцбурд) и полностью соответствует программе этого курса для первого и второго годов обучения. Кроме того, она может быть использована преподавателями и учащимися других техникумов при изучении программирования и вычислительной математики для всех тех специальностей, где такое изучение предусмотрено.

Ввиду большого объема курса учебник пришлось разбить на два выпуска:

вып. 1. Основы программирования. Алгоритмические языки,

вып. 2. Вычислительная математика. Программная реализация вычислительных методов.

Настоящий первый выпуск не требует от читателя никакой дополнительной подготовки, выходящей за пределы курса математики восьмилетней школы. Исключение составляют лишь несколько примеров и теорем, набранных мелким шрифтом, которые могут быть опущены при первоначальном изучении. Тем не менее, мы считаем их полезными для учащихся, закончивших 10 классов.

Первый выпуск состоит из четырех частей и приложения. Первая часть, «Приближенные вычисления», носит вводный характер. В ней излагаются правила действий над приближенными числами и основные приемы ручного счета, а также общие сведения о средствах вычислений (в том числе и математических машинах).

Вторая часть, «Основы программирования», посвящена основам программирования для вычислительных машин типа М-20 (к ним относятся машины М-20, БЭСМ-3, БЭСМ-4, М-220, М-222). Наш выбор был в основном обусловлен следующими двумя причинами. Во-первых, эти машины являются одними из наиболее распространенных отечественных машин. Во-вторых, «ручному» программированию легче всего учиться на трехадресной машине. Так как здесь рассматриваются основные команды машин типа М-20, то книга может быть использована как учебник программирования для этих машин. При этом мы пользуемся методом программирования в содержательных обозначениях. Полное описание системы команд машин типа М-20 и особенности каждой из них даны в приложении.

Третья часть книги — «Алгоритмические языки программирования». В ней рассматриваются машинно-ориентированный язык типа Автокод 1 : 1, являющийся, по существу, формализацией языка содержательных обозначений, и проблемно-ориентированный международный алгоритмический язык для вычислительных задач — алгол-60.

Изложение алгола является достаточно полным и подробным и сопровождается разбором большого числа примеров. Формально это изложение не зависит от всего предыдущего материала и можно изучать программирование, прямо начиная с алгола. Однако, по мнению авторов, порядок изучения программирования, используемый во второй и третьей частях нашей книги, является наиболее целесообразным. Во всяком случае, программирование на машинно-ориентированном языке типа Автокод 1 : 1 при всех обстоятельствах должно быть известно каждому специалисту-программисту, ибо создание математического обеспечения вычислительной машины обычно ведется на языках такого типа.

Наконец, последняя, четвертая часть книги — «Организация вычислительных систем» — посвящена описанию организации современных вычислительных систем. Этот материал до сих пор почти не излагался в учебной литературе, хотя он и является весьма актуальным для современного программирования. Здесь наибольшее внимание уделено вопросам структуры и логики работы технических средств вычислительной системы. На этой основе кратко излагаются важнейшие особенности общесистемного математического обеспечения (операционные системы).

Материал книги использовался в практике преподавания в Московском математическом техникуме, который является первым учебным заведением такого типа у нас в стране (организован в 1967 году).

Вместе с прямым своим назначением, как учебник для математических техникумов, книга может быть использована в качестве пособия по программированию и вычислительной математике для высших технических учебных заведений. Для этого достаточно лишь опустить материал, не входящий в соответствующие втузовские программы, что относится как к первому, так и ко второму выпуску. При этом не следует удивляться тому, что программа техникума оказывается заметно обширнее программы втуза: техникум *математический*. Не удивительно же, например, что техник связи знает об устройстве телефонного аппарата много больше и глубже, чем инженер-металлург.

Первая и вторая части книги, а также приложение написаны Р. С. Гутером и П. Т. Резниковским. Здесь широко использовалась наша предыдущая книга, предназначавшаяся для средних школ с математической специализацией (Р. С. Гутер, Б. В. Овчинский и П. Т. Резниковский, Программирование и вычислительная математика, «Наука», Главная редакция физико-математической литературы, 1965). В написании третьей части принимал участие также и С. М. Резник. Четвертая часть написана в основном П. Т. Резниковским.

Помещенные в конце книги синтаксические таблицы алгола заимствованы из книги: А. Л. Брудно, Алгол, изд. 2, 1971.

Для удобства пользования книгой мы ввели сплошную нумерацию глав и параграфов. Примеры, таблицы и фор-

мулы нумеруются в каждом параграфе заново; здесь используется двойная нумерация — п о с л е номера примера, таблицы или формулы указывается номер параграфа, в котором они находятся.

С. И. Шварцбурд и В. Л. Арлазаров внимательно ознакомились с рукописью и указали на ряд погрешностей, устранение которых существенно улучшило изложение материала. В большой степени этому же способствовала и работа редактора И. А. Румянцева. Большую помощь в работе над рукописью и подготовке ее к печати нам оказали Т. А. Муратова и З. П. Рант. С искренним удовольствием выражаем здесь всем им свою глубокую благодарность.

ЧАСТЬ ПЕРВАЯ

ПРИБЛИЖЕННЫЕ ВЫЧИСЛЕНИЯ

ГЛАВА I
ВЫЧИСЛЕНИЯ ПО ГОТОВОЙ ФОРМУЛЕ

§ 1. Приближенные числа. Способы записи чисел

Почти все числа, с которыми приходится иметь дело в процессе вычислений, являются *приближенными*. Действительно, исходные числовые данные обычно выражают значения тех или иных физических величин, полученные в результате измерений, которые, по самому существу своему, носят приближенный характер. Далее, действительные числа, хотя бы и рациональные, чаще всего не могут быть точно записаны в десятичной системе счисления. Их приходится записывать в виде конечной десятичной дроби, обрывая ее в том или ином месте. Наконец, арифметические действия в большинстве случаев тоже не могут быть выполнены точно.

В этих условиях серьезную роль играет вопрос о точности получаемых результатов и размерах возможных погрешностей.

Пусть A — некоторая величина, истинное значение которой известно или неизвестно. Число a, которое можно принять за значение величины A, мы будем называть *приближенным значением величины* A или просто *приближенным числом*. Число a называют *приближением по недостатку*, если оно меньше истинного значения, и *по избытку*, если оно больше. Так, 3,14 является приближенным значением числа π по недостатку, а 2,72 — приближением числа e по избытку. Чтобы охарактеризовать степень точности данного приближения, пользуются понятием *погрешности* или *ошибки*.

Абсолютная погрешность или *абсолютная ошибка* приближенного числа есть абсолютная величина разности между

истинным значением величины и данным ее приближенным значением. Чаще всего, когда речь идет об арифметических действиях с приближенными числами, употребляется термин *погрешность*, а когда говорят об измерениях, то употребляют термин *ошибка*. Нужно еще иметь в виду, что в обыденной речи этот термин употребляется совсем в ином смысле. Именно, в обыденной речи под о ш и б к о й понимается н е в е р н ы й результат. В вычислительной математике в этом смысле употребляется слово *просчет*.

Поскольку истинное значение величины обычно остается неизвестным, неизвестной остается также и абсолютная погрешность. Вместо нее приходится рассматривать *предельную абсолютную погрешность*, которая означает число, не меньшее абсолютной погрешности *).

Приближенные числа принято записывать таким образом, чтобы вид числа показывал его абсолютную погрешность. Исходные данные, получающиеся в результате измерений, а также числа в математических таблицах записывают так, чтобы абсолютная погрешность не превосходила половины единицы последнего разряда, сохраняемого при записи. Например, запись 3,1416 означает, что абсолютная погрешность этого приближенного числа не превосходит 0,00005. Для числа 2,150 абсолютная погрешность не превосходит 0,0005, а для числа 280 — величины 0,5. Если это число имеет бо́льшую точность, например, если абсолютная погрешность меньше 0,05, то следует писать не 280, а 280,0. Таким образом, при указанной форме записи числа все записанные цифры оказываются в е р н ы м и.

Если в результате действий получается большее, чем нужно, количество цифр, то число следует округлять, отбрасывая излишние цифры. При этом руководствуются известным правилом округления: если первая из отброшенных цифр 4 или меньше, то последняя цифра сохраняется без изменения; если первая из отброшенных цифр 5 или больше, то последняя оставшаяся цифра увеличивается на единицу.

---

*) Иногда, ради краткости речи, слово «предельная» опускают, понимая под абсолютной погрешностью именно предельную.

Исключением из этого правила является случай, когда отбрасывается только пятерка или же пятерка с нулями. Здесь можно выбрать любое правило округления, например, округлять всегда вверх или, наоборот, всегда вниз. Мы будем пользоваться правилом, предложенным К. Гауссом: последняя оставшаяся цифра сохраняется без изменения, если она четная, и увеличивается на единицу до четной, если она была нечетная. По этому правилу при округлении до целого числа 2,5 округляется до 2, а 3,5—до 4.

Точность данного приближенного числа не характеризуется его абсолютной погрешностью. Действительно, погрешность 0,5 *м* слишком велика при измерении длины комнаты, допустима при измерении участка, отводимого для постройки дома, и не может быть замечена при измерении расстояния между городами. Настоящим показателем точности результата измерения или вычисления является его относительная погрешность.

Относительной погрешностью приближенного значения величины называют абсолютную величину отношения его абсолютной погрешности к истинному значению этой величины. Часто эту относительную погрешность выражают в процентах.

Ввиду того, что фактически вместо абсолютной погрешности обычно рассматривается предельная, относительную погрешность также заменяют предельной относительной погрешностью, которая означает число, не меньшее относительной погрешности. Более того, при нахождении предельной относительной погрешности приходится заменять неизвестное истинное значение величины приближенным. Последняя замена обычно не отражается на величине относительной погрешности ввиду близости этих значений и малости абсолютной погрешности.

Например, для приближенного значения $\pi = 3{,}14$ предельная абсолютная погрешность составляет 0,0016, а относительная 0,00051, или 0,05%.

Для больших чисел, даже и при малых относительных погрешностях, абсолютные погрешности могут иметь порядок единиц, десятков, сотен и т. п. Например, для числа 234000 может не быть данных, указывающих на то, являются ли здесь нули действительно цифрами данного числа или, как принято говорить в таких случаях, з а м е щ а ю т

неизвестные цифры. Подчиняясь сформулированному ранее правилу, мы имеем право записать это число в виде 234000 только тогда, когда его абсолютная погрешность не превосходит 0,5. Если же абсолютная погрешность не превосходит пятидесяти, то последние два нуля замещают неизвестные цифры. Поэтому запись 234000 в этом случае является незаконной; число следует записывать в виде $2340 \cdot 10^2$ или $0{,}2340 \cdot 10^6$.

Абсолютная погрешность приближенного числа определяется числом десятичных знаков в его записи. Относительная погрешность определяется количеством *значащих цифр* в числе. При этом значащими цифрами являются все цифры числа, кроме тех нулей, которые употребляются для определения места других цифр в десятичной дроби или же замещают неизвестные цифры. Например, в числе 0,20003 значащими являются все цифры, кроме нуля целых. В числе 0,040 значащей является цифра 4, а также и нуль после нее, если число записано с соблюдением приведенного ранее правила и его абсолютная погрешность не превышает 0,0005. Два нуля впереди цифры 4 не являются значащими цифрами, так как они только определяют положение цифры 4. Можно не писать этих нулей, а записывать число в виде $40 \cdot 10^{-3}$, $4{,}0 \cdot 10^{-2}$ или же $0{,}40 \cdot 10^{-1}$.

Пользуясь термином «значащая цифра», мы можем упомянутое выше правило о записи приближенных чисел сформулировать таким образом: *приближенные числа следует записывать так, чтобы все цифры числа, кроме нулей впереди, если они есть, были значащими и верными цифрами.*

О числах $0{,}40 \cdot 10^{-1}$ и $0{,}2340 \cdot 10^6$ говорят, что они записаны в *нормальной* или *плавающей форме* (*с плавающей запятой*), в противоположность обычной записи, которую называют *естественной* или *фиксированной* (*с фиксированной запятой*). Как видно из приведенных примеров, запись числа в плавающей форме не определяется однозначно. Чтобы устранить неоднозначность, принято первый множитель брать меньшим единицы и состоящим только из значащих цифр (кроме нуля целых), так что его первая цифра после запятой отлична от нуля. В этом случае число называют *нормализованным*.

Таким образом, запись числа N в плавающей форме означает его представление в виде

$$N = N_0 \cdot 10^p;$$

обычно $|N_0| < 1$. Число N_0 называют *мантиссой* числа N, а число p — его *порядком*. Если число N нормализовано, то первая цифра мантиссы отлична от нуля, т. е.

$$1 > |N_0| \geqslant 0{,}1$$

и порядок p определен однозначно. Легко сообразить, что порядок числа, большего единицы, равен количеству целых разрядов числа до запятой, а порядок числа, меньшего единицы, отрицателен (либо равен нулю), и его абсолютная величина показывает количество нулей после запятой перед первой значащей цифрой числа.

При записи отрицательного числа в плавающей форме отрицательной считают его мантиссу.

Обычно в процессе вычислений стремятся иметь все числа с одинаковой точностью. Записывая все числа в фиксированной форме с одним и тем же количеством десятичных знаков, мы можем для всех чисел обеспечить одинаковую абсолютную погрешность. Запись всех чисел в нормализованной плавающей форме с одинаковым числом знаков в мантиссе позволяет обеспечить для всех одинаковую относительную погрешность. В зависимости от потребностей вычислений и употребляется та или иная форма записи.

Фиксированная форма более удобна для сложения и вычитания, чем плавающая, но переводить числа для сложения или вычитания в фиксированную форму не обязательно. Если слагаемые имеют одинаковый порядок, то сложение, как и вычитание, производится по обычным правилам и порядок суммы или разности остается равным порядку слагаемых. При этом надо только иметь в виду, что результат может не оказаться нормализованным и для его нормализации придется сдвинуть мантиссу в ту или иную сторону, изменив соответствующим образом порядок.

Легко понять, как это делается, рассмотрев следующие примеры:

$$0{,}347541 \cdot 10^2 + 0{,}532612 \cdot 10^2 = 0{,}880153 \cdot 10^2,$$
$$0{,}791564 \cdot 10^{-1} - 0{,}272316 \cdot 10^{-1} = 0{,}519248 \cdot 10^{-1},$$
$$0{,}631825 \cdot 10^{-1} + 0{,}723007 \cdot 10^{-1} = 1{,}354832 \cdot 10^{-1} =$$
$$= 0{,}135483\,(\cdot 10^0),$$
$$0{,}568188 \cdot 10^3 - 0{,}489313 \cdot 10^3 = 0{,}078875 \cdot 10^3 =$$
$$= 0{,}78875 \cdot 10^2,$$
$$0{,}713954 \cdot 10^4 - 0{,}711259 \cdot 10^4 = 0{,}002695 \cdot 10^4 =$$
$$= 0{,}2695 \cdot 10^2.$$

В первых двух случаях результат получился нормализованным. В трех следующих потребовалась последующая нормализация с изменением порядка и округлением.

Если слагаемые имеют различные порядки, то, прежде чем производить сложение или вычитание, необходимо выравнять порядки, сдвинув соответствующим образом мантиссу. Порядок выравнивается по большему (по абсолютной величине) числу, так что у меньшего числа, которое требуется «разнормализовать», мантисса сдвигается вправо. Обычно ее при этом округляют, как это видно из приведенных ниже примеров:

$$0{,}564137 \cdot 10^4 + 0{,}253912 \cdot 10^3 = 0{,}564137 \cdot 10^4 +$$
$$+ 0{,}025391 \cdot 10^4 = 0{,}589528 \cdot 10^4,$$
$$0{,}876839 \cdot 10^{-4} + 0{,}272731 \cdot 10^{-1} = 0{,}000877 \cdot 10^{-1} +$$
$$+ 0{,}272731 \cdot 10^{-1} = 0{,}273608 \cdot 10^{-1},$$
$$0{,}643995 \cdot 10^{-1} - 0{,}987414 \cdot 10^{-3} = 0{,}643995 \cdot 10^{-1} -$$
$$- 0{,}009874 \cdot 10^{-1} = 0{,}634121 \cdot 10^{-1}.$$

При умножении или делении чисел в плавающей форме порядки могут быть какими угодно. При умножении мантиссы сомножителей перемножаются по обычным правилам, а порядки складываются. Точно так же при делении мантиссы делятся, а порядки вычитаются. Не надо только забывать, что в результате действий могут получиться ненормализованные числа и может понадобиться сдвиг мантиссы для нормализации результата.

Примеры:

$$0{,}235642 \cdot 10^{2} \times 0{,}853165 \cdot 10^{-1} = 0{,}201042 \cdot 10^{1},$$

$$0{,}312317 \cdot 10^{-1} \times 0{,}292877 \cdot 10^{-2} = 0{,}0914705 \cdot 10^{-3} = 0{,}914705 \cdot 10^{-4},$$

$$0{,}156341 \cdot 10^{3} : 0{,}592658 \cdot 10^{2} = 0{,}263796 \cdot 10^{1},$$

$$0{,}828145 \cdot 10^{-2} : 0{,}327140 \cdot 10^{-1} = 2{,}53147 \cdot 10^{-1} = 0{,}253147\,(\cdot 10^{0}).$$

§ 2. Погрешности арифметических действий

В предыдущем параграфе шла речь о погрешностях в исходных данных. Кроме них, на погрешность результата влияют также и погрешности выполняемых нами арифметических действий. Поэтому в процессе вычислений сразу возникает вопрос о том, с какой точностью, т. е. с каким числом знаков, должны вестись вычисления. Часто такой вопрос решается однозначно: если исходные данные получены из опыта, нет необходимости вести промежуточные вычисления с большей точностью, чем точность исходных данных. Но в таких случаях нужно оценить, с какой точностью мы будем знать окончательный результат.

Настоящий параграф посвящен выяснению вопроса о погрешностях арифметических действий над приближенными числами.

Пусть a и b — приближенные значения величин A и B соответственно, с предельными абсолютными погрешностями α_1 и α_2, причем точные значения величин A и B могут быть известны или неизвестны. Тогда справедливы неравенства

$$a - \alpha_1 < A < a + \alpha_1,$$
$$b - \alpha_2 < B < b + \alpha_2.$$

Складывая оба эти неравенства, находим

$$a + b - (\alpha_1 + \alpha_2) < A + B < a + b + (\alpha_1 + \alpha_2), \quad (1.2)$$

так что приближенным значением суммы $A + B$ можно считать $a + b$, причем за предельную абсолютную погрешность можно принять $\alpha_1 + \alpha_2$. Точно так же, вычитая из неравенства $A > a - \alpha_1$ неравенство $B < b + \alpha_2$, а из

неравенства $A < a + \alpha_1$ неравенство $B > b - \alpha_2$, находим

$$a - b - (\alpha_1 + \alpha_2) < A - B < a - b + (\alpha_1 + \alpha_2). \quad (2.2)$$

Утверждения, выражаемые неравенствами (1.2) и (2.2), можно объединить одной формулировкой: *предельная абсолютная погрешность суммы или разности приближенных чисел равна сумме абсолютных погрешностей слагаемых.* Иногда говорят короче: *при сложении или вычитании абсолютные погрешности складываются.* Это утверждение без труда переносится на случай любого конечного числа слагаемых.

Если не обращать внимания на знаки слагаемых, то вместо сложения и вычитания можно, как известно, говорить только о сложении. Однако это возможно только при рассмотрении абсолютных погрешностей. При установлении предельной относительной погрешности суммы необходимо различать случаи одинаковых или различных знаков слагаемых.

Считая все слагаемые положительными, для предельной относительной погрешности суммы получаем

$$\delta = \frac{\alpha_a + \alpha_b + \ldots + \alpha_c}{a + b + \ldots + c}.$$

Обозначим через δ_a, δ_b, ..., δ_c предельные относительные погрешности слагаемых и через $\delta_{\min}$, $\delta_{\max}$ соответственно наименьшее и наибольшее из этих чисел. Так как $\delta_a = \alpha_a/a$, то $\alpha_a = \delta_a \cdot a$. Такое же равенство справедливо и для всех остальных слагаемых. Поэтому

$$\delta = \frac{\alpha_a + \alpha_b + \ldots + \alpha_c}{a + b + \ldots + c} = \frac{\delta_a \cdot a + \delta_b \cdot b + \ldots + \delta_c \cdot c}{a + b + \ldots + c} < \frac{\delta_{\max}(a + b + \ldots + c)}{a + b + \ldots + c} = \delta_{\max}$$

и, аналогично, $\delta > \delta_{\min}$. Таким образом,

$$\delta_{\min} < \delta < \delta_{\max}, \quad (3.2)$$

т. е. *относительная погрешность суммы слагаемых одного знака заключена между наименьшей и наибольшей относительными погрешностями слагаемых.*

Для слагаемых разных знаков дело обстоит иначе. Пусть $a > 0$, $b > 0$ и $c = a - b$. Тогда, сохраняя прежние обозначения, имеем

$$\delta = \frac{\alpha_a + \alpha_b}{|a - b|},$$

поэтому, если a и b близки между собой, то даже при очень малых погрешностях уменьшаемого и вычитаемого предельная относительная погрешность разности может оказаться весьма значительной, т. е. при вычитании близких величин происходит большая потеря точности.

Пример 1.2. Пусть $a = 1{,}363$, $b = 1{,}353$. Найдем относительную погрешность суммы и разности этих чисел.

Считая соблюденными правила записи приближенных чисел, получаем $\alpha_a = \alpha_b = 0{,}0005$, откуда $\delta_a \approx \delta_b = 0{,}04\%$. Поэтому для суммы находим $\alpha_{a+b} = 0{,}001$ и $\delta_{a+b} \approx 0{,}04\%$. Для разности же получаем $\delta_{a-b} = \dfrac{0{,}0005 + 0{,}0005}{0{,}01} \cdot 100\% = 10\%$, так что относительная погрешность разности превосходит относительную погрешность суммы в 250 раз.

Рассмотрим теперь умножение и деление приближенных чисел. Приближенное значение величины A можно представить в виде $a + \alpha_a = a(1 + \alpha_a/a) = a(1 + \delta_a)$. Поэтому для произведения двух множителей получаем

$$ab(1 + \delta_a)(1 + \delta_b) = ab(1 + \delta_a + \delta_b + \delta_a \delta_b).$$

Пренебрегая произведением малых относительных погрешностей по сравнению с их суммой, мы можем записать последнее выражение в виде

$$ab(1 + \delta_a + \delta_b),$$

откуда выводим, что *предельная относительная погрешность произведения равна сумме предельных относительных погрешностей сомножителей*,

$$\delta_{ab} = \delta_a + \delta_b. \tag{4.2}$$

Можно доказать, что относительная погрешность приближенных чисел a и $1/a$ одинакова. Отсюда следует, что формула (4.2) справедлива не только для умножения, но и для деления, т. е. что *при умножении или делении приближенных чисел их предельные относительные погрешности складываются.*

Так как абсолютная и относительная погрешности точного числа равны нулю, то из сказанного выше вытекает, что при умножении или делении на точное число относительная погрешность приближенного числа не меняется. В частности, отсюда следует, что относительная погрешность не зависит от положения запятой в числе, что и объясняет использование представления числа в плавающей форме, рассмотренное в предыдущем параграфе.

Пример 2.2. Сила взаимодействия между двумя зарядами e_1 и e_2, находящимися на расстоянии r друг от друга, выражается законом Кулона

$$F=\frac{e_1e_2}{\varepsilon r^2},$$

где ε — диэлектрическая постоянная среды. Зная, что величина ε определена с относительной погрешностью 5%, величины зарядов e_1 и e_2 — с относительной погрешностью 0,5%, а расстояние между ними r — с относительной погрешностью 1%, определим относительную ошибку величины F.

Согласно выражению для предельной относительной погрешности частного имеем

$$\delta_F=\delta_{e_1}+\delta_{e_2}+\delta_\varepsilon+2\delta_r,$$

откуда $\delta_F = 8\%$.

Рассмотренные свойства погрешностей арифметических операций являются частными случаями следующих двух общих теорем:

Теорема 1. *Предельная абсолютная погрешность функции $y = f(x)$ равна произведению абсолютной величины ее производной на предельную абсолютную погрешность аргумента.*

Доказательство. Пусть число x является приближенным значением величины X с абсолютной погрешностью $|\Delta x|$,

$$X=x+\Delta x.$$

Обозначим абсолютную погрешность функции через $|\Delta y|$.

Тогда

$$|\Delta y|=|f(X)-f(x)|=|f(x+\Delta x)-f(x)|.$$

Ввиду малости $|\Delta x|$ мы можем заменить приращение функции ее дифференциалом. Тогда получим

$$f(x+\Delta x)\approx f(x)+f'(x)\cdot\Delta x,$$

откуда

$$|\Delta y|\approx|f'(x)|\cdot|\Delta x|.$$

Обозначив предельную погрешность аргумента через α, т. е. считая, что $|\Delta x| \leqslant \alpha$, найдем

$$|\Delta y| \leqslant \alpha |f'(x)|.$$

Таким образом, предельную абсолютную погрешность β функции $f(x)$ можно принять равной

$$\beta = \alpha \cdot |f'(x)|. \tag{5.2}$$

Теорема доказана.

Доказанную теорему можно использовать и для получения относительной погрешности функции. Обозначим предельную относительную погрешность аргумента через δ_x и функции через δ_y. Тогда, учитывая, что $\delta_x = \alpha/|x|$, т. е. $\alpha = |x| \cdot \delta_x$, получим для δ_y выражение

$$\delta_y = \frac{\beta}{|f(x)|} = \frac{\alpha |f'(x)|}{|f(x)|} = \left| x \cdot \frac{f'(x)}{f(x)} \right| \delta_x.$$

Вспомнив выражение для логарифмической производной

$$\frac{d \ln f(x)}{dx} = \frac{f'(x)}{f(x)},$$

перепишем предыдущее равенство в виде

$$\delta_y = \left| x \frac{d \ln f(x)}{dx} \right| \delta_x. \tag{6.2}$$

Формулой (6.2) можно воспользоваться для нахождения относительных погрешностей ряда конкретных функций. Пусть, например, $f(x) = x^n$. Тогда

$$\delta_y = \left| x \cdot \frac{n x^{n-1}}{x^n} \right| \cdot \delta_x = |n| \cdot \delta_x, \tag{7.2}$$

т. е. *предельная относительная погрешность степени равна предельной относительной погрешности основания, умноженной на абсолютную величину показателя степени.*

Это утверждение для целого показателя степени n можно получить и без логарифмической производной, как следствие равенства (4.2). Фактически мы уже пользовались этим свойством в примере 2.2.

Положив в (7.2) $n = -1$, для функции $y = 1/x$ найдем, что $\delta_y = \delta_x$. Последнее равенство использовалось нами при выводе относительной погрешности частного.

Найдем еще предельную абсолютную погрешность β для логарифмической функции. Положив $f(x) = \ln x$ и учитывая (5.2), находим

$$\beta = \alpha \left| \frac{1}{x} \right| = \left| \frac{\alpha}{x} \right| = \delta_x,$$

так что *предельная абсолютная погрешность натурального логарифма равна предельной относительной погрешности аргумента.*

Теорема 2. *Абсолютная погрешность функции нескольких переменных равна сумме произведений абсолютных погрешностей аргументов на абсолютные величины соответствующих частных производных.*

Доказательство. Пусть x и y являются соответственно приближенными значениями величин X и Y с абсолютными погрешностями $|\Delta x|$, $|\Delta y|$. Следовательно,

$$X = x + \Delta x, \quad Y = y + \Delta y.$$

Тогда

$$|\Delta u| = |f(X, Y) - f(x, y)|$$

или

$$|\Delta u| = |f(x + \Delta x, y + \Delta y) - f(x, y)|.$$

Считая величины Δx и Δy малыми, заменим приращение полным дифференциалом du. Это приводит к приближенному равенству

$$\Delta u \approx \frac{\partial u}{\partial x}\Delta x + \frac{\partial u}{\partial y}\Delta y,$$

откуда

$$|\Delta u| \leqslant \left|\frac{\partial u}{\partial x}\right| \cdot |\Delta x| + \left|\frac{\partial u}{\partial y}\right| \cdot |\Delta y|.$$

Если обозначить предельные абсолютные погрешности аргументов x и y соответственно через α_x и α_y:

$$|\Delta x| \leqslant \alpha_x, \quad |\Delta y| \leqslant \alpha_y,$$

то

$$|\Delta u| \leqslant \left|\frac{\partial u}{\partial x}\right| \cdot \alpha_x + \left|\frac{\partial u}{\partial y}\right| \cdot \alpha_y,$$

и за предельную абсолютную погрешность β функции $f(x, y)$ можно принять величину

$$\beta = \left|\frac{\partial u}{\partial x}\right| \cdot \alpha_x + \left|\frac{\partial u}{\partial y}\right| \cdot \alpha_y. \tag{8.2}$$

Ясно, что аналогичное равенство имеет место и для дифференцируемой функции любого большего числа аргументов.

В качестве частных случаев из этой теоремы можно получить все рассматривавшиеся выше соотношения для погрешностей арифметических действий.

Заметим еще, что при вычислении значений функции абсолютная погрешность может существенно зависеть от того, каким образом записана расчетная формула и каково расположение операций в этой формуле. Для пояснения рассмотрим следующий пример.

Пример 3.2. Вычислим площадь кругового кольца с внутренним радиусом $r = 1{,}750$ и толщиной $h = 0{,}005$.

Здесь вычисления можно производить по формулам

$$S = \pi[(r + h)^2 - r^2]$$

и

$$S = \pi(2r + h)h.$$

Хотя алгебраически эти формулы тождественны, но для вычислений вторая во много раз лучше первой, так как при вычитании близких величин в первой формуле относительная погрешность сильно возрастает.

Действительно, подсчитаем предельную абсолютную погрешность величины S/π в том и другом случаях. В соответствии с правилом записи приближенных чисел следует считать, что предельная абсолютная погрешность величин r и h составляет 0,0005; при этом предельная абсолютная погрешность их суммы $r + h$ равна 0,001. Тогда предельные относительные погрешности величин r и $r + h$ равны соответственно 0,03 и 0,06%, а их квадратов — 0,06 и 0,12% (по второму правилу). Следовательно, предельная абсолютная погрешность величины $(r + h)^2$ равна 0,0037, а r^2 равна 0,0018. По первому правилу абсолютная погрешность их разности составляет $0{,}0037 + 0{,}0018 = 0{,}0055$, т. е. при расчете по первой формуле можно гарантировать лишь, что абсолютная погрешность величины S/π не превзойдет 0,0055.

Совсем иначе обстоит дело в случае второй формулы. Предельная абсолютная погрешность множителя $2r + h$ равна 0,0015, предельная относительная погрешность — 0,04%, а предельная относительная погрешность h — 10%. Поэтому предельная относительная погрешность отношения S/π составляет 10,04%, так что предельная абсолютная погрешность этого отношения при расчете по второй формуле равна 0,0018, что в три раза меньше, чем в предыдущем случае.

Пример 3.2 показывает, насколько важно правильно выбрать вид расчетной формулы. Поэтому к указанным выше правилам подсчета погрешностей следует присоединить важный практический совет:

Формулы для вычислений надо стараться приводить к такому виду, чтобы в них не было вычитания близких величин; последнее может привести к большой потере точности и большим относительным ошибкам.

Обращаем внимание читателя на то, что во всем предыдущем изложении был рассмотрен только один вопрос: *как, зная погрешности исходных данных, оценить погрешность результата расчета.* При этом мы совсем не касались вопроса о погрешностях округления. Точная оценка влияния погрешностей округления на окончательный результат пока еще не может быть сделана. Поэтому мы ограничимся следующим практическим советом: *для уменьшения погрешностей округления промежуточные действия рекомендуется производить, сохраняя один-два лишних знака, после чего окончательный результат следует округлять.*

В связи с ошибками округления мы можем снова сравнить две расчетные формулы, приведенные в примере 3.2, с другой точки зрения. Можно считать, что значения r и h являются точными, и интересоваться вопросом о том, с каким числом знаков нужно вести вычисления для достижения нужной точности. И с этой точки зрения вторая формула, не содержащая разностей близких величин, оказывается гораздо лучше, нежели первая.

§ 3. Расписка формулы

Основным, если не единственным, видом ручных расчетных работ является вычисление по готовой формуле. Вычисления по одной и той же формуле могут производиться в сравнительно большом числе точек либо в одной-двух

точках, если речь идет о контрольном расчете при проверке программы, составленной для электронной вычислительной машины.

Прежде чем приступать к непосредственным вычислениям, нужно продумать схему их выполнения и составить расчетную таблицу, в которой и должны выполняться все вычисления. Основное правило, о котором никогда нельзя забывать, состоит в следующем:

вычислять и записывать вычисления следует так, чтобы любой результат можно было всегда проверить.

Форма расчетной таблицы определяется видом заданной функции. В каждом ее столбце должны выполняться одни и те же действия. Столбцы нужно стараться располагать в таком порядке, чтобы результаты, полученные в одном из них, использовались при вычислении значений следующего. В заголовке каждого столбца, соответствующего промежуточным результатам, указывается, каким образом числа из этого столбца получаются через предыдущие. Если какой-либо промежуточный результат имеет свое обозначение, то это обозначение тоже нужно указать в заголовке.

Кроме наименования результатов, находящихся в данном столбце, в заголовке столбца следует стараться избегать употребления буквенных обозначений. Если речь идет о величинах, фигурировавших в предыдущих столбцах данной таблицы, то необходимо указывать не букву, а номер столбца. Если же речь идет о постоянной величине, ранее в таблице не встречавшейся и являющейся, например, слагаемым или множителем, то лучше в заголовке столбца помещать соответствующее числовое значение.

Форма расчетной таблицы зависит и от того, какими вычислительными средствами мы располагаем. Например, если пользоваться автоматической клавишной машиной, то некоторые промежуточные результаты можно не записывать и объединять несколько столбцов в один. Иногда бывает удобно воспользоваться какими-либо имеющимися таблицами, например, выписывать значения квадратов или квадратных корней из таблицы, вместо того чтобы производить соответствующие вычисления на машине.

Составленную расчетную таблицу обычно называют *распиской формулы*. При вычислениях по расписке рекомен-

дуется производить вычисления не по строке, а заполнять последовательно столбец за столбцом. Это, во-первых, ускоряет вычисления, так как при заполнении каждого столбца требуется выполнение однотипных операций, во-вторых, это облегчает контроль вычислений: благодаря монотонности изменения результатов, которая обычно имеет место, отдельные просчеты могут быть легче обнаружены.

Пример 1.3. Составим расписку для вычисления величины по формуле

$$y=\frac{4x}{3-x^2}-\sqrt{2x}.$$

Требуемая расписка приведена в табл. 1.3.

Так как в заголовках столбцов наряду с номерами предыдущих столбцов встречаются и числа, то они стоят в кавычках. Иногда их записывают также с индексом N (число).

Таблица 1.3

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
x	«2» · (1)	$\sqrt{(2)}$	(1)²	«3» — (4)	«2» · (2)	(6) : (5)	(7) — (3)

Пример 2.3. Амплитуда вынужденных колебаний груза, подвешенного на пружине, под воздействием периодической силы и с учетом сопротивления среды выражается формулой

$$a=\frac{h}{\sqrt{(\omega_0^2-\omega^2)^2+4p^2\omega^2}},$$

где ω — частота внешней периодической силы. Остальные параметры выражаются равенствами

$$\omega_0=\sqrt{\frac{k}{m}},\quad p=\frac{b}{2m},\quad h=\frac{F_0}{m},$$

где m — масса груза, k — коэффициент жесткости пружины, b — коэффициент сопротивления среды, F_0 — амплитуда

внешней силы. В свою очередь, частота внешней периодической силы может быть выражена через ее период T по формуле $\omega = 2\pi/T$.

Пусть масса груза $m = 200$ *г* $= 0{,}200$ *кг*, коэффициент жесткости $k = 0{,}423$ *кг/сек*^{2}, коэффициент сопротивления $b = 0{,}256$ *кг/сек* и амплитуда внешней силы $F_0 = 0{,}500$ *н*. Рассчитаем амплитуду вынужденных колебаний в зависимости от периода внешней силы T для значений T от 0,5 *сек* до 1 *сек* с шагом 0,1 *сек*.

Все вычисления приведены в табл. 2.3. Промежуточные вычисления ведутся с четырьмя значащими цифрами, окончательный результат должен быть округлен.

Для составления расписок удобно пользоваться бланками, напечатанными типографским путем. В зависимости от числа знаков, с которыми нужно вести вычисления, в бланках делают столбцы различной ширины.

Расположение расписки, приведенное в предыдущих примерах, удобно для самостоятельных ручных расчетов. Для расчетов, являющихся контрольными при отладке программ на машине, вычисления производятся лишь в одной-двух точках. Поэтому расписку удобнее располагать не по столбцам, а по строкам, для чего печатаются бланки другой формы.

Пример 3.3. Два вектора, $\boldsymbol{p}_1$ и $\boldsymbol{p}_2$, заданы своими модулями и направляющими косинусами относительно некоторой системы координатных осей. Требуется определить модуль вектора $\boldsymbol{p} = \boldsymbol{p}_1 + \boldsymbol{p}_2$, а также величины

$$E_i = \sqrt{|p_i|^2 + \mu^2}, \quad i = 1,2, \quad E = E_1 + E_2, \quad \varkappa = \sqrt{E^2 - |p|^2},$$

где μ — заданная постоянная величина.

Если все случаи рассчитываются вручную, то естественно пользоваться обычной распиской и обычной формой. Если случаи будут обрабатываться на электронной вычислительной машине, то необходимо провести контрольный ручной расчет для одного варианта. В этом случае удобнее пользоваться формой с другим расположением вычислений. Вычисления приведены в табл. 3.3. Первые строки таблицы содержат исходные числовые данные. Вычисления ведутся, как обычно для ручных расчетов при отладке программ, с восемью значащими цифрами, хотя это, казалось бы, противоречит сказанному в предыдущих параграфах (некоторые исходные данные имеют лишь две или три значащих цифры). Дело в том, что этот расчет служит для проверки программы, а машина воспринимает все введенные в нее знаки, в том числе — нули, как точные. Разумеется, окончательный результат при его использовании должен быть округлен в соответствии с правилами предыдущих параграфов.

Таблица 2.3

(1)	(2)	(3)	(4)	(5)	(6)	(7)
$\frac{k}{m}$	$\omega_0 = \sqrt{(1)}$	$p = \frac{b}{2m}$	«4» · (3)²	$h = \frac{F_0}{m}$	T	ω «6,28319» : (6)
2,115	1,454	0,64	1,6384	2,5	0,5	12,566
					0,6	10,472
					0,7	8,976
					0,8	7,854
					0,9	6,981
					1,0	6,283

(8)	(9)	(10)	(11)
(7)²	(1) — (8)	(9)²	(4) · (8)
$0{,}1579 \cdot 10^3$	$-0{,}1558 \cdot 10^3$	$0{,}2427 \cdot 10^5$	$0{,}2587 \cdot 10^3$
$0{,}1097 \cdot 10^3$	$-0{,}1076 \cdot 10^3$	$0{,}1158 \cdot 10^5$	$0{,}1797 \cdot 10^3$
$0{,}8057 \cdot 10^2$	$-0{,}7846 \cdot 10^2$	$0{,}6156 \cdot 10^4$	$0{,}1320 \cdot 10^3$
$0{,}6169 \cdot 10^2$	$-0{,}5958 \cdot 10^2$	$0{,}3550 \cdot 10^4$	$0{,}1011 \cdot 10^3$
$0{,}4873 \cdot 10^2$	$-0{,}4662 \cdot 10^2$	$0{,}2173 \cdot 10^4$	$0{,}7984 \cdot 10^2$
$0{,}3948 \cdot 10^2$	$-0{,}3737 \cdot 10^2$	$0{,}1397 \cdot 10^4$	$0{,}6468 \cdot 10^2$

(12)	(13)	(14)
(10) + (11)	$\sqrt{(12)}$	$a = \frac{(5)}{(13)}$
$0{,}2453 \cdot 10^5$	$0{,}1566 \cdot 10^3$	$0{,}160 \cdot 10^{-1}$
$0{,}1176 \cdot 10^5$	$0{,}1084 \cdot 10^3$	$0{,}231 \cdot 10^{-1}$
$0{,}6288 \cdot 10^4$	$0{,}7930 \cdot 10^2$	$0{,}315 \cdot 10^{-1}$
$0{,}3651 \cdot 10^4$	$0{,}6042 \cdot 10^2$	$0{,}414 \cdot 10^{-1}$
$0{,}2253 \cdot 10^4$	$0{,}4747 \cdot 10^2$	$0{,}527 \cdot 10^{-1}$
$0{,}1462 \cdot 10^4$	$0{,}3824 \cdot 10^2$	$0{,}654 \cdot 10^{-1}$

Таблица 3.3

p_1	(1)		0,78
$\cos\alpha_1$	(2)		0,32929
$\cos\beta_1$	(3)		0,54882
$\cos\gamma_1$	(4)		0,76835
p_2	(5)		3,12
$\cos\alpha_2$	(6)		0,42426
$\cos\beta_2$	(7)		0,70711
$\cos\gamma_2$	(8)		0,56569
x_1	(9)	(1) · (2)	0,2568462
y_1	(10)	(1) · (3)	0,4280796
z_1	(11)	(1) · (4)	0,5993130
x_2	(12)	(5) · (6)	1,3236912
y_2	(13)	(5) · (7)	2,2061832
z_2	(14)	(5) · (8)	1,7649528
x_p	(15)	(9) + (12)	1,5805374
y_p	(16)	(10) + (13)	2,6342628

Продолжение табл. 3.3

z_p	(17)	(11) + (14)	2,3642658
	(18)	$(15)^3$	2,4980985
	(19)	$(16)^2$	6,9393405
	(20)	$(17)^2$	5,5897528
$x_p^2 + y_p^2 + z_p^2 = p^2$	(21)	(18) + (19) + (20)	15,027192
p	(22)	$\sqrt{(21)}$	3,8764922
	(23)	$(1)^2$	0,6084
	(24)	(23) + «0,19485368 · 10^{-1}»	0,62788537
$E_1 = \sqrt{p_1^2 + \mu^2}$	(25)	$\sqrt{(24)}$	0,79239218
	(26)	$(5)^2$	9,7344
	(27)	(26) + «0,19485368 · 10^{-1}»	9,7538854
$E_2 = \sqrt{p_2^2 + \mu^2}$	(28)	$\sqrt{(27)}$	3,1231211
$E = E_1 + E_2$	(29)	(25) + (28)	3,9155133
	(30)	$(29)^2$	15,331244
	(31)	(30) — (21)	0,304052
$\varkappa = \sqrt{E^2 - \lvert \boldsymbol{p} \rvert^2}$	(32)	$\sqrt{(31)}$	0,55140910

§ 4. Контроль вычислений

Все результаты вычислений должны контролироваться, так как в процессе вычислений могут возникать просчеты. Причиной неверного результата могут служить неисправности в работе вычислительной машины. Однако и при правильно работающей машине возможны просчеты. Вычислитель может, особенно когда он начинает утомляться и его внимание рассеивается, записать или прочесть не ту цифру, взять число не из нужного места или записать его не в нужную колонку или строку, либо нажать не ту клавишу, которую надо, и не заметить этого. Неверное число войдет затем в другие вычисления, и в конечном счете многие из полученных результатов могут оказаться неверными.

Различают контроль *заключительный* и *текущий*. Заключительный контроль предполагает проверку окончательных результатов. Например, если целью вычислений было отыскание корня какого-либо уравнения, то в качестве заключительного контроля необходима подстановка полученного значения корня в уравнение и проверка того, что уравнение удовлетворяется с нужной степенью точности. Такая проверка корней о б я з а т е л ь н а в с е г д а, хотя бы уравнение было просто квадратным.

При построении таблиц функций хорошим заключительным контролем является составление табличных разностей. При отсутствии у функции особенностей, которые можно обнаружить, исследовав ее аналитическое выражение, разности между последовательными значениями функции должны изменяться плавно и (на небольших участках) быть практически монотонными. Нарушение правильности хода разностей может быть вызвано либо аналитическими особенностями функции, либо просчетами в вычислении ее значений.

Более детальные исследования разностей и, в особенности, привлечение разностей более высоких порядков позволяют довольно точно обнаружить неверный результат и даже исправить его. Этот вопрос будет рассмотрен во втором выпуске книги.

Аналогичную роль играет построение графика функции по вычисленным значениям. При отсутствии аналитических

особенностей построенные по результатам вычислений точки должны ложиться на плавную гладкую кривую.

К сожалению, заключительный контроль возможен не для всех типов вычислений. Кроме того, даже в тех случаях, когда он возможен, им нельзя ограничиваться: при сложных и громоздких расчетах слишком много работы будет потрачено зря, если пропустить просчет, допущенный на ранней стадии вычислений. Поэтому безусловно необходим также текущий контроль.

Наиболее совершенной формой текущего контроля является использование *контрольных соотношений*. Некоторые величины, получающиеся в результате независимых вычислений, могут быть связаны между собою какими-либо соотношениями, и для контроля необходимо проверить выполнение этих соотношений.

Например, при решении треугольников по двум сторонам a и b и углу C нужно прежде всего найти третью сторону c по теореме косинусов

$$c^2 = a^2 + b^2 - 2ab\cos C.$$

После этого нужно находить углы A и B по теореме синусов

$$\sin A = \frac{a\sin C}{c}, \qquad \sin B = \frac{b\sin C}{c}.$$

Контрольным соотношением будет равенство

$$A + B + C = \pi.$$

Нарушение этого равенства показывает наличие просчета в вычислениях. Некоторая излишняя вычислительная работа, которую нужно провести, окупается уверенностью в правильности полученных результатов.

Аналогичные соотношения используются при решении систем линейных уравнений. Эти вопросы будут подробно разобраны во втором выпуске.

Контрольные соотношения можно найти далеко не для всех вычислений. Иногда такие соотношения удается построить искусственно. Во всех тех случаях, когда можно воспользоваться контрольными соотношениями, это непременно следует сделать.

Самым распространенным способом контроля вычислений, хотя и не дающим полной гарантии их безошибочности, являются *вычисления в две руки*. Практически все вычисления должны вестись параллельно по одинаковой расписке

двумя вычислителями. Результаты вычислений должны сверяться, и нужно, чтобы эти результаты совпадали. Если вычисления производились на клавишных машинах одинакового типа и велись с одинаковым числом значащих цифр, то допустимым можно считать расхождение на одну-две единицы последнего (запасного) знака. При бо́льших расхождениях следует сверить все промежуточные результаты и обнаружить место и причину полученного расхождения.

Вычисления в две руки позволяют с очень большой надежностью исключить влияние просмотра, описки или очитки вычислителя и неправильностей в работе машины, так как вероятность одинаковых просчетов ничтожно мала. Тем не менее и при вычислениях в две руки можно получить совпадающие ошибочные результаты, например, в том случае, когда оба вычислителя пользовались одной и той же таблицей, в которой имеется опечатка.

При больших и сложных или громоздких расчетах необходимо сверять не только окончательные, но и промежуточные результаты, т. е. вести не только заключительный, но и текущий контроль. Но сверять каждое число или даже каждую колонку сразу после ее получения нельзя, так как при этом резко возрастает вероятность одинаковых просчетов. Сверку результатов лучше всего производить через каждые восемь — десять столбцов расписки.

Подчеркнем, что вычисления в две руки никак не отменяют и не заменяют других способов текущего и заключительного контроля: использования контрольных соотношений, если их можно получить, и заключительного контроля по разностям или по графику, если речь идет о таблице функций, или вычисления значения функции, если речь идет об отыскании корня. Все эти вычисления обязательно должны производиться, и также в две руки.

ГЛАВА II

СРЕДСТВА ВЫЧИСЛЕНИЙ

§ 5. Таблицы. Линейная интерполяция

Всевозможные таблицы функций являются одним из самых распространенных вспомогательных средств вычислений. Наиболее употребительными являются таблицы квадратов и кубов, квадратных и кубических корней, обратных величин, тригонометрических функций, логарифмов, показательной и других элементарных функций. Часто встречается необходимость и в разного рода таблицах специальных функций *).

Почти во всех случаях значения аргумента, которые приводятся в таблице, образуют арифметическую прогрессию; ее разность называют *шагом* таблицы. В некоторых таблицах для различных участков изменения аргумента выбирается различный шаг.

Значения функции обычно приводятся с одной и той же предельной абсолютной погрешностью, т. е. с фиксированным числом десятичных знаков. Реже встречаются таблицы, в которых применяется принцип постоянства относительной погрешности, т. е. дается определенное число значащих цифр.

Для практического ручного счета чаще всего бывает достаточно четырехзначных или пятизначных таблиц. Наиболее удобными и полными, а также и наиболее распространенными являются «Пятизначные математические таблицы»

---

*) К элементарным функциям относятся степенные, различного рода многочлены и алгебраические функции, логарифмическая, показательная, тригонометрические функции и обратные им, а также различные комбинации перечисленных функций. Остальные табулированные функции принято называть специальными.

Б. И. Сегала и К. А. Семендяева, «Четырехзначные математические таблицы» Милн-Томсона и Комри, а также четырехзначные таблицы из «Справочника по математике» И. Н. Бронштейна и К. А. Семендяева. К ним следует присоединить носящие более узкий характер, но также весьма употребительные таблицы «Специальные функции» Е. Янке, Ф. Эмде и Ф. Лёша. Можно пользоваться и пятизначными таблицами Л. С. Хренова, изданными для средней школы издательством «Просвещение».

Контрольные ручные расчеты для отладки программ на электронных вычислительных машинах ведутся уже со значительно большей точностью. Как правило, они бывают восьмизначными, а иногда и с большим числом знаков. Среди многозначных таблиц, используемых в таких расчетах, нужно прежде всего указать «Таблицы Барлоу», содержащие квадраты, кубы, квадратные и кубические корни, а также обратные величины. Там, где нельзя дать точных значений, эти таблицы являются восьмизначными. Существует также большое количество других многозначных таблиц элементарных и специальных функций.

Основной задачей, которая возникает при пользовании таблицами, является нахождение значений функции для тех значений аргумента, которые находятся между имеющимися в таблице. Эту задачу называют задачей *интерполяции*. Более подробно интерполяция функций будет рассматриваться во втором выпуске. Здесь мы рассмотрим лишь наиболее простой и наиболее широко распространенный способ — *линейную интерполяцию*.

Идея линейной интерполяции состоит в том, что функцию предполагают изменяющейся между двумя имеющимися в таблице значениями аргумента *линейно*. Иначе говоря, приращение функции на участке между двумя табличными значениями аргумента принимается пропорциональным приращению аргумента.

Нетрудно вывести общую формулу для линейной интерполяции. Пусть требуется найти значение функции в точке x, а x_0, x_1 — два табличных значения аргумента, в которых функция равна соответственно y_0 и y_1. Тогда на участке длины $x_1 - x_0$ функция изменяется на $y_1 - y_0$. На единицу длины приходится приращение $(y_1 - y_0)/(x_1 - x_0)$, а на участок длины $x - x_0$ — от точки x_0 до точки x — прира-

щение $(x - x_0)(y_1 - y_0)/(x_1 - x_0)$. Таким образом, в точке x функцию следует принять равной

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0). \qquad (1.5)$$

По большей части приращение функции $y_1 - y_0$ невелико и его считают в целых единицах последнего знака таблицы функции. Интерполяция обычно производится на один, следующий за имеющимся в таблице, десятичный знак, и поэтому разности $x_1 - x_0$ и $x - x_0$ также считают в целых единицах этого знака *), причем разность $x_1 - x_0$ в этом случае равна 10.

Пример 1.5. По табличным данным для функции $y = e^x$

$$x_0 = 0{,}83, \quad y_0 = 2{,}2933,$$
$$x_1 = 0{,}84, \quad y_1 = 2{,}3164,$$

найдем значение функции, соответствующее $x = 0{,}833$.

Приращение функции равно здесь 231 единице четвертого десятичного знака функции на 10 единиц третьего знака аргумента. Отсюда сразу получается, что на единицу третьего знака приходится 23, а на три единицы — 69 единиц четвертого знака функции. Таким образом, значение функции в точке $x = 0{,}833$ следует принять равным $2{,}2933 + 0{,}0069 = 2{,}3002$. Сравнение с более подробными таблицами показывает, что все полученные знаки верны.

Возможные погрешности интерполяции уменьшаются, если считать приращение на единицу следующего знака аргумента с одним запасным знаком, т. е. в нашем примере брать приращение на единицу третьего знака аргумента равным 23,1. Например, для значения $x = 0{,}837$ приращение следует считать равным $23{,}1 \cdot 7 = 162$ единицы четвертого знака функции (результат умножения округлен!), так что функцию следует принимать равной 2,3095.

Для облегчения интерполяции в таблицах часто печатаются разности между соседними значениями функции (*табличные разности*). Поскольку деление на 10 (или на 100, если аргумент имеет два лишних десятичных знака) достигается помещением запятой в соответствующем месте

*) Мы принимаем для простоты, что шаг таблицы, как это чаще всего бывает, равен 10^{-p}, т. е. единице какого-либо десятичного знака.

разности, остается только умножить полученное число на нужный множитель.

Впрочем, результаты умножения также можно заготовить заранее. Это сделано в специальной таблице, которую называют *таблицей пропорциональных частей*. Эта таблица печатается обычно на отдельном листе, который вкладывается в книгу. Она может быть использована для линейной интерполяции в любой таблице функций.

Пример 2.5. Таблица 1.5 содержит часть такой таблицы для значений разностей от 75 до 82.

Таблица 1.5

	75	76	77	78	79	80	81	82
1	7,5	7,6	7,7	7,8	7,9	8,0	8,1	8,2
2	15,0	15,2	15,4	15,6	15,8	16,0	16,2	16,4
3	22,5	22,8	23,1	23,4	23,7	24,0	24,3	24,6
4	30,0	30,4	30,8	31,2	31,6	32,0	32,4	32,8
5	37,5	38,0	38,5	39,0	39,5	40,0	40,5	41,0
6	45,0	45,6	46,2	46,8	47,4	48,0	48,6	49,2
7	52,5	53,2	53,9	54,6	55,3	56,0	56,7	57,4
8	60,0	60,8	61,6	62,4	63,2	64,0	64,8	65,6
9	67,5	68,4	69,3	70,2	71,1	72,0	72,9	73,8

Если, например, табличная разность равна 78 и аргумент имеет один лишний десятичный знак 6, то значение функции требуется увеличить на 47 единиц последнего знака. Если мы будем иметь два лишних десятичных знака, то, например, для аргумента $x = 0{,}3527$ значение функции в точке 0,35 нужно увеличить на $15{,}6 + 5{,}5 = 21$ (округлено!) единиц последнего знака (при той же разности 78).

Часто бывает выгодно вычислять непосредственно искомое значение функции, а не приращение, которое нужно прибавлять к табличному значению функции, как это делалось выше. Соответствующую формулу легко получить, исходя из уравнения прямой, проходящей через две данные точки, или путем алгебраического преобразования приведенной выше общей формулы (1.5).

Наиболее удобной формулой для получения значения функции является формула, полученная по так называемой *схеме Эйткина*. Пусть даны два значения аргумента

x_0 и x_1, в которых функция принимает соответственно значения y_0 и y_1. Значение функции в точке x может быть получено по формуле *)

$$P_{0,1}(x) = \frac{1}{x_1 - x_0} \begin{vmatrix} y_0 & x_0 - x \\ y_1 & x_1 - x \end{vmatrix}. \qquad (2.5)$$

Действительно, выражение $P_{0,1}(x)$ представляет собой многочлен первой степени относительно x, т. е. изменение функции на рассматриваемом участке предполагается линейным. Остается убедиться в том, что многочлен $P_{0,1}(x)$ принимает в заданных точках нужные значения. Чтобы это сделать, вычислим значения $P_{0,1}$ в точках x_0 и x_1:

$$P_{0,1}(x_0) = \frac{1}{x_1 - x_0} \begin{vmatrix} y_0 & 0 \\ y_1 & x_1 - x_0 \end{vmatrix} = y_0,$$

$$P_{0,1}(x_1) = \frac{1}{x_1 - x_0} \begin{vmatrix} y_0 & x_0 - x_1 \\ y_1 & 0 \end{vmatrix} = y_1.$$

Итак, функция $P_{0,1}(x)$ действительно является линейной функцией, совпадающей с заданной функцией в точках x_0 и x_1.

Схема Эйткина очень удобна для вычислений и дает возможность сразу получить требуемое значение функции.

Пример 3.5. По значениям функции $y = \sin x$ с аргументом, заданным в радианной мере,

$$x_0 = 0{,}40, \quad y_0 = 0{,}3894,$$
$$x_1 = 0{,}42, \quad y_1 = 0{,}4078,$$

найдем с помощью схемы Эйткина значение синуса для $x = 0{,}411$.

Пользуясь формулой (2.5), находим

$$P_{0,1} = \frac{1}{0{,}42 - 0{,}40} \cdot \begin{vmatrix} 0{,}3894 & -0{,}011 \\ 0{,}4078 & 0{,}009 \end{vmatrix} =$$
$$= \frac{0{,}3894 \cdot 0{,}009 + 0{,}4078 \cdot 0{,}011}{0{,}02} = 0{,}3995.$$

*) Здесь мы пользуемся выражением, которое называют *определителем второго порядка*; оно представляет сокращенную запись для разности произведений $y_0(x_1 - x) - y_1(x_0 - x)$.

§ 6. Функциональные шкалы

Кроме хорошо известного способа геометрического представления функции в виде графика, в вычислительной математике применяется также и другой способ изображения — *функциональные шкалы*.

Рассмотрим функцию $y = f(x)$, непрерывную и монотонную на некотором замкнутом интервале $[a, b]$. Возьмем ось OM, на которой будет строиться шкала, выберем на ней точку отсчета O и установим определенный масштаб μ, т. е. величину отрезка, принимаемого за единицу.

Функциональная шкала для функции $f(x)$ строится теперь следующим образом.

Разбив интервал $[a, b]$ на равные части, вычислим значение функции $f(x)$ в каждой из точек деления, включая начало и конец отрезка, и отложим на оси OM отрезок $\mu f(x)$. Полученная при этом точка снабжается отметкой x, т. е. откладывается в выбранном масштабе значение функции, а надписывается значение аргумента (рис. 1).

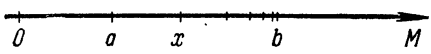


Рис. 1.

Иногда начало шкалы помещают в точку отсчета, т. е. точку с надписью a совмещают с точкой O. Тогда точка x будет находиться в конце отрезка $\mu [f(x) - f(a)]$.

Полученная шкала позволяет судить о поведении функции на рассматриваемом участке. Так, если функция возрастает, то надписи на шкале будут возрастать в положительном направлении на оси, а если убывает — в противоположном; большие промежутки между отметками укажут, что функция возрастает быстрее, чем там, где эти промежутки малы, и т. п.

Если наряду с функциональной шкалой построить также равномерную шкалу с отметками, содержащими истинные длины отложенных отрезков, то такую шкалу можно использовать для нахождения значений функции или значений обратной функции. Точность этих значений, естественно, определяется выбором масштаба и числа точек деления.

Построим для примера функциональную шкалу для функции $y = x^2$ на участке $[1, 2]$. Выбрав масштаб μ, мы

определим тем самым длину получающейся шкалы. Чаще поступают наоборот: задавшись заранее длиной шкалы l, определяют отсюда масштаб, что оказывается более удобным. Длина шкалы l и масштаб μ связаны соотношением $\mu\,[f(b) - f(a)] = l$. В нашем случае $f(a) = f(1) = 1$, $f(b) = f(2) = 4$, поэтому, выбрав $l = 6$ *см*, найдем $\mu = 2$ *см* на единицу измерения $f(x)$.

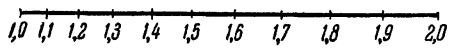


Рис. 2.

Для построения шкалы разобьем отрезок [1, 2] на десять частей и вычислим значения функции во всех точках деления, а затем подсчитаем длины отрезков, которые надо откладывать в выбранном масштабе. Все вычисления приведены в табл. 1.6. Перенеся полученные результаты на

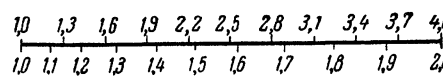


Рис. 3.

чертеж, получим функциональную шкалу для функции $y = x^2$ (рис. 2). Здесь начало шкалы как раз совпадает с точкой отсчета.

Таблица 1.6

(1)	x	1,00	1,10	1,20	1,30	1,40	1,50	1,60	1,70	1,80	1,90	2,00
(2)	$(1)^2$	1,00	1,21	1,44	1,69	1,96	2,25	2,56	2,89	3,24	3,61	4,00
(3)	(2) — «1»	0,00	0,21	0,44	0,69	0,96	1,25	1,56	1,89	2,24	2,61	3,00
(4)	«2» · (3)	0,00	0,42	0,88	1,38	1,92	2,50	3,12	3,78	4,48	5,22	6,00

На рис. 3 изображена та же функциональная шкала, дополненная равномерной шкалой (сверху). По ней можно

находить значение x^2 для $1 \leqslant x \leqslant 2$. Для этого следует найти x на нижней шкале и прочесть значение на верхней шкале, в случае необходимости интерполируя на глаз. Эта же шкала может быть использована для нахождения значений $\sqrt{x}$ при $1 \leqslant x \leqslant 4$, для чего следует найти x на верхней шкале и прочесть значение на нижней.

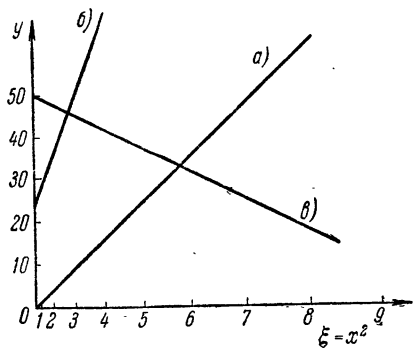


Рис. 4.

Примером функциональной шкалы является основная шкала логарифмической линейки. Она представляет шкалу, построенную для функции $y = \lg x$ на участке $1 \leqslant x \leqslant 10$. Масштаб совпадает здесь с длиной линейки, так как $\lg 10 = 1$. Чаще всего берется $\mu = 25$ *см*. Иногда встречаются также маленькие линейки с $\mu = 15$ *см* или, наоборот, большие, у которых $\mu = 50$ *см*. Действия на логарифмической линейке будут рассмотрены в § 7.

Функциональные шкалы находят широкое применение при обработке экспериментальных данных благодаря тому, что графики многих функций могут быть специальным подбором функциональных шкал преобразованы к прямолинейному виду. Познакомимся с некоторыми примерами.

Пример 1.6. Рассмотрим уравнения парабол:

а) $y = x^2$, б) $y = 3x^2 + 25$, в) $y = -0{,}5x^2 + 50$.

Построим теперь на оси Oy обычную равномерную шкалу, а на оси Ox — шкалу квадратов. Тогда получится сетка, изображенная на рис. 4. Построение такой шкалы эквивалентно замене переменных $x^2 = \xi$. Поэтому в новых координатах уравнения парабол будут иметь вид

а) $y = \xi$, б) $y = 3\xi + 25$, в) $y = -0{,}5\xi + 50$,

т. е. будут уравнениями первой степени, а значит, будут изображаться прямыми. Эти прямые и изображены на рис. 4.

Координатные сетки, построенные с помощью функциональных шкал, называют *функциональными сетками.*

Особенно часто используются логарифмические сетки, позволяющие «выпрямлять» графики показательных и степенных функций.

Действительно, рассмотрим показательную функцию $y = ae^{bx}$. Логарифмирование (по основанию 10) дает $\lg y = \lg a + b \lg e \cdot x$. Обозначим $\lg a = A$ и $b \cdot \lg e = B$. Если построить на оси Ox равномерную шкалу, а на оси Oy — логарифмическую, т. е. положить $\lg y = Y$, то получится линейная зависимость $Y = A + Bx$, которая выражается прямой линией.

Аналогично обстоит дело и для степенной функции $y = ax^b$. Логарифмирование дает $\lg y = \lg a + b \lg x$. Построив логарифмические сетки на обеих осях, т. е. полагая $\lg x = X$ и $\lg y = Y$ и обозначив $\lg a = A$, получим линейную зависимость $Y = A + bX$.

Такие сетки печатаются типографским способом и продаются наравне с миллиметровой бумагой под названием *полулогарифмической* и *логарифмической бумаги.*

§ 7. Логарифмическая линейка

Счетная логарифмическая линейка является наиболее простым и удобным, а потому и наиболее распространенным средством вычислений. Она позволяет вести вычисления с точностью трех значащих цифр (в среднем).

Принцип действия логарифмической линейки основан на сложении отрезков. Если взять две одинаковые равномерные

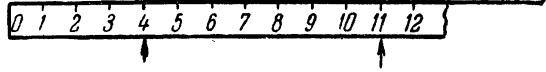


Рис. 5.

шкалы, то, сдвигая начальную точку одной из них на величину, равную одному слагаемому, и откладывая на сдвинутой шкале второе слагаемое, можно на неподвижной шкале найти соответствующую сумму (рис. 5).

Как было сказано в предыдущем параграфе, основной шкалой логарифмической линейки является не равномерная шкала, а логарифмическая. Поэтому сложение отрезков будет соответствовать уже не сложению чисел, а сложению

их л о г а р и ф м о в, т. е. умножению. Вычитанию отрезков соответствует деление. Таким образом, умножение и деление чисел производится с помощью основных шкал на линейке и на движке так, как это показано на рис. 6 и 7.

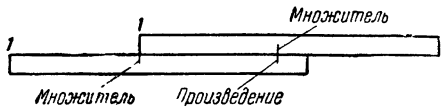


Рис. 6.

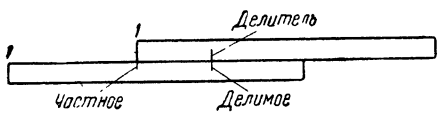


Рис. 7.

Для вычислений на логарифмической линейке числа удобнее всего задавать в плавающей форме, нормализованными. Порядок произведения в этом случае легко определяется; нужно только учитывать возможную необходимость нормализации результата. Впрочем, необходимость сдвига при нормализации легко определить по положению движка. Действительно, так как мантиссы нормализованных чисел заключены между 0,1 и 1 (по абсолютной величине), то их произведения содержатся между 0,01 и 1. Если произведение мантисс меньше, чем 0,1, то движок будет сдвинут вправо, а если больше 0,1, то влево. Во втором случае произведение получается нормализованным и его порядок равен сумме порядков сомножителей. В первом случае для нормализации мантиссу следует сдвинуть на один разряд влево, для чего необходимо уменьшить порядок на единицу, т. е. после нормализации порядок произведения на единицу меньше суммы порядков слагаемых.

Итак, мы приходим к следующему правилу: *порядок произведения равен сумме порядков сомножителей, если движок линейки сдвинут влево, и на единицу меньше этой суммы при сдвиге движка вправо.* Аналогичное правило справедливо также и для деления: *порядок частного равен разности порядков делимого и делителя, если движок линейки сдвинут*

влево, и на единицу больше этой разности при сдвиге движка вправо.

Кроме основных шкал, на логарифмической линейке имеется также еще ряд шкал, которые могут быть использованы для различных целей. В двух экземплярах — вверху движка и на корпусе линейки — имеется логарифмическая шкала, построенная в половинном масштабе и повторенная дважды. Она является шкалой квадратов, так как при переходе от основной шкалы к данной логарифм числа удваивается, а это соответствует возведению числа в квадрат. С помощью этой шкалы можно также и извлекать квадратные корни, переходя со шкалы квадратов на основную.

Легко сообразить, что *порядок квадрата равен удвоенному порядку числа, если квадрат находится в правой половине шкалы, и на единицу меньше удвоенного порядка числа, если квадрат находится в левой половине.* При извлечении квадратного корня *мантисса подкоренного выражения помещается на левой половине шкалы квадратов, если порядок числа нечетный, и на правой половине, если этот порядок четный.* Порядок корня определяется по следующему правилу: *порядок корня равен половине порядка подкоренного числа, если этот порядок четный, и на* 1/2 *больше ее, если порядок подкоренного числа нечетный.*

Благодаря тому, что шкала квадратов имеется и на движке, и на корпусе линейки, с помощью этих шкал можно производить умножение и деление, как и на основных. Это может быть полезным, например, в тех случаях, когда из произведения или частного нужно еще извлечь квадратный корень.

Самой верхней шкалой линейки обычно является утроенная логарифмическая шкала, которая служит шкалой кубов. С ее помощью можно возводить число в куб или извлекать кубический корень так же, как это описывалось выше для квадратов. Внизу на линейке помещается равномерная шкала. С ее помощью можно находить мантиссы десятичных логарифмов чисел или, наоборот, чйсла по их десятичным логарифмам, т. е. значения показательной функции $y = 10^x$. Это делается так же, как и по шкале квадратов, изображенной на рис. 3 в предыдущем параграфе. В середине движка часто помещается шкала

обратных величин, которая является той же логарифмической шкалой, только построенной справа налево.

На обороте движка нанесены шкалы для отыскания значений тригонометрических функций. Значение синуса для углов от 5°45′ до 90° или тангенса для углов от 5°45′ до 45° находится по основной шкале против отметки, соответствующей требуемому углу на шкале синусов или тангенсов. Так как для указанных углов синусы и тангенсы заключены в пределах от 0,1 до 1, то мы получаем нормализованную мантиссу и нулевой порядок. Для углов, меньших чем 5°45′, синус и тангенс в пределах точности линейки совпадают. Соответствующее значение угла ищется на средней шкале движка. Порядок найденного числа равен —1, мантисса находится по основной шкале.

Описанный способ предполагает, что движок перевернут тригонометрическими шкалами вверх. Благодаря рискам, нанесенным в прорезях с обратной стороны линейки, можно находить тригонометрические функции, не переворачивая движка. Для этого нужно установить требуемый угол на соответствующей шкале против риски; значение тригонометрической функции можно тогда прочесть на основной шкале движка, против единицы основной шкалы линейки.

Для свободного владения техникой вычислений на логарифмической линейке нужно уметь:

1) Быстро и безошибочно находить на любой шкале нужное число (его мантиссу) и устанавливать против него визирную линию или требуемую точку движка, а также легко считывать мантиссу полученного результата.

2) Легко подсчитывать порядок полученного результата.

3) Комбинировать действия таким образом, чтобы получать требуемый результат при наименьшем числе перебрасываний движка.

Соответствующие навыки приобретаются лишь при достаточной тренировке. Мы не станем описывать специальных приемов выполнения или комбинирования ряда операций, отсылая читателя к книгам, специально посвященным логарифмической линейке. Из них можно рекомендовать книгу Л. З. Румшиского «Счетная линейка» («Наука», 1964), а также книги К. А. Семендяева или Д. Ю. Панова с тем же названием, неоднократно издававшиеся.

§ 8. Арифмометры и клавишные машины

Основными средствами для ручных вычислений, позволяющими вести вычисления с точностью от четырех до восьми — десяти значащих цифр, являются в настоящее время арифмометры: ручные, полуавтоматические или автоматические. В основном, вычисления ведутся на полуавтоматических или автоматических арифмометрах, которые иначе называют клавишными машинами. Самыми распространенными клавишными машинами являются полноклавишные автоматы типа «Мерседес» и «Рейнметалл». По типу «Рейнметалла» (модель САР) сделаны автоматы ВММ-2 и полуавтоматы ВМП-2. Используются также десятиклавишные автоматы и полуавтоматы ВК-2. В последнее время широкое распространение получили электронные клавишные настольные вычислительные машины, математическое использование которых, в сущности, не отличается от электромеханических.

Приемы работы на различных клавишных машинах различны и описывать их здесь не стоит. Для свободного владения техникой вычислений на клавишных машинах нужно уметь:

1) Быстро и безошибочно набирать на нужных клавиатурах требуемые числа и считывать с соответствующих счетчиков полученные результаты. Удобно набирать числа на клавиатуре аккордом, нажимая одновременно несколько клавиш.

Некоторые вычислители рекомендуют набирать числа на клавиатуре левой рукой, чтобы правая была свободна для записей.

2) Твердо знать расположение клавиш соответствующих действий и находить их, не глядя на машину.

3) Широко пользоваться различными дополнительными возможностями, предоставляемыми машиной.

Клавишные машины следует считать, вообще говоря, машинами с плавающей запятой, поскольку положение запятой на них заранее не фиксировано. Однако для каждого отдельного действия число представляется в фиксированной форме, так что фактически в каждом *отдельном действии* клавишная машина работает как машина *с фиксированной запятой*. Положение

запятой может быть установлено с помощью специальных указателей, которые передвигаются вычислителем. Указатель запятой на счетчике результатов следует устанавливать до того, как нажата клавиша выполнения действия.

Клавишные машины позволяют выполнять все четыре арифметических действия. Кроме того, они дают возможность выполнения некоторых комбинированных действий. Например, благодаря возможности накопления на регистре (счетчике) результатов, можно получить сумму произведений, не выписывая промежуточных результатов. На некоторых машинах можно переносить числа с регистра результата на регистр установки множителя, благодаря чему можно выполнять несколько умножений подряд, также не выписывая промежуточных результатов.

При вычислениях на клавишных машинах широко используются различного рода математические таблицы. Их применение позволяет заменять выполнение многих операций выписыванием готовых результатов.

До появления клавишных машин очень большое значение для сложных многозначных расчетов имели вычисления с помощью таблиц логарифмов. Как известно, умножению или делению чисел соответствует сложение или вычитание их логарифмов, а эти действия легко выполнять вручную. В связи с этим был разработан ряд приемов преобразования выражений, особенно содержащих тригонометрические функции, к виду, удобному для логарифмирования.

Вследствие широкого распространения клавишных вычислительных машин вычисления с логарифмами играют уже далеко не ту первостепенную роль, как раньше. По сути дела, логарифмы в практике вычислений используются сейчас лишь для возведения в степень, если она дробная или большая, и для извлечения корня (кроме квадратного, который извлекается при помощи таблиц или непосредственно). Ввиду того, что вычисления с логарифмами потеряли массовый характер, преимущества десятичных логарифмов перед другими системами уже не играют никакой роли. Поэтому в настоящее время при вычислениях предпочтительнее пользоваться натуральными логарифмами, которые играют значительно бо́льшую теоретическую роль, нежели десятичные.

§ 9. Вычислительные машины непрерывного и дискретного действия

Наиболее мощным и широко применяемым в настоящее время средством вычислений являются вычислительные машины.

По принципу своей работы все вычислительные машины делятся на два больших класса: *машины непрерывного действия* (*моделирующие машины*) и *машины дискретного действия* (*цифровые машины*). В настоящем параграфе мы познакомимся с основными принципами работы машин этих классов.

В машинах и устройствах непрерывного действия числа представляются физическими величинами различного рода, которые могут меняться непрерывным образом. Это может быть длина отрезка или угол поворота вала, ток или напряжение в электрической цепи, температура, давление и т. п.

Простейшим примером вычислительного устройства непрерывного действия является *логарифмическая линейка*, на которой число изображается длиной отрезка. Работа линейки уже рассматривалась в § 7. Легко привести пример схемы, которую можно использовать для умножения и деления и в которой числа изображаются электрическими величинами.

Возьмем электрическую цепь, состоящую из батареи и омического сопротивления. Известно, что ток в такой цепи подчиняется закону Ома

$$I = U/R,$$

где I — ток и U — напряжение. Включив в цепь амперметр и вольтметр (рис. 8) и зная величину сопротивления R, мы можем по показаниям амперметра находить частное от деления U на R, а по показаниям вольтметра — произведение IR, придавая остальным величинам нужные значения. Таким образом, эта простейшая электрическая схема может быть использована для умножения и деления, причем числа изображаются здесь электрическими величинами: напряжением, величиной тока и сопротивлением.

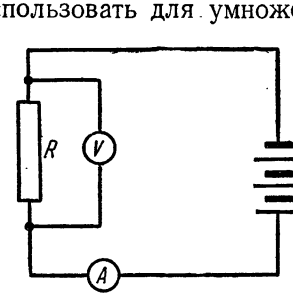


Рис. 8.

Из этого примера ясно, почему машины непрерывного действия называют *моделирующими машинами*. Машины непрерывного действия фактически не выполняют никаких вычислений. Они лишь воспроизводят (*моделируют*) процесс, который описывается данной функцией, уравнением или системой уравнений. Такое моделирование называют *математическим моделированием*, в отличие от физического моделирования, в котором изменяются лишь масштабы, но не физическая природа моделируемого процесса или явления.

Математические модели могут быть построены с использованием величин самой различной физической природы: механических, тепловых, гидродинамических, электрических, электромагнитных, электронных и т. п. Естественно, что чаще всего в машинах непрерывного действия используются электрические величины различного рода. Это объясняется компактностью соответствующих элементов, простотой их измерения и удобством осуществления нужных схем.

Существенным недостатком машин непрерывного действия является *сравнительно небольшая точность результатов*. Точность машины непрерывного действия определяется точностью измерительных приборов, измеряющих соответствующие физические величины, например, ток и напряжение в электрической цепи. Как правило, относительная ошибка измерения составляет несколько процентов или, в лучшем случае, несколько десятых процента. Это означает, что в показаниях прибора, а значит, и в результатах, выданных машиной, верными можно считать две или три значащих цифры. Дальнейшее повышение точности связано уже с серьезными затруднениями.

Другим недостатком моделирующих машин является их *специализированность*. Строго говоря, для каждой задачи необходимо иметь свою схему, т. е. свою модель, а значит, и другую машину. Для изменения схемы используются переключения, но эти возможности ограничены типом основных схем, имеющихся в данной машине. По этой причине машина непрерывного действия не может иметь универсального характера и область ее применения всегда ограничена задачами определенного класса.

Совсем иначе изображаются числа в машинах дискретного действия. Простейшим примером дискретного вычис-

лительного устройства являются хорошо известные *русские счеты*. Здесь число изображается в обычной цифровой форме в десятичной системе счисления. Для изображения каждого разряда числа отводится отдельная спица с надетыми на нее десятью косточками; чтобы изобразить какую-либо цифру, следует опустить соответствующее число косточек вниз. Вследствие того, что число в дискретных машинах изображается в цифровой форме, их называют также *цифровыми машинами*.

К цифровым машинам относятся и арифмометры, ручные и автоматические. В них числа изображаются также в десятичной системе счисления. Каждая цифра изображается углом поворота шестерни, которая может находиться в одном из десяти различных устойчивых состояний.

Часто используются также *счетно-аналитические* и *релейные вычислительные машины*. В счетно-аналитических машинах для изображения чисел используются пробивки на перфокартах (картонные карты с пробитыми отверстиями).

Релейные вычислительные машины являются промежуточным этапом между ручными (и счетно-аналитическими) машинами и электронными вычислительными машинами. В них для изображения чисел служат специальные электромеханические реле.

Скорости работы и возможности цифровых вычислительных машин весьма различны. С помощью автоматической клавишной вычислительной машины опытный вычислитель может выполнить до двух с половиной тысяч действий за семичасовую смену. Скорость работы счетно-аналитических и релейных машин составляет уже несколько операций в секунду. Советская релейная вычислительная машина РВМ-1 конструкции инженера Н. И. Бессонова, являвшаяся наиболее совершенной релейной машиной, обладала скоростью 20 арифметических операций в секунду.

Электронные вычислительные машины, с которыми мы подробно познакомимся дальше, в настоящее время обладают скоростями порядка десятков и сотен тысяч арифметических операций в секунду. Такое быстродействие связано с тем, что электронные устройства, применяемые в современных счетных машинах, не обладают инерцией механических или релейных элементов и время их срабатывания исчисляется миллионными долями секунды (микросекундами).

ЧАСТЬ ВТОРАЯ

ОСНОВЫ ПРОГРАММИРОВАНИЯ

ГЛАВА III

ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

§ 10. Структура программно-управляемой вычислительной машины

Процессу вычислений, который производится человеком, свойственны следующие основные элементы:

1. Хранение информации. Здесь под информацией подразумеваются исходные данные, промежуточные и окончательные результаты счета, а также алгоритм вычислений, т. е. формулы и способ счета, различного рода условия и т. п. Эта информация человеком частично запоминается, частично записывается на бумаге. Часть информации берется из различных справочников и таблиц. Память человека, бумага, справочники и таблицы являются различными видами запоминающих устройств.

2. Обработка информации. Для процесса вычислений это означает просто выполнение арифметических действий над числами. Обработка информации производится либо вручную (в уме), либо при помощи специальных счетных приборов, например, логарифмической линейки или арифмометра. При этом производится обмен информацией между устройством, предназначенным для выполнения арифметических действий, и запоминающим устройством: исходные данные берутся с листа бумаги, переносятся на арифмометр (или линейку), а затем результат вычислений снова записывается на бумаге или запоминается человеком.

3. Управление вычислительным процессом. Оно производится человеком. В соответствии с алгоритмом вычислитель производит вычислительные операции в определенной последовательности, каждый

раз решая, какие числа и в каком порядке переносить с бумаги на арифмометр или обратно.

Автоматическая программно-управляемая вычислительная машина должна быть устроена так, чтобы все перечисленные выше элементы процесса вычислений осуществлялись в ней без участия человека во время ее работы. В соответствии с этим требованием вычислительная машина должна содержать различные устройства, осуществляющие эти элементы процесса.

Основными частями вычислительной машины являются:

а) запоминающее устройство (*память*);

б) арифметическое устройство (*арифметика*);

в) устройство управления (*управление*);

г) устройства ввода и вывода (*ввод* и *вывод*).

Память машины преназначена для хранения всей требуемой информации, т. е. не только исходных, промежуточных и окончательных числовых значений, но также и алгоритма счета.

Памятью обладают все вычислительные устройства. Простые устройства, например арифмометр, обладают памятью в виде трех-четырех регистров (счетчиков) для исходных данных и результатов счета. В памяти современных вычислительных машин можно поместить тысячи и даже сотни тысяч чисел. Таким образом, память машины отличается от памяти арифмометра неизмеримо большим объемом. В памяти машины, помимо числового материала, помещается полный алгоритм счета, записанный в специальной форме. Его называют *программой* решения задачи.

Арифметика машины предназначена для переработки информации, т. е. для выполнения операций над числами, которые поступают туда из памяти. Это могут быть не только четыре основных арифметических операции, но и ряд других, о которых речь будет идти ниже. Главным отличием арифметики вычислительной машины от арифметических устройств других счетных приборов является громадное быстродействие. Современные машины выполняют десятки и сотни тысяч арифметических операций в секунду. В настоящее время вводятся в строй машины с быстродействием в миллионы операций в секунду.

Управление вызывает из памяти сведения о выполнении очередной операции, расшифровывает их, вызывает из

памяти нужные числа и засылает их в арифметическое устройство, а затем отсылает в память полученный результат, т. е. управляет ходом вычислительного процесса в соответствии с программой.

Ввод и *вывод* предназначены для общения человека с машиной и позволяют вводить в машину исходный материал и выводить из нее результаты счета.

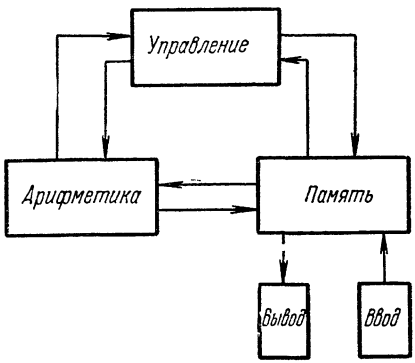


Рис. 9.

Общее представление о связи перечисленных устройств дает рис. 9, содержащий схему соединения основных частей (*блок-схему*) вычислительной машины. Проследим с помощью рис. 9 порядок решения задачи на машине.

Исходные числовые данные и программа задачи наносятся на перфоленту или перфокарты (бумажная лента или картонные карты, на которых цифры изображаются пробитыми в нужных местах отверстиями) и вводятся через устройство ввода в память. Через устройство управления в машину подается сигнал на начало счета. *Управление* вызывает из *памяти* сведения о первой элементарной операции программы и пересылает в *арифметику* исходные числа для данной операции и сведения, какую операцию над ними следует выполнить. После выполнения в *арифметике* этой операции *управление* пересылает результат снова в *память*, а затем извлекает оттуда сведения о следующей операции. Окончательные результаты счета из *памяти* по сигналу *управления* поступают в устройство вывода, которое печатает их на бумажной ленте.

§ 11. Команда в трехадресной машине. Арифметические операции

Память вычислительной машины состоит из большого числа N одинаковых ячеек, пронумерованных подряд от 0 до $N - 1$. Номер ячейки называется ее *адресом*. Каж-

дая ячейка содержит определенное число n разрядов. Поэтому в ячейке памяти может быть помещено любое n-разрядное целое число, которое мы в дальнейшем будем называть *словом*. *Слово* может быть использовано как число, как команда, либо какими-нибудь другими способами.

Ячейка памяти машины обладает той особенностью, что записанное в ней слово может храниться там и прочитываться любое число раз до тех пор, пока в эту ячейку не придется записать новое содержимое. *При записи в ячейку предварительно стирается прежнее содержимое, при чтении слова из ячейки оно сохраняется там без изменения.*

Рассмотрим более подробно использование слова как команды.

Арифметика каждой машины может выполнять определенное число *элементарных операций*. Для работы машины алгоритм решения задачи следует расписать в виде последовательности таких элементарных операций. Такую последовательность и называют *программой*. Программа состоит из *команд*, каждая из которых содержит сведения об одной элементарной операции.

Команда занимает одну ячейку памяти и состоит из двух основных частей — *операционной* и *адресной*. Операционная часть команды содержит сведения о характере операции. Для этой цели все элементарные операции машины нумеруются и в операционной части команды записывается номер операции, которую следует выполнить. Этот номер называют *кодом операции*.

Адресная часть команды содержит адреса (номера) ячеек, содержимое которых участвует в операции. У различных машин число адресов в адресной части команды может быть различным. В этой части курса мы ограничимся рассмотрением *трехадресных машин*, в адресной части которых указаны *адреса трех ячеек памяти*. Их называют первым, вторым и третьим адресами данной команды. В первом и втором адресах записываются номера ячеек памяти, где находятся исходные числа, над которыми нужно совершить данную операцию. Третий адрес указывает номер ячейки, куда следует записать результат *).

*) Мы описываем структуру команды, соответствующей арифметической операции. В других случаях использование адресов может быть иным.

В процессе работы машины команда, находящаяся в некоторой ячейке памяти, извлекается устройством управления и расшифровывается. В соответствии с кодом операции, арифметическое устройство выполняет нужную операцию; управление вызывает из памяти и направляет в арифметику требуемые числа, а после выполнения операции отсылает в память полученный результат, после чего извлекает из памяти следующую команду.

Для арифметических операций *следующая команда* всегда означает *команду, лежащую в следующей ячейке*, т. е. в ячейке, адрес которой на единицу больше. Поэтому для правильного выполнения программы последовательные команды должны размещаться в ячейках памяти подряд.

Основными элементарными операциями любой вычислительной машины являются арифметические: сложение, вычитание, умножение и деление. Эти операции производятся в машине над нормализованными числами с плавающей запятой.

Арифметические команды в трехадресной машине естественно записывать следующим образом:

$$a+b=c, \quad a\cdot b=c,$$
$$a-b=c, \quad a:b=c.$$

Здесь через a, b, c обозначены адреса ячеек памяти, содержимое которых также обозначено через a, b, c.

К примеру, запись $a : b = c$ означает команду: число из ячейки, адрес которой обозначен через a, разделить на число из ячейки b, а частное поместить в ячейку c.

Для рассматриваемой трехадресной машины элементарными являются еще две арифметические операции: вычитание абсолютных величин и извлечение корня. Эти операции записываются так:

$$|a|-|b|=c,$$
$$\sqrt{a} \quad =c.$$

При операции извлечения квадратного корня второй адрес не используется (это двухадресная операция).

Простейшая задача, которую может решать вычислительная машина, есть вычисление алгебраического выражения, образующегося при помощи четырех арифметических действий. Составить программу расчета означает здесь

расписать последовательность соответствующих элементарных операций.

Пример 1.11. В качестве первого примера рассмотрим вычисление значения квадратного трехчлена:

$$y = ax^2 + bx + c.$$

Сделаем расписку этой формулы по элементарным операциям:

1) $b \cdot x = R_1$ 4) $a \cdot R_3 = R_4$
2) $R_1 + c = R_2$ 5) $R_2 + R_4 = y$
3) $x \cdot x = R_3$

По такой схеме, в которой через R_1, R_2, R_3, R_4 обозначены результаты промежуточных действий, можно производить вычисления на арифмометре или на клавишной счетной машине. Эта же схема может служить и программой вычисления квадратного трехчлена на трехадресной машине.

Предположим, что числовые величины x, a, b, c помещены в некоторые ячейки машины; обозначим адреса этих ячеек теми же буквами, что и соответствующие величины. Кроме того, обозначим через R_1, R_2, R_3, R_4 адреса ячеек, в которых в машине будут записываться результаты промежуточных действий, а через y — адрес ячейки для значения квадратного трехчлена. Тогда формулу

$$1)\ b \cdot x = R_1$$

можно рассматривать как трехадресную команду: перемножить числа из ячеек b и x и произведение поместить в ячейку R_1. Аналогично, формулы 2) — 5) можно рассматривать как арифметические операции над числами, находящимися в соответствующих ячейках.

Поместим команды 1), 2), 3), 4), 5) в пять ячеек памяти подряд и заставим машину последовательно выполнить эти команды. Тогда в ячейке y окажется значение квадратного трехчлена. После команды 5) следует поставить команду остановки машины, которая записывается так:

$$6)\ стоп$$

(все три адреса у этой команды можно считать нулевыми).

В рассмотренной простейшей программе разные ячейки памяти используются по-разному: шесть ячеек, в которых

записаны команды 1) — 6), называются *командными*; ячейки для исходных данных x, a, b, c — *аргументами* программы; ячейки для промежуточных величин R_1, R_2, R_3, R_4 — *рабочими ячейками*, y — ячейкой *результата*. Всего программа занимает 15 ячеек:

1), 2), 3), 4), 5), 6), a, b, c, x, y, R_1, R_2, R_3, R_4,

которые могут быть расположены в любых местах памяти. Только командные ячейки 1)—6) должны быть расположены подряд.

При составлении программы счета квадратного трехчлена можно сократить количество рабочих ячеек от четырех до одной, если записывать промежуточные результаты в одну рабочую ячейку R и ячейку результата y:

1) $b \cdot x = y$ 4) $a \cdot R = R$
2) $y + c = y$ 5) $y + R = y$
3) $x \cdot x = R$ 6) *стоп*

Обратим внимание на команды 2), 4), 5), в которых одна и та же ячейка встречается и слева, и справа. Например, в команде 2) написано $y + c = y$. Если бы здесь буква y означала определенную величину, то такое равенство означало бы, что $c = 0$. На самом деле это не так, потому что в нашей записи команды 2) буква y означает адрес ячейки, которая отведена для величины y, а равенство $y + c = y$ не есть арифметическое равенство, а есть записанная в привычных арифметических обозначениях команда для машины: числа из ячеек, адреса которых обозначены буквами c и y, сложить и результат записать в ячейку y.

Можно вообще обойтись без рабочих ячеек, составляя программу по формуле $y = x(ax + b) + c$:

1) $a \cdot x = y$ 4) $y + c = y$
2) $y + b = y$ 5) *стоп*
3) $x \cdot y = y$

Таким образом, по сравнению с первоначальным вариантом мы сократили объем памяти, занимаемой программой, на пять ячеек: избавились от четырех рабочих ячеек и уменьшили количество команд с шести до пяти.

Пример 2.11. Составить программу вычисления алгебраического выражения:

$$z=(|x+1|-|x/y|)\sqrt{|x+2|}$$

и поместить величину z в две ячейки z_1 и z_2. Обозначим через «1» и «2» адреса ячеек, в которых расположены целые числа 1 и 2.

Аргументами программы является содержимое ячеек x, y, «1», «2», результатами — содержимое ячеек z_1, z_2. Составим программу, воспользовавшись тремя рабочими ячейками R_1, R_2, z и ячейкой «0», в которую поместим число нуль:

1) $x + $ «1» $= R_1$	6) $\sqrt{R_2} = R_2$
2) $x : y = R_2$	7) $R_1 \cdot R_2 = z$
3) $\lvert R_1\rvert - \lvert R_2\rvert = R_1$	8) «0» $+ z = z_1$
4) $x + $ «2» $= R_2$	9) «0» $+ z = z_2$
5) $\lvert R_2\rvert - \lvert$«0»$\rvert = R_2$	

Обычно при составлении программ число используемых ячеек (как командных, так и рабочих) стараются, по возможности, уменьшить. В частности, в приведенном примере легко можно было бы уменьшить программу на одну команду и обойтись без рабочих ячеек R_1, z, помещая промежуточные величины в результирующие ячейки z_1 и z_2. Отметим, однако, что *экономию ячеек памяти производят только тогда, когда это необходимо*. Часто важнее не сэкономить несколько ячеек, а сохранить в программе обозначения, более близкие к формульной записи программируемого выражения.

§ 12. Команды передачи управления. Разветвляющиеся программы

В большинстве вычислительных процессов мы сталкиваемся с тем, что выбор хода дальнейших вычислений определяется результатами предыдущих. Более точно можно сказать, что вычислительный процесс разбивается на ряд этапов, и переход от одного этапа к другому зависит от выполнения некоторых условий. Проверка выполнения этих условий тоже может рассматриваться как некоторый

этап вычислительного процесса, хотя он и не требует выполнения арифметических операций.

Например, процесс нахождения корней квадратного уравнения $ax^2 + bx + c = 0$ можно разбить на следующие этапы:

1) вычисление дискриминанта уравнения $D = b^2 - 4ac$,

2) проверка знака дискриминанта,

3) вычисление действительных корней уравнения (производится при $D \geqslant 0$),

4) вычисление мнимых корней уравнения (производится при $D < 0$).

Первый этап состоит из арифметических операций. Второй является проверкой логического условия и арифметических операций не требует. Это логическое условие можно сформулировать в виде: «верно ли, что дискриминант уравнения неотрицателен?» или «выполняется ли условие $D \geqslant 0$?». В зависимости от результатов вычислений, которые были произведены на первом этапе, мы получаем один из двух возможных ответов: «да» или «нет». В случае ответа «да» дальнейшие вычисления производятся по формулам, соответствующим третьему этапу — вычислению действительных корней. При ответе «нет» следует обратиться к формулам четвертого этапа.

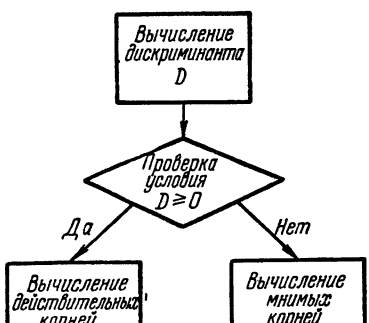


Рис. 10.

Разбиение этого процесса на этапы можно наглядно представить схемой, изображенной на рис. 10. Такие схемы называются *блок-схемами вычислительного процесса*.

Здесь мы имеем дело с вычислительным процессом, разветвляющимся в двух направлениях. Если допустить, что коэффициент a квадратного уравнения может обращаться в нуль, то вычислительный процесс будет разветвляться уже в трех направлениях. Его блок-схема имеет вид, показанный на рис. 11.

Другим примером разветвляющегося процесса является задача нахождения наибольшего из нескольких чисел.

К разветвляющимся процессам приводит и вычисление ряда физических величин. Например, при движении тела в воздухе величина силы сопротивления вычисляется по-разному, в зависимости от скорости движения; при малых скоростях сила сопротивления пропорциональна первой степени скорости, а при больших — второй.

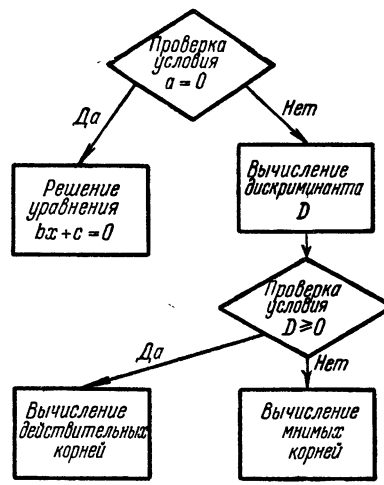


Рис. 11.

При ручном счете проверка выполнения логических условий, определяющая дальнейшее направление вычислительного процесса, производится вычислителем. Вычислительная машина работает с очень большой скоростью и автоматически, т. е. без участия человека. Поэтому в ней должна быть заложена возможность с помощью программы производить проверку выполнения логических условий и осуществлять, в зависимости от результатов этой проверки, разветвление вычислительного процесса.

Прежде всего в машине должен вырабатываться некоторый признак, показывающий, выполняется или не выполняется проверяемое условие. Для этой цели в машине имеется специальный регистр, в котором может быть записан 0 или 1. Содержимое этого регистра называют *управляющим сигналом* или *сигналом* ω.

Проверяемому логическому условию можно придать форму утверждения о знаке или абсолютной величине некоторого получающегося результата арифметического действия *), например, «верно ли, что $D \geqslant 0$», как при

*) Другие формы логических условий, не относящиеся к результатам арифметических операций, будут рассмотрены в следующих главах.

решении квадратного уравнения. Так как для этой цели может быть использован результат любого арифметического действия, то управляющий сигнал ω вырабатывается в машине при каждой арифметической операции.

Операции сложения, вычитания и вычитания абсолютных величин вырабатывают сигнал $\omega = 1$, если результат операции отрицателен, и $\omega = 0$, если он неотрицателен. После выполнения операций умножения, деления и извлечения квадратного корня вырабатывается сигнал $\omega = 0$, если результат меньше единицы по абсолютной величине, и $\omega = 1$ в противоположном случае.

Можно считать, что проверка выполнения логического условия (имеющего указанную выше форму) происходит одновременно с выполнением операции. К моменту окончания операции условие уже проверено и в машине выработался соответствующий признак. Однако мало получить этот признак, нужно еще иметь возможность им воспользоваться. Для этой цели в числе элементарных операций машины предусмотрены *операции условной передачи управления*.

Поясним сначала терминологию. Если машина выполняет команду, взятую из ячейки памяти с номером r, то говорят, что *управление находится в ячейке* r. Как было сказано в § 11, после выполнения арифметической операции машина переходит к выполнению команды, находящейся в следующей ячейке. Это означает, что арифметическая команда *всегда передает управление следующей ячейке*, так что естественный порядок выполнения команд сохраняется. *Операциями передачи управления называют команды, изменяющие естественную последовательность выполнения команд программы.*

В машине имеется две операции условной передачи управления по сигналу ω, которые мы будем обозначать через *У0* и *У1*. Здесь буква *У* обозначает условную передачу управления, а цифра 0 или 1 — значение сигнала ω, при котором передача управления происходит. В адресной части команды нужно указать адрес ячейки, которой, в зависимости от выполнения условия, следует передать управление. Мы будем предполагать, что этот адрес записывается во втором (среднем) адресе команды

передачи управления; первый и третий адреса будем п о к а считать нулевыми.

Команды условной передачи управления выполняются следующим образом. Если команда

$$У0 \quad n$$

лежит в ячейке с адресом r и в результате предыдущей операции выработался управляющий сигнал $\omega = 0$, то управление будет передано в ячейку n, т. е. следующей будет выполняться команда, записанная в ячейке n. Если же в результате предыдущей операции выработался сигнал $\omega = 1$, то управление передается ячейке с адресом $r + 1$.

Наоборот, команда

$$У1 \quad n,$$

лежащая в ячейке с адресом r, передает управление ячейке n, если в результате предыдущей операции выработался сигнал $\omega = 1$, и ячейке $r + 1$ при сигнале $\omega = 0$.

Кроме описанных команд условной передачи управления, в машине есть также команда *безусловной передачи управления*, которая в с е г д а передает управление ячейке, номер которой записан во втором адресе команды. Команду безусловной передачи управления мы будем обозначать так:

$$Б \quad n.$$

Первый и третий адреса этой команды также будем п о к а считать нулевыми.

Рассмотрим несколько программ разветвляющихся вычислительных процессов.

П р и м е р 1.12. Составить программу решения квадратного уравнения $ax^2 + bx + c = 0$.

Пусть всегда $a \neq 0$. Тогда, если $D = b^2 - 4ac \geqslant 0$, то

$$x_1 = -\frac{b}{2a} + \frac{\sqrt{D}}{2a}, \quad x_2 = -\frac{b}{2a} - \frac{\sqrt{D}}{2a},$$

если же $D < 0$, то

$$x_1 = -\frac{b}{2a} + i\frac{\sqrt{|D|}}{2a}, \quad x_2 = -\frac{b}{2a} - i\frac{\sqrt{|D|}}{2a}.$$

После работы программы в ячейки x_1, x_1', x_2, x_2' должны быть помещены действительные и мнимые части корней.

Блок-схему (схему счета) программы можно представить в виде рис. 12.

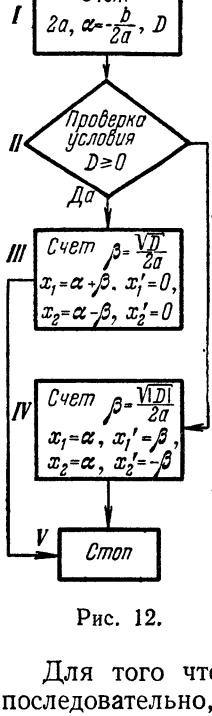


Рис. 12.

Программы для отдельных блоков (участков) задачи можно написать так:

Первый блок

1) $a+a=2a$
2) $b:2a=R_1$
3) «0» $-R_1=\alpha$
4) $b\cdot b=R_1$
5) $2a\cdot c=R_2$
6) $R_2+R_2=R_2$
7) $R_1-R_2=D$

Третий блок

1) $\sqrt{D}=R_1$
2) $R_1:2a=\beta$
3) $\alpha+\beta=x_1$
4) $\alpha-\beta=x_2$
5) «0» + «0» $=x_1'$
6) «0» + «0» $=x_2'$

Четвертый блок

1) «0» $-D=R_1$
2) $\sqrt{R_1}=R_1$
3) $R_1:2a=\beta$
4) «0» $+\alpha=x_1$
5) «0» $+\alpha=x_2$
6) «0» $+\beta=x_1'$
7) «0» $-\beta=x_2'$

Для того чтобы эти три счетных блока, записанные последовательно, объединить в программу решения квадратного уравнения, воспользуемся тем, что последняя команда первого блока вырабатывает сигнал $\omega = 1$, если $D < 0$, и сигнал $\omega = 0$, если $D \geqslant 0$.

Поэтому, расположив блоки в порядке, указанном на схеме рис. 12, следует после блока I поставить команду

У1 блок IV,

которая заставит обойти счет действительных корней (блок III) и сосчитать мнимые корни, если $\omega = 1$, т. е. $D < 0$.

При выполнении условия $D \geqslant 0$ после блока I сигнал ω окажется равным нулю, и команда «*У1* блок IV» передаст

управление на первую команду следующего за ней блока III вычисления действительных корней. Для того чтобы после счета действительных корней не считались мнимые, после блока III следует поставить команду безусловной передачи управления на конец счета:

Б стоп

Таким образом, программа вычисления корней квадратного уравнения имеет следующий вид:

1)	$a + a = 2a$	Блок I
2)	$b : 2a = R_1$	
3)	«0» $- R_1 = \alpha$	
4)	$b \cdot b = R_1$	
5)	$2a \cdot c = R_2$	
6)	$R_2 + R_2 = R_2$	
7)	$R_1 - R_2 = D$	
8)	*У*1 16)	Блок II
9)	$\sqrt{D} = R_1$	Блок III
10)	$R_1 : 2a = \beta$	
11)	$\alpha + \beta = x_1$	
12)	$\alpha - \beta = x_2$	
13)	«0» + «0» $= x_1'$	
14)	«0» + «0» $= x_2'$	
15)	*Б* 23)	
16)	«0» $- D = R_1$	Блок IV
17)	$\sqrt{R_1} = R_1$	
18)	$R_1 : 2a = \beta$	
19)	«0» $+ \alpha = x_1$	
20)	«0» $+ \alpha = x_2$	
21)	«0» $+ \beta = x_1'$	
22)	«0» $- \beta = x_2'$	
23)	*стоп*	Блок V

Проследим, как работает составленная программа. Сначала блок I (команды 1) — 7)) вычисляет величины $2a$, $\alpha = -b/(2a)$, $D = b^2 - 4ac$. Если дискриминант D оказывается отрицательным, то команда 7) вырабатывает сигнал $\omega = 1$, и по команде 8) управление передается на начальную команду 16) блока IV вычисления мнимых корней; вычислив мнимые корни (команды 16) — 22)), машина останавливается (команда 23)). Если же $D \geqslant 0$, то команда 7) вырабатывает сигнал $\omega = 0$ и команда 8) передает управление следующей за ней начальной команде 9) блока III счета действительных корней. Сосчитав эти корни (команды 10) — 14)), машина выполняет команду безусловной передачи управления 15), которая заставляет машину, обойдя блок IV счета мнимых корней, выйти на *стоп* (команда 23)).

До сих пор мы полагали первый и третий адреса команд передачи управления нулевыми. Эти команды работали у нас как одноадресные, в них использовался лишь второй адрес.

На самом деле, в машине предусмотрена возможность работы команд передачи управления как трехадресных. В этом более общем случае они записываются так:

$$У0\ \overrightarrow{a\ \ b\ \ c},$$
$$У1\ \overrightarrow{a\ \ b\ \ c},$$
$$Б\ \overrightarrow{a\ \ b\ \ c}.$$

Как и ранее, по этим командам осуществляется условная или безусловная передача управления команде с номером b или команде с номером $r + 1$; символы *У0*, *У1*, *Б* здесь имеют прежнее значение. Однако, кроме передачи управления, по этим командам происходит пересылка содержимого ячейки с номером a (первый адрес) в ячейку с номером c (третий адрес). Такое совмещение двух операций в одной команде не замедляет ее выполнения, ибо пересылка производится одновременно с передачей управления. При этом надо иметь в виду, что *пересылка выполняется всегда*, независимо от значения сигнала ω.

Покажем на примере программы решения квадратного уравнения, как использовать команды передачи управления в указанном более общем виде.

В этой программе имеются пять команд, являющихся, по существу, командами пересылки: 13), 14), 19), 20), 21). Пересылку 14) можно совместить с командой безусловной передачи управления 15), которая после этого будет выглядеть так:

$$Б\ \overrightarrow{«0»\ стоп\ x_2'}$$

Команду пересылки 19) можно совместить с командой условной передачи управления 8), что дает

$$У1\ \overrightarrow{\alpha\ блок\ IV\ x_1}$$

Заметим, кроме того, что в блоке IV можно еще избавиться от команды 21), помещая результат выполнения команды 18) в ячейку x_1'. После этих преобразований программа решения квадратного уравнения примет такой вид:

1)	$a + a = \rho$	11)	$\alpha + \beta = x_1$
2)	$b : \rho = R_1$	12)	$\alpha - \beta = x_2$
3)	$«0» - R_1 = \alpha$	13)	$«0» + «0» = x_1'$
4)	$b \cdot b = R_1$	14) $Б$	$\overrightarrow{«0»\ 20)\ x_2'}$
5)	$\rho \cdot c = R_2$	15)	$«0» - D = R_1$
6)	$R_2 + R_2 = R_2$	16)	$\sqrt{R_1} = R_1$
7)	$R_1 - R_2 = D$	17)	$R_1 : \rho = x_1'$
8)	$У1\ \overrightarrow{\alpha\ 15)\ x_1}$	18)	$«0» - x_1' = x_2'$
9)	$\sqrt{D} = R_1$	19)	$«0» + x_1 = x_2$
10)	$R_1 : \rho = \beta$	20)	*стоп*

Таким образом, программа сократилась на три команды; выполняться она будет быстрее, чем прежняя.

Отметим еще следующую особенность этой программы: по команде 8) одновременно с передачей управления в ячейку x_1 засылается величина α, вне зависимости от того, вычисляются дальше мнимые или действительные корни. Однако $x_1 = \alpha$ лишь в случае мнимых корней. Не получаются ли по нашей программе неправильные значения действительных корней? На самом деле это не так, ибо при работе блока вычисления действительных корней в ячейку x_1 посылается взамен α правильное значение $\alpha + \beta$ (команда 11)).

В написанной программе используются шесть рабочих ячеек: ρ, α, D, β, R_1, R_2. Можно вообще избавиться от рабочих ячеек, помещая промежуточные результаты в результирующие ячейки x_1, x_2, x_1', x_2'.

При записи программ в буквенном виде широко применяют стрелки вместо вторых адресов команд передачи управления. Эти стрелки направляются к той команде, к которой условно или безусловно передается управление. Нумерация команд при этом оказывается излишней.

При помощи команд передачи управления можно решать и некоторые логические задачи. Рассмотрим одну из простейших задач такого рода.

Пример 2.12. Из двух чисел a и b найти меньшее по абсолютной величине и поместить в ячейку m.

Чтобы сравнить абсолютные величины чисел a и b, выполним команду

$$|a|-|b|=0$$

Если $|a| \geqslant |b|$, эта команда выработает сигнал $\omega = 0$. Если $|a| < |b|$, сигнал ω будет равен 1. Поэтому при $\omega = 0$ в m следует послать b, а при $\omega = 1$ послать a.

Программа нахождения m имеет вид

1)	$\|a\|-\|b\|=0$	4)	*Б*	
2)	*У0*	5)	5)	«0» $+\, b = m$
3)	«0» $+\, a = m$	6)	*стоп*	

Можно сократить эту программу на две команды, совместив пересылку «0» $+\, a = m$ с условной передачей управления и опустив безусловную передачу управления:

	$\|a\|-\|b\|=0$	
У0	b	m
	«0» $+\, a = m$	
стоп		

Проследим, как работает эта программа. Если $|a| \geqslant |b|$, команда 1) вырабатывает сигнал $\omega = 0$ и команда 2) передает управление на *стоп*, одновременно засылая в m меньшее по абсолютной величине число b. Если же $|a| < |b|$, то команда 1) вырабатывает сигнал $\omega = 1$. После этого команда 2) передает управление команде 3), а в ячейке m,

вопреки требуемому, оказывается большее по абсолютной величине число b; однако следующая команда 3) посылает в ячейку m меньшее по абсолютной величине число a.

К разветвлению программ мы приходим в задачах, в которых искомая величина находится по-разному в зависимости от выполнения каких-либо условий.

Рассмотрим пример из механики.

Пример 3.12. Шар массы M с радиусом ρ и постоянной плотностью δ притягивает материальную точку массы m, находящуюся на расстоянии r от центра шара. Сила F притяжения точки шаром вычисляется так: если точка находится вне шара или на его поверхности, т. е. если $r \geqslant \rho$, то

$$F = f \cdot Mm/r^2,$$

где f — коэффициент, называемый постоянной тяготения; если же точка находится внутри шара, т. е. если $r < \rho$, то

$$F = f \cdot mM(r)/r^2,$$

где $M(r)$ — масса части данного шара с радиусом r. Очевидно,

$$M = (4/3)\,\pi\rho^3\delta, \qquad M(r) = (4/3)\,\pi r^3\delta.$$

Поэтому формулу для вычисления силы F можно представить в виде

$$F = \begin{cases} \gamma\rho^3/r^2, & \text{если } r \geqslant \rho, \\ \gamma r, & \text{если } r < \rho, \end{cases}$$

где $\gamma = (4/3)\,\pi f\delta m$ — постоянный коэффициент (эта величина имеет определенный физический смысл: она равна силе, с которой шар радиуса 1 и плотности δ притягивает точку массы m, находящуюся на его поверхности).

Условие $r \geqslant \rho$ можно записать и так: $r - \rho \geqslant 0$. Чтобы его проверить, следует выполнить команду $r - \rho = k$. Если величина $k \geqslant 0$, т. е. $\omega = 0$, следует считать F по первой формуле, если $k < 0$ ($\omega = 1$), то по второй.

Разность $k = r - \rho$ для расчета F не нужна, поэтому в качестве ячейки k можно использовать любую свободную ячейку. Обычно в качестве такой ячейки берут ячейку с нулевым номером. Команду, вырабатывающую сигнал ω, пишут тогда так:

$$r - \rho = 0.$$

Составим блок-схему расчета F. Она показана на рис. 13. Программа из 14 команд, составленная в соответствии с этой блок-схемой, имеет следующий вид:

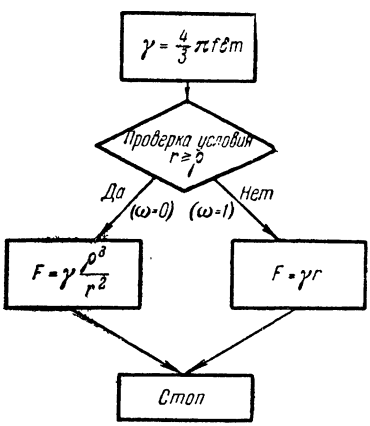


Рис. 13.

```
1)  «4/3» · «п» = γ
2)      γ · f  = γ
3)      γ · δ  = γ
4)      γ · m  = γ
5)      r − ρ  = 0
6)  У0      9)
7)      γ · r  = F
8)  Б
9)      γ : r  = F
10)     F : r  = F
11)     F · ρ  = F
12)     F · ρ  = F
13)     F · ρ  = F
14) стоп
```

Запишем формулы для определения F несколько иначе:

$$F = \begin{cases} \gamma r\,(\rho/r)^3, & \text{если } \rho/r < 1, \\ \gamma r, & \text{если } \rho/r \geqslant 1. \end{cases}$$

При такой записи естественно вести вычисления в следующем порядке: сначала вычислить значение F по второй из формул, затем проверить условие $\rho/r \geqslant 1$, и, если оно не выполняется, домножить прежнее значение F на $(\rho/r)^3$. Программа вычисления F указанным способом состоит из 11 команд и имеет такой вид:

```
1)  «4/3» · «п» = γ      7)  У1
2)  γ  · f  = γ          8)      F · k = F
3)  γ  · δ  = γ          9)      F · k = F
4)  γ  · m  = γ          10)     F · k = F
5)  γ  · r  = F          11) стоп
6)  ρ  : r  = k
```

Команда деления 6) в этой программе используется и для получения частного $\rho : r = k$, и для выработки сигнала ω. Если величина $k \geqslant 1$, то команда 6) вырабатывает сигнал $\omega = 1$ и команда 7) выводит нас на *стоп*, сохраняя вычисленное при помощи команд 1)—5) значение $F = \gamma r$. Если же $k < 1$, то после команды 6) вырабатывается сигнал $\omega = 0$, и команда 7) передает управление на команду 8). После этого, в полном соответствии с последней формулой, старое значение $F = \gamma r$ три раза умножается на $k = \rho/r$ и машина выходит на *стоп*.

Сравнивая эту программу с предыдущей, мы убеждаемся в том, что более рациональная запись формулы и использование команды деления для выработки сигнала ω дали возможность сэкономить три команды.

§ 13. Арифметические циклы

Как видно из примера предыдущего параграфа, характерной особенностью разветвляющихся программ является то, что отдельные команды или группы команд выполняются или не выполняются (обходятся) в зависимости от тех или иных условий. Однако во всех случаях каждая выполняемая команда выполняется только один раз.

Ясно, что задачи, программы для которых имеют такой характер, крайне невыгодны для решения на электронных вычислительных машинах, поскольку время написания одной команды неизмеримо превышает время ее выполнения. Наиболее выгодными для машины являются такие задачи, в которых одна и та же команда или группы команд выполняются многократно. Составление таких программ тем более возможно, что в команде указываются не числа, над которыми производятся операции, а адреса ячеек; содержимое же этих ячеек при повторном выполнении тех же команд может быть иным.

Группу команд, которая выполняется в процессе решения задачи несколько раз, принято называть *циклом* или *циклической программой*. Циклы являются основными частями всяких практически используемых программ, а потому организация цикла есть наиболее часто встречающаяся задача программирования. Как мы увидим, циклы состоят

обычно из одних и тех же частей, которые располагаются определенным образом.

Рассмотрим примеры программирования циклических вычислительных процессов.

Пример 1.13. Составим программу для вычисления n-й степени числа x (n — целое положительное число):

$$z = x^n.$$

Расписывая формулу для z по командам, получим программу:

$$\begin{array}{l} \text{«0»} + x = z \\ \left.\begin{array}{l} z \cdot x = z \\ z \cdot x = z \\ \ldots\ldots \\ z \cdot x = z \end{array}\right\} \; n-1 \text{ раз} \\ \textit{стоп} \end{array}$$

Таким образом, при программировании этой задачи потребовалась однообразная работа по выписыванию $n + 1$ команд. Между тем очевидно, что процесс вычисления z является циклическим: он состоит из многократного повторения команды умножения $z \cdot x = z$. Приведенная программа является бесцикловой в том смысле, что каждая из написанных команд выполняется только один раз.

Следует обратить внимание на то, что написанные в программе команды одинаковы. Все они (кроме первой) имеют вид $z \cdot x = z$, т. е. содержат тот же код операции умножения и те же адреса ячеек z и x. Однако выполняются они всякий раз иначе, так как содержимое ячейки z изменяется. Благодаря этому в ячейке z и появляются следующие степени числа x. Естественно попытаться написать программу так, чтобы эта команда была написана один раз, а выполнялась столько раз, сколько нужно.

Постараемся составить циклическую программу, т. е. такую, в которой команда цикла $z \cdot x = z$ будет написана только один раз, а выполняться многократно. Перепишем

сначала предыдущую программу в несколько ином виде, заменив первую команду двумя:

$$\left.\begin{array}{l} \text{«0»} + \text{«1»} = z \\ z \cdot x = z \\ z \cdot x = z \\ \ldots\ldots\ldots \\ z \cdot x = z \end{array}\right\} \; n \text{ раз}$$

стоп

Команда «0» + «1» = z служит для подготовки цикла, она заносит в ячейку результата z его начальное значение 1.

Чтобы повторить выполнение одной и той же команды несколько раз, нужно после нее поставить команду передачи управления, которая возвращала бы машину снова к выполнению той же команды. Однако безусловная передача управления здесь не годится, так как тогда управление передавалось бы всегда и нельзя было бы обеспечить выполнение команды нужное число раз.

Введем в рассмотрение специальную ячейку a («*счетчик*»), в которой будем считать, сколько раз выполняется нужная нам команда. При каждом выполнении команды будем прибавлять к содержимому ячейки a единицу. С помощью этой ячейки можно обеспечить повторение команды нужное число раз. Для этого достаточно каждый раз проверять содержимое ячейки a и передавать управление либо на повторение команды, пока a меньше n, либо на окончание, когда счетчик покажет выполнение команды нужное число раз.

Таким образом, мы приходим к следующей программе:

1)	«0» + «1» = z	5)	$a - n = 0$
2)	«0» + «0» = a	6)	*У1* 3)
3)	$z \cdot x = z$	7)	*стоп*
4)	a + «1» = a		

Проследим, как работает эта программа. Сначала выполняются команды 1), 2), составляющие подготовительную часть циклической программы. Затем первый раз выполняется команда 3), образующая рабочую часть цикла; в ячейке z оказывается x.

После выполнения команды 4) в счетчике a появляется 1; разность $a - n$ будет отрицательной, и команда 5) выра-

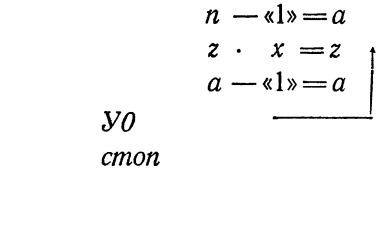
ботает сигнал $\omega = 1$; команда 6) передаст управление на команду 3), после выполнения которой в z окажется x^2.

Такое повторение цикла произойдет n раз. Действительно, после того как цикл повторится n — 1 раз, в z окажется величина x^{n-1}, а в счетчике a — число n — 1. Тогда разность $a - n$ будет отрицательной, сигнал $\omega = 1$ и команда 6) заставит машину выполнить команды 3), 4), 5) еще раз, в результате чего в ячейке z появится x^n, в a окажется n, разность $a - n$ станет равной нулю, сигнал ω первый раз станет нулем, и команда 6) передаст управление на *стоп*.

Сравним теперь циклическую и бесцикловую программы. Бесцикловая программа состоит из $n + 1$ команд, циклическая при любом n — из семи команд. Следовательно, при $n > 6$ циклическая программа короче бесцикловой. Кроме того, бесцикловую программу для произвольного n вообще невозможно написать.

Однако следует учитывать, что циклическая программа работает медленнее бесцикловой. Действительно, при работе бесцикловой программы фактически выполняется столько операций, сколько команд в программе, т. е. $n + 1$. При работе же циклической программы один раз выполняются лишь подготовительные команды 1), 2) и заключительная команда 7), основные же команды программы 3)—6) выполняются n раз. Таким образом, при работе циклической программы выполняется $4n + 3$ элементарных операции, т. е. в четыре раза больше, чем в бесцикловой программе. Это обстоятельство в тех случаях, когда важнее выгадать в быстродействии, чем в памяти, заставляет для циклических процессов составлять вместо коротких циклических длинные бесцикловые программы.

В рассматриваемой задаче можно сократить число команд в цикле, если изменять счетчик в обратном направлении, не от 0 до n, а от n — 1 до —1:



Этот вариант программы короче предыдущего на одну команду за счет того, что образование счетчика и выработка сигнала производятся одной командой a — «1» = a (в предыдущем варианте программы на это требовалось две команды — 4) и 5)). Вместе с тем эта программа будет работать быстрее предыдущей, ибо теперь для получения z требуется $3n + 3$ элементарных операций.

Пример 2.13. Запрограммируем вычисление произведения последовательных целых чисел от 1 до 15:

$$x = 15! = 1 \cdot 2 \cdot \ldots \cdot 15.$$

Процесс вычисления в этой задаче целесообразно организовать по такому плану:

а) в некоторую ячейку a заносим 0, а в ячейку x — единицу,

б) к a прибавляем 1, результат заносим в a,

в) x умножаем на a, результат заносится в x,

г) пункты б), в) повторяем 15 раз.

Бесцикловая программа имеет такой вид

$$\begin{array}{l} \text{«0»} + \text{«0»} = a \\ \text{«0»} + \text{«1»} = x \\ \left.\begin{array}{l} a + \text{«1»} = a \\ x \cdot a = x \\ \cdots\cdots \\ \cdots\cdots \\ a + \text{«1»} = a \\ x \cdot a = x \end{array}\right\} \text{15 пар команд} \\ \textit{стоп} \end{array}$$

и состоит из 33 команд. Процесс счета в рассматриваемой программе состоит из выполнения двух подготовительных команд:

1) «0» + «0» = a
2) «0» + «1» = x

и 15 пар одинаковых команд:

3) a + «1» = a
4) $x \cdot a = x$

Чтобы получить циклическую программу, следует, написав команды 3), 4) только один раз, заставить их выполняться 15 раз. Такое повторение можно осуществить при помощи двух команд:

5) $a-«15»=0$
6) *У1* 3)

Действительно, команда 5) будет вырабатывать сигнал $\omega = 1$ до тех пор, пока a не станет равным 15, поэтому команда 6) будет передавать управление на рабочую часть цикла 3), 4) 14 раз. Когда a окажется равным 15, по команде 4) в x образуется 15! Команда 5) выработает сигнал $\omega = 0$, а команда 6) передаст управление на команду *стоп*.

Таким образом, циклическая программа для нахождения x имеет вид

$«0»+«0»=a$
$«0»+«1»=x$
$a+«1»=a$
$x \cdot a=x$
$a-«15»=0$
У1
стоп

Аналогичный вид имеет программа для вычисления величины $x = n!$ при произвольном натуральном n:

$«0»+«0»=a$
$«0»+«1»=x$
$a+«1»=a$
$x \cdot a=x$
$a-n=0$
У1
стоп

Ячейка a служит в этой и предыдущей программах *счетчиком цикла*; когда рабочая часть цикла пройдена один раз, $a = 1$, два раза — $a = 2$ и т. д. Отметим, что в примере 2.13 ячейка a участвует в вычислениях, тогда как в

предыдущем примере 1.13 счетчик в вычислениях не участвовал, а был образован специально для проверки числа повторений цикла. По этой причине счетчик в последнем примере может быть назван *естественным*, тогда как в примере 1.13 его можно назвать *искусственным счетчиком*.

В обоих примерах ячейка n содержала число повторений цикла. Это число называют *эталоном цикла*. Проверка окончания цикла в наших примерах происходила путем сравнения счетчика с эталоном. Для обоих рассмотренных примеров характерно многократное повторение группы одинаковых команд, причем число повторений n известно заранее. Программа такого вычислительного процесса называется *арифметическим циклом*.

Рассмотрим еще физическую задачу, решение которой также приводит к программированию арифметического цикла.

Пример 3.13. Вдоль луча на равных расстояниях друг от друга находится 101 электрически заряженная частица. Заряды всех частиц одинаковы и положительны. Найти силу, с которой частица, находящаяся в начале луча, отталкивается остальными.

Из физики известно, что сила отталкивания двух положительных зарядов q_1 и q_2, находящихся на расстоянии $r_{1,2}$ друг от друга, определяется формулой

$$F_{1,2}=\gamma\frac{q_1q_2}{r_{1,2}^2}.$$

Поэтому силы отталкивания точки q_0 остальными точками q_1, q_2, ..., ..., q_{100} равны:

$$F_{0,1}=\gamma\frac{q_0q_1}{r_{0,1}^2},\quad F_{0,2}=\gamma\frac{q_0q_2}{r_{0,2}^2},\quad \ldots,\quad F_{0,100}=\gamma\frac{q_0q_{100}}{r_{0,100}^2}.$$

Поскольку все силы направлены по одной прямой и в одну сторону, то результирующая сила будет:

$$F_0=F_{0,1}+F_{0,2}+\ldots+F_{0,100}=\gamma q_0\left(\frac{q_1}{r_{0,1}^2}+\frac{q_2}{r_{0,2}^2}+\ldots+\frac{q_{100}}{r_{0,100}^2}\right).$$

По условию задачи все заряды и расстояния между парами соседних зарядов одинаковы. Поэтому $q_0=q_1=\ldots=q_{100}=q$, $r_{0,1}=\Delta$, $r_{0,2}=2\Delta$, ..., $r_{0,100}=100\,\Delta$ и, следовательно,

$$F=\gamma\frac{q^2}{\Delta^2}\left(\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{100^2}\right)$$

или

$$F=\gamma\,(q/\Delta)^2\,\Sigma,$$

где

$$\Sigma=\sum_{\rho=1}^{100}\frac{1}{\rho^2}.$$

Составим программу для вычисления F. Предположим, что мы уже вычислили сумму k слагаемых и в ячейке Σ лежит сумма

$$\frac{1}{1^2}+\frac{1}{2^2}+\ldots+\frac{1}{k^2}.$$

Для получения следующего значения суммы нужно выполнить команду

$$\Sigma+u=\Sigma,$$

причем в ячейку u следует предварительно положить значение $1/(k+1)^2$. По этой формуле нужно произвести вычисления 100 раз, полагая, что в начале цикла мы приняли $\Sigma = 0$. Каждый такой шаг можно сделать с помощью четырех команд

1) $k+«1»=k$	3) $u \cdot u = u$
2) $«1» : k = u$	4) $\Sigma + u = \Sigma$

Приведенная группа команд состоит из двух частей. Команда 1) служит для изменения содержимого ячейки k, которая является здесь и счетчиком, и основной рабочей ячейкой программы. Команды 2) — 4) служат *рабочей частью* цикла. Здесь вычисляется очередное слагаемое и оно прибавляется в ячейку Σ. Эту группу команд нужно повторить сто раз, что можно сделать при помощи следующих команд:

$$k-«100»=0$$
$$У1 \qquad 1)$$

Для подготовки цикла следует заслать нули в ячейки суммы Σ и счетчика k:

$$«0»+«0»=\Sigma$$
$$«0»+«0»=k$$

Собирая все эти команды вместе и добавляя команды для получения F из Σ, получим следующую программу:

1)	$«0»+«0» = \Sigma$	8)	$У1 \quad 3)$
2)	$«0»+«0» = k$	9)	$q : \Delta = u$
3)	$k+«1» = k$	10)	$\Sigma \cdot u = F$
4)	$«1» : k = u$	11)	$F \cdot u = F$
5)	$u \cdot u = u$	12)	$F \cdot \gamma = F$
6)	$\Sigma + u = \Sigma$	13)	*стоп*
7)	$k-«100»=0$		

Как видно из рассмотренных примеров, программы арифметических циклов всегда состоят из одних и тех же частей, которые соединяются между собой в определенной последовательности. Рассмотрим эти части на примере только что составленной программы.

Команды 1) и 2) засылают в рабочие ячейки программы k и Σ нули, т. е. их первоначальное содержимое. Эту группу команд называют командами *подготовки цикла*.

Благодаря наличию команд подготовки цикла выполнение программы можно повторять любое число раз, независимо от предыдущего состояния рабочих ячеек, без нового ввода программы в память. Программа работала бы так же, если опустить эти две команды, но одновременно с вводом программы ввести в ячейки k и Σ нули. Однако в этом случае мы не смогли бы повторить программу еще раз, выполняя ее сначала, так как в процессе ее работы содержимое ячеек k и Σ изменилось бы и, чтобы пустить программу еще раз, пришлось бы снова вводить ее в память.

Программу с командами 1) и 2) можно пускать любое число раз, выполнив ее до этого полностью или частично и прервав ее выполнение в любом месте. Команды 1) и 2), помещенные в начале программы, восстановят первоначальное содержимое рабочих ячеек k и Σ. Такие программы, которые можно пускать любое число раз без нового ввода, называют *самовосстанавливающимися программами* (иногда также *самовозобновляющимися*), а группу команд подготовки цикла, которые обеспечивают программе свойство самовосстановления, называют также *командами восстановления*. Из сказанного ясно, что команды восстановления или подготовки цикла *должны обязательно быть первой частью циклической программы*.

Команда 3) предназначена для изменения содержимого ячейки k, играющей роль счетчика и вместе с тем участвующей в вычислениях. С помощью этой команды мы переходим к вычислению следующего слагаемого, т. е. к выполнению следующего шага цикла. Ее называют поэтому командой *изменения*.

Дальше идет группа команд 4)—6), которые составляют *рабочую часть цикла*, т. е. основные команды, вычисляющие очередное слагаемое и прибавляющие его к сумме. Наконец, команды 7) и 8) образуют группу команд *проверки окончания цикла*. Эти команды сверяют содержимое счетчика с эталоном и, в зависимости от результатов, передают управление либо на изменение и рабочую часть для выполнения следующего шага цикла, либо на выход из цикла и дальнейшие вычисления, которые можно производить

после того, как цикл закончен. В нашей программе эти дальнейшие вычисления, не входящие в цикл, осуществляются командами 9)—13). Иногда эти команды объединяют в группу, условно называемую *досчет*.

Перечисленные части циклической программы расположены в программе так, как это показано на рис. 14. Однако эта схема не является наиболее удобной.

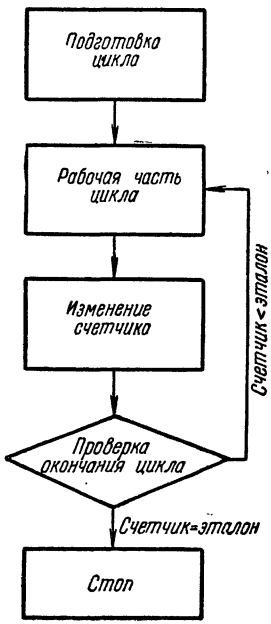


Рис. 14.

Обратим внимание на следующее обстоятельство. Эталон «100», используемый при проверке окончания цикла, совпадает с числом повторений цикла, что очень удобно. Но при подготовке цикла мы засылаем в счетчик не то значение, с которого фактически начинаются вычисления ($k = 1$), а предыдущее ($k = 0$). Это уже составляет неудобство, так как такое предшествующее значение счетчика надо еще вычислять, а это не всегда так легко, как в данном примере. Между тем при данной блок-схеме цикла иначе поступить нельзя, потому что изменение k происходит до рабочей части цикла.

Можно было бы в подготовке цикла полагать $k = 1$ и переходить сразу к рабочей части, переставив команду изменения ниже, между рабочей частью и проверкой окончания, т. е. поместив нынешнюю команду 3) между командами 6) и 7). Такая перестановка вполне естественна, но, как легко заметить, она потребовала бы изменения эталона окончания цикла.

Действительно, если *изменение* стоит после *рабочей части*, то проверка окончания происходит до того, как вычислен член с соответствующим значением k. Тогда после вычисления 99-го члена в ячейке k образуется уже число 100, и если оставить проверку окончания без изменения, то разность k — 100 будет уже равна нулю и команда 8)

передаст управление на выход из цикла, так что последний член останется невычисленным. Для верной работы программы нужно в качестве эталона брать уже не 100, а 101, т. е. не число повторений цикла, а следующее за ним, которое тоже надо еще вычислять.

Оказывается, все эти затруднения легко преодолеть, если оставить команду изменения в начале программы, но обходить ее в начале цикла таким образом, чтобы после подготовки цикла переходить сразу к выполнению рабочей части. В этом случае можно при подготовке цикла засылать в счетчик первое его значение и в качестве эталона для проверки окончания использовать число повторений цикла. Для того чтобы этого достигнуть, достаточно заменить команду 2) нашей программы такой:

$$2)\ Б\ \overline{«1»\quad 4)}\ \overrightarrow{k}$$

Блок-схема цикла при таком расположении его частей имеет вид, показанный на рис. 15. Это и есть наиболее простая и естественная схема, которая может быть рекомендована к использованию практически во всех случаях.

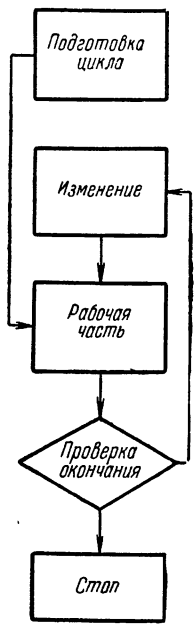


Рис. 15.

Впрочем, иногда можно, ценой некоторого отступления от этой схемы, сократить цикл на одну команду. Для этого нужно изменять счетчик в обратном направлении, как это делалось в последней программе примера 1.13. Если n — число повторений цикла, то команда подготовки цикла, приводящая счетчик в начальное состояние должна тогда выглядеть так:

$$n - «1» = счетчик,$$

а команды изменения и проверки окончания цикла могут быть совмещены и представлены в виде

$$\begin{array}{ll} счетчик - «1» = счетчик, \\ УО \qquad\qquad РЧ, \end{array}$$

где *РЧ* означает первую команду рабочей части. Такое сокращение возможно всегда с искусственным счетчиком, а с естественным лишь в том случае, когда вычисление соответствующих членов определяется только состоянием счетчика и не требует знания предыдущих. Блок-схема такого видоизмененного цикла показана на рис. 16.

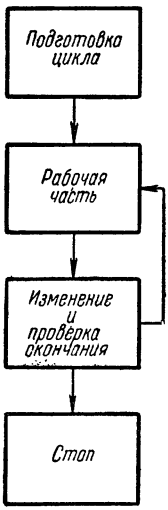


Рис. 16.

Не следует думать, что арифметические циклы должны быть организованы исключительно по счетчику. Следующий пример представляет собой арифметический цикл, организованный по иному способу.

Пример 4.13. Вычислим сумму

$$S = f(a) + f(a+h) + f(a+2h) + \ldots,$$

где $f(x) = (x^2 - 1)/(x^2 + 1)$, а сумма распространена на значения x, удовлетворяющие условиям $a \leqslant x \leqslant b$, $(b - a) / h = n$ целое.

Здесь мы имеем обычный арифметический цикл, хотя число повторений и неизвестно. Это число легко подсчитать, но в этом нет никакой нужды, так как проверку окончания цикла можно осуществить, сравнивая значения x и b*). Вычисления можно производить по следующей программе:

1)	«0» + «0» = S	7)	$v : u = u$
2)	Б a 4) x	8)	$S + u = S$
3)	$x + h = x$	9)	$x - b = 0$
4)	$x \cdot x = u$	10)	У1 3)
5)	u — «1» = v	11)	стоп
6)	u + «1» = u		

§ 14. Итерационные циклы

Во многих циклических вычислительных задачах число шагов, нужных для решения задачи, заранее определить либо невозможно, либо очень трудно.

*) При такой проверке могут возникнуть некоторые трудности. Подробнее о них будет сказано в гл. V второго выпуска.

Рассмотрим, например, как вычисляется число π. Из геометрии известно, что число π есть предел полупериметра вписанного в единичную окружность правильного многоугольника при неограниченном увеличении числа его сторон.

Для нахождения π с заданным числом верных десятичных знаков следует, последовательно увеличивая число сторон правильного многоугольника, вычислять его полупериметр. Здесь налицо циклический процесс. Когда же кончать цикл?

Заранее определить, сколько сторон нужно взять у многоугольника, чтобы получить π с нужной точностью, трудно. Однако можно поступить следующим образом: сравнивать два последовательных значения полупериметра и оканчивать цикл тогда, когда эти значения совпадут с заданной точностью.

Такие циклы, в которых *заранее неизвестно* число повторений и проверка окончания происходит не по счетчику, а по достижению нужной точности, называют итерационными. Очевидно, никаких изменений в структуре цикла при переходе от арифметических циклов к итерационным не происходит. Итерационный цикл состоит из тех же частей и строится по той же блок-схеме, что и арифметический.

Пример 1.14. Составить программу вычисления кубического корня из x:

$$y=\sqrt[3]{x}.$$

Пусть нам известно приближенное значение $y = y_0$. Постараемся найти, исходя из y_0, более точное значение кубического корня.

Прибавим к y_0 неизвестную поправку h и положим

$$y_1=y_0+h=\sqrt[3]{x}.$$

Возведя обе части этого равенства в куб, получим:

$$y_0^3+3y_0^2h+3y_0h^2+h^3=x.$$

Считая поправку h к приближенному значению y_0 малой, опустим в последнем равенстве величины h^2 и h^3. Тогда получим:

$$y_0^3+3y_0^2h\approx x.$$

Отсюда $h \approx x/(3y_0^2) - y_0/3$ и, следовательно,

$$y_1=y_0+h=\frac{x}{3y_0^2}+\frac{2y_0}{3}, \text{ или } y_1=\frac{1}{3}\left(\frac{x}{y_0^2}+2y_0\right).$$

Точно так же можно найти следующее приближение

$$y_2=\frac{1}{3}\left(\frac{x}{y_1^2}+2y_1\right).$$

Последовательные приближения для $\sqrt[3]{x}$ могут строиться по следующей формуле:

$$y_{n+1}=\frac{1}{3}\left(\frac{x}{y_n^2}+2y_n\right). \qquad (1.14)$$

Можно доказать *), что этот процесс последовательных приближений сходится к $\sqrt[3]{x}$ при любом начальном приближении $y_0>0$.

Однако процесс последовательных приближений является бесконечным. Для того чтобы его прервать, будем искать $\sqrt[3]{x}$ с заданной точностью, причем примем, что эта точность достигается при том значении n, при котором

$$|y_{n+1}-y_n|<\varepsilon. \qquad (2.14)$$

За начальное значение y_0 для простоты примем x:

$$y_0=x.$$

Составление программы начнем с подготовки цикла:

1) $0+x=y_n$.

Составим по формуле (1.14) рабочую часть цикла:

2) $y_n \cdot y_n=R_1$ 5) $R_1+ y_n =R_1$
3) $x : R_1=R_1$ 6) $R_1 \cdot$ «1/3» $=y_{n+1}$
4) $R_1+y_n=R_1$

Проверка окончания цикла осуществляется в соответствии с формулой (2.14):

7) $y_{n+1}-y_n=R_1$
8) $|R_1|-|\varepsilon|=0$
9) *У1* *стоп*

Если условие (2.14) не выполняется, то следует переслать y_{n+1} в y_n и перейти к началу цикла

10) «0» $+y_{n+1}=y_n$
11) *Б* 2)

*) См. § 7 второго выпуска.

И, наконец, окончание программы:

12) «0» $+ y_{n+1} = y$
13) *стоп*

Объединяя отдельные составные части программы, сократим ее на три команды; совмещая ячейки y_{n+1} и y, объединяя команды 9), 10) и опуская команды 11) и 12), получим

1)	$0 + x = y_n$	6)		$R_1 + R_2 = y$
2)	$y_n \cdot y_n = R_1$	7)		$y - y_n = R_1$
3)	$x : R_1 = R_1$	8)		$\lvert R_1 \rvert - \lvert \varepsilon \rvert = 0$
4)	$R_1 \cdot$ «1/3» $= R_1$	9)	*УО*	y 2) y_n
5)	$y_n \cdot$ «2/3» $= R_2$	10)	*стоп*	

Пример 2.14. Найти сумму бесконечного ряда

$$S = 1 - \frac{1}{1 \cdot 3} + \frac{1}{1 \cdot 3 \cdot 5} - \ldots + (-1)^n \frac{1}{1 \cdot 3 \cdot \ldots \cdot (2n+1)} + \ldots$$

с точностью до заданного ε.

Так как члены ряда монотонно убывают по абсолютной величине и изменяют знаки, то ряд сходится и ошибка от замены суммы ряда его частичной суммой не превосходит абсолютной величины первого из отброшенных членов. Это дает возможность организовать проверку окончания цикла. Если член ряда u_n с последним множителем $2n - 1$ в знаменателе уже вычислен, то следующий член можно получить по формуле

$$u_{n+1} = -u_n/(n+2),$$

причем $u_0 = 1$.

После сделанных замечаний напишем программу вычисления S:

1)		«0» $+$ «1» $= u$	6)		$u : n = u$
2)		«0» $+$ «1» $= n$	7)		$S + u = S$
3)	*Б*	«0» 6) S	8)		$\lvert u \rvert - \lvert \varepsilon \rvert = 0$
4)		$n +$ «2» $= n$	9)	*УО*	4)
5)		«0» $- u = u$	10)	*стоп*	

Здесь команды 1)—3) — команды подготовки цикла, команды 4)—5) — изменение, команды 6)—7) — рабочая часть и, наконец, команды 8)—9) — проверка окончания.

Для задач, решаемых итерационным методом, число шагов, которое нужно сделать для нахождения решения с заданной точностью, до решения задачи, как правило, определить невозможно. Поэтому для таких задач (в отличие от задач, приводящих к арифметическому циклу) невозможно написать программу без цикла.

ГЛАВА IV

ПЕРЕВОД ПРОГРАММЫ НА ЯЗЫК МАШИНЫ

§ 15. Системы счисления

Конструкция вычислительных машин и программирование на них тесно связаны с *системой счисления*. Так называют в математике *способы наименования и записи чисел.*

Все известные системы счисления можно разделить на две большие группы: *позиционные и непозиционные системы.* В позиционной системе одна и та же цифра может означать различные числа, в зависимости от места (*позиции*), где она стоит.

Хорошо известным примером непозиционной системы счисления является пришедшая к нам из Древнего Рима римская система, в которой для записи чисел используются буквы латинского алфавита. При этом буква I всегда означает единицу, буква V — пять, X — десять, L — пятьдесят, C — сто, D — пятьсот, M — тысячу и т. д. Число 267 запишется в римской системе в виде: CCLXVII. Каждая используемая буква в римской системе всегда означает одно и то же число. Поэтому для записи больших чисел введенных знаков будет нехватать и нужны будут новые, и сколько бы мы их ни ввели, всегда можно придумать число, которое уже введенными знаками изобразить трудно, как, например, трудно изобразить тысячу, имея лишь знаки I, V, X.

Совсем иначе обстоит дело в *позиционных* системах. Здесь имеется определенное количество знаков (цифр), каждый из которых может означать различные числа в зависимости от места (*позиции*), которое этот знак занимает.

Общепринятой системой счисления является десятичная позиционная система. Она была изобретена в Индии, затем

заимствована оттуда арабами и уже через арабские страны пришла в Европу. В этой системе для записи любого числа используется лишь десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Значение каждой цифры в позиционной системе счисления определяется как ею самой, так и местом, которое она занимает. Так, единица изображается цифрой 1, десять — той же цифрой, стоящей на втором месте справа, т. е. 10, сто — той же цифрой 1, но уже стоящей на третьем месте, и т. д.

Таким образом, каждое число разбивается на разряды, которые считаются справа налево, и единица каждого следующего разряда в определенное число раз превосходит единицу предыдущего. Отношение соседних разрядов называют основанием системы счисления. Принятая система счисления потому и называется десятичной, что ее основанием является число десять: каждый следующий разряд в десять раз больше предыдущего.

В разные исторические периоды человек использовал позиционные системы счисления, отличные от десятичной. Так, в Древнем Китае, а также у ряда африканских народов была распространена пятеричная система, а у населявших американский континент народностей — ацтеков и майя — двадцатеричная система. Эти системы, так же как и десятичная, имеют «анатомическое» происхождение — они произошли от счета по пальцам рук (или рук и ног).

Кроме пятеричной, десятичной и двадцатеричной систем, истории цивилизации известны и другие системы счисления.

В Древнем Вавилоне, например, применялась шестидесятеричная система счисления. Остатки ее мы находим в сохранившемся до наших дней обыкновении делить час или градус на 60 минут, минуту на 60 секунд, а круг на 360, т. е. шесть раз по 60 градусов.

Шестидесятеричная система возникла около двух тысячелетий до нашей эры при слиянии в одно государство двух древних народов — шумерийцев и аккадян. Во вновь образованном государстве остались в ходу единицы веса, используемые ранее тем и другим народом, причем одна из этих единиц была приблизительно в шестьдесят раз больше другой. Употреблялась также и двенадцатеричная система, следами которой является сохранившийся обычай

считать некоторые предметы дюжинами, деление года на 12 месяцев.

Для изображения числа в позиционной системе счисления требуется столько различных цифр, каково основание системы. Так, в десятичной системе для записи числа употребляется десять цифр, в пятеричной системе достаточно пяти цифр: 0, 1, 2, 3, 4. Основание системы — число *пять* — изобразится здесь как 10, поскольку оно является единицей следующего (второго) разряда. Легко понять, что так будет в любой системе счисления: *основание системы счисления в любой системе записывается как* 10.

В шестнадцатеричной системе имеющихся десяти цифр для изображения числа нехватает. Требуется ввести еще шесть цифр для изображения чисел, равных десяти, одиннадцати, двенадцати, тринадцати, четырнадцати, пятнадцати, которые в шестнадцатеричной системе являются однозначными. Вводя для этих чисел обозначения $\bar{0}$, $\bar{1}$, $\bar{2}$, $\bar{3}$, $\bar{4}$, $\bar{5}$, получим 16 цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, $\bar{0}$, $\bar{1}$, $\bar{2}$, $\bar{3}$, $\bar{4}$, $\bar{5}$. Число шестнадцать как основание системы будет иметь вид 10.

Чтобы различить, в какой системе счисления записано число, мы будем в дальнейшем, если это не ясно из текста, указывать основание системы счисления в виде индекса (при этом индекс всегда пишется в десятичной системе!). Например, число 173_{10} записано в обычной десятичной системе, а число 173_8 — в восьмеричной.

Запись целого числа в какой-либо системе счисления означает представление этого числа в виде суммы степеней основания с различными коэффициентами, меньшими основания. Эти коэффициенты и являются цифрами в записи числа. Например, запись числа 9078 в десятичной системе означает $9078 = 9 \cdot 10^3 + 0 \cdot 10^2 + 7 \cdot 10^1 + 8 \cdot 10^0$, а шестнадцатеричное число $\bar{0}52\bar{0}\bar{3}$ равно $10 \cdot 16^4 + 5 \cdot 16^3 + 2 \cdot 16^2 + 0 \cdot 16^1 + 13 \cdot 16^0 = 676365_{10}$.

До сих пор речь шла только о целых числах. Аналогично можно обращаться и с дробями. Основным способом записи дробей при вычислениях являются десятичные дроби; десятичная дробь есть представление дробного числа в виде суммы отрицательных степеней десяти.

Так, например, запись 0,30917 означает в десятичной системе счисления число

$$3\cdot 10^{-1}+0\cdot 10^{-2}+9\cdot 10^{-3}+1\cdot 10^{-4}+7\cdot 10^{-5}.$$

В пятеричной системе счисления запись 0,324 означает число

$$3\cdot 5^{-1}+2\cdot 5^{-2}+4\cdot 5^{-3}.$$

В десятичной системе это число равно 89/125 = 0,712.

Известно, что не всякая рациональная дробь может быть выражена конечной десятичной дробью. Такое же явление наблюдается и в других системах счисления. При этом может случиться, что рациональная дробь выражается конечной дробью в одной системе и бесконечной (периодической) в другой. Например, число 2/7 выражается в десятичной системе бесконечной дробью 0,2857142857142 ..., а в семеричной — конечной дробью 0,2. Наоборот, 1/5 в десятичной системе выражается конечной дробью 0,2, а в шестеричной — бесконечной дробью 0,111...

В системе счисления с основанием n любое число M записывается в виде

$$M=a_k a_{k-1}\dots a_1 a_0, a_{-1}a_{-2}a_{-3}\dots a_{-s}.$$

Здесь

$a_k a_{k-1},\dots,a_1 a_0$ — целая часть числа,

$a_{-1}a_{-2},\dots,a_{-s}$ — его дробная часть,

$a_k a_{k-1},\dots,a_{-s}$ — цифры, принимающие значения от 0 до $n-1$.

Развернутая запись числа имеет вид:

$$M_n=a_k n^k+a_{k-1}n^{k-1}+\dots+a_1 n^1+a_0 n^0+a_{-1}n^{-1}+ \\ +a_2 n^{-2}+\dots+a_{-s}n^{-s}.$$

Этой записью можно воспользоваться для перевода числа, записанного в системе счисления с основанием n, в десятичную систему.

Для такого перевода достаточно все цифры a_k, a_{k-1} ,...,a_{-s} и основание n числа M_n записать в десятичной системе и затем произвести (в десятичной системе) вычисление правой части последнего равенства. Такую операцию называют *подстановкой*.

Пример 1.15. Переведем число $401{,}324_5$ в десятичную систему: $401{,}324_5 = (4\cdot5^2 + 0\cdot5^1 + 1\cdot5^0 + 3\cdot5^{-1} + 2\cdot5^{-2} + 4\cdot5^{-3})_{10} = 101{,}712_{10}$.

Здесь основание системы счисления меньше 10, поэтому цифры числа и n при подстановке не изменились. Если же основание системы n больше 10, то n и каждую из цифр числа следует перед подстановкой перевести в десятичную систему. Так, в двенадцатеричной системе, помимо «обычных» десятичных цифр 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, используются еще две цифры $\overline{0}$, $\overline{1}$, которые в десятичной системе равны 10, 11, а основание n этой системы в десятичной системе соответствует 12.

Пример 2.15. $2\overline{0}7\overline{1},39_{12} = (2\cdot12^3 + 10\cdot12^2 + 7\cdot12^1 + 11\cdot12^0 + 3\cdot12^{-1} + 9\cdot12^{-2})_{10} = 4991{,}3125_{10}$.

Производить арифметические действия в непозиционной системе очень неудобно и сложно. В этом легко убедиться, если попробовать, например, сложить числа CCCLIX и CLXXIV или разделить числа CXVIII и LXIV, пользуясь лишь римской системой счисления. Поэтому непозиционная римская система счисления употребляется теперь очень редко и в тех случаях, когда над числами не требуется совершать действий: для нумерации веков в хронологии, глав в книгах, часов на циферблате и т. д.

Арифметические операции в позиционной системе счисления производятся по простым правилам. Здесь дело сводится к соответствующим действиям над однозначными числами в различных разрядах и переносу из младших разрядов в старшие (или наоборот) по определенным правилам.

§ 16. Двоичная арифметика

До создания быстродействующих вычислительных машин все вычислительные приборы и устройства создавались в основном на базе десятичной системы счисления. Для механических элементов, из которых состояли эти устройства, выбор системы счисления не очень важен: для пятеричной системы, например, можно надеть на спицу пять косточек (так и сделано на китайских счетах «суанпан»), а для двенадцатеричной — разделить цифровое колесо на 12 частей по 30°, вместо 10 делений по 36°. Вместе

с тем, поскольку при применении этих приборов вычислительный процесс не был полностью автоматизирован, естественно было применять в них употребляющуюся при ручном счете десятичную систему.

Первые «механические» вычислительные машины, основанные на десятичной системе, оказались очень сложными, неэкономичными и медленными.

При переходе к другим физическим принципам построения машин оказалось удобным изображать каждый разряд числа одним элементом с несколькими устойчивыми состояниями. При этом обнаружилось, что для большинства таких элементов характерно наличие двух устойчивых состояний — электромеханическое реле может быть замкнуто или разомкнуто, конденсатор может быть заряжен или разряжен, электронная лампа может проводить или не проводить ток.

Посмотрим, какая из позиционных систем счисления выгоднее всего с точки зрения экономии оборудования, применяемого для изображения чисел в машине или вычислительном приборе. Каждый разряд числа в k-ичной системе может изображаться в машине или приборе либо k элементами, либо k устойчивыми состояниями одного элемента (первый способ применяется, например, в русских счетах — десять косточек на разряд, второй — в арифмометре: в каждом разряде 10 положений цифрового колеса).

Пусть, для определенности, целые числа изображаются первым способом при помощи 30 элементов. В зависимости от выбора системы счисления, мы будем иметь возможность изображать этими элементами числа с различным количеством разрядов. Так, в десятичной системе на каждый разряд расходуется по десять элементов, а потому 30 элементов хватит для трех разрядов; в шестеричной каждый разряд потребует по шесть элементов, и мы сможем тридцатью элементами изобразить пятиразрядное шестеричное число и т. д. Тогда в приборе, работающем в десятичной системе, при помощи 30 элементов можно представить целые числа от 0 до 999, в шестеричной системе — числа от 0 до $6^5 - 1 = 7775$, в пятеричной системе от 0 до $5^6 - 1 = 15624$, в троичной системе от 0 до $3^{10} - 1 = 59048$ и наконец, в двоичной — от 0 до $2^{15} - 1 = 32767$.

Таким образом, при помощи одного и того же оборудования можно представить числа наиболее широкого диапазона в троичной системе; несколько менее широкого — в двоичной.

Этот вывод, полученный нами на примере, остается справедливым и в общем случае. Он остается верным и при втором способе изображения чисел, ибо изготовить элемент с k устойчивыми состояниями зачастую не менее трудно, чем k отдельных элементов. Поэтому, учитывая также трудности создания надежных физических элементов с тремя устойчивыми состояниями *), в большинстве современных быстродействующих вычислительных машин стали применять изображение чисел в двоичной системе счисления, т. е. в позиционной системе счисления с основанием два.

Для изображения чисел в этой системе требуется лишь две цифры — 0 и 1. Благодаря этому числа можно изображать с помощью элементов, имеющих два различных устойчивых состояния: одно из них принимается за изображение нуля, другое — за изображение единицы.

Основание двоичной системы «два» изображается в этой системе как 10. Прибавляя к этому числу единицу, мы получим 11, т. е. двоичную запись числа 3; прибавляя еще единицу, мы должны сделать перенос в двоичные «десятки», а затем в «сотни», что даст 100 — двоичное изображение числа 4.

Первые числа натурального ряда выражаются в двоичной системе следующим образом:

Десятичные числа	Двоичные числа	Десятичные числа	Двоичные числа
1	1	11	1011
2	10	12	1100
3	11	13	1101
4	100	14	1110
5	101	15	1111
6	110	16	10000
7	111	17	10001
8	1000	18	10010
9	1001	19	10011
10	1010	20	10100

*) У нас в стране на троичной системе счисления основана лишь одна электронная вычислительная машина «Сетунь», сконструированная в Московском университете.

Отсюда видно, что запись числа в двоичной системе значительно длиннее десятичной записи; так, двухразрядное десятичное число 19 записывается пятью двоичными разрядами, одноразрядное число 7 — тремя двоичными разрядами. Легко убедиться, что для записи произвольного целого числа в двоичной системе требуется примерно в три с половиной раза больше двоичных разрядов, чем в десятичной системе — десятичных. Поэтому при обычном ручном счете десятичная система удобнее двоичной.

Ценным качеством двоичной системы является чрезвычайная простота арифметических действий. Они выполняются по тем же правилам, что и в десятичной системе. Например, при сложении складываются соответствующие разряды, начиная с младших. Если в данном разряде образуется сумма, уже не умещающаяся в нем, то соответствующее превышение переносится в следующий старший разряд. Таким образом, фактически приходится использовать таблицу сложения для однозначных чисел, которая в двоичной системе имеет следующий вид:

$$\begin{aligned} 0+0&=0,\\ 0+1&=1,\\ 1+0&=1,\\ 1+1&=10. \end{aligned}$$

Сложение двух многозначных двоичных чисел выглядит так:

$$\begin{array}{r} 10011{,}0101 \\ +\ 1101{,}0111 \\ \hline 100000{,}1100. \end{array}$$

Таблица умножения в двоичной системе предельно проста. Поскольку умножение на нуль всегда дает нуль, можно считать, что здесь таблица умножения состоит лишь из одной строки

$$1\cdot 1=1.$$

Но умножение на единицу не меняет числа. Поэтому умножение многозначных чисел сводится в двоичной системе лишь к сдвигу и сложению.

Если в множителе несколько единиц, то между множителями и частичными произведениями полезно оставлять свободное место для записи единиц переноса.

Приведем пример на умножение:

$$\begin{array}{r} 11011 \\ \times \quad 100{,}1 \\ \hline 11011 \\ +11011 \\ \hline 1111001{,}1. \end{array}$$

Так же просто выполняются и обратные действия — вычитание и деление. Например,

$$\begin{array}{r} 1101001101 \\ -\ 11100110 \\ \hline 1001100111, \end{array}$$

$$\begin{array}{r|l} 10101110001111 & 11011 \\ \cline{2-2} -\ 11011 & 110011101. \\ \cline{1-1} 100001 & \\ -\ 11011 & \\ \cline{1-1} 110000 & \\ -\ 11011 & \\ \cline{1-1} 101011 & \\ -\ 11011 & \\ \cline{1-1} 100001 & \\ -\ 11011 & \\ \cline{1-1} 11011 & \\ -\ 11011 & \\ \cline{1-1} \end{array}$$

Приведенные примеры хорошо выясняют преимущества и недостатки двоичной системы счисления. С одной стороны, арифметические действия в двоичной системе очень просты; для представления чисел в машине двоичная система требует меньшего числа элементов, чем любая другая, кроме троичной. С другой стороны, запись чисел в двоичной системе очень громоздка и однообразна: для записи чисел на бумаге в двоичной системе требуется в три-четыре раза

больше разрядов, чем в десятичной, и число представляется в виде набора из одних только нулей и единиц.

Преимущества двоичной системы находят применение в конструкции электронных вычислительных машин. О том, как преодолеть ее недостатки, мы сообщим в следующих параграфах этой главы.

§ 17. Переход от одной системы счисления к другой. Смешанные системы

Кроме двоичной системы, при работе на вычислительных машинах используются и другие системы счисления. Поэтому нужно уметь переводить числа из одной системы в другую.

С одним способом такого перевода, называемым подстановкой, мы уже познакомились в предыдущем параграфе. По этому способу из числа в произвольной k-ичной системе можно было получить то же число в *десятичной* системе, при этом очень существенным является то обстоятельство, что все действия при подстановке производились в десятичной системе.

Способ подстановки годится и для перехода от любой системы с основанием n к системе с основанием a, если только все действия производить в системе с основанием a. При вычислениях, производимых в десятичной системе (или в системе с основанием a), для перехода от десятичной системы (или от системы с основанием a) к системе с основанием n способ подстановки непригоден. Здесь приходится пользоваться несколько иным приемом, с которым мы познакомимся сначала на примере.

Пример 1.17. Переведем в восьмеричную систему целое число 613_{10}. Разделив данное число на 8, найдем в частном 76 и в остатке 5:

$$613_{10} = 76 \cdot 8 + 5.$$

Это означает, что наше число, кроме некоторого количества восьмерок, содержит еще пять единиц, т. е. последняя цифра восьмеричной записи данного числа есть 5.

Для определения следующей (справа налево) цифры разделим на 8 полученное частное 76. Найдем, что частное

равно 9, а остаток 4, т. е. $76 = 9 \cdot 8 + 4$; иначе говоря, получаем

$$613_{10} = (9 \cdot 8 + 4) \cdot 8 + 5 = 9 \cdot 8^2 + 4 \cdot 8 + 5.$$

Таким образом, остаток 4 даст нам вторую справа цифру восьмеричного представления. Наконец, разделив новое частное 9 на 8, получим $9 = 8 \cdot 1 + 1$.

Все вычисления удобно объединить в следующую схему:

$$\begin{array}{r|l} 613 & 8 \\ \hline 53 & 76 \\ \hline 5 & 4 \end{array} \qquad \begin{array}{r|l} 8 & 8 \\ \hline 9 & \\ \hline 1 & \end{array} \qquad \begin{array}{r|l} 8 & \\ \hline 1 & \end{array}$$

Результат деления можно представить в виде:

$$613 = 76 \cdot 8 + 5 = (9 \cdot 8 + 4) \cdot 8 + 5 = ((8 \cdot 1 + 1) \cdot 8 + 4) \cdot 8 + 5 = \\ = 1 \cdot 8^3 + 1 \cdot 8^2 + 4 \cdot 8^1 + 5 \cdot 8^0.$$

Последнее означает, что число 613_{10} имеет в восьмеричной системе вид 1145_8.

Для проверки результата переведем число 1145_8 в десятичную систему способом подстановки:

$$1145_8 = (1 \cdot 8^3 + 1 \cdot 8^2 + 4 \cdot 8^1 + 5 \cdot 8^0)_{10} = \\ = (512 + 64 + 32 + 5)_{10} = 613_{10}.$$

Легко догадаться, что описанный прием является общим. Действительно, пусть целое число U записано в некоторой системе счисления с основанием a и его нужно перевести в новую систему счисления с основанием n.

Разделив U на n, получим:

$$U = b_0 n + a_0$$

(деление производится в системе счисления с основанием a). Разделив далее b_0 на n, найдем, что $b_0 = b_1 n + a_1$, откуда

$$U = (b_1 n + a_1)\, n + a_0 = b_1 n^2 + a_1 n + a_0.$$

Этот процесс будем продолжать до тех пор, пока последнее частное, которое мы обозначим через a_k, не станет меньше n.

Тогда получим

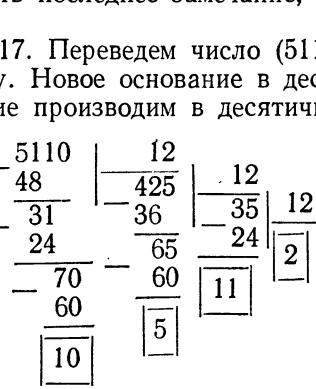
$$U = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_2 n^2 + a_1 n + a_0,$$

т. е. цифрами, представляющими число U в системе с основанием n, будут остатки, получающиеся при последовательном делении U на n, но записанные в обратном порядке. Этот прием перевода целых чисел назовем *алгоритмом последовательного деления.*

Деление в этом алгоритме производится в первоначальной системе счисления с основанием a. Если $n < a$, то делитель, а значит, и все остатки однозначны, и мы сразу получаем нужные цифры. Если же $n > a$, то в системе с основанием a делитель и, быть может, некоторые остатки будут содержать больше одной цифры. В новой системе счисления их следует заменить новыми цифрами, которые в системе с основанием a отсутствовали.

Чтобы пояснить последнее замечание, рассмотрим пример.

Пример 2.17. Переведем число $(5110)_{10}$ в двенадцатеричную систему. Новое основание в десятичной системе равно 12. Деление производим в десятичной системе:



Замечая, что $10_{10} = \bar{0}_{12}$, $11_{10} = \bar{1}_{12}$, получим $(5110)_{10} = (2\bar{1}5\bar{0})_{12}$.

Для контроля проделаем обратный перевод:

$$(2\bar{1}5\bar{0})_{12} = (2 \cdot 12^3 + 11 \cdot 12^2 + 5 \cdot 12 + 10)_{10} = 5110_{10}.$$

Для перевода правильных дробей вместо алгоритма последовательного деления используют так называемый *алгоритм последовательного умножения*, который мы также рассмотрим сначала на примере.

Пример 3.17. Пусть нужно перевести в двоичную систему десятичную дробь 0,4140625.

В двоичной системе счисления первая цифра после запятой означает половину. Поэтому она будет единицей или нулем, смотря по тому, будет ли данное число больше или меньше половины. Это легко установить, проверив, будет ли единицей или нулем целая часть числа после его удвоения. Таким же образом можно получить следующие двоичные цифры путем дальнейшего удвоения.

Производимые вычисления удобно записать подряд в виде следующей схемы:

0,	$\times \begin{matrix} 4140625 \\ 2 \end{matrix}$
0,	$\times \begin{matrix} 8281250 \\ 2 \end{matrix}$
1,	$\times \begin{matrix} 6562500 \\ 2 \end{matrix}$
1,	$\times \begin{matrix} 3125000 \\ 2 \end{matrix}$
0,	$\times \begin{matrix} 6250000 \\ 2 \end{matrix}$
1,	$\times \begin{matrix} 2500000 \\ 2 \end{matrix}$
0,	$\times \begin{matrix} 5000000 \\ 2 \end{matrix}$
1,	0000000

причем следует помнить, что удваивается всякий раз только дробная часть числа. Из приведенных вычислений видно, что $0{,}4140625_{10} = 0{,}0110101_2$.

Покажем, что этот алгоритм является общим. Пусть дано число

$$U = b_{-1}n^{-1} + b_{-2}n^{-2} + b_{-3}n^{-3} + b_{-4}n^{-4} + \ldots$$

Умножая это число на n, получим:

$$nU = b_{-1} + (b_{-2}n^{-1} + b_{-3}n^{-2} + \ldots).$$

Таким образом, целая часть произведения дает первую цифру дроби b_{-1}. Дробную часть $(b_{-2}n^{-1} + b_{-3}n^{-2} + \ldots)$ для получения следующей цифры следует умножить на n:

$$n\,(b_{-2}n^{-1} + b_{-3}n^{-2} + b_{-4}n^{-3} + \ldots) = b_{-2} + (b_{-3}n^{-1} + b_{-4}n^{-2} + \ldots)$$

и взять целую часть результата b_{-2} и т. д. Заметим, что в некоторых случаях этот процесс последовательного умножения может никогда не закончиться — это означает, что результатом является бесконечная дробь.

Если число является смешанным, т. е. его целая и дробная части отличны от нуля, то их следует переводить в другую систему счисления отдельно, каждое в соответствии с рассмотренными выше правилами.

Таким образом, *если действия над числами производятся в системе с основанием n, то для перевода числа из системы с основанием s в систему с основанием n используется алгоритм подстановки. Обратный перевод производится при помощи алгоритмов последовательного деления* (для целой части) *и последовательного умножения* (для дробной части).

При переходе от одной системы счисления к другой оказываются полезными смешанные формы записи чисел.

Если представить число в какой-либо системе счисления, а затем каждую цифру этого числа записать в другой системе, то мы получим смешанную форму записи числа. Практически используются двоично-десятичная и двоично-восьмеричная системы.

В двоично-десятичной системе число представляется в десятичной форме, а затем каждая десятичная цифра записывается в двоичной системе. При этом различные десятичные цифры требуют для своего двоичного написания различного числа двоичных разрядов от одного (для нуля и единицы) до четырех (для восьми и девяти). Чтобы не применять никаких разделительных знаков, для двоичного изображения десятичной цифры всегда выделяется четыре двоичных разряда. Такая группа из четырех двоичных разрядов, предназначенная для изображения одной десятичной цифры, называется *тетрадой*.

Двоично-десятичную форму числа легко получить, заменив каждую цифру десятичного числа соответствующей тетрадой. Так, например, число 9807 в двоично-десятичной

форме будет иметь вид 1001 1000 0000 0111. Для удобства чтения мы записали здесь тетрады с промежутками между ними. На самом деле все цифры могут быть поставлены рядом и надо только помнить, что каждая группа состоит из четырех разрядов. Например, двоично-десятичная запись 01011001011000000010 означает десятичное число 59602.

Из возможных шестнадцати различных тетрад 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 в двоично-десятичной системе используются только первые десять, остальные тетрады не означают никакой десятичной цифры и поэтому не имеют смысла в двоично-десятичной системе.

Как уже было сказано, элементы в вычислительной машине имеют два устойчивых состояния и поэтому не могут быть использованы для непосредственного изображения чисел в системе, требующей более двух цифр. Вместе с тем, исходные данные для вычислений естественно представляются в десятичной системе. Для записи десятичных чисел в машине и служит двоично-десятичная система.

Аналогично двоично-десятичной, двоично-восьмеричная форма записи числа состоит в том, что цифра восьмеричного числа записывается в двоичном виде.

Так как наибольшая цифра в восьмеричной системе есть 7, то для изображения любой восьмеричной цифры достаточно трех разрядов. Такая группа из трех разрядов, отведенная для изображения восьмеричной цифры, называется *триадой*.

Так, например, восьмеричное число 6124573 запишется в двоично-десятичной системе семью триадами:

110 001 010 100 101 111 011.

В отличие от тетрад, используемых в двоично-десятичной системе, в двоично-восьмеричной системе все возможные триады используются. Это обстоятельство составляет преимущество двоично-восьмеричной системы. Другим — и самым важным — преимуществом является то, что *двоично-восьмеричная запись числа совпадает с его двоичной записью.*

Убедимся сначала в справедливости этого утверждения на примере.

Пример 4.17. Пусть дано число 3804_{10}. Переведя его в двоичную систему с помощью алгоритма последовательного деления, найдем двоичную форму 111011011100_2. При переводе этого числа в восьмеричную систему получим:

$$\begin{array}{r|l|l|l}
3804 & 8 & & \\
\hline
32 & 475 & 8 & \\
\hline
60 & 40 & 59 & 8 \\
\hline
56 & 75 & 3 & 7 \\
\hline
44 & 72 & \cup & \cup \\
\hline
40 & 3 & & \\
\hline
4 & \cup & & \\
\hline
\cup & & &
\end{array}$$

т. е. 7334_8. Записав это восьмеричное число в двоично-восьмеричной форме, будем иметь 111011011100, что совпадает с полученным выше двоичным представлением.

Докажем это утверждение в общем виде. Для определенности ограничимся рассмотрением двоичного числа с шестью разрядами до и тремя после запятой. Пусть оно имеет вид:

$$N = b_5 b_4 b_3 b_2 b_1 b_0, b_{-1} b_{-2} b_{-3}.$$

В двоичной системе это число равно:

$$N = (b_5 \cdot 2^5 + b_4 \cdot 2^4 + b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} \cdot 2^{-2} + b_{-3} \cdot 2^{-3}).$$

Представим это число в виде

$$N = (b_5 \cdot 2^2 + b_4 \cdot 2^1 + b_3 \cdot 2^0) \cdot 2^3 + (b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0) + (b_{-1} \cdot 2^2 + b_{-2} \cdot 2^1 + b_{-3} \cdot 2^0) 2^{-3} = a_1 \cdot 8^1 + a_0 \cdot 8^0 + a_{-1} \cdot 8^{-1},$$

где

$$\begin{aligned} a_1 &= b_5 \cdot 2^2 + b_4 \cdot 2^1 + b_3 \cdot 2^0, \\ a_0 &= b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0, \\ a_{-1} &= b_{-1} \cdot 2^2 + b_{-2} \cdot 2^1 + b_{-3} \cdot 2^0. \end{aligned}$$

Из представления

$$N = a_1 \cdot 8^1 + a_0 \cdot 8^0 + a_{-1} \cdot 8^{-1}$$

следует, что восьмеричными цифрами числа N являются a_1, a_0, a_{-1}.

Вместе с тем равенство

$$a_1 = b_5 \cdot 2^2 + b_4 \cdot 2^1 + b_3 \cdot 2^0$$

показывает, что если записать восьмеричную цифру a_1 в виде триады двоичных цифр, то этими двоичными цифрами будут b_5, b_4, b_3. Поскольку аналогичные равенства имеют место также и для a_0, a_{-1}, тем самым доказано, что двоичная запись числа совпадает с двоично-восьмеричной записью.

Доказанное утверждение объясняет применение восьмеричной системы при программировании.

Элементы машины имеют два устойчивых состояния, и число в машине изображается в двоичной системе. Однако количество разрядов в двоичном числе обычно бывает довольно велико, а программисту иметь дело с длинным набором одних только нулей и единиц затруднительно.

Поэтому для сокращения записи двоичных чисел в программировании используют следующий способ. Двоичное число разбивают влево и вправо от запятой на тройки двоичных разрядов — триады, а затем каждую триаду заменяют соответствующей восьмеричной цифрой. От этого запись числа становится втрое короче, в результате чего облегчается переписывание и уменьшается вероятность описок и просчетов. В силу доказанного утверждения формально преобразованное таким образом число является восьмеричным представлением исходного двоичного числа.

Восьмеричную систему удобно использовать при переводе вручную чисел из десятичной системы в двоичную. Для этого сначала переводят десятичное число в восьмеричную систему, а затем каждую восьмеричную цифру заменяют соответствующей триадой. Аналогично можно производить и обратный перевод — из двоичной системы в десятичную. В этом случае следует, разбив двоичное число на триады, получить его восьмеричное изображение, а затем восьмеричное число перевести в десятичное.

Пример 5.17. Переведем в двоичную систему число $846{,}75_{10}$. Произведем перевод в восьмеричную систему отдельно целой и дробной части числа:

$$\begin{array}{r|l} 846 & 8 \\ \hline 6 & 105 \quad \big| \; 8 \\ \hline & 1 \quad\;\; 13 \quad \big| \; 8 \\ \hline & \quad\;\; 5 \quad\;\; 1 \end{array} \qquad \begin{array}{r|l} 0, & 75 \\ & \;\; 8 \\ \hline 6, & 00 \end{array}$$

Таким образом, $846{,}75_{10} = 1516{,}6_8$. Отсюда $846{,}75_{10} = 1101001110{,}11_2$. Сделаем обратный перевод:

$$1101001110{,}11_2 = 1516{,}6_8,$$

$$1516{,}6_8 = 1 \cdot 8^3 + 5 \cdot 8^2 + 1 \cdot 8^1 + 6 \cdot 8^0 + 6 \cdot 8^{-1} = 846{,}75_{10}.$$

Очевидно, этот способ значительно сокращает запись по сравнению с непосредственным переводом из десятичной системы в двоичную и обратно.

§ 18. Представление команды в машине и запись на бланке. Кодирование программы

В предыдущей главе программы составлялись в буквенной форме. Процесс решения задачи расчленялся на элементарные операции — команды, которые может выполнять машина. Каждая команда представлялась в буквенном виде, например:

$$a + b = c,$$

где a,b,c — обозначения I, II, и III адресов, а «+» — обозначение кода операции сложения. Однако машина не воспринимает букв и символов, а «понимает» лишь язык двоичных чисел. Поэтому, чтобы машина могла выполнить заданную последовательность элементарных операций, записанных нами в буквенном виде, необходимо эту программу перевести на двоичный язык машины. Сейчас мы и займемся таким переводом.

Числа и команды, образующие исходную информацию для решения задачи, должны располагаться в памяти машины. Память рассматриваемой нами машины содержит $2^{12} = 4096$ ячеек, в каждой из которых находится 45-разрядное двоичное число. Ячейки памяти занумерованы подряд от 0 до 4095_{10}. Номер ячейки называют ее *адресом*. Фактически в устройствах машины (*арифметике* и *управлении*) ячейки нумеруются не четырехразрядными десятичными числами, а двенадцатиразрядными двоичными числами; самый младший адрес равен: 000 000 000 000 = 0, самый старший изображается двенадцатью единицами: $111\ 111\ 111\ 111_2 = 2^{12} - 1 = 4095_{10}$.

Такая нумерация ячеек удобна для машины, ибо она работает в двоичной системе счисления. Однако двоичные

адреса очень громоздки и поэтому неудобны для нумерации ячеек человеком. Для записи адресов ячеек на бумаге используется восьмеричная система счисления.

Как было показано в предыдущем параграфе, для перевода целого двоичного числа в восьмеричное следует разбить двоичное число справа налево на триады, а каждую триаду заменить соответствующей восьмеричной цифрой. Двенадцатиразрядный двоичный адрес содержит, очевидно, четыре триады. Поэтому ячейки памяти получают восьмеричные адреса от 0000_8 до $7777_8 = 4095_{10}$. Так, первая ячейка памяти имеет адрес 0001, 58_{10}-я — адрес 0072_8, 3095_{10}-я — адрес 6027_8. В дальнейшем мы будем пользоваться только восьмеричной записью адресов ячеек памяти.

Рассмотрим теперь структуру ячейки памяти. Каждая ячейка состоит из сорока пяти двоичных разрядов. Эти разряды нумеруются (для наглядности) десятичными числами от 1 до 45 справа налево, как показано на рис. 17.

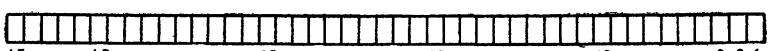


Рис. 17.

В ячейку может быть записан набор из сорока пяти двоичных цифр (нулей или единиц), который называют *машинным словом*. Особую роль играет ячейка с нулевым адресом — 0000. В этой ячейке всегда содержится *нулевое слово*, т. е. слово, состоящее из сорока пяти нулевых двоичных разрядов, и запись другого содержимого в эту ячейку невозможна. Таким образом, содержимое этой ячейки, в отличие от всех других ячеек памяти, не может изменяться.

Слово, содержащееся в ячейке памяти, может быть использовано в программе либо как команда, либо как двоичное число с плавающей запятой, либо как десятичное число с плавающей запятой, либо каким-нибудь иным способом.

Рассмотрим сначала, как представляется в трехадресной машине команда. Обычно команда в машине занимает одну ячейку памяти и делится на две основных части — операционную и адресную. Адресная часть ячейки в трехадресной

машине делится в свою очередь на три части: I, II и III адреса по схеме, показанной на рис. 18.

В трех адресах команды указываются обычно номера (адреса) ячеек памяти, участвующих в операции, код которой указан в операционной части. Как мы видели, для изображения номера (адреса) любой ячейки памяти достаточно 12 двоичных разрядов. Поэтому на адресную часть

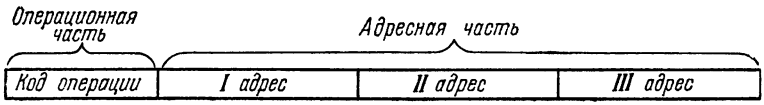


Рис. 18.

в командной ячейке отводится 36 двоичных разрядов (по 12 разрядов на каждый адрес). Разряды эти распределяются по следующей схеме (рис. 19): левый адрес (36—25 разряды) является первым, средний (24—13 разряды) — вторым, правый (12—1 разряды) — третьим.

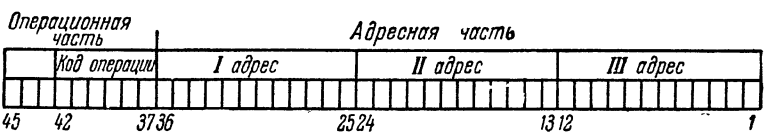


Рис. 19.

Разряды с 42 по 37 отводятся под код операции. Таким образом, код операции представляется шестиразрядным двоичным числом. Поскольку каждому такому числу может соответствовать определенная элементарная операция, то в машине может быть осуществлено не более $2^6 = 64_{10}$ элементарных операций (в предыдущей главе мы познакомились с десятью элементарными операциями). Разряды 45—43-й пока использоваться не будут, и мы будем полагать, что в них записаны нули.

Для того чтобы получить какую-либо команду, например,

$$a + b = c$$

в том виде, в котором ее может выполнить машина, следует прежде всего выделить в памяти ячейки для величин a, b, c,

т. е. приписать этим величинам определенные адреса. Пусть эти адреса суть

a	5712
b	0654
c	4361

Кроме того, нужно знать код операции сложения. Как и адреса ячеек, код тоже удобно записывать в восьмеричной форме. Группу разрядов, отведенных для записи кода, разбивают на две триады, каждая из которых записывается в виде восьмеричной цифры. Таким образом, код операции есть двузначное восьмеричное число. Код операции сложения изображается восьмеричным числом 01.

Команда $a + b = c$ в ячейке машины представляется следующим двоичным словом:

$$\underbrace{000\,000\,001}_{\text{код операции}}\ \underbrace{101\,111\,001\,010}_{\text{I адрес}}\ \underbrace{000\,110\,101\,100}_{\text{II адрес}}\ \underbrace{100\,011\,110\,001}_{\text{III адрес}}$$

При записи команды на бумаге команда представится в виде 001 5712 0654 4361 *).

В такой восьмеричной записи эта команда расшифровывается так: взять числа из ячеек с адресами 5712 и 0654, произвести над ними операцию с кодом 01 (т. е. сложение), результат (сумму) записать в ячейку с адресом 4361.

Подобная замена буквенной записи каждой команды программы восьмеричной записью называется *кодированием программы*. Буквенную запись программы называют записью *в содержательных обозначениях*.

Прежде чем произвести кодирование программы, надо выделить определенные ячейки памяти для исходных данных, рабочих ячеек, результатов счета и команд программы. Эта процедура называется *распределением памяти*.

---

*) Напомним, что самую левую восьмеричную цифру, соответствующую триаде из 45-43 разрядов, мы пока полагаем равной нулю. Об этой цифре речь будет идти в § 25.

Распределим, например, память для первого варианта программы в примере 1.11. Поставим в соответствие буквам a, b, c, x, y, R_1, R_2, R_3, R_4 следующие ячейки памяти:

a	2011	R_1	2001
b	2012	R_2	2002
c	2013	R_3	2003
x	2014	R_4	2004
y	2015		

Шесть команд программы 1)—6) поместим в последовательных ячейках памяти от 1000 до 1005.

Для того чтобы закодировать программу, надо еще знать коды используемых в ней элементарных операций.

Элементарным операциям машины, упомянутым в предыдущей главе, присвоены следующие коды (табл. 1.18).

Таблица 1.18

Название операции	Обозначение	Код
Сложение	$a + b = c$	01
Вычитание	$a - b = c$	02
Вычитание абсолютных величин . . .	$\|a\| - \|b\| = c$	03
Деление	$a : b = c$	04
Умножение	$a \cdot b = c$	05
Извлечение квадратного корня	$\sqrt{a} = c$	44
Остановка	*стоп*	77
Безусловная передача управления . .	*Б* $\overrightarrow{a\ b\ c}$	56
Условная передача управления при $\omega = 0$	*У0* $\overrightarrow{a\ b\ c}$	76
Условная передача управления при $\omega = 1$	*У1* $\overrightarrow{a\ b\ c}$	36

Разделим лист для программы на две половины — левую и правую. В левой части напишем буквенную программу, а в правой будем писать закодированную. Правую половину

листа разграфим на пять столбиков: адреса команд, код операции, IA, IIA, IIIA.

Приступая к кодированию, заполним прежде всего первый столбец правой части, занося в него, в соответствии с принятым распределением памяти, адреса команд 1000, 1001, 1002, 1003, 1004, 1005 (табл. 2.18).

Затем кодируем поочередно каждую команду, начиная с 1). Команда 1), записанная в левой части как $b \cdot x = R_1$, в кодированном виде должна выглядеть так:

05, адрес b, адрес x, адрес R_1

Согласно табличке распределения памяти

адрес $b = 2012$,
адрес $x = 2014$,
адрес $R_1 = 2001$.

Поэтому в правой части команду 1) следует писать в виде

005 2012 2014 2001

Так же кодируются и команды 2), 3), 4), 5). У команды 6) в правой части проставляется код операции 77 и нулевые адреса (табл. 2.18).

Таблица 2.18

	Вычисление значения квадратного трехчлена	Адреса команд	Код операции	IA	IIA	IIIA
1)	$b \cdot x = R_1$	1000	0 | 05	2012	2014	2001
2)	$R_1 + c = R_2$	1001	0 | 01	2001	2013	2002
3)	$x \cdot x = R_3$	1002	0 | 05	2014	2014	2003
4)	$a \cdot R_3 = R_4$	1003	0 | 05	2011	2003	2004
5)	$R_2 + R_4 = y$	1004	0 | 01	2002	2004	2015
6)	*стоп*	1005	0 | 77	0000	0000	0000

Программирование, распределение памяти и кодирование производят на специальных бланках, отпечатанных

типографским способом. В таблицах 3.18 и 4.18 приведены образцы программного бланка и части бланка для распределения памяти с программой решения квадратного уравнения (см. пример 1.12 стр. 65).

Для облегчения кодирования буквенную программу нужно писать четко и аккуратно, стараясь, чтобы буквы, соответствующие первому, второму и третьему адресам, располагались одна под другой, а буквы *У0*, *У1*, *Б*, заменяющие коды операции, находились ближе к левому краю бланка. Иногда левую часть бланка делят тонкими линиями на колонки, соответствующие разбиению правой части.

Программный бланк разделен на две группы по 12_{10} строк. В строках со второй по двенадцатую записываются команды программы. Место в первой строке слева от буквы *А* предназначено для *адресного слова*. Сюда записывается адрес ячейки памяти, в которую помещается команда, написанная в следующей (второй) строке. Записанные далее команды помещаются подряд в ячейки с последующими номерами.

Бланк распределения памяти представляет собой таблицу с написанными восьмеричными адресами ячеек. Бланки бывают различного формата и на различное количество адресов. Обычно один бланк содержит $1000_8 = 512_{10}$ или $2000_8 = 1024_{10}$ клеток и для всей памяти требуется соответственно восемь или четыре бланка. В некоторых случаях в каждой клетке проставляются цифры от 000 до 777, а первая цифра адреса, т. е. номер восьмеричной тысячи, проставляется вручную. Такой бланк с записанными в нем сведениями о данной задаче мы будем называть *памяткой*. Его называют также *словарем*. Впрочем, чаще всего такой бланк называют *шпаргалкой*.

В предыдущем параграфе было отмечено, что в ячейке с адресом 0000 (нулевой ячейке) всегда находится нулевое слово. Далее мы увидим, что это слово, использованное как двоичное число, означает нуль. Поэтому в командах 3), 13), 14), 15), 18) и 19) число «0» закодировано нулевым адресом.

Заметим также, что в рассматриваемой программе приходится кодировать, наряду с обозначениями рабочих ячеек, исходных данных и результатов счета, еще и номера 15), 20) команд, которым передается управление.

Таблица 3.18

					3040	*A*	
1)	$a + a = \rho$	3040	0	01	3000	3000	3010
2)	$b : \rho = R_1$	1	0	04	3001	3010	3013
3)	«0» $- R_1 = \alpha$	2	0	02	0000	3013	3011
4)	$b \cdot b = R_1$	3	0	05	3001	3001	3013
5)	$\rho \cdot c = R_2$	4	0	05	3010	3002	3014
6)	$R_2 + R_2 = R_2$	5	0	01	3014	3014	3014
7)	$R_1 - R_2 = D$	6	0	02	3013	3014	3007
8)	*У1* $\overrightarrow{\alpha\ 15)\ x_1}$	7	0	36	3011	3056	3003
9)	$\sqrt{D} = R_1$	3050	0	44	3007	0000	3013
10)	$R_1 : \rho = \beta$	1	0	04	3013	3010	3012
11)	$\alpha + \beta = x_1$	2	0	04	3011	3012	3003
					3053	*A*	
12)	$\alpha - \beta = x_2$	3053	0	02	3011	3012	3005
13)	«0» + «0» $= x_1'$	4	0	01	0000	0000	3004
14)	*Б* $\overrightarrow{\text{«0»}\quad 20)\quad x_2'}$	5	0	56	0000	3063	3006
15)	«0» $- D = R_1$	6	0	02	0000	3007	3013
16)	$\sqrt{R_1} = R_1$	7	0	44	3013	0000	3013
17)	$R_1 : \rho = x_1'$	3060	0	04	3013	3010	3004
18)	«0» $- x_1' = x_2'$	1	0	02	0000	3004	3006
19)	«0» $+ x_1 = x_2$	2	0	01	0000	3003	3005
20)	*стоп*	3	0	77	0000	0000	0000

Таблица 4.18

3000	a	3040	1)	3100
3001	b	3041	2)	3101
3002	c	3042	3)	3102
3003	x_1	3043	4)	3103
3004	x_1'	3044	5)	3104
3005	x_2	3045	6)	3105
3006	x_2'	3046	7)	3106
3007	D	3047	8)	3107
3010	ρ	3050	9)	3110
3011	α	3051	10)	3111
3012	β	3052	11)	3112
3013	R_1	3053	12)	3113
3014	R_2	3054	13)	3114
3015		3055	14)	3115
3016		3056	15)	3116
3017		3057	16)	3117
3020		3060	17)	3120
3021		3061	18)	3121
3022		3062	19)	3122
3023		3063	20)	3123
3024		3064		3124
3025		3065		3125
3026		3066		3126

Программа

Нумерация всех команд программы загромождает ее запись и является излишней. Вполне достаточно нумеровать лишь те команды, которым передается управление, ибо номера только этих команд участвуют в кодировании.

Более удобно применять не нумерацию, а различные условные обозначения команд. Эти номера или условные обозначения мы будем в дальнейшем называть *метками* команд. Метки команд пишут на бланке на месте номеров и наряду с буквенными обозначениями ячеек вносят в памятку.

§ 19. Машины с фиксированной и плавающей запятой

Как было сказано в § 1, числа могут представляться в форме с фиксированной и с плавающей запятой. При записи числа на бумаге вручную переход от одной формы к другой не вызывает ни малейших затруднений. Поэтому выбор формы записи не имеет существенного значения. Иначе обстоит дело, когда нужно выбрать форму записи числа в машине, так как выбранной формой придется пользоваться и в дальнейшем.

Наиболее привычной для нас является запись числа с фиксированной запятой. Такая форма записи чисел применялась в некоторых машинах, которые называют *машинами с фиксированной запятой*. При записи числа в ячейке памяти машины с фиксированной запятой положение запятой, отделяющей целую часть числа от дробной, фиксируется обычно перед первым разрядом. Это означает, что в ячейке машины с фиксированной запятой записываются только числа, меньшие единицы.

Каждый разряд ячейки памяти в такой машине всегда означает один и тот же разряд числа. Используя всю ячейку памяти, мы можем для всех употребляемых в вычислениях чисел обеспечить одну и ту же абсолютную погрешность. Сложение и вычитание чисел, записанных в ячейке машины с фиксированной запятой, можно производить сразу так, как эти числа записаны.

Недостатком машины с фиксированной запятой является ограниченность диапазона записываемых в них чисел. Для работы на машине с фиксированной запятой необходимо, чтобы все исходные данные, результаты промежуточных вычислений, а также окончательные результаты изображались числами, меньшими единицы. Это достигается выбором единиц измерения и подбором соответствующих масштабных коэффициентов, что существенно затрудняет программирование. Поэтому для сколько-нибудь сложных вычислительных задач, требующих достаточно разнообразных действий, применение машин с фиксированной запятой не выгодно.

Значительно шире диапазон чисел, которые можно записать в ячейке *машины с плавающей запятой*. В такой машине число записывается в нормальной форме, для чего одна группа разрядов ячейки

памяти отводится для записи мантиссы, а другая — для записи порядка. При этом мы имеем мантиссу постоянной длины, т. е. можем обеспечить для всех используемых чисел одну и ту же относительную погрешность. Диапазон чисел определяется здесь числом разрядов, отведенных для записи порядка.

Машины с плавающей запятой конструктивно сложнее, нежели с фиксированной, так как в последних все разряды (кроме знакового) равноправны и работают одинаково, тогда как для машин с плавающей запятой разряды порядка и разряды мантиссы в процессе вычислений играют существенно различную роль.

Арифметические действия в машине с плавающей запятой выполняются сложнее. Например, сложение или вычитание чисел с плавающей запятой не является в точном смысле элементарной операцией машины, а состоит из нескольких элементов. Прежде всего нужно сравнить порядки слагаемых. Если эти порядки различны, то одно из слагаемых надо «разнормализовать», сдвинув мантиссу таким образом, чтобы уравнять порядки. После этого можно производить сложение мантисс, а затем, быть может, понадобится еще раз сдвигать мантиссу для нормализации результата. Выравнивание порядка происходит по большему из них (по абсолютной величине), как это делается в примерах, приводимых в § 1. У многих машин с плавающей запятой в числе элементарных операций имеются и действия с фиксированной запятой, воспринимающие машинное слово, записанное в ячейке, или некоторую его часть как число с фиксированной запятой. Подробнее о таких действиях будет идти речь в следующей главе. Наоборот, для машин с фиксированной запятой, в тех более сложных случаях, когда не удается обычным масштабированием добиться того, чтобы все числа, участвующие в расчетах, укладывались в заданном диапазоне, создаются программы, при помощи которых можно выполнять арифметические действия с плавающей запятой. Иначе говоря, можно для машин с фиксированной запятой «запрограммировать плавающую запятую». Впрочем, такое программирование плавающей запятой приводит к значительному замедлению в выполнении операций.

§ 20. Представление чисел в машине и их запись на бланке

Арифметические действия в рассматриваемой машине производятся над двоичными числами с плавающей запятой, т. е. над числами, представленными в виде $M \cdot 2^p$, где p — двоичный порядок числа, а M — его мантисса.

Сорокапятиразрядная ячейка машины, содержащая двоичное число, делится на части по схеме, приведенной на рис. 20.

Мантисса числа занимает разряды с 1-го по 36-й, порядок — с 37-го по 43-й. 44-й разряд характеризует знак числа: если число положительное, в этом разряде ставится

нуль, если число отрицательное — единица. Самый левый (45-й) разряд содержит специальный признак числа (метку), который тоже может быть нулем или единицей. Обычно этот признак не используется. Мы будем всегда считать его нулевым.

Выясним теперь, с какой точностью при указанном распределении разрядов представляются в машине двоичные числа и каков возможный их диапазон.

Точность представления числа с плавающей запятой определяется, очевидно, числом разрядов в мантиссе.

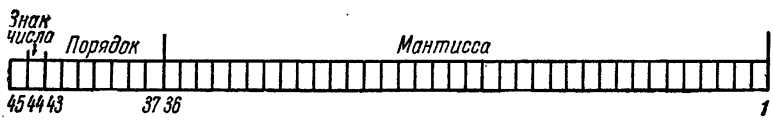


Рис. 20.

Наибольшее значение целого числа, которое может поместиться в разрядах мантиссы, равно $7777\ 7777\ 7777_8 = (8^{12} - 1)_{10} \approx 10^{11}$. Следовательно, машина работает с числами, имеющими (в переводе в десятичную систему) примерно одиннадцать значащих цифр.

Обычно машина работает с нормализованными числами, для которых, как мы знаем,

$$1/2 \leqslant M < 1.$$

Это означает, что самый старший (36-й) разряд мантиссы для нормализованного двоичного числа должен быть равен 1. Наибольшее значение мантиссы равно $1-2^{-36}$, и следовательно,

$$1/2 \leqslant M \leqslant 1 - 2^{-36}.$$

Диапазон представления чисел в машине определяется величиной порядка. *Машинный порядок* *) p' двоичного числа занимает в ячейке семь двоичных разрядов (с 37-го по 43-й) и, следовательно, может принимать целые положительные значения от $0\ 000\ 001_2 = 001_8$ до $1\ 111\ 111_2 = 177_8$:

$$1 \leqslant p' \leqslant 177_8.$$

*) Иногда говорят также *условный порядок*.

Машинный порядок p' связан с действительным порядком p соотношением

$$p' = p + 100_8.$$

Отсюда следует, что диапазон изменения действительного порядка p определяется неравенством

$$-77_8 \leqslant p \leqslant +77_8$$

или

$$-63_{10} \leqslant p \leqslant 63_{10}.$$

Поэтому все машинные двоичные числа x заключены в диапазоне

$$2^{-63} M \leqslant |x| \leqslant 2^{63} M.$$

Таким образом, для нормализованных двоичных чисел

$$2^{-64} \leqslant |x| \leqslant 2^{63}(1 - 2^{-36}).$$

Вычисляя 2^{63} при помощи таблицы логарифмов, получим приближенное неравенство

$$10^{-19,3} \leqslant |x| \leqslant 10^{19}. \qquad (1.20)$$

Неравенство (1.20) и указывает диапазон чисел, представленных в ячейке памяти рассматриваемой машины. Конструкцией машины предусмотрено, что получение результата с машинным порядком, большим 177_8, вызывает ее остановку, так называемый «стоп по переполнению» или «аварийный останов» (сокращенно «*авост*»). Если действительный порядок результата $p < -64$, то в ответе выдается «машинный нуль». Это происходит и в том случае, когда $p = -64$, т. е. машинный порядок $p' = 0$: все разряды мантиссы и знак автоматически гасятся и ответ воспринимается как нуль.

Заметим, что аварийный останов происходит не только при переполнении разрядной сетки, но также при попытке деления на нуль или извлечении квадратного корня из отрицательного числа.

Из формулы $p' = p + 100_8$, связывающей действительный и машинный порядки, видно, что при $p \geqslant 0$ имеем $p' \geqslant 100_8$; в этом случае в 43-м разряде ячейки, изобра-

жающей двоичное число, стоит 1. Если же $-77_8 \leqslant p \leqslant -1$, то $1 \leqslant p' \leqslant 77_8$ и в 43-м разряде стоит 0. Поэтому можно считать, что 43-й разряд характеризует знак действительного порядка; однако нуль в этом разряде будет обозначать знак минус, а единица — знак плюс (ср. с обозначениями знака числа в 44-м разряде на стр. 113).

Величина машинного порядка занимает при этом условии разряды от 37-го по 42-й (см. рис. 20), причем, если действительный порядок неотрицателен, то величины действительного и машинного порядка совпадают; если же действительный порядок отрицателен, то величина машинного порядка является дополнением величины действительного порядка до $1000000_2 = 100_8$.

Так, числа первого и второго положительного порядков будут иметь машинный порядок 101_8 и 102_8, при отрицательных первом и втором порядках машинные порядки оказываются равными 77_8 и 76_8; нулевому действительному порядку соответствует машинный порядок 100_8.

Исходные числовые данные записывают на бланке следующим образом. В левой части бланка пишется нужное число в десятичной системе счисления. Это число переводится в нормализованную двоично-восьмеричную форму. В правой части бланка в восьмеричной форме пишется мантисса: 12 цифр мантиссы располагаются по графам правой части бланка тремя группами по четыре восьмеричных цифры, т. е. так же, как адреса команды. Затем действительный порядок двоичного числа, записанный в восьмеричном виде, переводится в машинный порядок, «величина» машинного порядка в виде двух восьмеричных цифр записывается в графе «код операции». Знак же машинного порядка вместе со знаком числа и меткой в 45-м разряде кодируются одной восьмеричной цифрой, которая ставится перед величиной порядка.

Рассмотрим несколько примеров.

Нулевое слово, рассматриваемое как двоичное число, имеет мантиссу, равную нулю, и действительный двоичный порядок, равный

$$-1000\ 000_2 = -64_{10},$$

т. е. воспринимается как число, равное нулю. Ячейка с номером 0000, содержимое которой всегда нулевое, постоянно

содержит двоичный нуль. Нуль кодируется так:

000 0000 0000 0000.

Закодируем теперь числа $-(3/16)_{10}$, 1/2, 1, 2, —5, 9. Очевидно,

$$\begin{aligned} -3/16 &= -0{,}11 \cdot 10^{-10} \\ 1/2 &= 0{,}1 \cdot 10^{0} \\ 1 &= 0{,}1 \cdot 10^{1} \\ 2 &= 0{,}1 \cdot 10^{10} \\ -5 &= -0{,}101 \cdot 10^{11} \\ 9 &= 0{,}1001 \cdot 10^{100}. \end{aligned}$$

Машинные порядки этих чисел соответственно равны 076, 100, 101, 102, 103, 104. Учитывая знаки чисел и записывая мантиссы в восьмеричной форме, получим машинное представление:

276 6000 0000 0000
100 4000 0000 0000
101 4000 0000 0000
102 4000 0000 0000
303 5000 0000 0000
104 4400 0000 0000

Перевод вручную десятичных чисел в двоичные является довольно трудоемкой работой. Поэтому обычно числа подготавливаются к вводу в десятичном виде (а вводятся в машину в двоично-десятичном).

Как же изображаются в машине десятичные числа?

Десятичное число с плавающей запятой записывается в виде

$$M \cdot 10^{p},$$

где p — величина порядка числа, а M — мантисса. Ячейка, используемая для записи десятичного числа, делится на части по схеме, приведенной на рис. 21. Эта схема аналогична схеме распределения разрядов ячейки для двоичного числа (см. рис. 20). Действительно, мантисса также занимает разряды от 1-го по 36-й, порядок (вместе со знаком) — разряды с 37-го по 43-й, знак числа — 44-й разряд.

Для положительного десятичного числа в 44-м разряде ставится 0, для отрицательного 1. Так же кодируется и знак порядка: знаку плюс соответствует 0 в 43-м разряде, знаку минус — единица.

Для изображения цифр мантиссы и порядка в ячейке машины используется *двоично-десятичная система записи чисел*. В 36 разрядах мантиссы умещается девять тетрад,

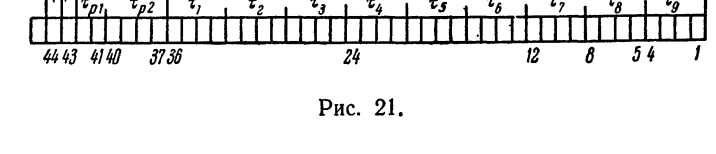


Рис. 21.

т. е. четверок двоичных цифр, изображающих отдельные десятичные цифры. Поэтому мантисса десятичного числа, вводимого в машину, должна содержать не более девяти десятичных цифр.

Каждая десятичная цифра мантиссы изображается в ячейке соответствующей тетрадой. Эти цифры заносятся в мантиссу, начиная с 36-го разряда, слева направо. Так, например, мантисса 0,957 000 000 изображается в ячейке в соответствии со следующей схемой (табл. 1.20).

Таблица 1.20

Номера разрядов	36	35	34	33	32	31	30	29	28	27	26	25	...	4	3	2	1
Двоичная мантисса	1	0	0	1	0	1	0	1	0	1	1	1	...	0	0	0	0
Десятичная мантисса	9				5				7				...	0			

В шести разрядах, отведенных для величины десятичного порядка, умещается одна тетрада (разряды с 37-го по 40-й) и одна *диада* (пара разрядов, 41-й и 42-й разряды). Поэтому максимально допустимая разрядной сеткой машины величина десятичного порядка равна 39_{10}. Однако, как

было выяснено в предыдущем параграфе, машина может производить арифметические операции лишь над числами, не превосходящими (по модулю) 10^{19}. Поэтому максимальная величина десятичного порядка равна 19, и следовательно, в 42-м разряде ячейки, изображающей десятичное число, всегда должен стоять нуль. Таким образом, величина десятичного порядка фактически изображается пятью разрядами — одной *монадой* (один 41-й разряд) и одной тетрадой (разряды с 37-го по 40-й).

На программном бланке десятичные числа кодируются в следующем порядке. В левой части бланка числа записываются в обычном виде (с запятой, фиксированной на любом месте) и нормализуются. Затем в графу правой части бланка, отведенную под код операции команды, заносятся последовательно знак числа ($\pm$), знак порядка ($\pm$) и две цифры десятичного порядка; после этого в графы, отведенные для I, II и III адресов команды, записываются цифры мантиссы, по три в каждом адресе.

Так как машина может производить счет лишь в двоичной системе счисления, то десятичные числа, введенные в указанном виде в машину, должны быть переведены в двоичную систему. Этот перевод производится специальной программой.

В предыдущем параграфе было показано, что действия над двоичными числами в машине фактически производятся примерно с одиннадцатью знаками (в переводе на десятичную систему). В то же время десятичные числа вводятся в машину (и выводятся из нее) с девятью знаками. Следовательно, *в трехадресной машине расчеты производятся с двумя запасными значащими цифрами.* Это дает возможность в подавляющем большинстве задач получить результаты расчетов с достаточной точностью.

В качестве примера кодирования программы вместе с числовыми исходными данными рассмотрим программу решения системы двух линейных алгебраических уравнений.

Пусть коэффициенты уравнений имеют следующие числовые значения:

$$a_1 = 0{,}01875, \quad b_1 = 1{,}8, \quad c_1 = -0{,}1,$$
$$a_2 = -20{,}5, \quad b_2 = 92{,}4, \quad c_2 = -0{,}06125.$$

Память распределим следующим образом (табл. 2.20). Заполним левую часть программного бланка (табл. 3.20) и закодируем программу обычным способом.

Таблица 2.20

1000	Программа	1040	a_1
1001		1041	b_1
1002		1042	c_1
1003		1043	a_2
1004		1044	b_2
1005		1045	c_2
1006		1046	x
1007		1047	y
1010		1050	D
1011		1051	D_1
1012		1052	D_2
1013		1053	R_1
1014		1054	R_2
1015		1055	

§ 21. Модификации арифметических операций

В машине с плавающей запятой предполагается, что арифметические действия совершаются над нормализованными числами и результат округляется и нормализуется. При этом, при расписке формулы, необходимо иметь в виду также и промежуточные результаты. Без этого можно выйти на «авост» не по существу или получить неверный ответ.

Таблица 3.20

					1000	A	
	$a_1 \cdot b_2 = R_1$	1000	0	05	1040	1044	1053
	$a_2 \cdot b_1 = R_2$	1	0	05	1043	1041	1054
	$R_1 - R_2 = D$	2	0	02	1053	1054	1050
	$b_2 \cdot c_1 = R_1$	3	0	05	1044	1042	1053
	$b_1 \cdot c_2 = R_2$	4	0	05	1041	1045	1054
	$R_1 - R_2 = D_1$	5	0	02	1053	1054	1051
	$a_1 \cdot c_2 = R_1$	6	0	05	1040	1045	1053
	$a_2 \cdot c_1 = R_2$	7	0	05	1043	1042	1054
	$R_1 - R_2 = D_2$	1010	0	02	1053	1054	1052
	$D_1 : D = x$	1	0	04	1051	1050	1046
	$D_2 : D = y$	2	0	04	1052	1050	1047

					1013	A	
	стоп	1013	0	77	0000	0000	000
					1040	A	
	$a_1 = 0{,}01875$	1040	0	73	4631	4631	4632
	$b_1 = 1{,}8$	1	1	01	7146	3146	3146
	$c_1 = -0{,}1$	2	2	75	6314	6314	6315
	$a_2 = -20{,}5$	3	3	05	5100	0000	0000
	$b_2 = 92{,}4$	4	1	07	5614	6314	6315
	$c_2 = -0{,}06125$	5	2	74	7656	0507	5341

Например, для трех десятичных чисел $a = 0{,}21 \cdot 10^{10}$, $b = 0{,}3 \cdot 10^{11}$ и $c = 0{,}5 \cdot 10^{7}$ при вычислении величины $x = a \cdot b/c$ можно располагать действия двумя способами, $x = (a \cdot b) : c$ и $x = (a : c) \cdot b$. В первом случае мы получим «*авост*» после умножения, так как $a \cdot b = 0{,}63 \cdot 10^{20}$, что выходит за пределы разрядной сетки машины. Между тем, действуя по второму способу, мы придем к верному результату $x = 0{,}126 \cdot 10^{14}$, укладывающемуся в разрядной сетке.

Аналогичные трудности могут встретиться и при очень малых числовых значениях. Например, если $a = 0{,}21 \cdot 10^{-12}$, $b = 0{,}7 \cdot 10^{7}$, $c = 0{,}5 \cdot 10^{9}$, то вычисляя снова величину $x = ab/c$ по схеме $x = (a : c) \cdot b$, получим после деления $a : c = 0{,}42 \cdot 10^{-21}$ и это число воспримется как машинный нуль. Поэтому и после умножения на b мы получим $x = 0$, что неверно. Если же действовать по первой формуле $x = (a \cdot b) : c$, то получим верный результат $x = 0{,}294 \cdot 10^{-14}$, отличный от машинного нуля и правильно записывающийся в разрядной сетке.

В некоторых специальных случаях программисту бывает выгодно или удобно отказаться от округления или нормализации. Для этой цели в машине предусмотрены специальные модификации арифметических действий.

В настоящем параграфе мы коротко познакомимся с такими модификациями, их возможностями и примерами применения. Для этого надо, прежде всего, несколько подробнее разобраться в том, как выполняются в машине арифметические действия с плавающей запятой, которые мы будем называть *плавающими действиями*.

Как уже говорилось в предыдущем параграфе, для записи мантиссы числа в ячейке памяти машины отводится 36_{10} разрядов. Кроме них, в регистрах чисел арифметического устройства имеются еще два дополнительных разряда — *старший дополнительный разряд* слева от 36-го и *младший дополнительный разряд* справа от первого (напомним, что разряды нумеруются справа налево).

При сложении двух нормализованных чисел одного знака и одинакового порядка в 36-х разрядах обоих слагаемых стоят единицы (кроме случая, когда слагаемые равны нулю) и в сумме получается единица переноса в старший разряд, которая и записывается в старший дополнительный разряд слева. Для нормализации мантисса сдвигается на один разряд вправо, а порядок увеличивается на единицу. Этот сдвиг называют *нормализацией вправо* и он происходит всегда, при выполнении любых модификаций сложения, если только старший дополнительный разряд оказывается отличным от нуля. Если при сдвиге вправо в младший дополнительный разряд попадает единица, то она просто отбрасывается.

Если складываются числа разных порядков (но одинаковых знаков), то предварительно происходит выравнивание порядков, для чего

мантисса меньшего по абсолютной величине числа сдвигается вправо. После выполнения сложения происходит округление, которое производится путем прибавления единицы к младшему дополнительному разряду. После этого в память записывается 36_{10} разрядов мантиссы, а содержимое младшего дополнительного разряда отбрасывается. Таким образом, если в младшем дополнительном разряде перед округлением был нуль, то последняя цифра мантиссы не изменится, а если была единица, то к последней цифре мантиссы прибавится 1.

Сложение чисел разных знаков может привести к тому, что один или несколько разрядов мантиссы слева могут оказаться нулями. Для нормализации результата в этом случае необходимо сдвигать мантиссу влево до тех пор, пока в 36-й разряд не попадет единица. При каждом сдвиге на разряд порядок результата уменьшается на единицу. Такую нормализацию называют *нормализацией влево*. Округление в этом случае производится путем простого отбрасывания младшего дополнительного разряда, без предварительного прибавления к нему единицы. Так же производится округление, когда одно из слагаемых равно нулю.

Мы описали порядок выполнения операции плавающего сложения с округлением и нормализацией. Кроме нее, машина имеет в числе своих элементарных операций еще три модификации сложения: сложение без округления с нормализацией, с округлением без нормализации, без округления и без нормализации. Эти операции мы будем обозначать тем же знаком плюс, указывая дополнительное условие после знака операции в скобках:

$a + (\text{БО})\, b = c$ — сложение с блокировкой округления,

$a + (\text{БН})\, b = c$ — сложение с блокировкой нормализации,

$a + (\text{БОН})\, b = c$ — сложение с блокировкой округления и нормализации.

Сложение без округления блокирует (выключает) округление результата, т. е. прибавление единицы в младший дополнительный разряд. Из сказанного выше следует, что сложение с округлением и с блокировкой округления выполняются одинаково в каждом из трех случаев:

а) слагаемые имеют одинаковые порядки;

б) слагаемые имеют разные знаки;

в) одно из слагаемых равно нулю,

поскольку в этих случаях прибавление единицы в младший дополнительный разряд все равно не производится. В остальных случаях результаты операций с округлением или с блокировкой округления могут оказаться различными.

Операции сложения без нормализации блокируют нормализацию в л е в о, так что в результате операций (БН) или (БОН) сумма может оказаться ненормализованной, т. е. один или несколько левых разрядов мантиссы результата могут равняться нулю. Нормализация вправо блокирована быть не может и обязательно выполняется (в случае необходимости).

Рассмотрим несколько примеров применения модификаций операции сложения.

Пример 1.21. Выделение целой части числа. Пусть в ячейке a записано некоторое число и требуется выделить его целую часть и записать ее в ячейку b. Это можно сделать с помощью одной команды, используя операцию сложения без округления, но с нормализацией.

Пусть в ячейке памяти записано слово, порядковая часть которого 145_8, а мантисса равна нулю. Обозначим это слово (145; 0,0,0). Тогда выделение целой части числа a можно выполнить так:

$$(145;\ 0,\ 0,\ 0) + (\text{БО})\ a = b.$$

Действительно, при выравнивании порядков мантисса числа a сдвинется на $45_8 - p$ разрядов, где p — истинный порядок числа a. Так как $45_8 = 37_{10}$, то в разрядной сетке машины, включая младший дополнительный разряд, останется ровно p разрядов мантиссы, т. е. целая часть, а дробная выйдет за ее пределы и погасится. При нормализации целая часть мантиссы вернется на место и порядок восстановится.

Сдвиг вправо мантиссы числа a произойдет в том случае, если $p < 45_8$. Если же $p \geqslant 45_8$, то мы получим $a = b$, так как ни сдвига, ни изменения порядка не произойдет. Но в этом случае число, записанное в ячейке a, само будет целым (дробная часть в ячейке не помещается), так что можно считать выделение целой части выполненным.

Пример 2.21. Нахождение ближайшего целого. Пусть требуется найти ближайшее целое для числа, записанного в ячейке a, и поместить его в ячейку b. Это можно сделать следующим образом:

$$a + (\text{БН})\ (144;\ 4000,\ 0,\ 0) = R,$$
$$R - \qquad (144;\ 4000,\ 0,\ 0) = b.$$

В самом деле, при выравнивании порядков для сложения мантисса числа a сдвинется на $44_8 - p$ разрядов (p — истинный порядок a), так что целая часть мантиссы a останется внутри разрядной сетки машины, а в младшем дополнительном разряде будет находиться первая после запятой цифра. Если эта цифра — нуль, то прибавление единицы к дополнительному разряду не изменит содержимого первого разряда, а при записи результата в память эта единица будет просто отброшена. Если же в младшем дополнительном разряде стояла единица, то прибавление единицы при округлении увеличит целую часть. Таким образом, после вычитания прибавленной константы (с нормализацией) в ячейке b будет записана либо целая часть числа a, если его дробная часть меньше половины, либо следующее целое число — в противном случае.

Другие примеры применения модификаций операции сложения будут рассмотрены ниже.

Аналогичным образом выполняется операция вычитания. Обычное вычитание происходит с округлением и с нормализацией. Округление совершается путем прибавления единицы к младшему дополнительному разряду результата, что делается всегда, кроме следующих случаев:

а) уменьшаемое и вычитаемое имеют одинаковые порядки;

б) уменьшаемое и вычитаемое имеют одинаковые знаки;

в) уменьшаемое или вычитаемое равно нулю.

Как и для сложения, для вычитания имеются три модификации, обозначаемые так же:

$$a - (\text{БО})\ \ b = c,$$
$$a - (\text{БН})\ \ b = c,$$
$$a - (\text{БОН})\ b = c.$$

При этом операции (БО) и (БОН) блокируют прибавление единицы к младшему дополнительному разряду (если оно должно производиться), а операции (БН) и (БОН) — нормализацию влево. Нормализация вправо при вычитании также не может быть заблокирована.

Такие же точно модификации имеет операция вычитания абсолютных величин. Впрочем, округление прибавлением единицы здесь не производится, так что операции без округления и с округлением выполняются совершенно одинаково.

Результат сложения, вычитания или вычитания абсолютных величин без блокировки нормализации всегда получается нормализованным, даже если исходные числа не были нормализованы. В частности, для нормализации числа достаточно сложить его с нулем с помощью обычной операции сложения.

При плавающем умножении порядки множителей складываются, а мантиссы перемножаются как числа нулевого порядка. Вследствие этого нормализация вправо здесь потребоваться не может. Нормализация влево производится сдвигом лишь на один разряд. Поэтому при умножении нормализованных чисел без блокировки нормализации результат всегда будет нормализованным, а при умножении ненормализованных чисел результат может оказаться ненормализованным, даже в том случае, когда нормализация производилась. Блокировка нормализации в операциях (БН) и (БОН) блокирует нормализацию влево.

Мантисса произведения получается в арифметическом устройстве с 72_{10} двоичными разрядами, затем переписывается в регистр чисел с младшим дополнительным разрядом, после чего округляется прибавлением в дополнительный разряд единицы. Блокировка округления в операциях (БО) и (БОН) блокирует это прибавление.

Если требуется знать все разряды произведения, то младшие 36_{10} разрядов можно вывести из арифметического устройства в ячейку памяти машины с помощью специальной команды, которую нужно помещать непосредственно после операции умножения с блокировкой округления. Эту команду мы будем записывать в виде

$$\text{младшие разряды} = c.$$

Она является одноадресной, т. е. в ней учитывается лишь один (третий) адрес; значение остальных адресов на выполнение команды не влияет, обычно в них пишут нули. По этой команде в порядковую часть ячейки c записывается порядок произведения, а в мантиссу — последние 36_{10} разрядов мантиссы произведения. Выборка из памяти при этом не производится.

Округление при делении и извлечении квадратного корня также производится путем прибавления единицы в младший дополнительный разряд. Для обеих этих операций имеется по одной модификации с бло-

кировкой округления. Возможность блокировки нормализации для деления и извлечения корня не предусмотрена.

Коды модификаций арифметических операций легко запомнить, так как они получаются из уже известных по общему простому правилу. Именно, код операции с блокировкой округления (БО) получается из кода основной операции прибавлением к нему 20_8. Для получения кода (БН) надо к коду основной операции прибавить 40_8, а для кода (БОН) — 60_8. Все эти коды приведены в табл. 1.21, в первой колонке которой указан код основной операции — с округлением и с нормализацией. Код операции вывода младших разрядов — 47.

Таблица 1.21

Операция	Основная	БО	БН	БОН
Сложение	01	21	41	61
Вычитание	02	22	42	62
Вычитание абсолютных величин	03	23	43	63
Умножение	05	25	45	65
Деление	04	24	—	—
Извлечение квадратного корня	44	64	—	—

Управляющий сигнал ω для всех модификаций арифметических операций вырабатывается так же, как и для соответствующих основных: для сложения, вычитания или вычитания абсолютных величин $\omega = 0$, если результат неотрицателен, и $\omega = 1$, если отрицателен, независимо от блокировки округления или нормализации. Для умножения, деления и извлечения квадратного корня $\omega = 1$, если машинный порядок результата больше или равен 101_8, и $\omega = 0$, если меньше.

§ 22. Перфорация и ввод

Большинство современных электронных машин не может «читать» цифры, записанные на бумаге. Поэтому, прежде чем вводить числа и команды в машину, их сначала переносят на перфокарты (картонные карточки с пробиваемыми на них отверстиями).

Стандартная перфокарта (рис. 22) состоит из 80 колонок (столбцов) и 12 позиций (строк). Из них две верхние строки размещаются на свободном месте вверху карты, а остальные строки помечены цифрами 0, 1, ..., 9. Над нулевой и между восьмой и девятой позициями мелким шрифтом напечатаны номера колонок.

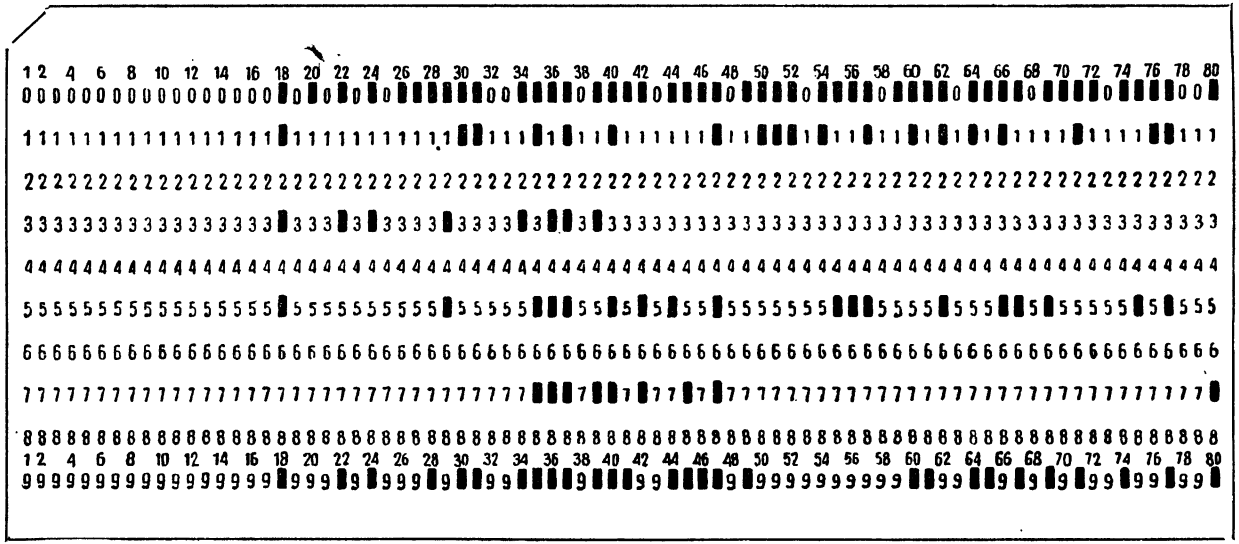


Рис. 22.

В каждую позицию может заноситься одно сорокапятиразрядное слово. Для изображения слова выделено 45 колонок. Пробивка в колонке соответствует двоичной цифре 1, отсутствие — цифре 0.

Кроме того, на перфокарте выделены две специальных маркерных колонки — основная с номером 18 и вспомогательная с номером 80. На рис. 22 в позиции 0 указаны пробивками 45 основных и две маркерных колонки.

В позицию (строку) перфокарты могут заноситься обычные и специальные слова. Обычные слова — это числа или команды, вводимые с перфокарт в память машины. Признаком того, что в позицию занесено обычное слово, является пробивка в основной маркерной дорожке.

На рис. 22 изображено, как пробивается на карте команда 003; 2501, 3625, 5043 (позиция 1), двоично-восьмеричное число 304; 5600, 0, 0, равное $-1011{,}1_2 = -13{,}4_8 = -11{,}5_{10}$ (позиция 3) и двоично-десятичное число $+7590{,}71385_{10}$ (позиция 5).

Кроме обычных слов, на перфокарте могут быть пробиты и специальные слова двух видов: адресные и контрольные.

Признаком *адресного слова* является пробивка на вспомогательной дорожке. Адресное слово занимает только колонки, соответствующие первому адресу в обычном командном слове (позиция 7 на рис. 22 содержит адресное слово 3725).

Как мы уже упоминали в § 18, обычные слова, идущие после адресного слова, записываются при вводе в машину в последовательные ячейки памяти, начиная с ячейки, указанной в адресном слове. Каждая перфокарта обычно начинается с адресного слова. На программном бланке размещается материал для пробивки двух перфокарт.

Контрольное слово (позиция 9 на рис. 22) отличается от других слов пробивками в обеих маркерных дорожках. Наличие этого слова дает сигнал машине о прекращении ввода материала с перфокарт в память *).

В качестве примера на рис. 23 приведена перфокарта с набитой на ней программой вычисления значения квадратного трехчлена (табл. 2.18, стр. 107).

*) Кроме того, это слово имеет и другое специальное назначение, о котором будет сказано ниже.

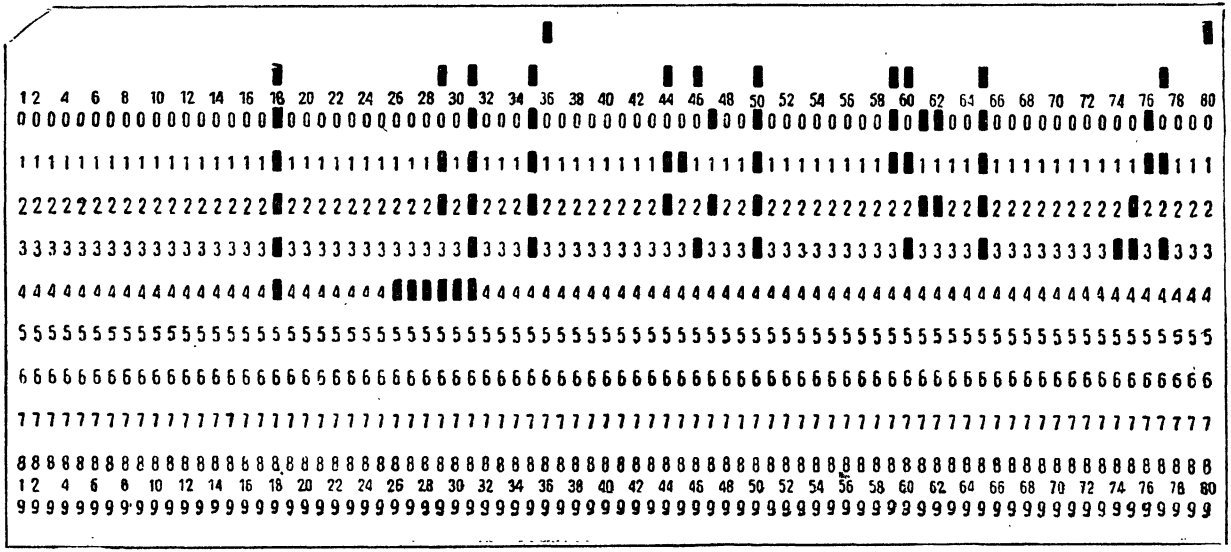


Рис. 23.

Пробивка программы на перфокартах производится при помощи специального перфоратора. Перфоратор может работать в двух режимах — восьмеричном и десятичном. Переход с одного режима на другой производится при помощи специального тумблера.

В восьмеричном режиме пробиваются команды и двоичные числа, причем при нажатии восьмеричной цифры на клавиатуре происходит автоматический перевод этой цифры в соответствующую триаду. В десятичном режиме (при пробивке десятичных чисел) происходит перевод нажатой на клавиатуре десятичной цифры в соответствующую тетраду (для пробивки знаков числа и порядка имеются специальные клавиши + и —).

Во избежание путаницы при пробивке, десятичные числа следует писать на отдельных бланках с надписью «числа».

Пробитые перфокарты контролируются одним из двух способов:

а) чтением при помощи специального шаблона;

б) выпечатыванием на бумажную ленту и сверкой с программой.

Колода перфокарт с набитыми на них командами и числами через специальное вводное устройство (*ввод*) поступает в память машины.

Кодирование, перфорация и ввод в машину команд и числового материала являются чисто технической работой, не вызывающей, при небольшом навыке, трудностей. Поэтому в следующих главах книги мы ограничимся, в основном, написанием содержательных частей программ, не производя их кодировки.

ГЛАВА V

ПЕРЕАДРЕСАЦИЯ

§ 23. Действия над числами с фиксированной запятой

Рассмотренные выше арифметические операции производятся над словами, изображающими число с плавающей запятой. Кроме них, существуют еще операции, в которых слово, записанное в ячейке, или некоторая его часть, рассматривается как целое неотрицательное число с фиксированной запятой. Такие операции называются фиксированными действиями *).

Мы рассмотрим фиксированные действия, которые совершаются не над полным содержимым ячейки, а над ее частями. Как было указано в предыдущей главе, при записи числа или команды ячейка делится на две неравные части, состоящие соответственно из 9 и 36 разрядов.

Правая часть ячейки, состоящая из разрядов 1-36, при записи команды отводится для ее адресной части, а при записи числа — для мантиссы. Операции сложения и вычитания этих частей слова, рассматриваемых как целые неотрицательные числа с фиксированной запятой, можно назвать

*) Как уже говорилось в § 19, некоторые вычислительные машины могут работать как в режиме с плавающей, так и в режиме с фиксированной запятой. У таких машин *фиксированными действиями* называют арифметические действия, в которых слово, записанное в ячейке, воспринимается как число с фиксированной запятой с учетом его знака. Таких действий у рассматриваемых машин типа М-20 нет. Разбираемые нами ниже действия имеют очень много общего с такими фиксированными действиями, хотя и имеют дело не со всей ячейкой, а лишь с частью ее, которая воспринимается как целое неотрицательное число с фиксированной запятой. Поэтому мы используем то же название фиксированных действий.

соответственно *сложением* и *вычитанием мантисс* или же *сложением* и *вычитанием адресных частей*.

Более распространенным является название *сложение мантисс*, хотя название *сложение адресных частей* следует считать более удачным, так как эта операция используется, главным образом, при преобразованиях команд, как это будет видно в следующих параграфах.

Мы будем обозначать эти операции следующим образом:

$$a+,\ b=c,$$
$$a-,\ b=c.$$

В операции $a+,\ b=c$ мантисса слова c образуется сложением мантисс a и b как целых неотрицательных чисел с фиксированной запятой, а порядок (операционная часть) слова c полагается равным порядку a. Таким образом, результаты операций $a+,\ b$ и $b+,\ a$ будут, вообще говоря, различными.

При сложении может возникнуть необходимость переноса единицы из 36-го разряда в старший разряд. Поскольку 37-й разряд ячейки не относится уже к адресной части, то этот перенос не происходит и единица переноса теряется. Признаком того, что такой перенос должен был произойти, является выработка управляющего сигнала $\omega = 1$. Если же сумма адресных частей a и b умещается в пределах 36 разрядов мантиссы c, то вырабатывается сигнал $\omega = 0$.

Операция $a-,\ b=c$ выполняется так: порядок слова полагается равным порядку уменьшаемого (a). Адресная часть c находится вычитанием адресных частей a и b, рассматриваемых как целые числа с фиксированной запятой. Если мантисса b не превосходит мантиссы a, то вычитание происходит обычным образом и вырабатывается сигнал $\omega = 0$. Если же мантисса a меньше мантиссы b, то вычитание происходит так, как будто к уменьшаемому слева была приписана единица 37-го разряда. В этом случае при вычитании адресных частей вырабатывается управляющий сигнал $\omega = 1$.

Аналогичные операции можно определить и для левых частей ячейки, состоящих из 37-45 разрядов. Эта часть ячейки при записи команды отводится для кода операции,

а при записи числа — для его порядка. Поэтому соответствующие операции, которые мы будем обозначать

$$a\ ,+b=c,$$
$$a\ ,-b=c,$$

носят названия *сложения* и *вычитания порядков* или же *сложения* и *вычитания операционных частей*.

Как и в предыдущем случае, название *сложение порядков* является более распространенным, а *сложение операционных частей* — более соответствующим существу операции.

В операции $a\ ,+\ b=c$ операционная часть слова c равна сумме операционных частей слов a и b, рассматриваемых как целые неотрицательные числа с фиксированной запятой. Мантисса слова c переносится целиком из мантиссы a. Если возникает необходимость переноса единицы из 45-го разряда, то единица переноса теряется, но вырабатывается управляющий сигнал $\omega = 1$, если же такой необходимости не возникало, то вырабатывается сигнал $\omega = 0$.

Точно так же, при вычитании операционных частей $a\ ,-\ b=c$ адресная часть слова c равна адресной части a, а операционная часть c получается вычитанием из порядка a порядка b. Если порядок b не превосходит порядка a, то происходит обычное вычитание и вырабатывается сигнал $\omega = 0$. В противном случае вырабатывается сигнал $\omega = 1$, а вычитание происходит так, как будто перед операционной частью a была приписана единица 46-го разряда.

Рассмотрим на примерах, как выполняются описанные выше фиксированные операции.

Пример 1.23. Пусть

$$a+,\ b=c,$$

причем a = 001 1005 2000 1500 и b = 105 7776 0000 0002. Тогда

$$c=001\ 1003\ 2000\ 1502$$

и вырабатывается сигнал $\omega = 1$. При том же содержимом указанных ячеек a и b по команде $b\ +,\ a=c$ получим c = 105 1003 2000 1502. Для операции $a\ -,\ b=c$, если a = 000 2735 0001 5007 и b = 012 2510 7776 3541, то c = 000 0224 0003 1246 и вырабатывается сигнал $\omega = 0$.

Если же $a = 075\ 1344\ 2752\ 0001$ и $b = 031\ 5121\ 3174\ 0002$, то $c = 075\ 4222\ 7555\ 7777$ и $\omega = 1$.

Для операции $a\ ,+\ b = c$ при $a = 154\ 2577\ 3143\ 2614$ и $b = 342\ 3217\ 2500\ 7441$ получаем $c = 516\ 2577\ 3143\ 2614$ и $\omega = 0$, а при том же a и при $b = 767\ 1243\ 1544\ 6704$ будем иметь $c = 143\ 2577\ 3143\ 2614$ и $\omega = 1$.

Операции сложения и вычитания адресных и операционных частей ячейки очень широко используются при программировании в самых разнообразных случаях. Покажем, прежде всего, их применение при составлении арифметических циклов.

Во всех рассмотренных в § 13 примерах в качестве счетчиков цикла использовались целые двоичные числа с плавающей запятой. Операции вычитания адресных и операционных частей дают возможность образовывать фиксированные целые счетчики, т. е. использовать в качестве счетчиков мантиссу или порядок слова. Эта возможность тем более полезна, что фиксированные действия происходят заметно быстрее плавающих.

В большинстве случаев максимальное значение счетчика не превышает $4095_{10} = 7777_8$. Поэтому обычно в качестве счетчика можно использовать не всю мантиссу слова, а какой-либо один адрес.

Покажем на примере, как это делается.

Пример 2.23. Вычислить

$$y=\underbrace{x+\sqrt{x+\sqrt{x+\ldots+\sqrt{x}}}}_{n\text{ членов}}.$$

Легко видеть, что величину y можно получить при помощи следующего процесса:

$$y_0=0,$$
$$y_k=x+\sqrt{y_{k-1}};\quad k=1,\ 2,\ \ldots,\ n;\quad y=y_n.$$

Этот процесс можно запрограммировать как арифметический цикл с рабочей частью, состоящей из двух команд:

$$\sqrt{y}\quad =R,$$
$$x+R=y.$$

В начале цикла следует положить $y = 0$, а затем выполнить написанные две команды n раз.

Цикл удобно организовать с фиксированным счетчиком, с обратным изменением счетчика, в соответствии со схемой, изображенной на рис. 16 (стр. 80).

Предположим, что в некоторой ячейке машины содержится слово, в котором в виде фиксированного числа единиц третьего адреса записано число n, а все остальные разряды содержат нуль. Такую ячейку мы будем обозначать через (0, 0, n). Пусть, кроме того, в некоторой ячейке записано слово 000 0000 0000 0001, которое мы будем коротко обозначать через (0, 0, 1). Тогда в качестве начального значения счетчика мы можем выбрать

$$(0,\ 0,\ n)\text{—},\ (0,\ 0,\ 1)=\nu,$$

а изменение счетчика вместе с проверкой окончания производить по команде

$$\nu\text{—},\ (0,\ 0,\ 1)=\nu.$$

Тогда арифметический цикл запишется так:

1)		0	+	0	$=y$	
2)		$(0,\ 0,\ n)$	—	$(0,\ 0,\ 1)$	$=\nu$	
3)		$\sqrt{y}$			$=R$	↑
4)		x	+	R	$=y$	|
5)		ν	—,	$(0,\ 0,\ 1)$	$=\nu$	|
6)	*У0*					
7)	*стоп*					

После того как цикл будет выполнен n — 1 раз, содержимое ячейки ν сделается равным нулю. После передачи управления на рабочую часть и вычисления окончательного значения y придется опять выполнить команду вычитания адресных частей, причем адресная часть вычитаемого будет превосходить адресную часть уменьшаемого. При этом вырабатывается сигнал $\omega = 1$ и команда *У0* передаст управление следующей ячейке, в которой находится команда *стоп*.

С тем же успехом в качестве счетчика можно было взять первый или второй адрес ячейки ν. Например, для использования в качестве счетчика второго адреса нужно иметь

ячейки (0, n, 0) и (0, 1, 0) и заменить команды 2) и 5) следующими командами:

$$(0,\ n,\ 0)\ —,\ (0,\ 1,\ 0) = \nu$$

и

$$\nu\ \ —,\ (0,\ 1,\ 0) = \nu.$$

Практика программирования показывает, что счетчики-адреса́ (*фиксированные счетчики*) удобнее счетчиков-чисел с плавающей запятой (*плавающих счетчиков*), так как вручную легче перевести начальное значение счетчика в целое восьмеричное число, чем в плавающее двоично-восьмеричное. Кроме того, из-за наличия ошибок округления применение плавающих счетчиков может в некоторых случаях привести к ошибкам в счете числа шагов цикла.

Если максимальное значение счетчика не превышает $511_{10} = 777_8$, то в качестве счетчика можно использовать вместо адресной операционную часть слова; в этом случае проверку окончания арифметического цикла следует осуществлять при помощи операции вычитания операционных частей.

Так, в рассмотренном примере цикл можно организовать так:

1) $0 + 0 = y$
2) $(n;\ 0,\ 0,\ 0),\ —\ (1;\ 0,\ 0,\ 0) = \nu$
3) $\sqrt{y} = R$
4) $x + R = y$
5) $\nu\ ,—\ (1;\ 0,\ 0,\ 0) = \nu$
6) *УО*
7) *стоп*

Отметим один частный прием, в котором используется операция сложения порядков. Предположим, что в слове c мы хотим получить порядок слова b и нулевую мантиссу. Этого можно достичь при помощи команды

$$0\ ,+b = c.$$

Действительно, в нулевой ячейке находится слово с нулевым порядком и нулевой мантиссой. Поэтому мантисса слова оказывается равной нулю, а порядок — сумме порядков нулевого слова и b, т. е. порядку слова b.

Аналогичным приемом, но уже применяя операцию сложения мантисс, можно выделить в слове b мантиссу, обращая в нуль порядок:

$$0+,\ b=c.$$

В следующих параграфах мы подробно рассмотрим главную область применения операций сложения и вычитания адресных частей — переадресацию переменных команд в цикле.

Для кодирования нужно знать коды фиксированных операций. Приводим соответствующую таблицу (табл.1.23).

Таблица 1.23

Название операции	Обозначение	Код
Сложение мантисс	$a+,\ b=c$	13
Вычитание мантисс	$a-,\ b=c$	33
Сложение порядков	$a\ ,+\ b=c$	53
Вычитание порядков	$a\ ,-\ b=c$	73

§ 24. Циклы с переадресацией

Рассмотрим задачу, аналогичную возведению числа x в степень n (§ 13). Найдем произведение

$$z=x_1\cdot x_2\cdot\ldots\cdot x_{100}$$

ста чисел, расположенных в ста последовательных ячейках памяти

$$x_1,\ x_2,\ \ldots,\ x_{100}.$$

Распишем формулу для z по командам

$$0+\text{«1»}=z$$
$$z\cdot x_1=z$$
$$z\cdot x_2=z$$
$$\ldots\ldots\ldots$$
$$z\cdot x_{100}=z$$
$$\textit{стоп}$$

Процесс вычислений имеет циклический характер. Однако, в отличие от задачи вычисления x^{100}, здесь нет повторяющихся одинаковых команд, а есть 100 выполняющихся подряд различных команд, хотя и схожих между собой.

Если бы мы, точно копируя способ образования цикла при вычислении x^{100}, сто раз повторили выполнение команды

$$z \cdot x_1 = z,$$

то вместо произведения $x_1 \cdot x_2 \cdot \ldots \cdot x_{100}$ получили бы степень x_1^{100}.

Для того чтобы правильно образовать цикл в рассматриваемой задаче, мы должны иметь возможность при каждом его повторении изменять команду

$$z \cdot x_1 = z,$$

заменяя в ней последовательно адрес x_1 на адрес x_2, x_2 на $x_3, \ldots, x_{99}$ на x_{100}, т. е. увеличивая на каждом шаге цикла второй адрес команды на единицу.

В результате такого изменения будут последовательно образовываться и выполняться такие команды $z \cdot x_1 = z$, $z \cdot x_2 = z, \ldots, z \cdot x_{100} = z$ и, в конце концов, в ячейке z окажется искомое произведение

$$x_1 \cdot x_2 \cdot \ldots \cdot x_{100}.$$

Важной особенностью электронных вычислительных машин является возможность изменения команд самой машиной. Такое изменение оказывается необходимым как в рассмотренном простом случае, так и во многих более сложных задачах.

Обозначим через (0, 1, 0) слово, у которого в коде, первом и третьем адресах находятся нули, а во втором адресе единица, и через K команду:

$$K: z \cdot x_1 = z.$$

Выполним команду

$$K +, (0, 1, 0) = K.$$

По этой команде к адресам z, x_1, z слóва, находящегося в ячейке (K), прибавляются соответственно адреса 0, 1, 0

сло́ва ячейки (0, 1, 0), и результат с кодом ячейки K записывается в ту же ячейку. В результате в ячейке K оказывается уже преобразованная команда

$$z \cdot x_2 = z.$$

Таким образом, происходит увеличение на единицу второго адреса команды K, т. е. ее *переадресация*; слово (0, 1, 0) называют *константой переадресации*.

Возвратимся теперь к задаче, сформулированной в начале параграфа.

Пример 1.24. Составим циклическую программу для вычисления величины

$$z = x_1 \cdot x_2 \cdot \ldots \cdot x_{100},$$

предполагая, что числа $x_1, x_2, \ldots, x_{100}$ расположены в ячейках памяти подряд. Проверку окончания цикла будем производить при помощи операции вычитания адресных частей.

Так как $100_{10} = 144_8$, то за начальное значение счетчика можно взять слово (0, 0, 144).

Подготовительную и рабочую части цикла запишем в виде

1) (0, 0, 144) —, (0, 0, 1) $= n$
2) 0 + «1» $= z$
3) z · x_1 $= z$

Для того чтобы при следующем повторении цикла в ячейке последовательно получать произведения $x_1 \cdot x_2$, $x_1 \cdot x_2 \cdot x_3$ и т. д., напишем команду увеличения на единицу второго адреса команды 3):

4) 3) +, (0, 1, 0) = 3)

Теперь нам нужно (условно) передать управление на рабочую часть цикла 3), повторив его 100 раз. Это делается тем же способом, что и ранее:

5) n —, (0, 0, 1) $= n$
6) *У0* 3)

Таким образом, циклическая программа для вычисления z имеет следующий вид:

1)		(0, 0, 144)	—,	(0, 0, 1)	$=n$
2)		0	+	«1»	$=z$
3)		z	·	x_1	$=z$
4)		3)	+,	(0, 1, 0)	$=3)$
5)		n	—,	(0, 0, 1)	$=n$
6)	*У0*				
7)	*стоп*				

Проследим, как работает составленная программа.

Команда 1) подготовляет счетчик цикла, а 2) заносит в z его первоначальное содержание — единицу. Команда 3) есть рабочая команда цикла. Сначала в ячейке 3) находится команда

$$3)\ z \cdot x_1 = z,$$

после выполнения которой в z оказывается величина x_1. Затем по команде переадресации 4) второй адрес команды 3) увеличивается на единицу, так что в ячейке 3) получается команда

$$3)\ z \cdot x_2 = z.$$

Затем происходит обычная проверка окончания арифметического цикла (команды 5), 6)) и управление опять переходит в 3). После вторичного выполнения команды 3) в ячейке z оказывается произведение $x_1 \cdot x_2$, затем команда 3) при помощи 4) опять переадресовывается, принимая вид $3)\ z \cdot x_3 = z$, после проверки окончания цикла 4), 5) и третьего выполнения команды 3) в z получаем величину $x_1 \cdot x_2 \cdot x_3$.

После того как цикл прокрутится 99 раз, в ячейке z окажется произведение $x_1 \cdot x_2 \cdot \ldots \cdot x_{99}$, команда 3) примет вид

$$3)\ z \cdot x_{100} = z,$$

а счетчик n станет равным нулю. При этом выработается сигнал $\omega = 0$ и команда 6) еще раз передаст управление на команду 3), после выполнения которой в z образуется искомое произведение. Затем команда 5) выработает сигнал $\omega = 1$ и по команде 6) машина выйдет на *стоп*.

Рассмотренная программа является простейшим примером арифметического цикла с переменными командами или *арифметического цикла с переадресацией*.

Команда $z \cdot x_1 = z$ в этой программе меняется с каждым повторением цикла, принимая в конце концов вид

$$z \cdot x_{100} = z.$$

Заметим, что по написанной программе можно произвести расчет величины

$$z = x_1 \cdot x_2 \cdot \ldots \cdot x_{100}$$

только один-единственный раз. Действительно, если мы после однократного вычисления z второй раз обратимся к этой программе, то в ячейке 3) будет находиться команда

$$z \cdot x_{100} = z$$

и программа не сможет правильно работать.

Однако в большинстве задач, решаемых на машинах, один и тот же участок программы должен работать многократно, выполняя одни и те же функции (вычисляя, например, произведение чисел, находящихся в ячейках $x_1, x_2, \ldots, x_{100}$). Поэтому программы должны писаться так, чтобы они были *самовосстанавливающимися* (см. определение на стр. 77). Это особенно важно для циклов, содержащих переменные команды, которые изменяются в процессе работы машины. *Переменные команды всегда должны восстанавливаться.*

В нашем примере мы могли бы достичь своей цели, записав где-либо, например, после команды *стоп*, начальное состояние команды 3) и поместив перед командой 7) команду засылки этого начального состояния в ячейку, отведенную для команды 3). Программа приняла бы тогда вид

	(0, 0, 144)	—,	(0, 0, 1)	$=n$
	0	+	«1»	$=z$
T	z	·	x_1	$=z$
	T	+,	(0, 1, 0)	$=T$
	n	—,	(0, 0, 1)	$=n$
УО				
	T_0	+,	0	$=T$
стоп				
T_0	z	·	x_1	$=z$

Рассмотренный способ называют *конечным восстановлением*: программа восстанавливается к первоначальному виду после ее работы. Однако такое восстановление не обеспечивает возможностей нормальной работы программы во всех случаях.

Дело в том, что при конечном восстановлении переменная команда восстановится только в том случае, если программа дойдет до конца. Если же выполнение программы будет прервано до окончания цикла по каким бы то ни было причинам, например, в результате случайного сбоя машины, то команда не восстановится и пустить ее сначала без нового ввода в память будет невозможно.

Чтобы обеспечить восстановление переменной команды во всех случаях, следует поместить команду $T_0 +, 0 = T$ в начале цикла, в группе команд подготовки цикла. Это есть частный случай общего правила, которым должен руководствоваться программист, имея дело с переменными командами: *восстановление переменной команды должно происходить до ее первого исполнения.*

Общая структура цикла с переадресацией ничем, собственно, не отличается от структуры обычного арифметического цикла. Как и там, чтобы начинать цикл с первого фактически используемого значения счетчика и в качестве эталона проверки брать число повторений цикла, удобно ставить переадресацию и изменение до рабочей части цикла и обходить их при первом выполнении. Программа, написанная по такой схеме, будет иметь следующий вид:

	0	$+$	«1»	$= z$	
	$(0,\ 0,\ 144)$	$-,$	$(0,\ 0,\ 1)$	$= n$	
Б	T_0			T	
	T	$+,$	$(0,\ 1,\ 0)$	$= T$	
T	$(z$	$\cdot$	x_1	$= z)$	н. п.
	n	$-,$	$(0,\ 0,\ 1)$	$= n$	
УО					
стоп					
T_0	z	$\cdot$	x_1	$= z$	

Заметим, что команда $z \cdot x_1 = z$ в ячейке T может и не быть написана, потому что при выполнении третьей команды

программы туда будет заслана команда T_0. Поэтому на месте команды T можно ничего не писать или писать все, что угодно. Обычно на месте такой команды пишут значок н. п., что означает *не перфорировать*. Впрочем, в левой части лучше написать первоначальное состояние команды (в скобках, чтобы показать, что ее не нужно перфорировать). В правой части ставится н. п. Машинное слово T_0 называют *восстановителем*. Восстановитель лучше помещать в конце программы, после *стопа*. Однако иногда его помещают перед командой переадресации таким образом, чтобы управление ему не передавалось, т. е. чтобы оно не воспринималось как команда; в нашем примере его можно было бы поместить между третьей и четвертой командами.

Пример 2.24. Вычислим значения $y_1, y_2, \ldots, y_{20}$ следующей функции

$$y=\frac{ax^2+bx+c}{dx^2+ex+f}$$

при $x = x_1, x_2, \ldots, x_{20}$.

Рабочая часть цикла программы для вычисления величин $y_1, y_2, \ldots, y_{20}$ имеет такой вид:

1) $a \cdot x_1 = R$		6) $S + e = S$	
2) $R + b = R$		7) $x_1 \cdot S = S$	
3) $x_1 \cdot R = R$		8) $S + f = S$	знаменатель
4) $R + c = R$	числитель	9) $R : S = y_1$	
5) $d \cdot x_1 = S$			

и содержит, как легко видеть, пять переменных команд 1), 3), 5), 7), 9). На восстановление, переадресацию и команды-восстановители циклической программы с написанной рабочей частью потребуется дополнительно 15 ячеек памяти (по три ячейки на каждую переменную команду). Однако для этой и подобных программ можно упростить циклы следующим способом.

Введем стандартную ячейку x, в которую будем засылать поочередно значения аргумента $x_1, x_2, \ldots, x_{20}$. Полученные в стандартной ячейке y значения функции будем затем последовательно пересылать в ячейки $y_1, y_2, \ldots, y_{20}$.

Тогда цикл будет содержать лишь две переменных команды:

засылка в стандартную ячейку $0+x_1=x$ и пересылка из стандартной ячейки $0+y=y_1$. Программа запишется в виде

		(0, 0, 24)	—,	(0, 0, 1)	$=n$	
		X_0	+,	0	$=X$	
	Б	Y_0			Y	
		X	+	(0, 1, 0)	$=X$	
		Y	+	(0, 0, 1)	$=Y$	
X		(0	+	x_1	$=x$)	н. п.
		a	·	x	$=R$	
		R	+	b	$=R$	
		x	·	R	$=R$	
		R	+	c	$=R$	
		d	·	x	$=S$	
		S	+	e	$=S$	
		S	·	x	$=S$	
		S	+	f	$=S$	
Y		R	:	S	$=y$	
		(0	+	y	$=y_1$)	н. п.
		n	—,	(0. 0, 1)	$=n$	
	У0					
	стоп					
X_0		0	+	x_1	$=x$	
Y_0		0	+	y	$=y_1$	

Заметим, что эту программу можно сократить на одну ячейку, совместив пересылку y в y_1 с условной передачей управления. Разумеется, при этом придется несколько изменить восстановитель. Предоставляем читателю сделать это самостоятельно.

Рассмотренный в этом примере прием засылки в стандартные ячейки и пересылки из стандартных ячеек широко применяется при программировании.

Команды сложения и вычитания адресных частей применялись в рассмотренных циклических программах для образования счетчиков и переадресации переменных команд. В большинстве случаев массивы для исходных данных и результатов счета цикла расположены в идущих подряд ячейках. В этих случаях изменение каждого из адресов переменной команды происходит на единицу. Поэтому наиболее часто применяют следующие семь констант переадресации (0, 0, 1); (0, 1, 0); (0, 1, 1); (1, 0, 0); (1, 0, 1); (1, 1, 0); (1, 1, 1).

Эти константы, называемые *стандартными константами переадресации*, целесообразно всегда вводить в память машины, а не присоединять к каждой отдельной программе. Обычно их размещают так, чтобы последняя цифра адреса ячейки совпадала с восьмеричной цифрой, которая получится, если константу переадресации прочитать как триаду. Например, константа (1, 0, 1), прочитанная как триада, означает цифру 5, поэтому ее помещают в ячейку, адрес которой заканчивается пятеркой. Это облегчает запоминание расположения констант переадресации.

Мы будем предполагать, что они размещены в следующих ячейках памяти:

(0, 0, 1)	7721	(1, 0, 1)	7725
(0, 1, 0)	7722	(1, 1, 0)	7726
(0, 1, 1)	7723	(1, 1, 1)	7727
(1, 0, 0)	7724		

Используя эти адреса для констант переадресации, закодируем программу последнего примера (табл. 1.24). Команды программы расположим, начиная с ячейки 4000. В памятку (табл. 2.24) внесем обозначения рабочих ячеек R, S, x, y, счетчика n, исходных данных a, b, c, d, e, f, x_1, x_2,..., x_{20} и результатов счета y_1, y_2,..., y_{20}. Внесем в этот бланк также метки X,Y переадресуемых команд, команд-констант X_0, Y_0 и обозначение начального значения восьмеричного счетчика (0, 0, 24).

Команды, которым передается управление, в этой программе не нужно снабжать метками, ибо передачи управления обозначены стрелками. Второй адрес команды передачи управления в этом случае кодируется адресом команды,

Таблица 1.24

						4000	*A*	
	$(0, 0, 24) -, (0, 0, 1) = n$	4000	0	33	4025	7721	4030
	$X_0 +, \quad 0 = X$	1	0	13	4023	0000	4005
	Б $Y_0 \quad \vert \quad Y$	2	0	56	4024	4005	4017
	$X + (0, 1, 0) = X$	3	0	13	4005	7722	4005
	$Y + (0, 0, 1) = Y$	4	0	13	4017	7721	4017
X	$(0 + x_1 = x)$ (н. п)	5	0	77	0000	0000	0000
	$a \cdot x = R$	6	0	05	4070	4033	4031
	$R + b = R$	7	0	01	4031	4071	4031
	$x \cdot R = R$	4010	0	05	4033	4031	4031
	$R + c = R$	1	0	01	4031	4072	4031
	$d \cdot x = S$	2	0	05	4073	4033	4032

						4013	*A*	
	$S + e = S$	4013	0	01	4032	4074	4032
	$S \cdot x = S$	4	0	05	4032	4033	4032
	$S + f = S$	5	0	01	4032	4075	4032
	$R : S = y$	6	0	04	4031	4032	4034
Y	$(0 + y = y_1)$ (н. п)	7	0	77	0000	0000	0000
	$n -, (0, 0, 1) = n$	4020	0	33	4030	7721	4030
	У0	1	0	76	0000	4003	0000
	стоп	2	0	77	0000	0000	0000
X_0	$0 + x_1 = x$	3	0	01	0000	4041	4033
Y_0	$0 + y = y_1$	4	0	01	0000	4034	4101
	$(0, \quad 0, \quad 24)$	5	0	00	0000	0000	0024

Таблица 2.24

4000	4040	4100	4140
4001	4041 x_1	4101 y_1	4141
4002	4042 x_2	4102 y_2	4142
4003	4043 x_3	4103 y_3	4143
4004	4044 x_4	4104 y_4	4144
4005 X	4045 x_5	4105 y_5	4145
4006	4046 x_6	4106 y_6	4146
4007	4047 x_7	4107 y_7	4147
4010	4050 x_8	4110 y_8	4150
4011	4051 x_9	4111 y_9	4151
4012	4052 x_{10}	4112 y_{10}	4152
4013	4053 x_{11}	4113 y_{11}	4153
4014	4054 x_{12}	4114 y_{12}	4154
4015	4055 x_{13}	4115 y_{13}	4155
4016	4056 x_{14}	4116 y_{14}	4156
4017 Y	4057 x_{15}	4117 y_{15}	4157
4020	4060 x_{16}	4120 y_{16}	4160
4021	4061 x_{17}	4121 y_{17}	4161
4022	4062 x_{18}	4122 y_{18}	4162
4023 X_0	4063 x_{19}	4123 y_{19}	4163
4024 Y_0	4064 x_{20}	4124 y_{20}	4164
4025 (0, 0, 24)	4065	4125	4165
4026	4066	4126	4166
4027	4067	4127	4167
4030 n	4070 a	4130	4170
4031 R	4071 b	4131	4171
4032 S	4072 c	4132	4172
4033 x	4073 d	4133	4173
4034 y	4074 e	4134	4174
4035	4075 f	4135	4175
4036	4076	4136	4176
4037	4077	4137	4177

Программа

который стоит в правой части в одной строке с концом стрелки. Константа (0, 0, 24) в правой части записывается так:

000 0000 0000 0024

Команды в этой программе кодируются тем же способом, что и в ранее рассмотренных случаях.

Сделаем еще одно дополнительное замечание. Как уже было указано, восстанавливаемые команды можно либо совсем не перфорировать, либо вводить туда любое содержимое. Очень удобно вводить на место восстанавливаемой команды команду *стоп*. Если вследствие ошибки программиста переменная команда не восстановится, то, дойдя до этой команды, машина остановится, указав тем самым на необходимость исправить ошибку.

Проще всего условиться, что знак н. п. в правой части означает, что нужно перфорировать

077 0000 0000 0000,

т. е. команду *стоп*. Обычно так и поступают. Если же это невозможно, то при кодировке в правой части кодируют *стоп*, а знак н. п. оставляют в левой части программы, однако не на месте первоначального состояния команды, а рядом с ним (см. табл. 1.24).

В приведенных примерах переадресации переменных команд цикла команды сложения адресных частей работали так, что сложение производилось поадресно. Заметим, что так будет только тогда, когда при поадресном сложении третьих адресов не произойдет переноса во второй адрес, или вторых адресов — в первый адрес (мантиссы складываются не поадресно, а как целые числа). Аналогичное замечание относится и к операции вычитания адресных частей.

Иногда при переадресации полезно использовать указанную особенность команд сложения и вычитания адресных частей. Приведем пример такого рода.

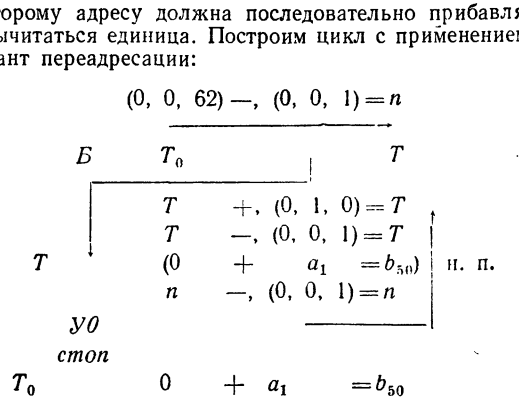
Пример 3.24. В идущих подряд ячейках памяти находятся числа последовательности $a_1, a_2, a_3, \ldots, a_{50}$. Переставить эти числа в обратном порядке, поместив их в ячейки $b_1, b_2, b_3, \ldots, b_{50}$. Единственной рабочей переменной командой цикла в этом примере является, очевидно, команда пересылки; которая сначала имеет вид

$$0+a_1=b_{50}.$$

Затем эта команда должна последовательно принимать вид

$$0+a_2=b_{49},$$
$$0+a_3=b_{48},$$
$$\ldots\ldots\ldots$$
$$0+a_{50}=b_1,$$

т. е. ко второму адресу должна последовательно прибавляться, а из третьего вычитаться единица. Построим цикл с применением стандартных констант переадресации:



Здесь цикл содержит пять команд и вся программа для своего выполнения требует 251 операцию. Постараемся теперь подобрать такую (нестандартную) константу переадресации, чтобы переадресовывать переменную команду цикла в одну операцию, а не в две.

Третий адрес команды T нам нужно уменьшать на 1. Покажем, что это можно сделать, прибавляя к третьему адресу восьмеричное число 7777_8.

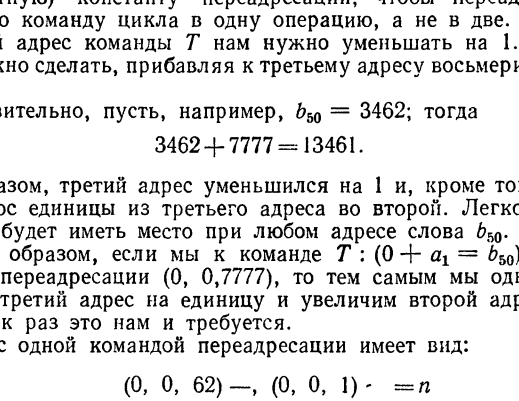
Действительно, пусть, например, $b_{50} = 3462$; тогда

$$3462 + 7777 = 13461.$$

Таким образом, третий адрес уменьшился на 1 и, кроме того, произошел перенос единицы из третьего адреса во второй. Легко показать, что то же будет иметь место при любом адресе слова b_{50}.

Таким образом, если мы к команде T : $(0 + a_1 = b_{50})$ прибавим константу переадресации (0, 0,7777), то тем самым мы одновременно уменьшим третий адрес на единицу и увеличим второй адрес на единицу, а как раз это нам и требуется.

Цикл с одной командой переадресации имеет вид:



Теперь для решения задачи требуется выполнить 202 операции вместо 251 в первом варианте программы.

Рассмотренный пример показывает, что при помощи операции сложения мантисс можно не только увеличивать, но и уменьшать адреса переменных команд. Делается это специальным подбором констант переадресации.

Часто встречающуюся константу 7777_8 мы будем обозначать буквой F и вместо (0, 0, 7777) писать (0, 0, F).

Можно составить предыдущий цикл, применяя для переадресации одну операцию вычитания адресных частей. Для этого следует в качестве константы переадресации использовать слово (7777, 7777, 1), или, в других обозначениях, (F, F, 1). Рекомендуем читателю самостоятельно убедиться в этом.

Все рассмотренные в этом параграфе программы циклов содержали переменные команды. Присутствие таких команд потребовало введения в цикл команд переадресации, начального восстановления, специальных констант переадресации и команд-восстановителей.

Как уже было сказано, общая структура цикла с переадресацией принципиально не отличается от структуры обычного арифметического цикла. Блок-схема цикла с переадресацией показана на рис. 24.

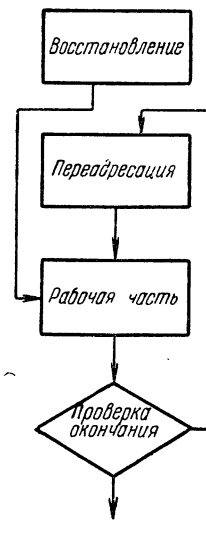


Рис. 24.

§ 25. Индексный регистр (регистр адреса). Операции с регистром адреса

Характерной особенностью рассмотренных нами циклических вычислительных процессов является работа с числовыми последовательностями.

Так, в примерах циклов без переадресации мы встречались с последовательностями целых степеней числа x (§ 13), частичных сумм степенного ряда (§ 14) и т. д.

В примерах циклов с переадресацией (§ 24) подсчитывалось произведение членов последовательности $x_1, x_2, \ldots, x_{100}$, из членов последовательности $x_1, x_2, \ldots, x_{20}$ образовывались члены последовательности

$$y_i = \frac{ax_i^2 + bx_i + c}{dx_i^2 + bx_i + f} \quad (i = 1, 2, \ldots, 20) \qquad \text{и т. д.}$$

Отметим, что в программах циклов без переадресации величины членов каждой последовательности помещались в одну ячейку памяти. С другой стороны в программе циклов с переадресацией для каждого члена последовательности выделялась отдельная ячейка. Это приводило к тому, что при работе цикла приходилось переадресовывать команды, содержащие адреса членов последовательности. Обычно члены последовательности нумеруются целыми числами от 0 до n или от 1 до n, причем номер члена обозначается индексом. Выясним связь между значениями индекса и адресами членов последовательности в памяти.

Пусть нам дана числовая последовательность x_0, x_1, x_2... ..., x_n. Поместим члены этой последовательности в идущие подряд ячейки памяти и обозначим адреса этих ячеек через

$$«x_0», \quad «x_1», \quad «x_2», \ldots, «x_n»*).$$

Эти адреса связаны со значениями индекса очевидными соотношениями:

$$\begin{aligned} «x_1» &= «x_0» + 1, \\ «x_2» &= «x_0» + 2, \\ &\ldots\ldots\ldots \\ «x_j» &= «x_0» + j, \\ &\ldots\ldots\ldots \\ «x_n» &= «x_0» + n. \end{aligned}$$

Таким образом, адрес произвольного члена последовательности x_j определяется адресом начального члена x_0 и значением индекса j. Из того, что вся последовательность должна умещаться в памяти машины, следует, что j есть двоичное число, меньшее 2^{12}. Таким образом, для *записи любого индекса достаточно одного адреса машинного слова.*

---

*) Ранее мы обозначали адрес ячейки, содержащей некоторую величину, той же буквой, что и эту величину. Исключение делалось для ячеек, содержащих известные и никакой буквой в программе не обозначенные числа. Адреса таких ячеек мы обозначали теми же числами, взятыми в кавычках. Например, «1» означало «адрес ячейки, в которой лежит число 1». Так как в данном случае важны не числа x_0, x_1, ..., а адреса ячеек, в которых они лежат, то мы и здесь пользуемся таким обозначением. Как и для чисел, «x_0» означает адрес ячейки, в которой лежит число x_0.

Для облегчения работы с числовыми последовательностями, члены которых занумерованы при помощи индексов, в машине имеется *индексный регистр*.

Индексный регистр, называемый также *регистром адреса*, — это устройство, в которое можно поместить любое двоичное число длиной в один адрес (например, индекс).

В рассматриваемой трехадресной машине регистр адреса состоит из 12 разрядов и является частью устройства управления. Двенадцатиразрядное двоичное число, находящееся в этом регистре, так же как и сам этот регистр, мы будем обозначать буквами PA.

Регистр адреса применяется при программировании в основном для образования адресов членов числовой последовательности. Предположим, что нам нужно образовать адрес «x_j» члена последовательности x_j.

Так как

$$«x_j» = «x_0» + j,$$

то для этого следует в регистр адреса занести значение индекса j, а затем к адресу начального числа последовательности прибавить PA. Для этой операции — добавления PA к адресу машинного слова x_0 — мы введем специальное обозначение:

$$«x_0»^* = «x_0» + PA.$$

Очевидно, если $PA = 0$, то «x_0» * = «x_0»; если $PA = 0001$, то «x_0» * = «x_0» + 1 = «x_1»; если $PA = j$, то «x_0» * = = «x_0» + j = «x_j».

Таким образом, звездочный адрес «x_0»* при соответствующем значении PA можно превратить в адрес любого члена последовательности.

Посмотрим, как выполняется команда со звездочным адресом на примере команды

$$K: \quad «1» + x_0{}^* = a.$$

Пусть перед выполнением этой команды $PA = 0003$. Тогда «x_0» * = «x_0» + 0003 = «x_3», и команда будет выполняться в виде

$$K^{(1)}: \quad «1» + x_3 = a.$$

Как же происходит переадресация (переход от K к $K^{(1)}$ и выполнение команды $K^{(1)}$)?

В памяти машины команда находится в виде K. В тот момент, когда управление передается на K, эта команда «переносится» из памяти в устройство управления машины, где и происходит увеличение помеченного звездочкой адреса на значение PA. Таким образом, оставляя команду K в памяти неизменной, машина производит переадресацию (изменение адреса) непосредственно в *управлении*.

Прибавление PA к адресам команды мы будем называть *переадресацией по PA*.

Переадресовать по PA можно один, два или все три адреса команды. Так, например, команда

$$x_0^* \cdot x_1^* = c$$

при $PA = 0004$ исполняется в виде

$$x_4 \cdot x_5 = c.$$

Из изложенного видно, что при переадресации по PA нет необходимости в начальном восстановлении переадресуемой команды, ибо эта команда в памяти машины не меняется. Как мы увидим далее, это обстоятельство дает возможность значительно упростить построение циклов с переадресацией.

Рассмотрим теперь вопрос о том, как изображаются в ячейке памяти машины команды, использующие регистр адреса, т. е. содержащие звездочные адреса.

Для указания звездочных адресов в машинном слове, изображающем команду, используются 45-й, 44-й и 43-й разряды. Если звездочным является первый (левый) адрес, то в 45-м (левом) разряде ставится единица, если второй (средний) адрес — звездочный, то единицу ставят в 44-м (среднем) разряде, третьему (правому) звездочному адресу соответствует единица в 43-м (правом) разряде.

Разряды 45-й, 44-й и 43-й обозначаются соответственно через π_1, π_2, π_3 и называются *признаками изменения адресов команды по PA* (или просто *признаками*). Таким образом, машинное слово, используемое как команда, разбивается на части по схеме, указанной на рис. 25.

При записи команды в ячейке эти три признака объединяются в одну восьмеричную цифру, которая и ставится впереди кода операции. Это и есть та самая цифра, которую мы до сих пор полагали равной нулю.

Обозначим через A_1, A_2, A_3 адреса команды в памяти машины *(действительные адреса)*, а через A_1', A_2', A_3' — адреса той же команды при ее исполнении в устройстве управления *(исполнительные адреса)*. Из определения звездочных адресов следует, что

$$A_1' = A_1 + \pi_1 PA,$$
$$A_2' = A_2 + \pi_2 PA,$$
$$A_3' = A_3 + \pi_3 PA.$$

Отсюда видно, что если $\pi_1 = \pi_2 = \pi_3 = 0$ или если $PA = 0$, то исполнительные адреса совпадают с действительными и команда исполняется в том виде, в каком она записана в ячейке памяти. Если же хотя бы один из признаков π_1, π_2, π_3 отличен от нуля и $PA \neq 0$, то команда при исполнении в устройстве управления переадресуется по PA.

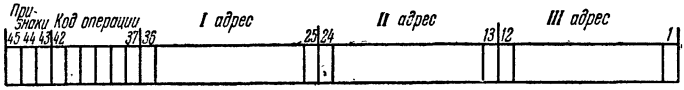


Рис. 25.

При образовании исполнительных адресов по предыдущим формулам может произойти переполнение двенадцатиразрядной сетки соответствующего адреса; в этом случае образовавшийся тринадцатый разряд не переносится в следующий слева адрес, а теряется. Например, если звездочный адрес команды равен 4574, а $PA = 5312$, то соответствующий исполнительный адрес будет равен 2106.

Переадресация по PA применяется главным образом при образовании циклов с переменными командами. В качестве примера такого цикла рассмотрим следующую простую задачу.

Пусть имеются две последовательности

$$u_0, u_1, u_2, \ldots, u_n$$

и

$$v_0, v_1, v_2, \ldots, v_n$$

и требуется образовать новую последовательность

$$w_0, w_1, w_2, \ldots, w_n,$$

попарно перемножая члены двух исходных,

$$w_0 = u_0 \cdot v_0,$$
$$w_1 = u_1 \cdot v_1,$$
$$w_2 = u_2 \cdot v_2,$$
$$\ldots\ldots$$
$$w_n = u_n \cdot v_n.$$

Произвольный j-й член последовательности определяется формулой

$$w_j = u_j \cdot v_j.$$

Для его вычисления в машине следует занести в регистр адреса значение индекса j, а затем выполнить команду

$$u_0^* \cdot v_0^* = w_0^*.$$

Для того чтобы получить все члены последовательности w_0, w_1, w_2,..., w_n, следует эту команду включить в цикл, во время работы которого PA должен принимать последовательно значения 0000, 0001, 0002, ... n. Цикл следует кончать тогда, когда PA примет значение n.

Из рассмотренного примера видно, что для образования цикла с переадресацией по PA необходимо иметь возможность:

1) заносить в регистр адреса любое двенадцатиразрядное двоичное число,

2) изменять значение PA,

3) сравнивать PA с заданным 12-разрядным эталоном и, в зависимости от результата сравнения, передавать управление в ту или иную ячейку памяти.

Для работы с регистром адреса в машине предусмотрены специальные команды. Две из них предназначены для первоначального занесения в регистр нужного содержимого; восемь других — для сравнения содержимого регистра с эталоном и передачи управления.

Первую из этих команд мы будем обозначать так:

$$PA \quad \overline{b}.$$

При выполнении этой команды машина заносит в регистр адреса фиксированное двенадцатиразрядное двоичное число b, которое записывается во втором адресе команды. Чер-

точку над буквой b мы пишем, чтобы показать, что при кодировке в среднем адресе нужно записать то самое число, которое стоит в содержательной части программы (разумеется, в восьмеричной форме). Первый и третий адреса этой команды будем п о к а полагать нулёвыми. Об их использовании речь будет идти в следующем параграфе.

Здесь мы впервые встречаемся с командой, адрес в которой не означает адреса ячейки памяти, а некоторое условное число. С такими командами нам еще придется встречаться в дальнейшем, и не только при работе с регистром адреса.

Команда

$$PA \quad 0$$

очищает регистр адреса (заносит в него нуль), а команда

$$PA \quad \overline{3}$$

заносит в регистр адреса число 0003. После выполнения двух команд

$$PA \quad \overline{3}$$
$$u_0^* \cdot v_0^* = w_0^*$$

в регистр адреса будет занесено число 0003, а затем в ячейку w_3 произведение $u_3 \cdot v_3$.

Рассматриваемая команда может применяться не только для занесения в PA нужного содержимого, но и для его изменения. Например, команда

$$PA \quad \overline{1}^*$$

будет выполняться так: звездочный второй адрес при поступлении в устройство управления преобразуется в исполнительный адрес, равный

$$0001^* = 0001 + PA,$$

поэтому в регистр адреса занесется старое значение PA, увеличенное на единицу.

Часто бывает необходимо занести в регистр адреса число, неизвестное программисту в момент написания программы, а записанное в некоторой ячейке. Для этой цели используется команда

$$[PA] \quad b.$$

При выполнении этой команды число b, записанное в ее втором адресе, рассматривается как адрес некоторой ячейки и в регистр заносится двенадцатиразрядное двоичное число, содержащееся во втором адресе этой ячейки b.

Например, команда

$$[PA] \quad 0$$

заносит в регистр нуль, так как в нулевой ячейке всегда записано нулевое слово. Команда

$$[PA] \quad b,$$

если в ячейке b записано число (0, 1, 0), занесет в регистр число 0001. Левый и правый адреса этой команды, как и предыдущей, мы будем п о к а считать нулевыми.

Рассмотренные команды используются, главным образом, для подготовки цикла, занося в начале его в регистр нужное содержимое. Две другие команды, о которых сейчас будет идти речь, предназначены для сравнения содержимого регистра адреса с эталоном и передачи управления. Кроме того, они дают возможность одновременно менять содержимое регистра адреса. Эти команды обычно применяются в качестве команд проверки окончания цикла.

Первую из них мы будем записывать в виде

$$PA < \bar{a} \qquad b \qquad \bar{c},$$

где a и c — целые двенадцатиразрядные двоичные числа, а b — адрес некоторой ячейки памяти. Команда выполняется так: проверяется справедливость написанного слева неравенства, т. е. содержимое PA сравнивается с числом $\bar{a}$. Если неравенство $PA < \bar{a}$ справедливо, то управление передается ячейке b. Если, наоборот, $PA \geqslant \bar{a}$, то управление передается следующей ячейке. Одновременно с передачей управления в регистр адреса заносится число c, стоящее в третьем адресе выполняемой команды.

Как правило, эта команда используется несколько иначе, а именно, в виде

$$PA < \bar{a} \qquad b \qquad \bar{c}^*.$$

При ее выполнении третий звездочный адрес переадресуется в управлении машины на прежнее содержимое PA, поэтому

после выполнения команды в регистр адреса запишется число $c^* = c + PA$, т. е. *содержимое регистра адреса увеличится на с двоичных единиц.*

Вторая команда для работы с регистром адреса имеет вид

$$PA \geqslant \bar{a} \quad b \quad \bar{c}$$

и выполняется точно так же, как и предыдущая, с той только разницей, что проверяется справедливость другого неравенства. Управление передается ячейке b, если $PA \geqslant \bar{a}$, и следующей ячейке, если $PA < \bar{a}$. Как и предыдущая команда, она чаще всего используется в виде

$$PA \geqslant \bar{a} \quad b \quad \bar{c}^*,$$

так что после ее выполнения к содержимому регистра адреса прибавляется число $\bar{c}$.

Учитывая, что при переполнении разрядной сетки регистра адреса образовавшийся тринадцатый разряд теряется, можно подобрать число $\bar{c}$ так, чтобы содержимое регистра не увеличивалось, а у м е н ь ш а л о с ь. Например, чтобы при проверке окончания содержимое регистра адреса уменьшилось на две единицы, достаточно выполнить команду

$$PA \geqslant \bar{a} \quad b \quad \overline{7776}^*.$$

Читатель легко убедится в справедливости этого утверждения.

Наряду с рассмотренными командами, употребляющимися очень часто, в машине имеется еще одна группа команд проверки окончания цикла, в которых проверка окончания происходит не только по содержимому PA, но одновременно и по управляющему сигналу ω. Эти команды обозначаются так:

$$\begin{array}{llll} (0)\ PA < \bar{a} & b & \bar{c}, \\ (1)\ PA < \bar{a} & b & \bar{c}, \\ (0)\ PA \geqslant \bar{a} & b & \bar{c}, \\ (1)\ PA \geqslant \bar{a} & b & \bar{c}. \end{array}$$

Передача управления ячейке b происходит здесь при одновременном выполнении двух условий: справедливости написанного неравенства и совпадения управляющего сигнала, выработанного в результате

выполнения предыдущей операции, с указанным в команде. Так, команда

$$(0)\ PA < \overline{1} \qquad b \qquad \overline{c}$$

передаст управление ячейке b, если $PA = 0$ и после предыдущей операции выработался сигнал $\omega = 0$. Если же $\omega = 1$ или $PA \geqslant \overline{1}$, то управление передается следующей ячейке. Примеры применения этих команд будут приведены в следующих параграфах.

Во всех приведенных выше командах содержимое PA сравнивается с написанным в самой команде эталоном. Это вполне удобно для арифметических циклов, но может представить неудобства в том случае, когда значение эталона заранее (при написании программы) неизвестно. Чтобы обойти указанные затруднения, на некоторых машинах сделаны еще две команды *), позволяющие сравнивать текущее значение PA с эталоном, записанным не в самой команде, а в некоторой ячейке памяти. Эти команды обозначаются

$$[PA] < a \qquad b \qquad \overline{c},$$
$$[PA] \geqslant a \qquad b \qquad \overline{c}.$$

При выполнении каждой из них текущее значение PA сравнивается со вторым адресом a_2 ячейки a. Если указанное неравенство справедливо (одновременно предполагается, что a_1 и a_3, т. е. первый и третий адреса ячейки a, содержащей эталон, являются нулевыми), то управление передается ячейке b, если нет, то следующей ячейке. Кроме того, в PA заносится исполнительный третий адрес, т. е. число c, либо старое значение PA, увеличенное на c, если третий адрес в команде звездочный.

Таблица 1.25

Обозначение команды	Код
$PA \quad \overline{a} \quad \overline{b} \quad c$	52
$[PA] \quad \overline{a} \quad b \quad c$	72
$PA < \overline{a} \quad b \quad \overline{c}$	12
$PA \geqslant \overline{a} \quad b \quad \overline{c}$	32
(1) $PA < \overline{a} \quad b \quad \overline{c}$	11
(1) $PA \geqslant \overline{a} \quad b \quad \overline{c}$	31
(0) $PA < \overline{a} \quad b \quad \overline{c}$	51
(0) $PA \geqslant \overline{a} \quad b \quad \overline{c}$	71
$[PA] < a \quad b \quad \overline{c}$	40
$[PA] \geqslant a \quad b \quad \overline{c}$	60

Коды операций с регистром адреса приводятся в таблице 1.25. Заметим, что команды занесения в регистр мы пишем здесь со всеми тремя адресами, хотя левый и правый адреса будем пока считать нулевыми. Об их использовании речь будет идти в § 27.

*) Эти две команды сделаны на машинах БЭСМ-4 и М-220. На серийных машинах М-20 этих команд нет.

§ 26. Использование регистра адреса при программировании

Рассмотрим примеры применения регистра адреса при программировании циклов. Самым простым является применение регистра адреса для арифметических циклов с искусственными счетчиками.

Пример 1.26. Рассмотрим циклическую программу для нахождения $z = x^{100}$, которую мы уже разбирали (см. пример 1. 13). Используя регистр адреса в качестве счетчика, мы можем написать такую программу:

1)	PA			0	
2)		0	$+$	«1»	$=z$
3)		z	$\cdot$	x	$=z$ ↑
4)	$PA < \overline{143}$				$\overline{1}$*
5)	*стоп*				

Этот цикл не содержит переменных команд, и все адреса команд, кроме правого адреса команды 4), не являются звездочными, т. е. выполняются точно так, как написаны. Стрелка, проведенная из среднего адреса команды 4) вверх, показывает, что при выполнении условия $PA < \overline{143}$ управление передается команде 3).

Программа работает так: команды 1) и 2) являются командами подготовки цикла. Команда 1) очищает регистр адреса, а команда 2) заносит в ячейку z первоначальное содержимое. Команда 3) составляет рабочую часть цикла. После ее первого выполнения в ячейке z будет находиться x, а $PA = 0$. Неравенство, написанное слева в команде 4), будет выполнено, и команда 4) передаст управление команде 3). Одновременно содержимое регистра адреса увеличится на 1, т. е. будет $PA = 0001$.

Каждый раз при выполнении команды 4) к содержимому PA будет прибавляться единица. Таким образом, команда 3) будет выполняться при $PA = 0, 1, 2, \ldots, 143$, т. е. $144_8 = 100_{10}$ раз. При $PA = 143$ неравенство, написанное в команде 4), не выполнится, и команда 4) передаст управление уже не команде 3), а команде 5), которая и остановит машину. Заметим, между прочим, что прибавление единицы в регистр

все равно произойдет, так что после окончания цикла содержимое регистра адреса будет $PA = 0144$.

Использование регистра адреса оказалось здесь очень удобным по ряду причин. Во-первых, команда 4) одновременно выполняет три обязанности, для которых раньше приходилось расходовать две или три команды: сравнение счетчика с эталоном, изменение счетчика и передачу управления рабочей части. Во-вторых, отпала необходимость иметь специальные ячейки для хранения эталона и константы изменения счетчика, так как эти числа пишутся в самой команде.

В следующих примерах регистр адреса используется уже не только как счетчик, но и для переадресации команд.

Пример 2.26. Перемножить две последовательности

$$u_0,\ u_1,\ u_2,\ \ldots,\ u_n$$

и

$$v_0,\ v_1,\ v_2,\ \ldots,\ v_n,$$

члены которых расположены в идущих подряд ячейках памяти. Полученную последовательность поместить в идущие подряд ячейки

$$w_0,\ w_1,\ \ldots,\ w_n.$$

Для решения задачи требуется выполнить последовательно $n + 1$ команд умножения:

$$\begin{aligned} u_0 \cdot v_0 &= w_0, \\ u_1 \cdot v_1 &= w_1, \\ &\ldots\ldots \\ u_n \cdot v_n &= w_n. \end{aligned}$$

Каждая следующая из этих команд может быть получена из предыдущей увеличением всех трех индексов на единицу. Поэтому для построения цикла в этом примере следует сначала выполнить две команды:

$$\begin{aligned} PA \quad & 0, \\ & u_0^* \cdot v_0^* = w_0^*, \end{aligned}$$

а затем n раз *) повторить выполнение второй из этих команд, увеличивая каждый раз PA на единицу.

*) Здесь n предполагается целым восьмеричным числом.

Осуществляется это при помощи следующей программы:

1) PA 0
2) $u_0^* \cdot v_0^* = w_0^*$
3) $PA < \overline{n}$ $\overline{1}^*$
4) *стоп*

Проследим, как работает эта программа. Сначала по команде 1) очищается регистр адреса. Так как $PA = 0$, то

$$«u_0»^* = «u_0», \quad «v_0»^* = «v_0», \quad «w_0»^* = «w_0»$$

и команда 2) исполняется в виде $u_0 \cdot v_0 = w_0$. Сравнение нулевого значения PA с первым адресом команды 3) приводит к передаче управления на 2), при этом одновременно в регистр адреса заносится третий исполнительный адрес команды 3):

$$\overline{1}^* = 0000 + 0001 = 0001.$$

Таким образом, на следующем шаге цикла $PA = 0001$ и в команде 2):

$$«u_0»^* = «u_0» + 1, \quad «v_0»^* = «v_0» + 1, \quad «w_0»^* = «w_0» + 1.$$

Поэтому команда 2), не изменяясь в памяти, фактически исполняется в виде

$$u_1 \cdot v_1 = w_1.$$

Проследив далее выполнение программы, легко убедиться в том, что на каждом следующем шаге цикла PA будет увеличиваться на единицу, а команда 2) будет последовательно исполняться в виде

$$u_2 \cdot v_2 = w_2,$$
$$u_3 \cdot v_3 = w_3,$$
$$\ldots\ldots$$

При этом цикл будет работать до тех пор, пока выполняется условие: $PA < \overline{n}$. Рассмотрим тот этап цикла, когда перед выполнением команды 3) имеем $PA = n - 1$. Тогда команда 3) передает управление на 2), а к PA прибавляется единица. При $PA = n$ команда 2) исполняется в виде

$$u_n \cdot v_n = w_n,$$

т. е. образуется последний член нужной последовательности. Так как $PA = n$, то выполняемая после 2) команда 3) передает управление на *стоп*, т. е. цикл заканчивается. Как и в предыдущем примере, к моменту окончания цикла PA оказывается равным $n + 1$, ибо команда 3) прибавит при последнем исполнении единицу к прежнему значению PA, равному n.

Рассматриваемую задачу можно запрограммировать, используя вместо регистра адреса переадресацию переменной команды: $u_0 \cdot v_0 = w_0$ при помощи константы (1, 1, 1). Самовосстанавливающаяся программа, составленная таким способом, имеет вид:

```
        (0, 0, n)+,      0    = v
Б       T0                    T
        T      +, (1, 1, 1) = T
T       (u0    ·      v0    = w0)   н. п.
        v      −, (0, 0, 1) = v
    У0
    стоп
T0      u0     ·      v0    = w0
```

Эта программа в два раза длиннее предыдущей и, кроме того, работает в два раза медленнее. Из рассмотренного примера видно, что применение регистра адреса для переадресации переменных команд в цикле позволяет значительно сократить программу и ускорить ее работу.

Это достигается благодаря следующим обстоятельствам:

а) переменная команда цикла в памяти остается неизменной, а переадресуется в устройстве управления, поэтому оказывается излишним начальное восстановление;

б) переадресация производится при помощи PA и, следовательно, устраняется необходимость держать в памяти константы переадресации и вводить в программу команды переадресации;

в) проверка окончания цикла происходит по значению PA, играющего роль счетчика цикла, поэтому нет нужды вводить специальный счетчик.

Пример 3.26. Найти силу F, с которой положительный заряд отталкивается $n + 1$ положительными зарядами $q_0, q_1, q_2, \ldots, q_n$, находящимися на луче, выходящем из заряда q, на расстояниях $r_0, r_1, r_2, \ldots, r_n$ от него.

По закону Кулона заряд q_j отталкивает заряд q с силой

$$F_j = k\frac{qq_j}{r_j^2}.$$

Поэтому равнодействующая всех сил отталкивания определяется формулой

$$F = kq\left(\frac{q_0}{r_0^2} + \frac{q_1}{r_1^2} + \ldots + \frac{q_n}{r_n^2}\right),$$

или

$$F = kq\,\Sigma,$$

где

$$\Sigma = \frac{q_0}{r_0^2} + \frac{q_1}{r_1^2} + \ldots + \frac{q_n}{r_n^2}.$$

Очевидно, Σ можно получить при помощи следующих формул:

$$\Sigma_0 = 0,$$
$$\Sigma_j = \Sigma_{j-1} + q_j/r_j^2 \quad (j = 1, \ldots, n),$$
$$\Sigma = \Sigma_n.$$

Мы будем предполагать, что число n задано в виде константы N (0, n, 0).

Приступая к составлению программы, очистим сначала ячейку Σ:

$$0 + 0 = \Sigma.$$

Для того чтобы из величины Σ_{j-1}, находящейся в ячейке Σ, получить Σ_j, достаточно выполнить три команды:

$$q_j : r_j = f$$
$$f : r_j = f$$
$$\Sigma + f = \Sigma$$

Для нахождения Σ следует эти команды включить в цикл, в котором j должно меняться от 0 до n. Заменим в этих командах адрес q_j на q_0^* и адрес r_j на r_0^*:

$$q_0^* : r_0^* = f$$
$$f : r_0^* = f$$
$$\Sigma + f = \Sigma$$

Изменяя PA от 0000 до n, мы получим цикл нахождения Σ:

$$\begin{array}{rl} & 0 + 0 = \Sigma \\ PA & 0 \\ & q_0^* : r_0^* = f \\ & f : r_1^* = f \\ & \Sigma + f = \Sigma \\ [PA] < N & \boxed{1^*} \end{array}$$

Присоединим к этим командам две команды получения F из Σ и *стоп*:

$$q \cdot \Sigma = F$$
$$k \cdot F = F$$
стоп

Заметим еще, что команду очистки ячейки Σ можно поместить после очистки регистра адреса.

Перепишем эту программу в левую часть программного бланка и закодируем (табл. 1.26).

Таблица 1.26

				4200	A	
PA 0 0 0	4200	0	52	0000	0000	0000
$0+0=\Sigma$	1	0	01	0000	0000	4212
$q_0^* : r_0^* = f$	2	6	04	4240	4300	4213
$f : r_0^* = f$	3	2	04	4213	4300	4213
$\Sigma + f = \Sigma$	4	0	01	4212	4213	4212
$[PA] < N$ $\overline{\vert\ \bar{1}^*}$	5	1	40	4215	4202	0001
$q \cdot \Sigma = F$	6	0	05	4220	4212	4213
$k \cdot F = F$	7	0	05	4221	4213	4213
стоп	4210	0	77	0000	0000	0000

Программу, исходные данные, результат F и рабочие ячейки распределим в памяти согласно памятке (табл. 2.26). В переадресуемой по PA команде 4202 признаки $\pi_1 = 1$, $\pi_2 = 1$, $\pi_3 = 0$. Объединяя три признака π_1, π_2, π_3 в триаду π и переводя ее затем в одну восьмеричную цифру, получим $\pi = 110_2 = 6_8$. Это значение π записываем в специально отведенную для признаков графу правой части бланка. Аналогично кодируются признаки в команде 4203 ($\pi_1 = 0$, $\pi_2 = 1$, $\pi_3 = 0$, $\pi = 010_2 = 2_8$) и в команде 4205 ($\pi_1 = 0$, $\pi_2 = 0$, $\pi_3 = 1$, $\pi = 001_2 = 1_8$).

Отметим, что составленную программу можно использовать для вычисления силы F при любом числе зарядов, не превышающем $40_8 = 32_{10}$ (для q_0, q_1, q_2, ..., r_0, r_1, r_2, ... выделены поля 4240—4277, 4300—4337). Так, например, в случае $n = 23_{10}$ достаточно в ячейку N (4215) поместить константу (0, 27, 0).

Таблица 2.26

4200	Программа		4240	q_0	4300	r_0
4201			4241	q_1	4301	r_1
4202			4242	q_2	4302	r_2
4203			4243	.	4303	.
4204			4244	.	4304	.
4205			4245	.	4305	.
4206			4246	.	4306	.
4207			4247	.	4307	.
4210			4250	.	4310	.
4211			4251	.	4311	.
4212	Σ		4252	.	4312	.
4213	f		4253	.	4313	.
4214	F		4254	.	4314	.
4215	N		4255	.	4315	.
4216			4256	.	4316	.
4217			4257	.	4317	.
4220	q		4260	.	4320	.
4221	k		4261		4321	
4222			4262		4322	

Пример 4.26. Вычислить сумму пятнадцати комплексных чисел

$$z_1,\ z_2,\ \ldots,z_{15}.$$

Поместим действительную часть числа z_1 в ячейку u_1, мнимую — в следующую ячейку v_1, действительную и мнимую части числа z_2 в следующие две ячейки u_2, v_2 и т. д. Тогда числа последовательности u_1, u_2,..., u_{15} будут расположены в ячейках памяти через одну, так что

$$«u_j»=«u_1»+2(j-1)\ \ (j=1,2,\ldots,\ 15). \qquad (1.26)$$

Аналогично для последовательности $v_1, v_2, \ldots, v_{15}$

$$«v_j» = «v_1» + 2(j-1). \tag{2.26}$$

Действительную и мнимую части суммы

$$z = z_1 + z_2 + \ldots + z_{15}$$

мы расположим в ячейках u и v. Тогда

$$\begin{aligned} u &= u_1 + u_2 + \ldots + u_{15}, \\ v &= v_1 + v_2 + \ldots + v_{15}. \end{aligned} \tag{3.26}$$

Приступая к составлению программы, очистим ячейки u и v:

1) $0 + 0 = u$,
2) $0 + 0 = v$.

Для получения u и v по формулам (3.26) следует в цикле для $j = 1, 2, 3, \ldots, 15$ выполнить команды

$$\begin{aligned} u + u_j &= u, \\ v + v_j &= v. \end{aligned}$$

На первом шаге цикла имеем

$$\begin{aligned} u + u_1 &= u, \\ v + v_1 &= v. \end{aligned}$$

Будем переадресовывать эти команды при помощи регистра адреса:

3) $PA \quad 0$
4) $\quad u + u_1^* = u$
5) $\quad v + v_1^* = v$

Согласно (1.26) и (2.26) в рассматриваемом примере, в отличие от двух предыдущих, PA на каждом шаге цикла следует увеличивать не на 0001, а на 0002. Кончать цикл нужно тогда, когда выполнится равенство

$$«u_1»^* = «u_{15}»,$$

а это будет при $PA = 2 \cdot 14_{10} = 2 \cdot 16_8 = 34_8$. Поэтому окончание цикла и изменение PA следует осуществлять так:

$$6)\ PA < \overline{34} \qquad 4) \qquad \overline{2}^*$$

Таким образом, программа получения z имеет вид

```
PA          0
          0 + 0 = u
          0 + 0 = v
          u + u1* = u   ↑
          v + v1* = v   |
PA < 34   |       2*    |
стоп
```

Рассмотренный пример характерен тем, что члены обрабатываемых последовательностей расположены в памяти не подряд (с шагом 1), а через одну ячейку (с шагом 2).

Пусть в цикле следует «переработать» все члены некоторой последовательности $a_1, a_2, \ldots, a_n$, причем в памяти эта последовательность расположена с шагом Δ. Цикл для такой последовательности следует организовывать по схеме

```
PA          0
        Рабочая часть   ↑
PA < Δ(n−1) |  Δ*       |
```

В трех рассмотренных примерах элементы последовательностей обрабатывались в порядке возрастания индексов, начиная с самого первого члена до последнего.

В некоторых задачах оказывается удобнее перебирать члены последовательности, начиная с конца. Если применять в этом случае для переадресации регистр адреса, то придется уменьшать адреса членов последовательности, т. е. в цикле уменьшать PA.

Пример 5.26. В последовательности комплексных чисел

$$z_1, z_2, \ldots, z_{25}$$

найти последнее отличное от нуля действительное число w и разделить все члены последовательности на это число. Задача здесь распадается на два этапа:

I) нахождение числа w,

II) образование последовательности:

$$\zeta_1 = z_1/w, \quad \zeta_2 = z_2/w, \quad \ldots, \quad \zeta_{25} = z_{25}/w.$$

Будем находить w, перебирая члены последовательности с конца. Пусть

$$z_1 = u_1 + iv_1, \quad z_2 = u_2 + iv_2, \quad \ldots, \quad z_{25} = u_{25} + iv_{25}$$

и ячейки памяти заняты в следующем порядке:

$$u_1,\ v_1,\ u_2,\ v_2,\ \ldots,\ u_{25},\ v_{25}.$$

Проверку условия: является ли число z_j действительным и отличным от нуля, можно осуществить при помощи команд:

```
1)'      |0|—|v_j|=0   4)'  У0       7)'
2)'  У1        7)'      5)'    0+  u_j  =w
3)'      |0|—|u_j|=0   6)'  Б      II этап
```

Эти команды следует включить в цикл, выполняя их для $j = 25, 24, 23, \ldots$ до тех пор, пока мы не выйдем из цикла по команде безусловной передачи управления. Для того чтобы производить переадресацию по $РА$, перепишем эти команды в виде:

```
1)''     |0|—|v_1*|=0   4)''  У0       7)''
2)''  У1       7)''      5)''    0+  u_1*  =w
3)''     |0|—|u_1*|=0   6)''  Б      II этап
```

На первом шаге цикла «v_1» * должно равняться «v_{25}»:

$$«v_1»^* = «v_{25}» = «v_1» + 2 \cdot 24_{10},$$

т. е. $РА = 2 \cdot 24_{10} = 48_{10} = 60_8$. Таким образом, перед началом цикла следует поставить команду

$$РА \quad \overline{60}.$$

На втором шаге цикла должно быть

$$«v_1»^* = «v_{2i}» = «v_1» + 2 \cdot 23_{10},$$

т. е. PA должен уменьшиться на 2. Такое уменьшение PA и передачу управления на начало цикла можно осуществить при помощи команды

$$7)''\ PA \geqslant 0 \qquad 1)'' \qquad \overline{7776}$$

Объединяя все выписанные команды и совмещая пересылку 5)'' с передачей управления 6)'', получим программу вычисления числа w:

0) PA	$\overline{60}$	4) *У0*	6)
1)	$\|0\|-\|v_1{}^*\|=0$	5) *Б*	$\overrightarrow{u_1^*\ \text{II})\ w}$
2) *У1*	6)	6) $PA \geqslant 0$	1) $\overline{7776^*}$
3)	$\|0\|-\|u_1{}^*\|=0$		

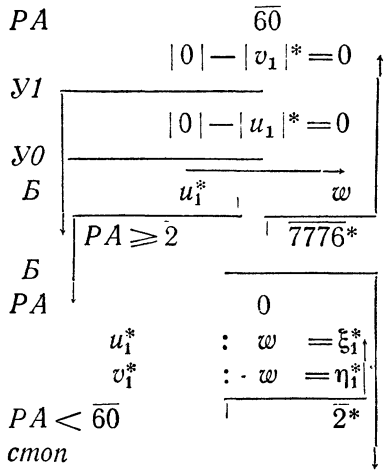
Написанная программа может зациклиться (не выйти из цикла), если среди чисел последовательности нет действительных чисел, отличных от нуля. В самом деле, команда 6) фактически является безусловной передачей управления, а выход из цикла происходит только по команде 5), когда будет обнаружено отличное от нуля действительное число. Чтобы избежать зацикливания, изменим команду 6) таким образом, чтобы выход из цикла был обеспечен после окончания перебора всех членов последовательности, после чего можно поместить безусловную передачу управления на *стоп*. Для этой цели можно воспользоваться командами

6) $PA \geqslant \overline{2}$	1)	$\overline{7776^*}$
7) *Б*	*стоп*	

Действительно, после того как цикл пройдет 24_{10} раза, и после выполнения команды 6) мы получим $PA = 0$, так что при следующем переходе к той же команде управление перейдет команде 7).

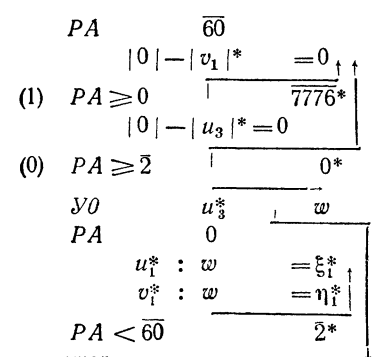
Второй этап решения задачи, нахождение последовательности $\xi_1, \xi_2, \ldots, \xi_{25}$, программируется тем же способом, что и в предыдущем примере.

Программа решения всей задачи имеет следующий вид:



Здесь через ξ_1, η_1, ξ_2, η_2,..., ξ_{25}, η_{25} обозначены адреса последовательных ячеек памяти, содержащих действительные и мнимые части искомых комплексных чисел.

Эту программу можно сократить, если воспользоваться командами проверки окончания цикла по *РА* и ω. В этом случае ее можно записать так:



§ 27. Цикл в цикле

Рассматривавшиеся нами ранее циклические программы (циклы) можно разделить на *циклы без переменных команд*, в которых все команды цикла в процессе их выполнения не изменяются, и *циклы с переадресацией*. В зависимости от способа проверки окончания циклы без переменных команд делятся на арифметические и итерационные. Все эти типы циклов достаточно подробно рассматривались нами ранее. Различные циклы обычно являются составными частями более сложных программ. В частности, такие циклы могут находиться внутри других циклов, являясь их рабочими частями.

Таким образом, обычный цикл (его называют в таких случаях *внутренним*) оказывается «вложенным» в другой *(внешний)* цикл.

Рассмотрим примеры задач такого рода.

Пример 1.27. Вычислим сумму

$$S=\frac{1}{\sqrt[3]{1}}+\frac{1}{\sqrt[3]{2}}+\frac{1}{\sqrt[3]{3}}+\ldots+\frac{1}{\sqrt[3]{n}}.$$

Обозначим

$$S_k=\frac{1}{\sqrt[3]{1}}+\frac{1}{\sqrt[3]{2}}+\ldots+\frac{1}{\sqrt[3]{k}}.$$

Тогда, очевидно,

$$S_{k+1}=S_k+1/\sqrt[3]{k+1} \text{ и } S=S_n.$$

Значение S, очевидно, можно получить при помощи простого арифметического цикла. Однако, поскольку в машине нет операции извлечения кубического корня, то для получения слагаемых (т. е. внутрь арифметического цикла) придется вставить итерационный цикл для нахождения $\sqrt[3]{k}$. . При этом за начальное приближение можно принимать вычисленное ранее (на предыдущем шаге внешнего цикла) значение $\sqrt[3]{k-1}$.

Начнем с команд восстановления для внешнего цикла. Если отвести для кубического корня ячейку q, то команды восстановления будут иметь вид

$$\text{«0»}+\text{«1»}=q$$
$$\text{«0»}+\text{«0»}=S$$
$$\text{«0»}+\text{«1»}=k$$

Впрочем, последнюю команду надо заменить пересылкой с передачей управления, чтобы обойти команду изменения

$$k + «1» = k$$

После команды изменения идет рабочая часть, которая содержит внутренний итерационный цикл для вычисления кубического корня; его можно полностью заимствовать из примера 1.14. После него необходимы еще команды

$$«1» : q = u$$

$$S + u = S$$

а затем две обычные команды проверки окончания внешнего цикла. Окончательно требуемую программу можно записать так:

	«0» + «1» = q		$R_1 + R_2 = q$
	«0» + «0» = S		$q - q_1 = R_1$
	Б «1» $\rightarrow k$		$\lvert R_1 \rvert - \lvert \varepsilon \rvert = 0$
T_1	k + «1» = k	У0	$q \rightarrow T_2 \rightarrow q_1$
	«0» + q = q_1		«1» : q = u
T_2	$q_1 \cdot q_1 = R_1$		$S + u = S$
	k : $R_1 = R_1$		$k - n = 0$
	$R_1 \cdot$ «1/3» = R_1	У1	T_1
	$q_1 \cdot$ «2/3» = R_2	стоп	

Пример 2.27. Вычислим значение выражения:

$$S = x_1^9 + x_2^9 + x_3^9 + \ldots + x_{25}^9.$$

Для вычисления величины S нужно составить внутренний цикл нахождения девятой степени x_1, x_2, ..., x_{25} и внешний цикл нахождения S суммированием. Арифметический цикл для вычисления $y_1 = x_1^9$ «перепишем» из примера 1.23, проверяя, однако, окончание цикла при помощи фиксированного счетчика:

1)	0	+	«1»	$= y$
2)	(0, 0, 11)	—,	(0, 0, 1)	$= n$
3)	x_1	·	y	$= y$
4)	n	—,	(0, 0, 1)	$= n$
5)	У0			

Команды 1) — 5) служат для вычисления отдельного значения $y = x_1^9$ и образуют внутренний цикл.

Для того чтобы получить все значения степеней $x_1^9, x_2^9, \ldots$ $\ldots, x_{25}^9$, следует этот внутренний цикл повторить 25 раз, включив его во внешний цикл суммирования этих степеней. При этом, чтобы при втором прохождении внутреннего цикла считалась величина x_2^9, нужно переадресовать команду 3), придав ей вид

$$3)\ x_2 \cdot y = y.$$

Переадресацию следует производить во внешнем цикле каждый раз после вычисления очередного значения y. Обозначим фиксированный счетчик внешнего цикла через m и будем изменять его также в обратном направлении.

Программа для вычисления S имеет вид

	«0» + «0» $=S$		
	(0, 0, 31) —, (0, 0, 1) $=m$		Восстановление внешнего цикла
Б	T^0 | T		
	T +, (1, 0, 0) $=T$	↑	Переадресация внешнего цикла
	0 + «1» $=y$	↓	Восстановление внутреннего цикла
	(0, 0, 11) —, (0, 0, 1) $=n$		
T	$(x_1 \cdot y = y)$		Рабочая часть внутреннего цикла (н. п.)
	n —, (0, 0, 1) $=n$		Изменение и проверка окончания внутреннего цикла
У0	T		
	S + y $=S$		Рабочая команда внешнего цикла
	m —, (0, 0, 1) $=m$		Изменение и проверка окончания внешнего цикла
У0			
стоп			
T^0	$x_1 \cdot y = y$		Восстановитель

Эту программу можно сократить, если для переадресации и счета шагов внешнего цикла воспользоваться

регистром адреса:

	«0»	+	«0»	$=S$
	«0»	+	«1»	$=y$
РА	0		0	0
	(0, 0, 11)	—,	(0, 0, 1)	$=n$
	x_1^*	·	y	$=y$
	n	—,	(0, 0, 1)	$=n$
У0				
	S	+	y	$=S$
РА	$<\overline{30}$			$\overline{1}^*$

стоп

В рассмотренном примере и внутренний, и внешний цикл оказались арифметическими, причем внешний цикл требовал переадресации.

В первом варианте программы мы использовали во внутреннем и внешнем цикле фиксированные счетчики, во втором, более коротком, варианте и счетчиком, и средством переадресации внешнего цикла служил регистр адреса.

Для того чтобы использовать регистр адреса как во внешнем, так и во внутреннем цикле, необходимо иметь возможность при переходе к внутреннему циклу запоминать соответствующее значение регистра и восстанавливать его при обращении к командам внешнего цикла, т. е. после выхода из внутреннего. Этой цели и служит использование крайних адресов команд занесения в регистр.

Как уже было сказано в § 25, команды занесения в регистр являются трехадресными. Команда

$$РА \quad \overline{a} \ \overline{b} \ c$$

выполняется так: в регистр адреса заносится число b. Одновременно в ячейку с адресом c записывается команда занесения в регистр числа a, т. е. команда

$$РА \quad \overline{a}.$$

Обычно команда занесения в регистр используется в виде

$$РА \quad 0^* \ 0 \ c.$$

Первый адрес команды является звездочным. Исполнительный адрес формируется еще при старом значении регистра. Поэтому в ячейку c запишется команда

$$PA \quad 0 \quad PA_{\text{старое}} \; 0,$$

а регистр очистится (если же в среднем адресе написать любое число b, то оно занесется в регистр). Когда управление придет в ячейку c, то находящаяся там команда восстановит прежнее содержимое регистра адреса.

Точно так же используются первый и третий адреса и в команде

$$[PA] \quad \bar{a} \; b \; c \; :$$

в регистр адреса заносится второй адрес ячейки b, а в ячейку c — команда

$$PA \quad \bar{a}.$$

Как и предыдущую, эту команду чаще всего используют в виде

$$[PA] \quad 0^* \; b \; c.$$

Рассмотрим примеры применения этих команд с использованием крайних адресов.

Пример 3.27. Вычислим девятые степени чисел $x_1, x_2, \ldots, x_{30}$ и поместим в ячейки $y_1, y_2, \ldots, y_{30}$. Мы имеем

$$y_1 = x_1^9, \quad y_2 = x_2^9, \quad \ldots, \quad y_{30} = x_{30}^9.$$

В отличие от предыдущего примера во внутреннем цикле (получение x_1^9) в качестве счетчика используем PA:

1) PA 0 0 0		3)	$y_1 \cdot x_1 = y_1$
2)	«0» + «1» $= y_1$	4) PA	$<\overline{10}$ $\overline{\quad 1^*}$

Для того чтобы получить все значения y от y_1 до y_{30}, следует этот внутренний цикл повторить 30 раз, включив его во внешний цикл.

При этом, чтобы при втором прохождении внутреннего цикла считалась величина $y_2 = x_2^9$, нужно переадресовать команды 2), 3), придав им вид

$$2) \quad 0 + \text{«1»} = y_2$$
$$3) \quad y_2 \cdot x_2 = y_2$$

Это можно было бы сделать переадресацией при помощи фиксированных операций, что потребовало бы двух команд

восстановления, двух команд переадресации и двух констант-восстановителей. Однако целесообразнее для переадресации во внешнем цикле использовать PA. Для этого «перенесем» переадресацию во внешний цикл, используя рассмотренный в § 26 прием засылки в стандартные ячейки

$$\begin{array}{lll} 0 + x_1 = x & & y \cdot x = y_1 \\ PA\ \ 0\ \ 0\ \ 0 & PA < \overline{10} & \overline{1^*} \\ \text{«0»} + \text{«1»} = y & & 0 + y = y_1 \end{array}$$

Организуя переадресацию во внешнем цикле при помощи PA, получим следующую программу

$$\begin{array}{llll} 1)\ PA\ \ 0\ \ 0\ \ 0 & 5) & & y \cdot x = y_1 \\ 2)\ \ \ 0 + x_1^* = x & 6)\ PA & < \overline{10} & \overline{1^*} \\ 3)\ PA\ \ 0\ \ 0\ \ 0 & 7) & & y = y_1^* \\ 4)\ \ \ \text{«0»} + \text{«1»} = y & 8)\ PA & < \overline{35}\ \ 2) & 1^* \end{array}$$

Однако в таком виде программа будет работать неверно.

Действительно, во внешнем цикле PA должен меняться от 0000 до 0035, увеличиваясь на каждом шаге цикла на 0001. Однако каждый раз при входе во внутренний цикл PA очищается (см. команду 3)), а при выходе из внутреннего цикла PA становится равным 0011 (см. команду 6)), в результате чего команды 7) и 8) будут выполняться не так, как следует.

Для правильной работы внешнего цикла следует при входе во внутренний цикл запоминать значение PA для внешнего цикла и восстанавливать его после выхода из внутреннего.

Заменим команду 3) командой

$$PA\ \ 0^*\ 0\ BP,$$

где BP — адрес ячейки, которую следует оставить для восстановления регистра после команды 6) внутреннего цикла.

Написанная команда, очищая PA для начала работы внутреннего цикла получения y, одновременно заготовляет в ячейке BP команду

$$PA\ \ PA_{\text{старое}},$$

при помощи которой после выхода из внутреннего цикла восстанавливается значение PA для внешнего цикла.

Таким образом, программа вычисления последовательности $y_1, y_2, \ldots, y_{30}$ записывается в следующем виде

```
      PA    0    0    0
                0 + x1* = x
      PA    0*   0    BP
           «0» + «1» = y
            y  ·  x  = y
      PA   <10        1*
BP          н. п.
                0 + y = y1*
      PA   <35        1*
      стоп
```

Пример 4.27. Вычислим значения $z=\sqrt{ax+by+c}$ для всех возможных пар значения x и y из следующих последовательностей

$$x=x_1,\ x_2,\ \ldots,\ x_8,$$
$$y=y_1,\ y_2,\ \ldots,\ y_6.$$

Легко видеть, что нам нужно вычислить всего 48 значений функции z, которые мы будем помещать в 48 последовательных ячейках.

Положим сначала $y = y_1$. Тогда нам нужно будет получить восемь значений z:

$$z_{11}=\sqrt{ax_1+by_1+c},$$
$$z_{21}=\sqrt{ax_2+by_1+c},$$
$$\ldots\ldots\ldots\ldots$$
$$z_{81}=\sqrt{ax_8+by_1+c}.$$

Эти значения вычисляются при помощи простого цикла с переадресацией по PA:

```
      PA   0     0    0
           a  ·  x1* = R1
U          b  ·  y1 = R2
           R1 + R2 = R1
           R1 +  c = R1
V     √R1          = z11*
      PA <7        1*
```

Чтобы получить следующую серию из 8 значений z:

$$z_{12}=\sqrt{ax_1+by_2+c},$$
$$z_{22}=\sqrt{ax_2+by_2+c},$$
$$\cdots\cdots\cdots\cdots$$
$$z_{82}=\sqrt{ax_8+by_2+c},$$

следует, очевидно, повторить вычисление по написанному внутреннему циклу, заменив, однако, в нем y_1 на y_2, т. е. увеличивая второй адрес команды U на 1. Аналогично можно получить и все следующие серии по 8 значений z.

Внешний цикл должен состоять из получения шести серий значений z (по числу значений y). В этом цикле должны переадресовываться, а следовательно, и восстанавливаться команды U и V, поэтому цикл должен содержать команды восстановления

$$U_0+,\ 0=U$$
$$V_0+,\ 0=V$$

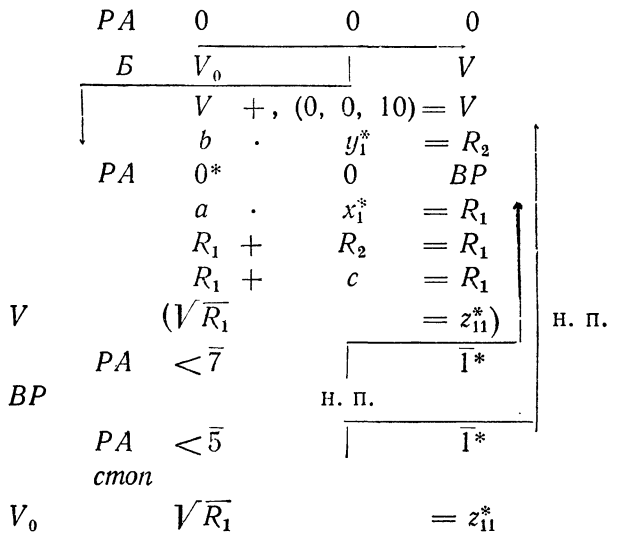
и команды переадресации:

$$U+,\ (0,\ 1,\ 0)=U$$
$$V+,\ (0,\ 0,\ 10)=V$$

Образуя внешний цикл при помощи PA, получим следующую программу:

```
      PA  0          0        0
          U0  +,     0      = U
      Б   V0                  V
      |   U   +, (0, 1, 0)  = U
      |   V   +, (0, 0, 10) = V
      |   PA  0*         0      BP
      |       a    ·     x1*    = R1
U     |       (b   ·     y1     = R2)   н. п.
      |       R1  +      R2     = R1
      |       R1  +      c      = R1
V     |         (√R1             = z11*) н. п.
      |   PA  <7                 1*
BP    |                 н. п.
      |   PA  <5                 1*
```

Следует обратить внимание на не совсем обычный характер переадресации команд U и V. Команда V переадресуется как во внешнем, так и во внутреннем цикле (по PA), команда же U переадресуется только во внешнем цикле. Поэтому команду U можно вынести во внешний цикл, переадресуя ее по PA. Таким образом получим более короткую программу:



Эта программа в закодированном виде приведена в табл. 1.27, а ее памятка — в табл. 2.27.

Команды программы расположены, начиная с ячейки 4400. В памятку внесены обозначения рабочих ячеек R_1, R_2, исходных данных a, b, c, x_1, x_2, ..., x_8, y_1, y_2, ..., y_6 и результатов счета z_{11}, z_{21}, ..., z_{86}.

Рассмотренный пример характерен тем, что для переадресации цикла в цикле оказалось недостаточным применения регистра адреса; пришлось переадресовывать команду V в памяти при помощи операции сложения адресных частей. Этот пример показывает, что *регистр адреса не является универсальным средством переадресации.*

Заметим, что регистр адреса удается использовать лишь тогда, когда все переменные адреса переадресуемых команд цикла должны изменяться (увеличиваться или уменьшаться) с одним и тем же шагом.

Таблица 1.27

						4400	*A*	
	PA 0 0 0	4400	0	52	0000	0000	0000
	Б V_0 *V*	1	0	56	4415	4403	4410
	V +, (0, 0,10) = *V*	2	0	13	4410	4416	4410
	b · y_1^* = R_2	3	2	05	4421	4431	4424
	PA 0* 0 *BP*	4	4	52	0000	0000	4412
	a · x_1^* = R_1	5	2	05	4420	4441	4423
	R_1 + R_2 = R_1	6	0	01	4423	4424	4423
	R_1 + *c* = R_1	7	0	01	4423	4422	4423
V	н.п.($\sqrt{R_1}$ = z_{11}^*)	4410	0	77	0000	0000	0000
	PA < $\bar{7}$ $\bar{1}$*	1	1	12	0007	4405	0001
BP	н. п.	2	0	77	0000	0000	0000

						4413	*A*	
	PA < $\bar{5}$ $\bar{1}$*	4413	1	12	0005	4402	0001
	стоп	4	0	77	0000	0000	0000
V_0	$\sqrt{R_1}$ = z_{11}^*	5	1	44	4423	0000	4461
	(0 , 0 , 10)	6	0	00	0000	0000	0010

Большинство встречающихся при программировании циклов имеет именно такой характер. Однако встречаются задачи и другого рода. Пусть, например, нам нужно переставить члены последовательности

$$a_1, a_2, \ldots, a_{50}$$

в обратном порядке, размещая новую последовательность в ячейках

$$b_1, b_2, \ldots, b_{50}$$

(см. пример 3.24).

В этом случае следует в цикле при $i = 1, 2, \ldots, 50$ выполнить команду

$$a_i = b_{51-i}.$$

Очевидно, переадресовывать такую команду при помощи *PA* не удастся.

Таблица 2.27

4400	4440	4500 z_{82}	4540 z_{86}
4401	4441 x_1	4501 z_{13}	4541
4402	4442 x_2	4502 z_{23}	4542
4403	4443 x_3	4503 ·	4543
4404	4444 x_4	4504 ·	4544
4405	4445 x_5	4505 ·	4545
4406	4446 x_6	4506 ·	4546
4407	4447 x_7	4507 ·	4547
4410 V	4450 x_8	4510 z_{83}	4550
4411	4451	4511 z_{14}	4551
4412 BP	4452	4512 z_{24}	4552
4413	4453	4513 ·	4553
4414	4454	4514 ·	4554
4415 V_0	4455	4515 ·	4555
4416 (0, 0, 10)	4456	4516 ·	4556
4417	4457	4517 ·	4557
4420 a	4460	4520 z_{84}	4560
4421 b	4461 z_{11}	4521 z_{15}	4561
4422 c	4462 z_{21}	4522 z_{25}	4562
4423 R_1	4463 z_{31}	4523 ·	4563
4424 R_2	4464 ·	4524 ·	4564
4425	4465 ·	4525 ·	4565
4426	4466 ·	4526 ·	4566
4427	4467 z_{71}	4527 ·	4567
4430	4470 z_{81}	4530 z_{85}	4570
4431 y_1	4471 z_{12}	4531 z_{16}	4571
4432 y_2	4472 z_{22}	4532 z_{26}	4572
4433 y_3	4473 ·	4533 ·	4573
4434 y_4	4474 ·	4534 ·	4574
4435 y_5	4575 ·	4535 ·	4575
4436 y_6	4476 ·	4536 ·	4576
4437	4477 ·	4537 ·	4577

Программа

В рассмотренных примерах составлялись программы, имеющие вид цикла в цикле. Такие программы называют *двойными циклами*. Иногда приходится иметь дело и с *тройным* циклом, т. е. с двойным циклом, заключенным внутри обычного внешнего. Встречаются циклы и большей кратности.

Пример 5.27. В ячейках c_0, c_1, ... подряд записаны 500 значений некоторой случайной величины; все они принадлежат интервалу (a, b). Интервал разбит на 12 равных участков (a, d_1), (d_1, d_2), ..., (d_{11}, b). Требуется подсчитать число значений c_j, попавших в каждый из этих интервалов.

Пусть u_0, u_1, ... — двенадцать ячеек, отведенных для счетчиков, в которых мы будем в плавающей форме записывать количество значений c_j, попавших в каждый из интервалов. Прежде всего необходимо очистить эти ячейки, что легко сделать с помощью следующего цикла:

```
PA    0    0    0
      0  +0 = u0*  ↑
PA  < 13   |   1*
```

Далее должен идти цикл с переадресацией, в котором мы снова воспользуемся приемом засылки в стандартную рабочую ячейку. Такой цикл можно написать в виде

```
PA          0 .   0      0
            0  +  c0*  = cраб  ↑
   внутренний   цикл            |
PA       < 763    |        1*
стоп
```

Задачей «внутреннего цикла» будет являться нахождение номера участка, которому принадлежит число, записанное в ячейке $c_{раб}$, и прибавление единицы в счетчик u_i, соответствующий именно этому участку.

Внутренний цикл можно написать следующим образом. Предположим, что числа d_1, d_2, ... лежат подряд в ячейках с тем же наименованием, а в ячейке d_{12} лежит число b. Будем сравнивать $c_{раб}$ с этими числами по команде $c_{раб} - d_1^* = 0$ до тех пор, пока разность не сделается отрицательной. В этот момент в регистре будет записан номер участка. Поэтому программу внутреннего цикла можно написать так

```
PA    0*     0     BP
 |    cраб -  d1* = 0   ↑
 У1                     |
 |  PA  < 13    |    1*
 ↓       u0* + «1» = u0*
BP              н. п.
```

Работа начинается с очистки регистра адреса и его запоминания в ячейке BP. Затем содержимое ячейки $c_{раб}$ сравнивается с d_1. Если $c_{раб} < d_1$, то данное значение лежит в интервале (a, d_1). В этом случае

команда условной передачи управления сразу выведет из цикла с $PA = 0$ и к счетчику u_0 прибавится единица. Если $c_{раб} \geqslant d_1$, то управление перейдет команде проверки окончания цикла и после прибавления единицы в регистр $c_{раб}$ будет сравниваться с d_2 и т. д. до тех пор, пока не будет выполнено неравенство $c_{раб} < d_i$ с каким-либо i. При этом выход из цикла будет происходить только по команде условной передачи управления.

Непосредственное сокращение программы с помощью команды проверки окончания по PA и по ω затруднительно, так как эта команда, даже при передаче управления на следующую ячейку, увеличивает регистр, поэтому при выполнении команды $u_0^* + $ «1» $= u_0^*$ не удастся, например, получить $PA = 0$ ни при каких обстоятельствах. Эту трудность, однако, можно обойти, заменив адрес u_0 на адрес непосредственно предшествующей ячейки, которую мы обозначим через u_{-1}. Тогда программу можно будет записать так:

```
          PA      0*      0      BP
                  cраб —  d1*  = 0   ↑
(0)   PA  <       13      |      1*
                  u*-1 +  «1»  = u*-1
BP                        н. п.
```

Читатель разберется в работе этой программы самостоятельно.

Используя модификации арифметических операций, можно достичь цели определения номера интервала, в котором находится $c_{раб}$, гораздо быстрее и без цикла, причем число действий, требуемых для нахождения номера, не будет зависеть от номера, как это имело место в приведенных выше программах.

Для этого нового приема нам не потребуются величины d_1, d_2, ..., а лишь длина участка (d_i, d_{i+1}), которую мы запишем в ячейке h. После выполнения команд

$$c_{раб} - a = R$$
$$R : h = R$$

в рабочей ячейке R будет записано число, целая часть которого равна требуемому номеру участка. С помощью его сложения со специально подобранной константой можно получить этот номер в виде числа единиц второго адреса. Этого можно достичь одной командой

$$(130;\ 0,\ 0,\ 0) + (БН)\, R = R.$$

В самом деле, машинный порядок 130 имеет число, истинный порядок которого равен 30_8, так что разряды мантиссы, соответствующие целой части такого числа, занимают первый и второй адреса ячейки. При выравнивании порядков слагаемых мантисса числа в ячейке R расположится так же и при сложении не изменится, поскольку мантисса первого слагаемого равна нулю. Блокировка нормализации воспрепятствует обратному сдвигу мантиссы влево. Теперь для окончательного достижения наших целей достаточно выполнить команды

```
[PA]  0*      R      BP
      u0* +  «1»  = u0*
BP            н. п.
```

Напомним, что по команде $[PA]$ 0* R BP в PA заносится число единиц второго адреса ячейки R с одновременной записью в ячейку BP команды восстановления PA.

Таким образом, окончательное решение задачи дается программой

PA	0		0	0
	0	+	0	$= u_0^*$ ↑
PA	$< \overline{13}$			$\overline{1}$*
PA	0		0	0
	0	+	c_0^*	$= c_{\text{раб}}$ ↑
	$c_{\text{раб}}$	—	a	$= R$
	R	:	h	$= R$
	(130; 0, 0, 0)	+ ($БН$)	R	$= R$
$[PA]$	0*		R	BP
	u_0^*	+	«1»	$= u_0^*$
BP			н. п.	
PA	$< \overline{763}$			$\overline{1}$*
стоп				

§ 28. Циклические действия и операции с порядками

Среди элементарных операций рассматриваемых нами машин имеются еще две группы операций, в какой-то мере родственные фиксированным действиям сложения и вычитания адресных или операционных частей. Для более полного знакомства с системой команд мы разберем коротко эти операции.

Очень близкими к фиксированному сложению и вычитанию являются циклические действия: *циклическое сложение* и *циклическое вычитание*. В содержательных обозначениях мы будем применять для этих действий обозначения

$$a \oplus b = c,$$
$$a \ominus b = c.$$

По команде циклического сложения $a \oplus b = c$ происходит одновременное фиксированное сложение операционных и адресных частей ячеек a и b. Кроме того, если при сложении адресных частей образуется единица переноса в старший разряд, то она не пропадает, а добавляется к младшему разряду адресной части, т. е. к первому. Аналогично, единица переноса из 45-го разряда при сложении операционных частей, если она образуется, также не пропадает, а прибавляется к младшему разряду операционной части — 37-му. Циклическое сложение используется в программах контрольного суммирования, которые применяются для проверки правильности обмена между различными видами памяти: при вводе с перфокарт в оперативную память, при записи из оперативной памяти во внешнюю и наоборот и т. п.

Аналогично, при циклическом вычитании $a \ominus b = c$ происходит одновременное вычитание операционной части ячейки b из операцион-

ной части ячейки a и адресной части ячейки b из адресной части ячейки a. Если в той или иной части вычитаемое больше уменьшаемого, то недостающая единица занимается из младшего разряда соответствующей части ячейки — из первого, если это требуется для адресной части (мантиссы), и из 37-го для операционной (порядка). Связи между этими двумя частями ячейки при циклических действиях нет, как и при сложении и вычитании адресных или операционных частей.

Например, если

$$a = 125\ 6430\ 7512\ 2631,$$
$$b = 346\ 3745\ 2674\ 1315,$$

то после выполнения команды $a \oplus b = c$ получится

$$c = 473\ 2376\ 2406\ 4147,$$

а после команды $a \ominus b = c$

$$c = 556\ 2463\ 4616\ 1314.$$

Рекомендуем читателю самостоятельно убедиться в правильности приведенных результатов.

Управляющий сигнал ω вырабатывается при операциях циклического сложения и вычитания по тем же правилам, что и при сложении или вычитании адресных частей: $\omega = 1$, если происходит перенос из 36-го разряда в первый или — при вычитании — из первого в 36-й. Если сумма адресных частей не переполняет разрядной сетки, а при вычитании уменьшаемая адресная часть не меньше соответствующего вычитаемого, то вырабатывается сигнал $\omega = 0$. Переносы из 37-го разряда в 45-й (и обратно), если они есть, т. е. результаты действий с операционными частями, на управляющий сигнал не влияют.

Совсем другой характер носят операции с порядками, к знакомству с которыми мы сейчас переходим. С фиксированными действиями, которые рассматривались в § 23, их объединяет только то, что эти операции имеют дело не со всем машинным словом, а лишь с его частью.

Действия с порядками, по существу, являются арифметическими действиями и эквивалентны умножению или делению числа, записанного в ячейке, на число 2^s, где s — целое положительное или отрицательное число, задаваемое в разных операциях по-разному. При выполнении этих действий к порядку числа прибавляется или из порядка вычитается число s.

В операциях сложения порядка с адресом и вычитания из порядка адреса соответствующее число записывается как машинный порядок в первом адресе команды. Команду сложения порядка с адресом мы будем обозначать

$$\bar{a}\ (+)\ b = c,$$

а команду вычитания из порядка адреса

$$\bar{a}\ (-)\ b = c.$$

Эти команды представляют собою еще один пример команд, в которых адрес команды не является адресом некоторой ячейки памяти, а представляет собою некоторое условное число. С такими командами мы впервые встретились в § 25, при рассмотрении операций

с регистром адреса. Первый адрес в этих командах мы пишем с чертой наверху. В фактической команде здесь будет стоять число, которое без изменения должно быть переписано в правую (кодированную) часть программы.

При выполнении этих команд используется содержимое лишь семи младших разрядов числа a, т. е. первого адреса, иначе говоря, число, не превосходящее 177_8. Это число рассматривается как машинный порядок, а соответствующий ему истинный порядок прибавляется к порядку числа, записанного в ячейке b, или вычитается из него. Мантисса ячейки b переписывается на место мантиссы в ячейку c без изменения. Так как истинный порядок p связан с условным порядком p' сложением $p' = p + 100_8$, то можно сказать, что к порядку числа b прибавляется (или из него вычитается) число $\bar{a} - 100_8$.

Например, при выполнении команды

$$\overline{105}\,(+)\,b = c$$

к порядку b прибавится число 5, а при выполнении команды

$$\overline{75}\,(+)\,b = c$$

— число (—3), так что эта команда равносильна другой

$$\overline{103}\,(-)\,b = c,$$

где из порядка b вычитается число 3. При этом нужно иметь в виду, что наличие единиц в старших разрядах первого адреса не влияет на выполнение команды; команды

$$\overline{105}\,(+)\,b = c$$
$$\overline{505}\,(+)\,b = c$$
$$\overline{2305}\,(+)\,b = c$$

совершенно равносильны и выполняются одинаково: к порядку числа b прибавляется число 5, т. е. число умножается на 2^5.

Аналогично выполняются и две другие операции: сложение порядка с порядком и вычитание из порядка порядка, которые мы будем обозначать так:

$$a\,[+]\,b = c$$
$$a\,[-]\,b = c$$

При выполнении этих команд к порядку числа, записанного в ячейке b, прибавляется или из него вычитается истинный порядок числа, находящегося в ячейке a. Если, например, в ячейке a записано число, условный порядок которого равен 104, то после выполнения команды $a\,[+]\,b = c$ в ячейке c будет записано число с мантиссой b и порядком, на 4 бо́льшим порядка b.

Следует обратить внимание на то, что содержательные обозначения команд

$$\bar{a}\,(-)\,b = c$$
$$a\,[-]\,b = c$$

не соответствуют смыслу выполняемых операций, так как фактически вычитание происходит не так, как написано; на самом деле, из порядка b

вычитается либо число $\bar{a}$, либо порядок числа из ячейки a. Однако мы предпочитаем такое отступление, создающее некоторую трудность для содержательной записи команды, чтобы сохранить порядок следования адресов в команде и не создавать трудностей при кодировке.

Управляющий сигнал ω при всех описанных операциях с порядками вырабатывается, как и при операциях плавающего умножения и деления, в зависимости от порядка результата: если условный порядок c превосходит 100_8, т. е. если результат не меньше единицы по абсолютной величине, то $\omega = 1$; в противном случае $\omega = 0$. Если условный порядок результата превосходит 177_8, т. е. результат не помещается в разрядной сетке машины, то происходит *авост*, как и при обычных плавающих арифметических операциях.

Заметим, что если мантисса числа в ячейке b является нулевой, то результатом любой из операций изменения порядка будет машинный нуль.

Коды рассмотренных в настоящем параграфе циклических операций и операций с порядками приведены в табл. 1.28.

Таблица 1.28

Обозначение команды	Коды
$a \oplus b = c$	07
$a \ominus b = c$	27
$\bar{a}\ (+)\ b = c$	06
$a\ [+]\ b = c$	26
$\bar{a}\ (-)\ b = c$	46
$a\ [-]\ b = c$	66

ГЛАВА VI

ОПЕРАЦИИ НАД МАШИННЫМИ СЛОВАМИ

§ 29. Машинное слово

Как уже было сказано в § 18, ячейка памяти может хранить набор из сорока пяти двоичных цифр (нулей или единиц), который мы и называем *машинным словом*. До сих пор мы рассматривали машинные слова, которые употреблялись как ч и с л о (плавающее или фиксированное) или как к о м а н д а. В обоих случаях отдельные разряды слова определенным образом связаны между собой.

Операции, которые рассматривались в предыдущих главах, были операциями над числами или над командами. При этом, в зависимости от вида операции, в ней участвовала вся ячейка памяти (как в плавающих арифметических действиях или в циклических операциях) или только некоторая часть ячейки (как в сложении или вычитании адресных или операционных частей и действиях с порядками). В настоящей главе будут разобраны операции над словами, характер которых безразличен, т. е. которые рассматриваются именно как н а б о р нулей и единиц, записанных в определенном порядке и никак не связанных между собою.

Мы рассмотрим два типа таких операций: операции сдвига и логические операции. В операциях сдвига действия происходят над некоторым машинным словом, рассматриваемым как единый набор независимых друг от друга двоичных цифр. В логических операциях действия происходят фактически даже не со словами, а с отдельными разрядами этих слов.

§ 30. Сдвиги

Сдвиг заключается в том, что машинное слово целиком сдвигается внутри разрядной сетки ячейки влево или вправо на определенное число разрядов. Двоичные разряды, выходящие за разрядную сетку, теряются. На место освобождающихся разрядов ставятся нули.

Так, например, слово
011 011 111 101 100 011 000 100 110 001 111 010 110 111 001
при сдвиге на 15 разрядов вправо превращается в слово
000 000 000 000 000 011 011 111 101 100 011 000 100 110 001,
а при сдвиге на 15 разрядов влево — в такое слово:
011 000 100 110 001 111 010 110 111 001 000 000 000 000 000.

Если рассматривать слово как целое двоичное число, то сдвиг вправо на один разряд означает уменьшение числа вдвое. Поэтому сдвиг вправо называют *отрицательным*, а сдвиг влево, увеличивающий слово, называют *положительным сдвигом*. Адрес сдвигаемого слова указывается во втором адресе команды сдвига, а результат сдвига записывается по третьему адресу. Первый адрес команды сдвига используется для указания направления сдвига и числа разрядов, на которое нужно сдвинуть слово. Это можно сделать различными способами.

Проще всего написать число разрядов, на которое нужно произвести сдвиг, прямо в виде фиксированного числа единиц первого адреса. Здесь мы еще раз встречаемся с командой, адрес которой не означает адреса ячейки памяти, а представляет собой некоторое условное число (такую же роль играли, например, крайние адреса в командах проверки содержимого регистра).

Такую операцию называют *сдвигом по адресу*. Для записи этой команды в левой части воспользуемся следующим обозначением. Сдвиг будем обозначать стрелкой, а перед нею указывать *условное число*, равное $100_8 + \nu$, где ν — число разрядов сдвига со знаком, зависящим от того, положительным или отрицательным является сдвиг.

Так, команда

$$\overline{54} \rightarrow a = b$$

означает, что слово, содержащееся в ячейке a, надо сдвинуть на $24_8 = 20_{10}$ разрядов вправо ($100_8 - 24_8 = 54_8$) и

результат записать в ячейку b. Команда

$$\overline{137} \rightarrow a = b$$

означает, что слово a сдвинется на $37_8 = 31_{10}$ разрядов в л е в о ($100_8 + 37_8 = 137_8$). При кодировке этой команды соответствующее условное число записывается в первом адресе.

Кроме сдвига всего слова, используется также операция *сдвига мантиссы* или, что то же самое, *сдвига адресной части* слова. Она обозначается так же, как и предыдущая, только по аналогии с фиксированными действиями для указания сдвига мантиссы мы будем ставить запятую у стрелки, означающей сдвиг. Например, команда

$$\overline{53} \rightarrow, \ a = b$$

означает сдвиг мантиссы ячейки a на $25_8 = 21_{10}$ разрядов вправо. При кодировке первый адрес команды должен содержать условное число 0053. При выполнении команды сдвига мантиссы порядок остается на месте, а мантисса сдвигается на указанное число разрядов.

Сдвиг по адресу можно применять тогда, когда величина сдвига известна программисту заранее. Возможны, однако, случаи, когда эта величина заранее не известна, а определяется в процессе работы программы. Тогда применяются операции *сдвига по порядку*.

Команду сдвига по порядку мы будем обозначать так:

$$a[\rightarrow]b = c.$$

При выполнении этой команды слово из ячейки b сдвигается и результат сдвига кладется в ячейку c. Величина сдвига принимается равной истинному порядку числа, записанного в ячейке a. Направление сдвига определяется знаком порядка этого числа. Таким образом, семь младших разрядов порядка ячейки используются здесь так же, как и первый адрес команды сдвига по адресу.

Если, например, машинный порядок числа в ячейке a равен 114, то это означает положительный сдвиг (влево) на $14_8 = 12_{10}$ разрядов (при этом истинный порядок числа равен, как известно, 14_8). Если машинный порядок есть 070, то истинный порядок равен -10_8 и сдвиг происходит на $10_8 = 8_{10}$ разрядов вправо.

Кроме описанной, существует также операция *сдвига мантиссы по порядку*. Она обозначается так:

$$a\,[\rightarrow,]\,b = c,$$

и выполняется так же, как и сдвиг по порядку, но только сдвигается не все слово, а лишь его адресная часть (мантисса).

При операциях сдвига управляющий сигнал ω вырабатывается по следующим правилам. Он принимает значение $\omega = 1$ в тех случаях, когда в результате сдвига получилось нулевое слово, если сдвигалось все слово, либо слово с нулевой мантиссой, если сдвигалась только мантисса. Во всех остальных случаях вырабатывается сигнал $\omega = 0$.

Коды описанных операций приведены в табл. 1.30.

Таблица 1.30

Название операции	Обозначение	Код
Сдвиг мантиссы по адресу	$\bar{a} \rightarrow, b = c$	14
Сдвиг мантиссы по порядку	$a\,[\rightarrow,]\,b = c$	34
Сдвиг слова по адресу . . .	$\bar{a} \rightarrow b = c$	54
Сдвиг слова по порядку	$a\,[\rightarrow]\,b = c$	74

Следующие примеры показывают использование операций сдвига.

Пример 1.30. Рассмотрим задачу *сжатия таблицы*. Пусть в машину введена таблица 1000 двоичных чисел x_0, x_1, x_2, ..., x_{999}, каждое из которых известно с точностью, не превышающей 14 двоичных знаков (4 — 5 десятичных знаков).

Сожмем эту таблицу вдвое, помещая в каждую из 500 ячеек a_0, ..., a_{499} по два числа из исходной таблицы.

Посмотрим сначала, каким образом можно два двоичных числа x_0 и x_1, заданных с указанной точностью, поместить в одну ячейку. Двоичное число x_0 состоит из восьми разрядов порядковой части (знак числа и порядок) и 36 разрядов мантиссы. Из мантиссы, согласно условию, достаточно оставить 14 левых разрядов (с 23-го по 36-й), а правые 22

разряда отбросить. Таким образом, в числе x_0 существенными являются 22 левых разряда *). То же можно сказать и о числе x_1.

Если выделить в числах x_0 и x_1 по 22 разряда, то их можно разместить в одной ячейке a_0 по схеме, указанной на рис. 26.

45	44 ← Разряды → 23	22 ← Разряды → 1
0	x_0	x_1

Рис. 26.

Сделать это можно при помощи следующих операций:

1) $\overline{52} \rightarrow x_0 = R$ 3) $\overline{52} \rightarrow x_1 = a_0$

2) $\overline{126} \rightarrow R = R$ 4) $R +, a_0 = a_0$

Команды 1) и 2) превращают в нуль младшие 22 разряда числа x_0 и помещают затем старшие 22 разряда в левую часть ячейки R. При помощи команды 3) старшие 22 разряда числа x_1 помещаются в правую половину ячейки a_0. Наконец, команда 4) ставит в левую часть ячейки a_0 число x_0.

Чтобы сжать всю таблицу, следует эти четыре команды включить в цикл, который должен повториться 500 раз.

Программа цикла с использованием регистра адреса будет иметь вид

	PA	0	0	0
	$Б$	T^0		T
		$T +, (0, 0, 1) = T$		
		$\overline{52} \rightarrow$	$x_0^* = R_1$	
		$\overline{126} \rightarrow$	$R_1 = R_1$	
		$\overline{52} \rightarrow$	$x_1^* = R_2$	
T		$(R_1 +,$	$R_2 = a_0)$	н. п.
	$PA <$	$\overline{1746}$		2^*
	стоп			
T^0		$R_1 +,$	$R_2 = a_0$	

*) Напомним, что 45-й разряд ячейки памяти принимается всегда равным нулю.

Пример 2.30. Разберем задачу, обратную рассмотренной в предыдущем примере. Пусть дана сжатая таблица $a_0, a_1, \ldots, a_{499}$, причем в каждой из ячеек a_i лежит пара чисел x_{2i}, x_{2i+1}, занимающих по 22_{10} разряда каждое. Пусть, кроме того, задана ячейка s, в третьем адресе которой указан некоторый восьмеричный номер n нужного числа

$$s=(0,\ 0,\ n).$$

Требуется выбрать это число из таблицы и записать его в ячейку x.

Выберем сначала слово a_k, содержащее число x_n. Так как числа x_0, x_1 лежат в ячейке a_0, числа x_2, x_3 — в ячейке a_1 и т. д., то число x_n лежит в ячейке a_k с номером k, равным целой части $n/2$. Целую часть $n/2$ можно получить, сдвинув ячейку s на один разряд вправо,

$$\overline{77} - s=q.$$

Если теперь T^0 означает команду $a_0 +$, $0 = x$, то можно переслать слово a_k в ячейку x с помощью следующей программы:

	$\overline{77} \rightarrow$	$s=q$
	$\overline{130} \rightarrow$	$q=q$
	$T^0 +$,	$q=T$
T		н.п.

Остается узнать, в какой половине ячейки x находится нужное число. Для этого надо определить четность числа n: при четном n число x_n находится в левой половине ячейки, а при нечетном — в правой. Четность n определяется значением первого разряда ячейки s. Поэтому можно выяснить четность n, сдвинув ячейку на $44_{10} = 54_8$ разрядов влево. Тогда в ячейке останется лишь последний разряд. Если n — число четное и этот разряд равен нулю, то результатом сдвига будет нулевое слово и выработается сигнал $\omega = 1$. Если же n нечетно, то в сдвинутом слове останется единица последнего разряда, так что управляющий сигнал ω будет равен нулю.

Программу выборки можно теперь закончить командами

1) $\overline{154} \rightarrow s=0$
2) *УО* ——
3) $\overline{52} \rightarrow x=x$ |
4) $\overline{126} \rightarrow x=x$ ↓
5) *стоп*

Действительно, при четном n команда 2) передаст управление команде 3), которая сдвинет ячейку на 22 разряда вправо, в результате чего сотрется правое число в ячейке. После этого команда 4) вернет оставшееся нужное число на место. При нечетном n управление перейдет сразу команде 4), которая сдвинет нужное число налево, стерев ненужное предыдущее.

Таблица 2.30

					1000	*A*	
	$\overline{77} \rightarrow s=q$	1000	0	54	0077	1012	1021
	$\overline{130} \rightarrow q=q$	1	0	54	0130	1021	1021
	$T^0+, q=T$	2	0	13	1011	1021	1003
T	$(a_0+, 0=x)$	3				н. п.	
	$\overline{154} \rightarrow s=0$	4	0	54	0154	1012	0000
	УО ——	5	0	76	0000	1007	0000
	$\overline{52} \rightarrow x=x$ |	6	0	54	0052	1020	1020
	$\overline{126} \rightarrow x=x$ ↓	7	0	54	0126	1020	1020
	стоп	1010	0	77	0000	0000	0000
T^0	$a_0+, 0=x$	1	0	13	2000	0000	1020
s	(0, 0, 17)	2	0	00	0000	0000	0017

Окончательный вид программы приведен в табл. 2.30 в кодированном виде. Здесь предположено, что таблица

находится в ячейках, начиная с 2000; ячейкам s, x и q приданы соответственно адреса 1012, 1020 и 1021, а $n = 17_8$.

С помощью регистра адреса ту же программу можно написать короче:

```
1)        113 →   s=q
2) [PA]           q
3)        a₀* +,  0=x
4)        154 →   s=0
5) УО             ------
6)         52 →   x=x |
7)        126 →   x=x |
8) стоп
9)        (0,     0,  n)
```

Действительно, при сдвиге ячейки s на $13_8 = 11_{10}$ разрядов во втором адресе ячейки q окажется целая часть $n/2$. Она берется в регистр и затем команда 3) переадресуется по PA, перенося в ячейку x нужное слово.

§ 31. Логические операции

В математической логике рассматриваются операции исчисления высказываний, которые с формальной точки зрения эквивалентны операциям над двоичными переменными, принимающими лишь два значения: 0 или 1.

Основным операциям исчисления высказываний соответствуют машинные операции над словами, которые и называются *логическими операциями* машины. В логических операциях каждое слово рассматривается как набор двоичных цифр, не связанных между собою. Значения отдельных разрядов слова a будем обозначать через a_i ($i = 1, 2, \ldots, 45$). Они являются двоичными переменными, поскольку могут принимать значения 0 или 1.

Логические операции выполняются, собственно, не над словами, а над каждым разрядом слова в отдельности, т. е. *поразрядно*. У машины есть три логических операции, которые определяются следующим образом.

Логическим сложением слов a и b называется операция $a \vee b = c$, определяемая формулой

$$a_i \vee b_i = c_i \quad (i = 1, 2, \ldots, 45).$$

Иначе говоря, слово c, являющееся логической суммой слов a и b, образуется в результате логического сложения соответствующих разрядов слов-слагаемых в соответствии со следующей таблицей:

$$\left.\begin{array}{l} 0 \vee 0 = 0 \\ 0 \vee 1 = 1 \\ 1 \vee 0 = 1 \\ 1 \vee 1 = 1 \end{array}\right\} \qquad (1.31)$$

Логическим умножением слов a и b называется операция $a \wedge b = c$, выполняемая в соответствии с формулой

$$a_i \wedge b_i = c_i \quad (i = 1, 2, \ldots, 45),$$

причем значения отдельных разрядов образуются по следующей таблице:

$$\left.\begin{array}{l} 0 \wedge 0 = 0 \\ 0 \wedge 1 = 0 \\ 1 \wedge 0 = 0 \\ 1 \wedge 1 = 1 \end{array}\right\} \qquad (2.31)$$

Сверкой (отрицанием равнозначности) слов a и b называется операция $a \not\sim b = c$, выполняемая в соответствии с формулами:

$$a_i \not\sim b_i = c_i \quad (i = 1, 2, \ldots, 45)$$

$$\left.\begin{array}{l} 0 \not\sim 0 = 0 \\ 0 \not\sim 1 = 1 \\ 1 \not\sim 0 = 1 \\ 1 \not\sim 1 = 0 \end{array}\right\} \qquad (3.31)$$

Из (1.31) — (3.31) видно, что все три логических операции коммутативны, т. е. подчиняются переместительному закону.

При выполнении любой из трех перечисленных логических операций вырабатывается управляющий сигнал $\omega = 0$

во всех случаях, кроме того, когда результатом операции является нулевое слово. В последнем случае получаем $\omega = 1$.

Коды логических операций приведены в таблице 1.31.

Таблица 1.31

Название операции	Обозначение	Код
Логическое сложение .	$a \vee b = c$	75
Логическое умножение .	$a \wedge b = c$	55
Сверка	$a \nsim b = c$	15

Отметим основные случаи применения логических операций.

Логическое сложение выполняется значительно быстрее, чем плавающее и фиксированное, потому что не требует времени для переноса единиц из разряда в разряд. По этой причине его удобно применять *для пересылки слов из одной ячейки в другую*, необходимость в которой встречается довольно часто. В предыдущих главах для пересылки слов мы пользовались плавающим или фиксированным сложением с нулем. Более удобно делать это путем логического сложения с нулем *).

Действительно, при выполнении команды $0 \vee b = c$, в соответствии с табл. 1.31, находим, что $b_i = c_i$ ($i = 1,2, \ldots$ ..., 45), т. е. слово из ячейки b пересылается в ячейку c без изменения. В дальнейшем мы будем широко использовать эту возможность и введем для команды $0 \vee b = c$ сокращенное обозначение $b = c$.

Операция логического сложения применяется также для формирования некоторого слова (например, команды) из отдельных частей, находящихся в различных ячейках. Например, если слово a имеет вид (α, 0, 0), а слово b — вид (0, β, 0), то путем логического сложения $a \vee b = c$ мы получим слово c, имеющее вид (α, β, 0).

*) Применение плавающего сложения с нулем для пересылки слов приводит к *нормализации* этих слов, рассматриваемых как двоичные числа. Это в ряде случаев может быть нежелательным.

Операцию логического умножения часто называют *пересечением* или *выделением*. Чаще всего ее применяют для того, чтобы высечь из слова некоторую группу разрядов, заменив остальные разряды нулями. Для этого нужно логически умножить данное слово на такое, в котором на месте выделяемых разрядов стоят единицы, а в остальных разрядах нули.

Операция сверки применяется главным образом для проверки логических условий, которым можно придать вид совпадения двух слов. Действительно, если слова a и b совпадают, то результат операции $a \not\sim b$ будет, как видно из (3.31), нулевым словом и выработается сигнал $\omega = 1$. Если же a и b отличаются хотя бы одним разрядом, то $\omega = 0$.

Перейдем теперь к рассмотрению некоторых примеров применения логических операций.

Пример 1.31. Числу a присвоить знак числа b, результат записать в ячейку c.

Знак двоичного числа характеризуется 44-м разрядом слова, изображающего это число. Поэтому для решения сформулированной задачи следует в числе b выделить знаковый 44-й разряд, в числе a все остальные разряды (т. е. взять модуль a) и результаты этих двух операций логически сложить.

Знак числа b можно выделить, логически умножая это число на константу e_{44}, имеющую единицу в 44-м разряде и нули в остальных:

$$b \wedge e_{44} = c.$$

Для нахождения $|a|$ можно логически умножить слово a на константу, имеющую единицы во всех разрядах, кроме знакового 44-го; это слово записывается в восьмеричном виде так: 577 7777 7777 7777. Обозначая это слово через $e_{\overline{44}}$, напишем команду нахождения модуля a:

$$a \wedge e_{\overline{44}} = |a|.$$

Присвоение числу $|a|$ знака числа b осуществляется при помощи логического сложения:

$$c \vee |a| = c.$$

Таким образом, задача решается при помощи трех команд:

$$b \wedge e_{44} = c$$
$$a \wedge e_{\overline{44}} = |a|$$
$$c \vee |a| = c$$

и двух констант *):

$$e_{44} : (200;\ 0000,\ 0000,\ 0000),$$
$$e_{\overline{44}} : (577;\ 7777,\ 7777,\ 7777).$$

Пример 2.31. Подсчитать число нулевых двоичных разрядов в слове a; полученное число поместить в порядковую часть слова u.

Очистим сначала счетчик числа нулей:

$$1)\ 0 = u.$$

Для того чтобы узнать, есть ли нуль в первом разряде слова a, выполним операцию логического умножения:

$$2)\ a \wedge (0,\ 0,\ 1) = 0.$$

Если в первом разряде a есть нуль, то эта операция выработает сигнал $\omega = 1$; если в этом разряде единица, то $\omega = 0$. Поэтому после 2) следует поместить две команды:

$$3)\ \textit{У0} \qquad\qquad 5)$$
$$4) \qquad u\ , + (1;\ 0,\ 0,\ 0) = u$$

Сдвинем теперь слово a на один разряд вправо:

$$5)\ \overline{77} \rightarrow a = a.$$

Тогда второй разряд займет место первого и после повторения команд 2) — 4) в u появится число нулей в двух разрядах слова.

Рабочую часть цикла следует повторить $45_{10} = 55_8$ раз. Объединяя команды 1) — 5), организуем проверку окончания цикла по счетчику, записанному в порядковой части

*) Впрочем, для нахождения $|a|$ можно воспользоваться командой вычитания абсолютных величин и тогда константа $e_{\overline{11}}$ не потребуется, но эта операция выполняется дольше, чем логическое умножение.

ячейки v. Кроме того, чтобы программа была самовосстанавливающейся, слово a в памяти не должно «портиться», т. е. изменяться. Поэтому его следует переслать в ячейку α и дальнейшие операции производить над словом α.

Программу можно записать в таком виде:

1)		$(55;\ 0,\ 0,\ 0),$	$-$	$(1;\ 0,\ 0,\ 0)=v$
2)				$0\ \ =u$
3)				$a\ \ =\alpha$
4)		α	$\wedge$	$(0,\ 0,\ 1)=0$
5)	*У0*			
6)		u	$+$	$(1;\ 0,\ 0,\ 0)=u$
7)		$\overline{77}$	$\rightarrow$	$\alpha\ \ =\alpha$
8)		v	$-$	$(1;\ 0,\ 0,\ 0)=v$
9)	*У0*			
10)	*стоп*			

Более короткую и быструю программу можно получить, пользуясь другим алгоритмом. Сложим все разряды слова a. Полученная таким образом сумма будет равна количеству единиц слова. Число нулей можно получить, вычитая эту сумму из числа 55_8 разрядов машинного слова. Это можно осуществить с помощью такой программы:

1)			a	$=\alpha$
2)			$(0,\ 0,\ 55)$	$=u$
3)		$\alpha\ \wedge$	$(0,\ 0,\ 1)$	$=q$
4)		$u\ -,$	q	$=u$
5)		$\overline{77}\rightarrow$	α	$=\alpha$
6)	*У0*			
7)		$\overline{144}\rightarrow$	u	$=u$
8)	*стоп*			

Пример 3.31. Из последовательных чисел a_1, a_2, ..., a_{90}, расположенных в идущих подряд ячейках памяти, найти первое ненулевое и заслать в ячейку u. Если все числа последовательности нулевые, в ячейку u заслать нуль.

Составим программу, образуя цикл при помощи регистра адреса и используя операцию сверки с нулем для нахождения первого ненулевого элемента последовательности:

$$\begin{array}{lcccc} & PA & 0 & 0 & 0 \\ & & a_1^* & \not\sim & 0 = u_1 \\ (1) & PA & < \overline{131} & & \overline{1^*} \\ & стоп & & & \end{array}$$

Для пояснения программы заметим, что при сверке с нулем слово не изменяется; поэтому вторая команда всякий раз записывает в ячейку u очередное слово a. Если оно отлично от нуля, то после этой команды $\omega = 0$ и команда проверки окончания передает управление на стоп при любом PA. Если же слова были нулевыми, то после окончания цикла в ячейке u будет записано нулевое слово.

§ 32. Логические шкалы

При помощи операций логического умножения и сдвига можно организовать работу с *логическими шкалами*.

Логической шкалой называется двоичное слово, в котором каждый разряд имеет значение истинности некоторого высказывания. При помощи одной логической шкалы, содержащейся в ячейке нашей машины, можно записать значения истинности не более 45 высказываний. Если число высказываний больше 45, то под логическую шкалу можно занять несколько ячеек.

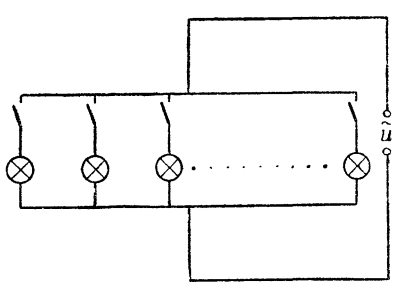


Рис. 27.

Логическими шкалами при программировании пользуются тогда, когда в задаче требуется проверка многих однотипных логических условий.

Пример 1.32. В электрическую цепь параллельно включены 20 лампочек с сопротивлениями R_1, R_2, ..., R_{20} (рис. 27). Каждая лампочка снабжена своим выключателем.

Составить программу расчета мощности, потребляемой этой сетью, при произвольном положении выключателей.

Если все выключатели находятся в положении *включено*, то величина потребляемой мощности W находится по формулам (сопротивлением проводов пренебрегаем)

$$W=\frac{U^2}{R},\quad \frac{1}{R}=\frac{1}{R_1}+\frac{1}{R_2}+\ldots+\frac{1}{R_{20}},$$

где U — напряжение в цепи.

Величина проводимости $1/R$ подсчитывается при помощи простого арифметического цикла:

$$\begin{array}{rcl} PA & 0 & 0 \quad 0 \\ & & 0 = 1/R \\ & \text{«1»} : & R_1^* = k \\ & 1/R + & k = 1/R \\ PA < & \overline{23} & \overline{1^*} \end{array}$$

после чего W определяется двумя арифметическими операциями:

$$U \cdot U = k,$$
$$k \cdot 1/R = W.$$

Рассмотрим теперь более сложный случай, когда часть лампочек отключена от сети, а часть включена. Положение выключателей охарактеризуем 20 двоичными переменными $\lambda_1, \lambda_2, \ldots, \lambda_{20}$. Пусть значение $\lambda_i = 1$ соответствует включенной i-й лампочке, а $\lambda_i = 0$ — выключенной. Тогда проводимость цепи определится формулой

$$\frac{1}{R}=\frac{\lambda_1}{R_1}+\frac{\lambda_2}{R_2}+\ldots+\frac{\lambda_{20}}{R_{20}}.$$

Состояние цепи, т. е. положение выключателей, естественно задать при помощи логической шкалы, для которой мы используем 20 младших разрядов (от 1-го до 20-го) ячейки λ. В первый разряд мы занесем значение λ_1, во второй — λ_2, ..., в двадцатый — λ_{20}. Счет проводимости по последней формуле организуем так: очистим ячейку $1/R$, в которой подсчитывается проводимость:

$$1)\ 0 = 1/R.$$

Проверим при помощи операции логического умножения значение λ_1 первого разряда ячейки λ; если там единица, то прибавим к $1/R$ величину проводимости первой лампочки:

2)		λ	$\wedge$ (0, 0, 1)	$=0$
3)	*У1*		6)	
4)		«1» :	R_1	$=k$
5)		$1/R$ +	k	$=1/R$

Сдвинем теперь шкалу λ на один разряд вправо

$$6)\ \overline{77} \rightarrow \lambda = \lambda,$$

увеличим второй адрес команды 4) на 1 и передадим управление на 2). Тогда к величине $1/R$ прибавится второе слагаемое λ_2/R_2.

Продолжая таким образом далее, мы получим величину проводимости, а затем и мощность W.

Используя регистр адреса, получим следующую программу расчета мощности:

1)	*РА*	0	0	0
2)			0	$=1/R$
3)			λ	$=\lambda_{раб}$
4)		$\lambda_{раб}$ $\wedge$ (0, 0, 1)		$=0$
5)	*У1*			
6)		«1» :	R_1^*	$=k$
7)		$1/R$ +	k	$=1/R$
8)		$\overline{77}\rightarrow$	$\lambda_{раб}$	$=\lambda_{раб}$
9)	*РА* <	$\overline{23}$		$\overline{1}$*
10)		U ·	U	$=k$
11)		k ·	$1/R$	$=W$
12)	*стоп*			

Пример 2.32. Ассортимент n магазинов одного и того же профиля состоит из k товаров ($k \leqslant 45$). Каждый из m покупателей хочет, обойдя все магазины, купить некоторые товары.

Выяснить по совокупности заявок от покупателей и имеющихся в наличии товаров, какие товары являются ходовыми, какие — неходовыми и какие — дефицитными (сведения по совокупности n магазинов). Каждому покупателю выдать справку о дефицитных (по его заявке) товарах.

Условия задачи удобнее всего записать в виде логических шкал. Припишем товарам номера 1, 2, ..., $\varkappa$ ($\varkappa \leqslant 45$). Наличие товаров в i-м магазине охарактеризуем шкалой H_i, в которой в r-м разряде поставим 1, если соответствующий товар имеется в продаже, и 0, если его нет *). Таким же способом составим шкалу $З_j$ для заявки j-го покупателя: в r-м разряде этой шкалы поставим единицу, если r-й товар требуется покупателю, и 0, если он не нужен. Таким образом, наличие товаров в магазинах характеризуется n шкалами H_1, H_2, ..., H_n, а заявки покупателей m шкалами: $З_1$, $З_2$, $З_3$,..., $З_m$, которые предполагаются заданными.

Введем следующие определения:

1) товар называется з а я в л е н н ы м, если он присутствует в заявке хотя бы одного из покупателей;

2) товар и м е е т с я в н а л и ч и и, если он есть хотя бы в одном из магазинов;

3) товар называется х о д о в ы м, если он заявлен и имеется в наличии;

4) товар называется н е х о д о в ы м, если он не заявлен, но имеется в наличии;

5) товар называется д е ф и ц и т н ы м, если он заявлен, но его нет в наличии.

Заявленные, имеющиеся в наличии, ходовые, неходовые и дефицитные товары мы охарактеризуем соответственно шкалами $З$, $Н$, $Х$, $НХ$, $Д$; в каждой шкале на j-м месте должна стоять 1, если j-й товар обладает соответствующим свойством, и 0 — в противоположном случае.

Ответом на вопрос задачи могут служить шкалы $Х$, $НХ$ и $Д$. Кроме того, нужно найти шкалы $Д_1$, $Д_2$, ..., $Д_m$ дефицита для отдельных покупателей.

Из приведенных определений следует, что шкалы $З$, $Н$, $Х$, $НХ$, $Д$, $Д_j$ связаны следующими соотношениями:

$$\begin{aligned} &З \wedge Н = Х, \\ &\overline{З} \wedge Н = НХ, \\ &З \wedge \overline{Н} = Д, \\ &З_j \wedge \overline{Н} = Д_j \qquad (j = 1, 2, \ldots, m). \end{aligned} \tag{1.32}$$

Так как в машине нет операции отрицания и у нас нет шкал $\overline{Н}$ и $\overline{З}$, то последние три соотношения лучше заменить другими. Легко проверить **) справедливость соотношения

$$\bar{a} \wedge b = (a \wedge b) \nsim b.$$

Поэтому

$$(З \wedge Н) \nsim Н = НХ, \tag{2.32}$$

$$(З \wedge Н) \nsim З = Д, \tag{3.32}$$

$$(З_j \wedge Н) \nsim З_j = Д_j \qquad (j = 1, 2, \ldots, m). \tag{4.32}$$

*) Заметим, что такая шкала хранит значения истинности $\varkappa$ высказываний вида: «Товар есть в магазине».

**) Предлагаем читателю убедиться в этом самостоятельно, давая a и b значения 0 и 1 и вычисляя при этом значения левой и правой частей равенства.

Для решения задачи в силу формул (1.32) — (4.32) достаточно найти шкалу заявок $З$ и шкалу наличия $Н$. Из определения 1) следует, что r-й товар заявлен, если в r-м разряде хотя бы в одной из шкал $З_1$, $З_2$, ..., $З_m$ стоит 1. Поэтому шкалу $З$ можно получить логическим суммированием шкал $З_1$, $З_2$, ..., $З_m$

$$\begin{array}{cccc} & 0=З & & З \quad \vee \quad \overline{З_i^*=З}\uparrow \\ РА & 0\ 0\ 0 & РА < \overline{m-1} & \overline{\quad i^*\quad} \end{array}$$

Шкала $Н$, очевидно, также может быть найдена логическим суммированием шкал $Н_1$, $Н_2$, ..., $Н_n$:

$$\begin{array}{cccc} & 0=Н & & Н \quad \vee \quad \overline{Н_i^*=Н}\uparrow \\ РА & 0\ 0\ 0 & РА < \overline{n-1} & \overline{\quad i^*\quad} \end{array}$$

Теперь составим программу нахождения шкал $Х$, $НХ$, $Д$, $Д_j$ ($j = 1, 2, ..., m$)

$$\begin{array}{cccc} & З \wedge Н = Р & & З_i^* \quad \wedge \quad Н = Р \uparrow \\ & Р \not\sim Н = НХ & & Р \quad \not\sim З_i^* = Д_i^* \downarrow \\ & Р \not\sim З = Д & РА < \overline{m-1} & \overline{\quad i^*\quad} \\ РА & 0\ 0\ 0 & & \end{array}$$

Для полного решения задачи остается только объединить все написанные выше команды.

В задачах с более сложными логическими условиями, чем предыдущие, шкалу делят на группы из нескольких разрядов. Каждая такая группа содержит информацию о тех или иных условиях.

§ 33. Некоторые логические задачи

Наряду с вычислительными задачами на машинах решают и самые разнообразные невычислительные: программируют игру в шашки и шахматы, диагностику заболеваний, доказательство теорем и другие логические задачи. Такого рода программы, как правило, почти не содержат арифметических операций. Содержимое ячеек в них обычно не является числами, а играет роль некоторой информации, над которой производятся главным образом логические операции.

Рассмотрим несколько примеров, дающих представление о возможностях программирования игры в шахматы.

Прежде всего необходимо условиться о способе задания информации. Для изображения шахматной доски выделим восемь ячеек памяти a_0, a_1, ..., a_7 и в каждой из ячеек — первые восемь разрядов. При этом каждая ячейка соответствует горизонтали шахматной доски, а каждый из выделенных разрядов — полю. Положение фигур на доске можно задать, расставив единицы на всех занятых полях и нули на свободных. Такое задание положения мы будем называть растром фигур (см. рис. 28).

С другой стороны, для каждой отдельной фигуры можно задать ее координаты, указав номер горизонтали и номер поля в ней, на котором стоит фигура. Будем считать, что горизонтали нумеруются сверху

вниз и их номера i меняются в пределах $0 \leqslant i \leqslant 7$, а поля — справа налево и их номера k удовлетворяют условиям $1 \leqslant k \leqslant 8$, что соответствует выбранным обозначениям для ячеек и нумерации разрядов. Координаты будем записывать в фиксированной форме в одной ячейке: номер горизонтали во втором, а номер поля — в третьем адресе, т. е. $(0, i, k)$. Например, полю $e4$ в обычной шахматной кодировке будут соответствовать координаты, записанные машинным словом (0, 3, 5), а полю $g3$ — координаты (0, 2, 7).

Теперь мы можем перейти к примерам.

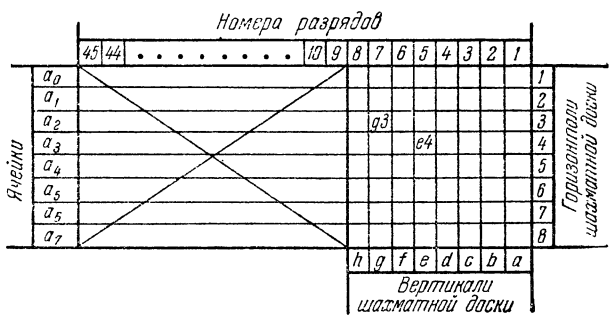


Рис. 28.

Пример 1.33. Определим координаты единственной фигуры, стоящей на шахматной доске, т. е. номер ячейки и номер разряда, где стоит единственная единица в растре фигур.

Номер ячейки можно определить с помощью программы

$$\begin{array}{lll} PA & 0 & 0 \quad 0 \\ & a_0^* \wedge (0, 0, 377) = R_1 \uparrow & \\ (1)\quad PA < \overline{7} & | \qquad \overline{1}^* & \end{array}$$

Действительно, константа 377_8 состоит из восьми единиц в правых разрядах. Поэтому команда $a_0^* \wedge (0, 0, 377) = R_1$ перекладывает соответствующие восемь разрядов ячейки a_i в рабочую ячейку R_1, одновременно проверяя, есть ли в них хотя бы одна единица. В том случае, когда единица находится в данной ячейке a_i, в ячейку R_1 запишется слово, отличное от чисто нулевого, поэтому выработается управляющий сигнал $\omega = 0$ и команда проверки окончания цикла передаст управление не назад, а на следующую. При этом содержимое регистра адреса увеличится на единицу.

Записать в средний адрес ячейки R_2 номер горизонтали, в которой стоит фигура, можно с помощью команды

$$PA \quad F^* \quad 0 \quad R_2,$$

так как прибавление числа $F = 7777$ уменьшит содержимое регистра на единицу и сделает его снова равным номеру перекладываемой ячейки a_i.

Остается проанализировать содержимое ячейки R_1 и выяснить, в каком именно разряде этой ячейки стоит единица. Учитывая, что предыдущая команда уже очистила регистр, это можно сделать так:

```
                (0, 0, 1)=K
        R1 ∧    (0, 0, 1)= 0   ↑
У0                             |
|       K +,    (0, 0, 1)=K    |
|       7̄7̄ →,     R1    =R1    |
|PA < 7̄                 1̄*     |
↓       K +,     R2    =K
```

Последняя команда образует в ячейке K координаты фигуры в требуемой форме, оставляя операционную часть ячейки K равной нулю, хотя в ячейке R_2 имелся отличный от нуля код (052).

Пример 2.33. Пусть нам даны координаты фигуры в ячейке K. Построим растр, т. е. занесем единицу в разряд, соответствующий данному полю требуемой горизонтали; иначе говоря, выполним задание, противоположное рассмотренному в предыдущем примере. Нулевой горизонтали соответствует ячейка a_0.

Читателю будет очень полезно самостоятельно убедиться в том, что это можно сделать с помощью следующей программы:

```
PA      0        0        0
        K ∧ (0, 0, F)=R1
             (0, 0, 1)=R2
T       R1 ∧     R2     = 0
У0                      |
|       1̄0̄1̄ →,    R2     =R2
|PA <   7̄         T       1̄*
↓[PA]   0        K       0
                 R2     =a0*
```

Пример 3.33. На шахматной доске стоит единственная ладья. Построим растр для полей, находящихся под боем этой ладьи, т. е. пометим единицами все поля, бьющиеся этой ладьей.

Допустим, что нам известны и растр, и координаты ладьи. Тогда можно действовать таким образом. Чтобы получить все поля по вертикали, надо во все ячейки a_i разослать содержимое той горизонтали, где стоит ладья. Это можно сделать с помощью команд

```
[PA]   0   K      0
           a0* = R1
PA     0   0      0
           R1 = a0*  ↑
PA <7̄          |    |
               1̄*
```

Все единицы в нужную горизонталь можно теперь заслать двумя командами

$$[PA]\ 0 \qquad K \qquad 0$$
$$(0,\ 0,\ 377) = a_0^*$$

Заметим, кстати, что полученный нами растр одновременно определяет возможные ходы ладьи на свободной доске. Аналогичные растры возможных ходов можно строить и для других фигур, хотя и по несколько более сложным правилам.

Нужно сказать, что выбранный нами способ задания информации удобен для работы, но не рационален, так как в каждой ячейке используется только 8 разрядов из 45. Можно сократить требуемую память, объединив содержимое четырех ячеек в одну. Тогда в ячейке будет занято 32 разряда и можно будет поместить полный растр в двух ячейках памяти. Правда, это потребует некоторого усложнения программы.

ГЛАВА VII

ОРГАНИЗАЦИЯ ПРОГРАММЫ

§ 34. Операция безусловной передачи управления с возвратом. Блоки и подпрограммы

Во многих задачах, решаемых на машине, приходится неоднократно вычислять одни и те же функции. Приведем простейший пример такого рода.

Пример 1.34. При заданных значениях x, y, z вычислить величину

$$w=\frac{2x^3-5x^2+6x-4}{2y^3-5y^2+6y-4}-9\,(2z^3-5z^2+6z-4).$$

Кубический многочлен в числителе запишем так

$$[(2x-5)x+6]x-4.$$

Аналогично переписываются и два других многочлена.

Программа вычисления w, написанная «в лоб», имеет такой вид:

1) «2» · x = R_1	12) R_2 — «4» = R_2
2) R_1 — «5» = R_1	13) «2» · z = R_3
3) R_1 · x = R_1	14) R_3 — «5» = R_3
4) R_1 + «6» = R_1	15) R_3 · z = R_3
5) R_1 · x = R_1	16) R_3 + «6» = R_3
6) R_1 — «4» = R_1	17) R_3 · z = R_3
7) «2» · y = R_2	18) R_3 — «4» = R_3
8) R_2 — «5» = R_2	19) R_1 : R_2 = R_1
9) R_2 · y = R_2	20) «9» · R_3 = R_3
10) R_2 + «6» = R_2	21) R_1 — R_3 = w
11) R_2 · y = R_2	22) *стоп*

Легко понять, что вычисление w запрограммировано здесь очень нецелесообразно. При программировании мы не воспользовались тем, что в формулу для w входят три значения одного и того же кубического многочлена:

$$Q(\alpha) = 2\alpha^3 - 5\alpha^2 + 6\alpha - 4.$$

Формулу для w можно записать так:

$$w = Q(x)/Q(y) - 9Q(z).$$

Из этой записи видно, что программирование счета w естественно вести следующим образом: отдельно выписать программу расчета $Q(\alpha)$; затем, обращаясь к этой программе, при $\alpha = x$ получить числитель, при $\alpha = y$ — знаменатель; после этого подсчитать дробь $Q(x)/Q(y)$, и, обращаясь к программе $Q(\alpha)$ при $\alpha = z$, найти $Q(z)$; наконец, получить по последней формуле w.

Напишем прежде всего программу вычисления $Q(\alpha)$:

Счет Q

Q	«2»$\cdot\ \alpha = R_0$	$R_0 + $«6»$= R_0$
	$R_0 - $«5»$= R_0$	$R_0 \cdot \alpha = R_0$
	$R_0 \cdot \alpha = R_0$	$R_0 - $«4»$= \gamma$
	к.Q	н. п.

Здесь через γ обозначена ячейка результата. Первая команда программы помечена буквой Q, последняя, конечная, — буквами к. Q (как мы увидим дальше, эта свободная ячейка нужна для организации счета w).

Для того чтобы получить $Q(x)$, зашлем x во входную ячейку α программы *Счет Q* и передадим управление на начало этой программы:

Счет w

1) $x = \alpha$
2) *Б Q*

Тогда после выполнения шести команд программы *Счет Q* в ячейке γ окажется величина $Q(x)$.

Для продолжения вычислений необходимо, чтобы после окончания работы программы счета Q управление передавалось следующей команде 3) нашей основной программы счета w. Это можно обеспечить, заслав в ячейку к. Q команду

безусловной передачи управления, которую нужно еще заготовить. Однако этим дело не ограничится, потому что к программе счета Q нам придется обращаться еще дважды и всякий раз при таком обращении необходимо обеспечивать возврат из программы счета Q в различные места программы счета w.

Возможность возврата в нужное место основной программы можно обеспечить с помощью специальной команды *безусловной передачи управления с возвратом*. Эту команду мы будем записывать в виде

$$БВ \quad a \quad b \quad c.$$

Выполняется она так: управление, как и при выполнении команды $Б$, передается ячейке b. Одновременно в ячейку c заносится не содержимое ячейки a, как при выполнении команды $Б$, а *команда передачи управления ячейке* a

$$БВ \quad 0 \quad a \quad 0.$$

Таким образом, если при обращении к программе счета Q воспользоваться вместо команды 2) этой командой в виде

$$БВ \quad 3) \quad Q \quad к.Q,$$

то одновременно с передачей управления ячейке Q в ячейку $к.Q$ запишется команда передачи управления команде 3) программы счета w, т. е. возврат в эту программу будет обеспечен.

Нужную нам программу можно теперь записать таким образом:

Счет w

1)	$x=\alpha$	7)	$z=\alpha$
2)	$БВ$ 3) Q к.Q	8)	$БВ$ 9) Q к.Q
3)	$\gamma=R_1$	9)	«9»$\cdot\gamma=R_2$
4)	$y=\alpha$	10)	$R_1-R_2=w$
5)	$БВ$ 6) Q к.Q	11)	*стоп*
6)	$R_1:\gamma=R_1$		

Программа вычисления w состоит из написанных здесь 11 команд и 7 команд, служащих для счета Q. Программа счета Q называется *блоком*, а команды 2), 5), 8) — *командами обращения к блоку*.

Блоки, к которым приходится обращаться из нескольких мест программы, мы будем называть *подпрограммами*.

Из рассмотренного примера видно, что выделение счета в отдельную подпрограмму позволило сократить программу. Экономия памяти при выделении в подпрограммы счета функций, которые приходится вычислять в десятках мест программы, окажется, конечно, еще более существенной. Не менее существенным является еще то обстоятельство, что выделение часто встречающихся функций в подпрограммы не только экономит память, но и дает возможность записывать программу более естественно и наглядно.

При обращении к блоку управление передается началу блока, а команда возврата засылается в его конец, т. е. обращение происходит по команде

БВ a Начало блока Конец блока

Эта команда дает возможность возвратиться к любой команде *a*. Однако в большинстве случаев возврат осуществляется на следующую команду основной программы.

Для упрощения записи таких команд введем следующее условие. Обозначим через *Я* адрес команды обращения к блоку, в которой находится управление в данный момент. Тогда *Я* + 1 будет адресом следующей команды и обращение к блоку запишется так:

БВ Я+1 Начало блока Конец блока

Команды 2), 5), 8) предыдущей программы запишутся при таком условии одинаково:

БВ Я+1 *Q* к.*Q*.

При этом для команды 2) *Я* + 1 будет иметь значение 3), для команды 5) — значение 6), а для команды 8) — значение 9).

Заметим, что это обращение имеет то неудобство, что для кодировки надо знать д в а а д р е с а: адрес начала блока *Q* и адрес конца блока к. *Q*. Можно устранить это неудобство, вводя для всех блоков, которые могут встретиться при программировании, одну и ту же ячейку конца Ω.

Тогда обращение к блоку примет вид

БВ Я+1 *Q* Ω,

Таблица 1.34

	Счет w				1200	A	
	$x = \alpha$	1200	0	75	0000	1231	0140
	БВ Я+1 Q Ω	1	0	16	1202	1220	0007
	$\gamma = R_1$	2	0	75	0000	0160	0011
	$y = \alpha$	3	0	75	0000	1232	0140
	БВ Я+1 Q Ω	4	0	16	1205	1220	0007
	$R_1 : \gamma = R_1$	5	0	04	0011	0160	0011
	$: z = \alpha$	6	0	75	0000	1233	0140
	БВ Я+1 Q Ω	7	0	16	1210	1220	0007
	«9» $\cdot \gamma = R_2$	1210	0	05	0111	0160	0012
	$R_1 - R_2 = w$	1	0	02	0011	0012	1234
	стоп	2	0	77	0000	0000	0000

	Счет Q				1220	A	
Q	«2» $\cdot \alpha = R_0$	1220	0	05	0102	0140	0010
	$R_0 -$ «5» $= R_0$	1	0	02	0010	0105	0010
	$R_0 \cdot \alpha = R_0$	2	0	05	0010	0140	0010
	$R_0 +$ «6» $= R_0$	3	0	01	0010	0106	0010
	$R_0 \cdot \alpha = R_0$	4	0	05	0010	0140	0010
	$R_0 -$ «4» $= \gamma$	5	0	02	0010	0104	0160
	Б Ω	6	0	56	0000	0007	0000

а для того, чтобы обеспечить возврат из блока Q к основной программе, следует в последнюю (ранее свободную) ячейку блока Q поставить команду *Б Ω*

$$
\begin{array}{l|l}
Q & \text{«2»} \cdot \alpha = R_0 \qquad R_0 \cdot \alpha = R_0 \\
 & R_0 - \text{«5»} = R_0 \qquad R_0 - \text{«4»} = \gamma \\
 & R_0 \cdot \alpha = R_0 \qquad \textit{Б} \quad \Omega \\
 & R_0 + \text{«6»} = R_0
\end{array}
$$

В табл. 1.34 приведена программа счета w в содержательных обозначениях и закодированном виде. В левой части снята ненужная здесь нумерация команд. Операция *БВ* имеет код 16. При кодировке за величинами, входящими в программу, закреплены следующие ячейки:

x	1231	R_1	0011	«2»	0102
y	1232	R_2	0012	«4»	0104
z	1233	Ω	0007	«5»	0105
w	1234	α	0140	«6»	0106
R_0	0010	γ	0160	«9»	0111

§ 35. Блочное программирование

На машине обычно решают задачи, программы которых содержат многие сотни, а иногда и тысячи команд. В таких задачах всегда следует выделять части, имеющие самостоятельное значение, и в программе эти части писать как отдельные блоки.

Выделение отдельных частей программы в блоки следует производить не только в сложных, но и в простых программах. Так, например, если рабочая часть цикла содержит более двух-трех команд, ее следует выделять в отдельный блок. Тем более целесообразно выделять в отдельный блок внутренние циклы в двойных и многократных циклах.

Пример 1.35. Рассчитать таблицу значений функции

$$y=(x-1/x)^2+x^3$$

для $x = 1, 2, 3, \ldots, 20$.

Напишем прежде всего блок вычисления y при заданном значении x. При этом, как было указано в § 34, обращаясь к блоку, мы будем засылать команду возврата в стандартную ячейку Ω и кончать блок командой передачи управления этой ячейке:

$y(x)$	«1» : $x=R_1$	$R_2 \cdot x = R_2$	
	$x-R_1=R_1$	$R_1+R_2=y$	
	$R_1 \cdot R_1 = R_1$	*Б*	Ω
	$x \cdot x = R_2$		

Теперь при помощи регистра адреса образуем цикл счета y для различных значений x:

Таблица	*РА*	0	0	0
	Б	«1»	*Я*+2	x
		x + «1» = x		
	БВ	*Я*+1	$y(x)$	Ω
			$y = y_1^*$	
	РА	$<\overline{23}$		$\overline{1}$*
	стоп			

В этом примере программа, в которой счет y не был бы выделен в отдельный блок, была бы короче на три ячейки. Однако потеря трех ячеек в написанной программе с лихвой компенсируется тем обстоятельством, что программа выигрывает в простоте и наглядности. Отметим, в частности, что *команда обращения к блоку*

$$БВ\ \ Я+1\ \ y(x)\ \ \Omega$$

выглядит в программе как «элементарная» операция: счет $y\ (x)$. Тип этой элементарной операции зависит от того, что делает сам блок *счет* $y\ (x)$.

В случае необходимости можно в рассмотренном цикле протабулировать при $x = 1, 2, 3, \ldots, 20$ другую функцию $z\ (x)$. Для этого достаточно заменить в программе обращение к блоку

$$БВ\ \ Я+1\ \ y(x)\ \ \Omega$$

на

$$БВ\ \ Я+1\ \ z(x)\ \ \Omega$$

и написать блок счета $z\ (x)$.

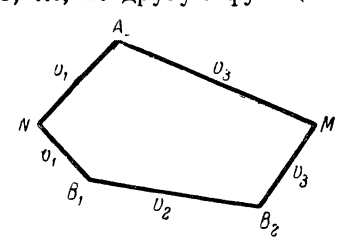


Рис. 29.

Пример 2.35. Самолет должен вылететь из пункта N, взять груз в некоторых пунктах и доставить его в пункт M. Пунктами погрузки может быть либо пункт A, либо два пункта B_1 и B_2 (рис. 29). Расстояния NA, AM, NB_1, B_1B_2, B_2M заданы. Самолет без груза летит со скоростью v_1, с неполным грузом — со скоростью v_2, с полным грузом — со скоростью v_3.

В пункте B_1 на погрузку тратится τ_1 минут, в пункте B_2 — τ_2 минут, в пункте A — τ_3 минут. По какому из путей, NAM или NB_1B_2M, должен лететь самолет, чтобы быстрее доставить груз?

Здесь естественно выделить в программе три блока: счет времени полета через пункт A (блок *Через A*), счет времени полета через пункты B_1 и B_2 (блок *Через* B_1, B_2) и, наконец, сравнение этих двух времен (блок *Сравнение*).

Блок *Сравнение* должен в результате своей работы выработать некоторый признак, по которому можно судить о том, по какому пути должен лететь самолет. Пусть например, в ячейку v засылается нуль, если выгоден путь через A, и (0, 0, 1), — если через B_1B_2.

Блоки программы имеют следующий вид:

Через A

$NA : v_1 = R_1$
$R_1 + \tau_3 = R_1$
$AM : v_3 = t_A$
$t_A + R_1 = t_A$
Б Ω

Через B_1B_2

$NB_1 : v_1 = R_1$ $\quad R_2 + \tau_2 = R_2$
$R_1 + \tau_1 = R_1$ $\quad R_1 + R_2 = R_1$
$B_1B_2 : v_2 = R_2$ $\quad B_2M : v_3 = R_2$
$R_1 + R_2 = t_B$
Б Ω

Сравнение

$t_A - t_B = 0 \rightarrow$
У1 0 Я+2 v
(0, 0, 1) = v
Б Ω

Чтобы объединить эти три блока в единую программу, напишем *собирающую программу*, состоящую из обращений к этим трем блокам и *стопа*:

Собирающая | *БВ Я*+1 *Через A* Ω
| *БВ Я*+1 *Через* B_1B_2 Ω
БВ Я+1 *Сравнение* Ω
стоп

Заметим, что при принятом способе обращения к блокам команды обращения (в содержательных обозначениях) отличаются лишь вторыми адресами. Поэтому для сокращения записи и для наглядности можно писать в собирающей только обозначения вторых адресов. Тогда собирающая рассмотренного примера примет вид

Собирающая | *Через A*
| *Через* B_1B_2
Сравнение
стоп

Программа решения рассмотренной задачи состоит из *Собирающей* и трех блоков: *Через A*, *Через* B_1B_2, *Сравнение*. Для решения задачи все эти четыре программы вместе с исходными данными нужно закодировать, ввести в память и передать управление первой команде собирающей.

Блочная структура программы дает возможность в ряде случаев переходить от решения простых задач к решению более сложных, не меняя написанных блоков, а внося в них лишь некоторые добавления и исправления и дописывая новые блоки.

Пример 3.35. Пусть в задаче предыдущего примера известны не расстояния NA, AM, NB_1, B_1B_2, B_2M, а прямоугольные декартовы координаты точек N, A, M, B_1, B_2. Тогда для решения задачи следует, ничего не меняя в блоках *Через A*, *Через* B_1B_2, *Сравнение* предыдущего примера, написать еще четвертый блок, определяющий нужные расстояния, а собирающую дополнить обращением к этому блоку.

Новая собирающая | *Расстояния*
| *Через A*
Через B_1B_2
Сравнение
стоп

В предыдущей задаче обращения к блокам содержались лишь в собирающей программе. В данном случае блок *Расстояния* удобно написать как собирающую, в которой последовательными обращениями к блоку *Длина* будут вычисляться нужные расстояния. Этот блок *Длина* вычисляет длину отрезка K_1, K_2 по координатам концов отрезка K_1 (x_1, y_1), K_2 (x_2, y_2) в соответствии с формулой

$$l = K_1K_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

и имеет вид

Длина

$x_2 - x_1 = R_1$ $\quad$ $R_1 + R_2 = R_1$
$y_2 - y_1 = R_2$ $\quad$ $\sqrt{R_1} \quad = l$
$R_1 \cdot R_1 = R_1$ $\quad$ *Б* $\quad$ Ω
$R_2 \cdot R_2 = R_2$

Собирающая *Расстояния* пишется теперь в обычном виде:

Расстояния

$x_N = x_1$	*БВ Я*+1 *Длина* Ω
$y_N = y_1$	$l = B_2M$
$x_A = x_2$	$x_{B_1} = x_1$
$y_A = y_2$	$y_{B_1} = y_1$
БВ Я+1 *Длина* Ω	*БВ Я*+1 *Длина* Ω
$l = NA$	$l = B_1B_2$
$x_M = x_1$	$x_N = x_2$
$y_M = y_1$	$y_N = y_2$
БВ Я+1 *Длина* Ω	*БВ Я*+1 *Длина* Ω
$l = AM$	$l = NB_1$
$x_{B_2} = x_2$	*Б* $\quad$ Ω
$y_{B_2} = y_2$	

Блок *Длина*, к которому имеется пять обращений, можно назвать подпрограммой.

Посмотрим теперь, как будет работать блок *Расстояния* при обращении к нему из *Новой собирающей*.

Это обращение имеет вид

1) *БВ Я*+1 *Расстояния* Ω
2) . . . . . .

Команда 1) передает управление на начало блока *Расстояния* и засылает в Ω возврат на команду 2)

БВ 2)

Затем после засылки y_A в y_2 команда

БВ Я+1 *Длина* Ω

передает управление на подпрограмму *Длина*, заслав в Ω возврат на команду

$$l = NA.$$

Таким образом, команда возврата в *Новую собирающую*, ранее засланная в ячейку Ω, окажется безвозвратно потерянной и после выполнения последней команды блока *Расстояния* мы возвратимся к предпоследней команде этого блока, а не к команде 2) *Новой собирающей*, что необходимо для правильной работы программы.

Отсюда видно, что блок, содержащий обращение к другим блокам, нельзя заканчивать командой

Б Ω.

Чтобы обеспечить правильную работу такого блока, следует оставить в его конце свободную ячейку *конец*, а в начале блока выполнить команду пересылки

$$\Omega = конец.$$

Теперь в начале работы блока *Расстояния* команда возврата в основную программу, записанная в ячейке Ω, сразу же пересылается *в конец этого блока*. Поэтому любые дальнейшие обращения к другим блокам (в частности, к стандартным подпрограммам) изнутри данного не испортят команду возвращения в основную программу.

Отметим, что расмотренный прием сохранения команды возврата при оформлении блока является общим. К любому блоку можно обращаться при помощи команды

БВ S Начало блока Ω.

При этом первой командой блока должна быть пересылка

$$\Omega = конец,$$

где *конец* — последняя ячейка данного блока, которую необходимо оставлять свободной. В этом случае внутри

данного блока можно в любом количестве ставить подобные же обращения к другим блокам, которые в свою очередь следует оформлять таким же способом.

Самые внутренние блоки, которые не обращаются ни к каким другим, можно было бы не начинать с команды

$$\Omega = конец,$$

а заканчивать командой

$$Б \quad \Omega.$$

Однако следует оформлять все блоки, в том числе и самые внутренние, указанным *стандартным образом*.

Такая стандартизация нужна для того, чтобы гарантировать себя от возможных ошибок. Кроме того, начиная писать блок, никогда нельзя быть уверенным в том, что некоторую его часть не нужно будет выделить в качестве отдельного блока.

Из рассмотренных примеров видно, что блоки могут быть весьма различными. Одни из них состоят просто из расписки формул или цикла, другие содержат обращение к подпрограммам, третьи — проверку каких-либо логических условий. Вообще, блок может быть сколь угодно сложным образованием. Единственным свойством блока, которое можно считать его определением, является следующее:

При обращении к блоку при помощи команды

$$БВ \quad S \text{ Начало блока } \Omega$$

можно быть уверенным в том, что после его работы управление перейдет к ячейке S.

Обращения к блокам программы, расположенные в нужной последовательности, образуют *блок-программу*, которую мы уже ввели ранее под названием *собирающей*.

Если обращения к некоторым блокам являются условными, то блок-программа должна содержать проверку соответствующих условий. При этом, если передача управления некоторому блоку зависит от результатов работы предыдущего, предыдущий блок должен быть устроен так, чтобы он вырабатывал признаки, *проверка которых должна производиться в собирающей, но не в блоке*. Это требование непосредственно вытекает из данного выше определения блока.

Рассмотрим пример с условным обращением к блокам из собирающей.

Пример 4.35. Резервуар с горючим имеет форму усеченного конуса с высотой l и радиусами оснований R и r, завершенного цилиндром с радиусом основания R (рис. 30). В резервуар налито v литров горючего. Найти высоту λ уровня жидкости, отсчитываемую от дна резервуара.

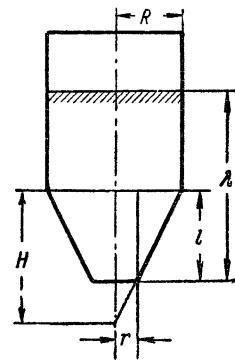


Рис. 30.

Очевидно, высота λ должна вычисляться по разным формулам в зависимости от того, больше или меньше величина v объема усеченного конуса $v_{УК}$. Поэтому, прежде всего, при помощи блока, который мы назовем *Усеченный конус*, следует вычислить объем $v_{УК}$ усеченного конуса. Затем, если $v < v_{УК}$, то λ следует подсчитывать как высоту наполненной жидкостью части усеченного конуса (блок *По конусу*), если же $v \geqslant v_{УК}$, то λ равно высоте наполненной жидкостью части цилиндра, сложенной с l (блок *По цилиндру*). В соответствии с этим планом напишем блок-программу задачи:

Резервуар		*Усеченный конус*		
	$v-$	$v_{УК}=$		Δv
	У0		$Я+2$	
	БВ	$Я+2$	*По конусу*	Ω
			По цилиндру	
	стоп			

Здесь мы воспользовались уже указанным ранее способом сокращенной записи: если при обращении к блоку возвращение происходит на следующую команду ($Я+1$), то в обращении записывается лишь второй адрес, т. е. название блока.

Объем усеченного конуса будем находить как разность объемов двух полных конусов. Для того чтобы это сделать, следует предварительно определить высоту большего конуса H. Поэтому блок *Усеченный конус* мы построим как собирающую, состоящую из обращения к блоку *Высота*, вычисляющему H, двух обращений к блоку *Конус* со входными

ячейками ρ (радиус основания) и h (высота) и выходной ячейкой v_K и команды вычитания объемов большего и меньшего конусов:

Усеченный конус
$\Omega = конец$
Высота
$H = h$
$R = \rho$
Конус
$v_K = v_{зап}$
$H - l = h$
$r = \rho$
Конус
$v_{зап} - v_K = v_{УК}$
конец

Напишем теперь блоки *Высота* и *Конус*. Из подобия треугольников, получаемых в сечении малого и большого конусов, имеем

$$H = lR/(R-r).$$

Поэтому блок *Высота* записывается в виде

Высота
$\Omega = конец$
$l \cdot R = k_1$
$R - r = k_2$
$k_1 : k_2 = H$
конец

Объем конуса определяется как одна треть площади основания, умноженная на высоту:

Конус
$\Omega = конец$
$\pi \cdot \rho = q$
$q \cdot \rho = q$
$q \cdot h = q$
$q \cdot «1/3» = v_K$
конец

Таким образом, программа нахождения объема усеченного конуса состоит из собирающей (*Усеченный конус*) и двух блоков — *Высота* и *Конус*.

Составим теперь блок *По цилиндру*, работающий лишь при $v \geqslant v_{УК}$. В этом блоке следует вычислить площадь основания цилиндра S, разделить вычисленный в собирающей

Резервуар излишек объема Δv на S и сложить эту величину с l:

По цилиндру	$\Omega = конец$	$\Delta v : S = \lambda$
	$\pi \cdot R = S$	$\lambda + l = \lambda$
	$S \cdot R = S$	*конец*

При $v < v_{УК}$ должен работать блок *По конусу*. При составлении этого блока мы воспользуемся тем обстоятельством, что в блоке *Усеченный конус* уже вычислены высота h и объем v_K малого конуса.

Между искомой величиной λ и известными величинами v, v_K, h имеет место соотношение

$$v + v_K = (1/3)\,\pi\rho_1^2\,(h + \lambda),$$

где ρ_1 — радиус поверхности жидкости. Из подобия треугольников имеем

$$\rho_1 / r = (h + \lambda)/h.$$

Поэтому

$$v + v_K = (1/3)\,\pi r^2\,(h + \lambda)^3/h^2.$$

Отсюда находим

$$\lambda = \sqrt[3]{\frac{3\,(v + v_K)\,h^2}{\pi r^2}} - h.$$

Блок *По конусу*, вычисляющий по этой формуле λ, должен содержать обращение к блоку *Извлечение кубического корня*, начало которого мы обозначим через $\sqrt[3]{\ }$ *).

По конусу	$\Omega = конец$	$t_1 \cdot t_2 = t_3$
	$v + v_K = t_1$	$t_3 \cdot t_2 = \alpha$
	«3» $\cdot\, t_1 = t_1$	$\sqrt[3]{\ }$
	$t_1 : \pi = t_1$	$\gamma - h = \lambda$
	$h : r = t_2$	*конец*

Таким образом, программа нахождения уровня жидкости в резервуаре состоит из собирающей (*Резервуар*) и пяти блоков (*Усеченный конус*, *Высота*, *Конус*, *По цилиндру*,

*) Программа извлечения кубического корня составлялась нами в § 14 и использовалась в § 27.

По конусу), из которых первый фактически также является собирающей, а последний содержит обращение к блоку $\sqrt[3]{\ }$.

В предыдущих главах при составлении программ мы под промежуточные результаты счета занимали рабочие ячейки R_0, R_1, R_2, ... причем в одни и те же ячейки помещали совершенно различные величины. При блочном программировании во всех рассмотренных примерах мы для каждого блока выделяли свои рабочие ячейки, а для промежуточных величин также брали свои ячейки, обозначая их наиболее естественным образом. Такой способ выбора рабочих ячеек делает программу более наглядной и упрощает ее проверку. Правда, он требует дополнительных рабочих ячеек, но к нему следует прибегать всегда, когда программа свободно размещается в памяти.

§ 36. Стандартные подпрограммы с входными и выходными ячейками

В процессе решения каждой конкретной задачи математик выделяет часто встречающиеся ему функции в «свои» подпрограммы. Однако есть функции, которые встречаются при решении самых разнообразных задач. Так, например, во многих задачах приходится иметь дело с тригонометрическими функциями. Естественно для таких функций раз и навсегда написать *стандартные подпрограммы*, которыми сможет воспользоваться любой программист. Правила обращения к стандартной подпрограмме должны быть естественными и простыми, а следовательно, легко запоминающимися.

Все стандартные подпрограммы по способу обращения к ним делятся на две группы: *подпрограммы с входными и выходными ячейками* и *подпрограммы с информацией*.

Подпрограммы первой группы служат для вычисления функций одного или двух независимых переменных, например тригонометрических, показательной, показательно-степенной и др. Покажем на примере как составляются такие подпрограммы.

Пример 1.36. Составить стандартную подпрограмму вычисления кубического корня $y=\sqrt[3]{x}$.

Программа должна удовлетворять следующему требованию. Если в некоторую *стандартную входную ячейку* α

заслать аргумент

$$x=\alpha$$

и обратиться к подпрограмме

$$БВ \quad Я+1 \quad \sqrt[3]{\ } \quad Ω,$$

то после того как работа этой программы закончится и управление будет передано ячейке $Я+1$, *в стандартной выходной ячейке* γ окажется значение кубического корня из α. Это значение должно быть вычислено с машинной точностью. Таким образом, для вычисления $y=\sqrt[3]{x}$ нужно написать три команды:

$$\begin{array}{llll} & & & x=\alpha \\ БВ & Я+1 & \sqrt[3]{\ } & Ω \\ & & & \gamma=y \end{array}$$

Здесь $\sqrt[3]{\ }$ — метка начальной ячейки подпрограммы. Ранее нами была составлена программа вычисления кубического корня с заданной точностью ε (см. пример 1.14). Чтобы превратить эту программу в стандартную, поставим на место команды *стоп* команду

$$Б \quad Ω,$$

заменим обозначения x на α, y на γ, y_n на R_0, и организуем проверку окончания счета на машинную точность. Тогда получится следующая программа *):

$$\begin{array}{lll|l} \sqrt[3]{\ } & & & \\ & \alpha \cdot & «1/3» & =R_0 \\ & R_0 \cdot & R_0 & =R_1 \\ & \alpha : & R_1 & =R_1 \\ & R_1 \cdot & «1/3» & =R_1 \\ & R_0 \cdot & «2/3» & =R_2 \\ & R_1 + & R_2 & =\gamma \\ & \gamma \not\sim & R_0 & =0 \\ \hline УО & \gamma & | & R_0 \\ Б & & Ω & \end{array}$$

*) Для экономии памяти мы отступаем в стандартных подпрограммах от обязательной пересылки Ω в конец блока, а заканчиваем блок передачей управления ячейке Ω, что в данном случае не нарушает работы программы.

По тому же образцу, что и для подпрограммы $\sqrt[3]{\ }$, строятся обращения к другим подпрограммам, вычисляющим функции одного независимого переменного. Так, для вычисления $y = e^x$ следует обратиться к подпрограмме вычисления показательной функции следующим образом:

$$\begin{array}{c} x=\alpha \\ БВ \quad Я+1 \quad \exp \quad \Omega \\ \gamma=y \end{array}$$

Аналогично находится $y = \ln x$

$$\begin{array}{c} x=\alpha \\ БВ \quad Я+1 \quad \ln \quad \Omega \\ \gamma=y \end{array}$$

При использовании программ для нахождения кубического корня, показательной и логарифмической функций обычно не засылают аргумент в α, а при вычислении этого аргумента (если его нужно вычислять и не требуется запомнить) прямо образуют его в ячейке α, результат же не пересылают из γ, а используют в последующих вычислениях.

Пример 2.36. Вычислить $y = 5e^{2\sqrt[3]{x-1}}$. Здесь y легко находится по программе

$$\begin{array}{llll} x & -\text{«1»}=\alpha & БВ \quad Я+1 \quad \exp & \Omega \\ БВ \quad Я+1 & \sqrt[3]{\ } & \Omega & \text{«5»} \cdot \gamma = y \\ \text{«2»} \cdot \gamma = \alpha & стоп & & \end{array}$$

Мы рассмотрели обращения к трем стандартным подпрограммам: $\sqrt[3]{\ }$, exp, ln. Все эти подпрограммы, с точки зрения их использования в рабочих программах, построены одинаково. Каждая из них вычисляет функцию одного независимого переменного

$$\gamma = f(\alpha),$$

причем аргумент берется из стандартной входной ячейки α, а результат помещается в стандартную выходную ячейку γ. Обращение к подпрограмме имеет вид

$$БВ \quad Я+1 \quad f \quad \Omega$$

где Ω — стандартная ячейка возврата, а f — метка начальной команды подпрограммы.

Заметим, что, обращаясь к подпрограмме, можно возвратиться и не на следующую ($Я + 1$), а на любую команду S:

$$БВ \quad S \quad f \quad Ω.$$

Помимо указанных функций, в практике вычислений часто встречаются еще тригонометрические и обратные тригонометрические функции. Тригонометрические функции $\cos x$ и $\sin x$ обычно в формулах встречаются парами. Далее мы увидим *), что и алгоритмы вычисления этих функций таковы, что удобно получать сразу пару функций $\cos\alpha$ и $\sin\alpha$ одного независимого переменного α. Поэтому для тригонометрических функций обычно используется одна подпрограмма вычисления $\cos\alpha$ и $\sin\alpha$, которая по аргументу, находящемуся в стандартной входной ячейке α, вычисляет в двух стандартных выходных ячейках γ_0 и γ_1 косинус и синус α, так что

$$\gamma_0 = \cos\alpha,$$
$$\gamma_1 = \sin\alpha$$

(в ячейке с четным индексом 0 получается четная функция $\cos\alpha$, в ячейке с нечетным индексом 1 — нечетная функция $\sin\alpha$).

Обращение к этой стандартной подпрограмме имеет вид

$$БВ \quad Я+1 \quad \cos,\sin \quad Ω,$$

где cos,sin — метка начальной ячейки подпрограммы. Точно так же пишутся обращения к стандартным подпрограммам, вычисляющим пары обратных тригонометрических функций:

$$БВ \quad Я+1 \quad \text{arccos,arcsin} \quad Ω$$

и

$$БВ \quad Я+1 \quad \text{arcctg,arctg} \quad Ω,$$

причем в результате работы первой из них получаем

$$\gamma_0 = \arccos\alpha,$$
$$\gamma_1 = \arcsin\alpha,$$

*) См. гл. I второго выпуска.

а после работы второй

$$\gamma_0 = \operatorname{arcctg}\alpha,$$
$$\gamma_1 = \operatorname{arctg}\alpha.$$

Таким образом, все эти подпрограммы вычисляют пары функций одного аргумента

$$\gamma_0 = f_0(\alpha),$$
$$\gamma_1 = f_1(\alpha).$$

Обращение же к ним пишется в виде

БВ *Я*+1 f_0, f_1 Ω.

Составление стандартных программ для вычисления элементарных функций будет рассмотрено в гл. I второго выпуска.

Пример 3.36. Вычислить величину

$$y = \sin x - 4\cos x + 3\operatorname{tg} 5x.$$

По определению, $\operatorname{tg} x = \sin x/\cos x$. Программа вычисления имеет вид

		x	$=\alpha$	*БВ*	*Я*+1		cos,sin		Ω
БВ	*Я*+1	cos,sin	Ω			γ_1	:	γ_0	$=R$
	«4»	· γ_0	$=R$			«3»	· R		$=R$
	γ_1	— R	$=y$			y	+ R		$=y$
	«5»	· x	$=\alpha$	*стоп*					

В некоторых задачах приходится вычислять функции двух переменных, например, показательно-степенную функцию

$$z = x^y,$$

где x и y — действительные числа, причем $x > 0$.

Обращение к стандартной подпрограмме, вычисляющей эту функцию двух независимых переменных, строится следующим образом:

$x = \alpha_0$
$y = \alpha_1$
БВ *Я*+1 ↑ Ω
$\gamma = z$

Здесь ↑ — метка начальной команды подпрограммы, вычисляющей в стандартной ячейке *) γ по величинам, заданным в стандартных ячейках α_0, α_1, величину

$$\gamma = \alpha_0^{\alpha_1}.$$

§ 37. Стандартные подпрограммы с информацией. Формирование команд

Программы с входными и выходными ячейками, рассмотренные в предыдущем параграфе, вполне удобны для использования, однако не всегда удается ими ограничиться. Чтобы в этом убедиться, рассмотрим сначала следующий пример.

Пример 1.37. Программа должна работать с массивом исходных числовых данных, введенных в память в двоично-десятичном виде в ячейки от a_1 до a_n подряд. Для работы программы необходимо перевести эти числа в двоичную систему и разместить их в ячейках памяти подряд, начиная с ячейки b_1. Мы будем предполагать, что уже составлена стандартная подпрограмма «10→2», переводящая одно двоично-десятичное число из ячейки α в двоичную систему и помещающая результат перевода в ячейку γ.

Напишем теперь блок, который обеспечит перевод всех чисел заданного массива в двоичную систему и размещение их в нужных местах: Это будет обычный цикл с переадресацией, содержащий обращение к программе «10→2»

	PA	0	0	0
T_1			a_1^*	$= \alpha$
	$БВ$	$Я+1$	«10→2»	Ω
T_2			γ	$= b_1^*$
T_3	PA	$< \overline{n-1}$		$\overline{1}^*$

Таким образом, нам пришлось написать блок, занимающий 5 команд. Ясно, что этот блок является необходимым

*) Ячейки γ и γ_0, так же как α и α_0, мы будем считать совпадающими.

для любой задачи, работающей с массивом входных данных. Но можно ли сделать так, чтобы *этот блок был стандартным* и чтобы каждому программисту не пришлось писать его для себя заново?

В таком виде, как этот блок написан, его можно считать стандартным только в том случае, если ячейки a_1, b_1 и число $n-1$ известны заранее и зафиксированы, т. е. тоже являются стандартными. Но тогда программисту все равно придется пересылать числа из своих ячеек в ячейки $a_1, \ldots, a_n$, а затем из ячеек $b_1, \ldots, b_n$ в нужные для своей программы ячейки, так что *никакой экономии не получается.*

Гораздо удобнее иметь стандартную программу, которая будет брать исходные числа из тех ячеек, где они лежат, и помещать их сразу на нужное место. В такой программе команды T_1 и T_2 не могут быть написаны заранее. *Их необходимо сформировать по заданной информации.*

Информация к этой программе составляет три не известных нам заранее адреса: адреса a_1 и a_n начальной и конечной ячеек массива десятичных чисел и адрес b_1 первой ячейки для размещения двоичных чисел. Предположим, что эти адреса записаны в некоторой ячейке памяти

$$i\colon\ (0;\ a_1,\ a_n,\ b_1).$$

Пример 2.37. Пусть задана ячейка i, содержащая указанную выше информацию. Составим стандартную программу, переводящую десятичные числа из ячеек $a_1, \ldots, a_n$ в двоичную систему и размещающую двоичные числа в ячейки, начиная с b_1.

Для этой цели можно воспользоваться составленной в предыдущем примере программой. Нужно только сформировать команды T_1, T_2 и T_3. Надобность в восстановлении этих переменных команд в данном случае отпадает, поскольку они в начале работы программы все равно формируются. Кроме того, нужно еще позаботиться о восстановлении PA.

Команду T_3 можно сформировать при помощи константы

$$T_3^{\circ}\quad (PA<0\ T_1\ \bar{1}^*)$$

следующим образом:

$$
\begin{array}{llll}
1) & i & \wedge & (F,\ 0,\ 0) = R_1 \\
2) & \overline{114} & \rightarrow, & i = R_2 \\
3) & R_2 & \wedge & (F,\ 0,\ 0) = R_2 \\
4) & R_2 & -, & R_1 = R_3 \\
5) & T_3^0 & +, & R_3 = T_3
\end{array}
$$

Действительно, в результате работы команд 1), 2), 3) в ячейках R_1 и R_2 оказываются константы $(a_1,\ 0,\ 0)$, $(a_n,\ 0,\ 0)$ и так как $a_n - a_1 = n - 1$, то в результате выполнения команд 4), 5) в ячейке T_3 сформируется нужная команда. Команды T_1 и T_2 формируются при помощи констант

$$
\begin{array}{l}
T_1^0\ (0^* = \alpha) \\
T_2^0\ (\gamma = 0^*)
\end{array}
$$

следующим образом:

$$
\begin{array}{llll}
6) & \overline{64} & \rightarrow, & R_1 = R_1 \\
7) & i & \wedge & (0,\ 0,\ F) = R_2 \\
8) & R_1 & \vee & T_1^0 = T_1 \\
9) & R_2 & \vee & T_2^0 = T_2
\end{array}
$$

В самом деле, после сдвига командой 6) в среднем адресе ячейки R_1 будет a_1, а пересечение по команде 7) оставляет лишь в правом адресе ячейки R_2 адрес b_1, и их следует только прибавить к нужным командам-заготовкам.

Команды 1) — 9) и константы T_1^0, T_2^0, T_3^0 образуют формирующую часть стандартной подпрограммы. Объединяя формирующую часть и счетный цикл, написанный в предыдущем примере, получим программу группового перевода «*Перевод*».

При составлении этой программы мы считали, что информация к ней находится в некоторой известной ячейке i, которую можно считать стандартной. Тогда при обращении к программе необходимо предварительно заслать в ячейку i соответствующую информацию. Более удобно помещать информацию рядом с командой обращения к стандартной программе, так, чтобы стандартная программа сама переносила ее в нужное место.

Обращение к такой стандартной программе принято писать так:

$$\begin{array}{ll} Я & : БВ \quad Я+2 \quad \text{Перевод} \quad \Omega \\ Я+1 & : 0 \qquad a_1 \qquad\quad a_n \qquad b_1 \\ Я+2 & : \text{что делать дальше} \end{array}$$

При этом в ячейку Ω засылается команда $БВ\ Я+2$, т. е. возврат происходит в ячейку, содержащую следующую команду программы. Ячейка же $Я+1$ содержит информацию для стандартной программы, к которой мы обращаемся. Ее называют строкой информации.

Пример 3.37. Составить стандартную программу «Перевод», в результате обращения к которой

$$\begin{array}{llll} БВ & Я+2 & \text{Перевод} & \Omega \\ 0 & a_1 \qquad & a_n \qquad & b_1 \end{array}$$

двоично-десятичные числа из массива (a_1, a_n) переводятся «$10 \to 2$» и помещаются в массив (b_1, b_n).

Для составления нужной программы достаточно произвести перенос информации из ячейки $Я+1$ в ячейку i и присоединить к команде переноса программу, составленную в предыдущем примере. Команда переноса может быть сформирована следующим образом:

$$U \left| \begin{array}{l} U^0+, \quad \Omega = U \\ (\qquad Я+1=i) \end{array} \right.$$

где U^0 — константа, имеющая вид

$$U^0 | F \vee F = i.$$

Таким образом, стандартная программа состоит из двух команд переноса строки информации, команд формирования 1) — 9), пяти команд счетной части и команд-констант U^0, T_1^0, T_2^0, T_3^0. Соединив все эти составляющие вместе и позаботившись о сохранении Ω и $РА$ *), мы получим

*) Ячейка для восстановления регистра в блоке обычно помещается непосредственно перед его концом; дадим этой ячейке название *предконец*.

стандартную программу группового перевода «10 → 2» в следующем виде:

	1)		U^0	+ ,	Ω	=	U		
	2)			(	$Я+1$	=	i)		н. п.
	3)		i	$\wedge$	$(F,\ 0,\ 0)$	=	R_1		
	4)		$\overline{114}$	→ ,	i	=	R_2		
	5)		R_2	$\wedge$	$(F,\ 0,\ 0)$	=	R_2		
	6)		R_2	— ,	R_1	=	R_3		
	7)		T_3^0	+ ,	R_3	=	T_3		
	8)		$\overline{64}$	→ ,	R_1	=	R_1		
	9)		i	$\wedge$	$(0,\ 0,\ F)$	=	R_2		
	10)		R_1	$\vee$	T_1^0	=	T_1		
	11)		R_2	$\vee$	T_2^0	=	T_2		
	12)				Ω	=	*конец*		
	13)	PA	0*		0		*предконец*		
T_1	14)				$(a_1^*$	=	$\alpha)$		н. п.
	15)	$БВ$	$Я+1$		$10 \to 2$		Ω		
T_2	16)				$(\gamma$	=	$b_1^*)$		н. п.
T_3	17)	$(PA$	$< \overline{n-1}$		14)		$\overline{1}^*)$		н. п.
	18)		*предконец*						н. п.
	19)		*конец*						н. п.
U^0	20)		F	$\vee$	F	=	i		
T_1^0	21)				0*	=	α		
T_2^0	22)				γ	=	0*		
T_3^0	23)	PA	<0		14)		$\overline{1}^*$		

Так как подпрограмма «Перевод» должна применяться во всех вычислительных задачах, желательно, чтобы она занимала меньше места в памяти. Эту программу можно значительно сократить, если при формировании и образовании цикла более искусно использовать регистр адреса. Сокращенная программа в содержательных обозначениях и в закодированном виде приведена в табл. 1.37. При кодировке мы поместим программу «Перевод», начиная с ячейки 1000, а рабочим ячейкам I_0, I_1, I_2, I_3, N поставим

в соответствие адреса 0010, 0011, 0012, 0013, 0014. Адрес начала программы «10 → 2» принят равным 7140.

Таблица 1.37

					1000	A	
		Ω = *конец*	1000	0 75	0000	0007	1020
$[PA]$	0*	Ω *предконец*	1	4 72	0000	0007	1017
		$\overline{7777}^* = I_0$	2	2 75	0000	7777	0010
	$\overline{114}\rightarrow$,	$I_0 = I_3$	3	0 14	0114	0010	0013
	$\overline{64}\rightarrow$	$I_0 = I_1$	4	0 54	0064	0010	0011
		$I_1 \wedge (0, F, 0) = I_1$	5	0 55	0011	7732	0011
		$I_0 \wedge (0, F, 0) = I_2$	6	0 55	0010	7732	0012
		I_2 —, $I_1 = N$	7	0 33	0012	0011	0014
$[PA]$	0	I_3 0	1010	0 72	0000	0013	0000

					1011	A	
$[PA]$	$\overline{1}^*$	I_1 I_3	1011	4 72	0001	0011	0013
		$0^* = \alpha$	2	2 75	0000	0000	0140
БВ	$Я + 1$	$10 \rightarrow 2$ Ω	3	0 16	1014	7140	0007
$[PA]$	$\overline{1}^*$	I_3 I_1	4	4 72	0001	0013	0011
		N —, (0, 1, 0) = N	5	0 33	0014	7722	0014
УО	γ	$\overline{7777}^*$	6	1 76	0160	1011	7777
предконец		(н. п.)	7	0 77	0000	0000	0000
конец		(н. п.)	1020	0 77	0000	0000	0000

Составленная нами стандартная программа состоит из следующих основных частей:

1) *перенос информации*, находящейся в ячейке $Я + 1$, в стандартную ячейку i;

2) *формирование* команд рабочей части программы, зависящих от адресов ячейки i;

3) *рабочая часть* программы (в нашем примере в рабочую часть входит еще стандартный блок перевода одного числа);

4) *константы формирования.*

Наличие этих четырех частей характерно для любой стандартной программы с информацией. При этом первая часть (перенос информации) организуется во всех программах совершенно одинаково.

Команда обращения к подпрограмме играет для программ с информацией двоякую роль. Во-первых, как и при обращении к любому блоку, она обеспечивает возврат в основную программу. Во-вторых, записывая в ячейке Ω адрес места, куда следует возвратиться, она тем самым сообщает стандартной программе местонахождение строки информации.

Такие стандартные программы называют *программами с принудительным концом*, в отличие от *программ со свободным концом*, которыми являются программы без информации; последние допускают обращение

$$БВ \quad S \quad СП \quad \Omega,$$

где *СП* — начало стандартной программы, а S — произвольная ячейка. К программе с информацией такое обращение возможно лишь тогда, когда информация находится в ячейке $S - 1$.

Некоторые стандартные программы требуют не одной строки информации, а двух или трех. В таком случае в ячейку Ω засылается возврат к $Я + 3$ или $Я + 4$.

Впрочем, для удобства кодирования обычно при обращении к стандартной программе в Ω засылают возврат на $Я + 1$, а в самой стандартной программе видоизменяют ячейку Ω таким образом, чтобы возвратиться к команде основной программы, следующей за строкой информации. Точная форма обращения к каждой библиотечной подпрограмме указывается в соответствующем описании.

Можно привести большое число различных примеров стандартных программ с информацией. Одной из наиболее важных таких программ является программа печати чисел. Как мы знаем, результаты вычислений получаются в машине в двоичной системе счисления. Программа печати обеспечивает вывод полученных результатов. Она переводит окончательные данные в десятичную систему, обращаясь к программе «2 → 10», и печатает их. Обычное обращение к этой программе имеет вид

$$\begin{array}{lllll} Я: & БВ\ Я+1 & Печать\ чисел & \Omega \\ Я+1: & a_1 & a_n & q \end{array}$$

где a_1, a_n — адрес первой и последней ячейки печатаемого массива. Число q, стоящее в правом адресе ячейки информации, показывает, что полученные числа нужно печатать

группами по q штук в каждой. Это делается для удобства ориентировки в табулограмме.

Мы познакомились, таким образом, с *формированием команд* машиной. Так принято называть *одно или несколько действий, целью которых является получение команды.* Для формирования команд применяются, как мы видели, фиксированные действия, сдвиги и логические операции. Частным случаем формирования команд является *переадресация*, рассматривавшаяся нами в гл. V.

§ 38. Библиотека стандартных подпрограмм

Совокупность всех стандартных подпрограмм образует библиотеку стандартных подпрограмм. Библиотечные подпрограммы располагают обычно в конце памяти.

В библиотеку, кроме подпрограмм, включают еще часто встречающиеся константы: плавающие числа от «1» до «10», «1/2», «0,1», $\pi/2$, π, 2π, константы переадресации, константы выделения.

Расположение некоторых констант в памяти указано на памятке (табл. 1.38). Кроме того, на памятке помечают буквами R_1, R_2, ..., R_{17}, Ω, $\Omega - 1$, α_0, α_1, γ_0, γ_1 — рабочие ячейки библиотеки. Библиотеку снабжают *каталогом*, т. е. перечнем всех стандартных подпрограмм с краткими пояснениями к ним.

По форме обращений подпрограммы делятся на два типа:
а) подпрограммы с входными α_0, α_1 и выходными γ_0, γ_1 ячейками — обращение

$$БВ \quad Я+1 \quad f \quad \Omega,$$

где f — обозначение вычисляемой функции,

б) подпрограммы с информацией — обращение

$$\begin{matrix} БВ & Я+1 & U & \Omega, \\ k & a & b & c \end{matrix}$$

где U — название (обозначение) данной подпрограммы, а (k; a, b, c) — строка информации к ней.

В таблице 2.38 приведен каталог библиотеки, содержащей наиболее употребительные стандартные подпрограммы.

Таблица 1.38

7640		7700		7740	*Щ*
7641		7701	«1»	7741	(100; 0, 0, 0)
7642		7702	«2»	7742	(200; 0, 0, 0)
7643		7703	«3»	7743	(300; 0, 0, 0)
7644		7704	«4»	7744	(400; 0, 0, 0)
7645		7705	«5»	7745	(500; 0, 0, 0)
7646		7706	«6»	7746	(600; 0, 0, 0)
7647		7707	«7»	7747	(700; 0, 0, 0)
7650		7710	«8»	7750	
7651		7711	«9»	7751	
7652		7712	«10»	7752	
7653		7713	$\pi/2$	7753	
7654		7714	π	7754	(114; 0, 0, 0)
7655		7715	2π	7755	(130; 0, 0, 0)
7656		7716	$\pi/180$	7756	(144; 0, 0, 0)
7657		7717		7757	(145; 0, 0, 0)
7660		7720	(1; 0, 0, 0)	7760	
7661		7721	(0, 0, 1)	7761	
7662		7722	(0, 1, 0)	7762	
7663		7723	(0, 1, 1)	7763	
7664		7724	(1, 0, 0)	7764	
7665		7725	(1, 0, 1)	7765	
7666		7726	(1, 1, 0)	7766	
7667		7727	(1, 1, 1)	7767	
7670	«1/2»	7730	(77; 0, 0, 0)	7770	
7671	«10^{-1}»	7731	(0, 0, *F*)	7771	
7672	«10^{-2}»	7732	(0, *F*, 0)	7772	
7673	«10^{-3}»	7733	(0, *F*, *F*)	7773	
7674	«10^{-4}»	7734	(*F*, 0, 0)	7774	
7675	«10^{-5}»	7735	(*F*, 0, *F*)	7775	
7676	«10^{-6}»	7736	(*F*, *F*, 0)	7776	
7677	«10^{-7}»	7737	(*F*, *F*, *F*)	7777	

Таблица 2.38

Каталог библиотеки стандартных подпрограмм

	f		Результат
Подпрограммы с входными и выходными ячейками	exp		$\gamma_0 = \exp \alpha_0 \cdot (= e^{\alpha_0})$
	ln		$\gamma_0 = \ln \alpha_0$
	$\sqrt[3]{}$		$\gamma_0 = \sqrt[3]{\alpha_0}$
	cos, sin		$\gamma_0 = \cos \alpha_0, \quad \gamma_1 = \sin \alpha_0$
	arccos, arcsin		$\gamma_0 = \arccos \alpha_0, \ \gamma_1 = \arcsin \alpha_0,$
	arcctg, arctg		$\gamma_0 = \operatorname{arcctg} \alpha_0, \ \gamma_1 = \operatorname{arctg} \alpha_0$
	ch, sh		$\gamma_0 = \operatorname{ch} \alpha_0, \quad \gamma_1 = \operatorname{sh} \alpha_0$
	$\uparrow$		$\gamma_0 = \alpha_0^{\alpha_1}$
Подпрограммы с информацией	Название подпрограммы U	Строка информации	Пояснение
	Перевод плав. чисел	0; a_1, a_n, b_1	Массив плавающих чисел от a_1 до a_n переводится из двоично-десятичной системы в двоичную и ставится, начиная с ячейки b_1
	Печать плав. чисел	0; a_1, a_n, q	Массив плавающих чисел от a_1 до a_n переводится из двоичной системы в двоично-десятичную и печатается группами по q штук

Продолжение табл. 2.38

	Название подпрограммы U	Строка информации	Пояснение
Подпрограммы с информацией	Перевод целых чисел	0; a_1, a_n, b_1	Массив целых чисел (знак в 44-м разряде, десят. цифры в мантиссе) от a_1 до a_n переводится 10 → 2 и ставится с ячейки b_1
	Печать целых чисел	0; a_1, a_n, q	Массив целых чисел от a_1 до a_n переводится 2→ 10 и печатается группами по q штук
	Печать команд	0; a_1, a_n, 0	Содержимое ячеек от a_1 до a_n печатается в виде команд
	Перенос	0; a_1, a_n, b_1	Массив чисел от a_1 до a_n переносится на другое место оперативной памяти, начиная с b_1

По выполняемым функциям все библиотечные подпрограммы можно разделить на две большие группы — обслуживающие (сервисные) подпрограммы и подпрограммы методов вычислений. Обслуживающие подпрограммы не выполняют собственно вычислительной работы, а служат для того, чтобы создать определенные удобства при общении человека с машиной. К этой группе подпрограмм принадлежат, в частности, рассмотренные нами подпрограммы печати и перевода, а также подпрограммы обмена с внешней памятью, отладки, ввода и вывода алфавитно-цифровой информации и др. *).

*) Некоторые из этих подпрограмм мы рассмотрим в дальнейшем.

Группа библиотечных подпрограмм методов вычислений содержит обычно некоторый минимальный набор программ, дающий возможность вычислять элементарные функции, производить действия над комплексными числами, решать системы линейных алгебраических уравнений, вычислять определенные интегралы и т. д. Ряд таких программ будет рассмотрен во втором выпуске.

Составление библиотечных подпрограмм является высококвалифицированной работой, так как к ним предъявляются очень жесткие требования. Они должны быть по возможности быстрыми и короткими. Эти два требования обычно противоречат друг другу; так, если алгоритм программы имеет циклический характер, то самой короткой будет программа с циклом, а самой быстрой — программа без цикла. Это обстоятельство заставляет составителей стандартных подпрограмм идти при удовлетворении указанных требований на разумный компромисс.

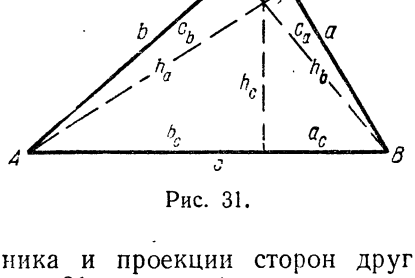


Рис. 31.

В заключение главы рассмотрим один пример, в котором используются несколько библиотечных подпрограмм.

Пример 1.38. В треугольнике ABC даны две стороны a, b и угол C между ними. Вычислить высоты треугольника и проекции сторон друг на друга. При помощи рис. 31 получим формулы, определяющие искомые отрезки:

$$\begin{array}{lll} h_a = b\sin C, & b_a = b\cos C, & c_a = a - b_a, \\ h_b = a\sin C, & a_b = a\cos C, & c_b = b - a_b, \\ c = \sqrt{c_b^2 + h_b^2}, & A = \arccos c_b/c, & \\ h_c = b\sin A, & b_c = b\cos A, & a_c = c - b_c. \end{array}$$

Мы будем предполагать, что величины a, b, C находятся в последовательных ячейках $a_{(10)}$, $b_{(10)}$, $C_{(10)}$ в двоично-десятичной форме, причем угол $C_{(10)}$ задан в градусах и долях градуса.

Эти три величины перед расчетом по формулам следует перевести в двоичную систему счисления, а C, кроме того, перевести в радианы. Результаты расчета поместим в памяти в следующем порядке:

$$h_a,\ h_b,\ h_c;\quad a_b,\ a_c;\quad b_a,\ b_c;\quad c_a,\ c_b,$$

переведем в десятичную систему и напечатаем.

Программа решения задачи имеет следующий вид:

БВ	*Я*+1		*Перевод пл.*	Ω
	$a_{(10)}$		$C_{(10)}$	a
	C	·	«π/180»	$=\alpha$
БВ	*Я*+1		cos,sin	Ω
	b	·	γ_1	$=h_a$
	b	·	γ_0	$=b_a$
	a	—	b_a	$=c_a$
	a	·	γ_1	$=h_b$
	a	·	γ_0	$=a_b$
	b	—	a_b	$=c_b$
	c_b	·	c_b	$=R_1$
	h_b	·	h_b	$=R_2$
	R_1	+	R_2	$=R_1$
	$\sqrt{R_1}$			$=c$
	c_b	:	c	$=\alpha$
БВ	*Я*+1		arccos,arcsin	Ω
			γ_0	$=\alpha$
БВ	*Я*+1		cos,sin	Ω
	b	·	γ_1	$=h_c$
	b	·	γ_0	$=b_c$
	c	—	b_c	$=a_c$
БВ	*Я*+1		*Печать пл.*	Ω
	h_a		h_c	$\overline{3}$
БВ	*Я*+1		*Печать пл.*	Ω
	a_b		c_b	$\overline{2}$
стоп				

§ 39. Работа с внешней памятью

Память машины ограничена и во многих задачах имеющегося количества ячеек не хватает для того, чтобы разместить всю необходимую информацию.

Так, в некоторых задачах приходится обрабатывать таблицы, содержащие много тысяч чисел, в процессе решения других задач появляется большое количество промежуточных результатов, третьи имеют такой громоздкий алгоритм решения, что в памяти не помещается программа.

Память машины, о которой мы говорили до сих пор, является оперативной, так как она непосредственно обменивается информацией с арифметическим устройством. Назовем оперативное запоминающее устройство *внутренней памятью*. У машины, наряду с внутренней памятью, существует еще и внешняя память. Внешняя память связана с арифметикой только через внутреннюю.

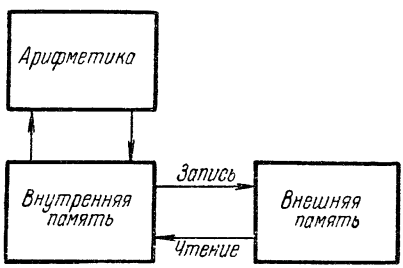


Рис. 32.

Перенос информации из внутренней памяти во внешнюю называется *записью*, а из внешней памяти во внутреннюю — *чтением*. На рис. 32 представлена схема взаимодействия запоминающих и арифметического устройств машины.

Внешняя память машины состоит из нескольких устройств, которые называют *устройствами обмена*. Они могут быть *односторонними* и *двусторонними*.

Односторонние устройства обычно предназначены для общения человека с машиной. Так, одним из носителей информации для машины являются уже известные нам перфокарты. В таком случае, *устройство ввода* с перфокарт является односторонним устройством обмена, переносящим информацию с перфокарт в оперативную память. Другим односторонним устройством обмена служит печатающее, которое переносит информацию из оперативной памяти на внешний носитель — бумажную ленту. Выходной перфоратор — еще одно одностороннее устройство обмена, вы-

водящее информацию из оперативной памяти на перфокарты.

Двусторонние устройства обмена применяются для хранения больших массивов информации при их использовании в процессе счета. Они состоят из *магнитных барабанов* и *магнитных лент* (магнитофонов).

На магнитном барабане информация записывается на поверхности вращающегося круглого цилиндра. За один полный оборот барабана можно записать из внутренней памяти (или считать в нее) информацию в любое место. Для магнитной ленты, устройство обмена с которой устроено как обычный магнитофон, доступ к любому ее месту осуществляется путем перематывания от данного места до требуемого.

У машины М-20 имеется четыре магнитных барабана и четыре магнитофона. Каждый барабан имеет номер (от 0 до 3). По емкости он равен оперативной памяти и разделен на сорокапятиразрядные ячейки с номерами от 0000 до 7777_8.

На магнитной ленте информация записывается *зонами*. Каждая зона имеет свой номер (от 1 до 1000_{10}) и может содержать различное количество, от 1 до 4095_{10}, сорокапятиразрядных машинных слов. В отличие от барабана, при работе с которым объем обмениваемой информации может быть любым, обмен с лентой может производиться только целыми зонами. Лента, естественно, своего номера не имеет и может быть помещена на любой магнитофон. Последние, как и барабаны, имеют номера от 0 до 3. Машины БЭСМ-4 и М-220 имеют иное число барабанов большей емкости (см. приложение).

В системе команд рассматриваемой машины предусмотрены элементарные операции, осуществляющие перенос информации из оперативной памяти в любые устройства внешней и обратно, т. е. запись и чтение. Эти операции объединены в две команды, которые помещаются обязательно одна за другой и записываются в виде

$$\begin{array}{llll} M(a) & \overline{УЧ} & b_1 & a_n \\ M(б) & a_1 & c & \Sigma \end{array}$$

Здесь a_1 и a_n означают адреса первой и последней ячеек массива оперативной памяти, с которым должна идти работа,

b_1 — адрес первой ячейки соответствующего массива во внешнем устройстве, $\overline{УЧ}$ — условное число, содержащее указание характера операции — печать, перфорация, запись (или чтение) на барабан или ленту, номер их и т. п. Об использовании второго и третьего адресов команды M ($б$) мы скажем несколько позже (см. стр. 245). Коды команд M ($а$) и M ($б$) суть соответственно 50 и 70.

Мы не станем сколько-нибудь подробно рассматривать здесь эти операции *), так как обычно программист с ними дела не имеет: обмен с внешней памятью производится при помощи специальных стандартных библиотечных программ, внутри которых и содержатся эти команды. Программисту остается лишь написать обращение к требуемой библиотечной программе по обычным правилам.

Так, для работы с барабанами имеются программы записи и чтения, обращение к которым имеет вид: программа записи на магнитный барабан

$$\begin{array}{cccc} БВ\ \ Я+1 & & ЗМБ & \Omega \\ k & a_1 & a_n & b_1 \end{array}$$

По этой программе содержимое ячеек оперативной памяти от a_1 до a_n переписывается на барабан с номером k в ячейки, начиная с b_1. Аналогичная программа чтения, с обращением

$$\begin{array}{cccc} БВ\ \ Я+1 & & ЧМБ & \Omega \\ k & a_1 & a_n & b_1 \end{array}$$

переносит с k-го барабана из ячеек, начинающихся с b_1, информацию в ячейки оперативной памяти от a_1 до a_n.

Для магнитной ленты программы записи и чтения имеют такое же обращение, с той только разницей, что вместо адреса начальной ячейки на барабане b_1 указывается номер зоны. Таким образом, программа записи на ленту с обращением

$$\begin{array}{cccc} БВ\ \ Я+1 & & ЗМЛ & \Omega \\ k & a_1 & a_n & Q \end{array}$$

*) См. приложение, § 68.

записывает содержимое ячеек оперативной памяти от a_1 до a_n в зону с номером Q магнитной ленты, находящейся на k-м магнитофоне. Аналогично работает и программа чтения, обращение к которой

БВ	*Я* + 1	*ЧМЛ*	Ω
k	a_1	a_n	Q

В библиотеке существуют и подпрограммы работы с односторонними устройствами внешней памяти (перфорация, печать). Обращения к подпрограммам печати мы уже рассмотрели ранее.

К подпрограмме выдачи материала на перфокарты можно обратиться следующим образом:

БВ	*Я* + 1	*Перф*	Ω
0	a_1	a_n	0

где (a_1, a_n) — границы во внутренней памяти перфорируемого массива.

Внешняя память машины работает, вообще говоря, менее надежно, чем внутренняя. Правильность выполнения операций обмена контролируется в машине следующим образом. Все слова обмениваемого массива в процессе выполнения соответствующей операции (записи или чтения) суммируются в арифметическом устройстве при помощи циклического сложения, описанного нами в § 28. При записи, вслед за последней ячейкой записанного массива b_n, на барабане или ленте записывается циклическая сумма массива от a_1 до a_n. Следовательно, во внешней памяти оказываются занятыми не n, а $n + 1$ ячеек — b_1, b_2, ..., b_n, b_{n+1}. Одновременно эта сумма записывается в ячейку оперативной памяти Σ, адрес которой указан в третьем адресе команды M (*б*) (см. стр. 243).

При чтении записанного таким образом массива сумма, накопленная в процессе чтения в арифметическом устройстве, поразрядно сравнивается с содержимым ячейки b_{n+1} внешней памяти. Если эти слова совпадают, то выполняется команда, следующая за командой с кодом M (*б*). Если же

они не совпадают хотя бы в одном разряде, машина останавливается. Если после этой остановки снова пустить машину, то управление будет передано ячейке памяти *c*, адрес которой указан в среднем адресе команды *M* (*б*) (см. стр. 243). Отметим, что этот *стоп* может свидетельствовать либо о неисправности машины (неправильное выполнение операций обмена), либо об ошибке в программе. Так, например, возможен случай, когда в операции *M* (*б*) неправильно указывается начало массива во внешней памяти. Тогда, естественно, контрольная сумма и содержимое ячейки, следующей за считываемым массивом, не совпадут, и машина остановится. В некоторых модификациях этих команд, определяемых значением условного числа (*УЧ*), при несовпадении сумм машина не остановится, а сразу передаст управление команде с адресом *c*.

При выводе материала на перфокарты (запись) вслед за обычными словами в конце колоды перфорируется контрольное слово (циклическая сумма отперфорированного массива).

Ввод информации с перфокарт во внутреннюю память обычно производится при помощи элементарной операции *ввод со стопом*, записываемой так:

$$ВС\ a\ b\ \Sigma$$

(код операции — 10). При выполнении этой операции обычные слова, набитые на перфокартах, находящихся на устройстве ввода, заносятся в ячейки памяти подряд, начиная с *a*. Если же перфокарта начинается с адресного слова, то адрес *a* в команде ввода игнорируется и ввод происходит в соответствии с адресным словом, набитым на перфокарте.

В процессе ввода обычные и адресные слова циклически суммируются, а полученная сумма заносится в ячейку Σ. Ввод прекращается в тот момент, когда в устройство ввода поступает контрольное слово. Это слово сравнивается с суммой, накопленной при вводе в арифметическом устройстве. При совпадении этих сумм управление передается следующей команде, при несовпадении машина останавливается. Если в последнем случае пустить машину дальше, то управление будет передано команде, находящейся в ячейке *b*.

Другая операция ввода — ввод без останова

$$B\ a\ b\ \Sigma$$

(код операции 30) отличается от предыдущей лишь тем, что при несовпадении сумм машина не останавливается, а сразу передает управление команде b.

Рассмотренные ранее особенности операций обмена используются в стандартных подпрограммах. При работе стандартных подпрограмм обмена (запись, чтение) во внешней памяти занимается не n, а $n + 1$ ячеек и чтение контролируется указанным образом.

Рассмотрим пример, в котором необходимо использование внешней памяти.

Пример 1.39. Рассмотрим задачу, связанную с прогнозом погоды. Основой прогноза служит синоптический архив, составляемый следующим образом. Синоптическая ситуация для какого-либо пункта на данный день характеризуется четырьмя числами: температурами T_0 и T_1 и атмосферными давлениями H_0 и H_1 у поверхности Земли и на некоторой определенной высоте.

В архиве накапливаются ежедневные синоптические ситуации за продолжительный срок. Часть архива, относящаяся к данному году, называется годичным архивом, к данному месяцу — месячным архивом.

Пусть задана синоптическая ситуация на сегодняшний день и требуется сделать синоптический прогноз, т. е. предсказать синоптические ситуации на ближайшие дни. Одним из этапов прогноза является отбор месячных архивов, содержащих ситуации, совпадающие с сегодняшней.

Отбор этот производится следующим образом. Из всего архива выбираются годичные архивы за последние 50 лет. Из каждого годичного архива берется месячный архив того же месяца, что и текущий. Все ситуации этого месяца сравниваются с сегодняшней и те ситуации, которые точно совпадают, выписываются вместе с их датами (год, день, месяц). Месячные же архивы, содержащие хотя бы одну совпадающую ситуацию, запоминаются для дальнейшей обработки. Таким образом, задачу синоптического прогноза можно разбить на три этапа:

1) составление синоптического архива,
2) отбор сходных ситуаций и месячных архивов,
3) статистическая обработка отобранной информации.

Займемся программированием второго из перечисленных этапов составления синоптического прогноза. Решение такого рода задач на машине целесообразно начинать с организации информации, т. е. в данном случае с построения синоптического архива. Условимся считать величины T_0, H_0, T_1, H_1, год, день и месяц данной синоптической ситуации целыми десятичными числами. Знак каждого такого числа определяется 44-м разрядом слова. Таким образом, сегодняшняя ситуация занимает 7 ячеек памяти.

Рассмотрим структуру годичного архива, составленного из ежедневных ситуаций. В начале архива ставится его год, занимающий одну

ячейку памяти. Для каждой ситуации в годичном архиве не нужно указывать ее дату, ибо она однозначно определяется местом этой ситуации в общем массиве информации годичного архива. Поэтому для ситуации в годичном архиве мы отведем по 4 ячейки памяти (T_0, H_0, T_1, H_1). Всего эти ситуации займут $4\times366 = 1464$ ячейки памяти *), а весь годичный архив — 1465 ячеек.

Для решения задачи синоптического прогноза нам потребуется 50 годичных архивов, т. е. память емкостью 1465×50 ячеек. Такой объем информации не помещается не только во внутренней памяти машины, но и на четырех магнитных барабанах. Поэтому синоптический архив приходится помещать на магнитную ленту, выделяя для годичных архивов последовательных лет последовательные зоны одной и той же магнитной ленты.

Отметим, что при решении задачи нужно отбирать и где-то хранить по одному месячному архиву из тех годичных архивов, в которых нашлись совпадающие ситуации. Для этого требуется не более

$$4\times31\times50 = 6200 \text{ ячеек памяти.}$$

Массивы такой величины тоже не помещаются во внутренней памяти. Поэтому мы будем собирать месячные архивы на магнитном барабане. Предположим для простоты, что нам потребуется не более 30 месячных архивов. Тогда для массива месячных архивов достаточно одного магнитного барабана.

Приступая к составлению программы, условимся, что исходная и текущая информация организованы следующим образом.

На отдельной перфокарте помещается сегодняшняя ситуация. На магнитной ленте помещаются годичные архивы — в зоне 1 — за первый год полувека, в зоне 2 — за второй год и т. д.

Во внутренней памяти, кроме поля для программы, выделим еще два поля — одно для вызываемого с ленты годичного архива (1465 ячеек p_1 — p_{1465}), другое — для отбираемых месячных архивов (124 ячейки m_1 — m_{124}).

Отобранные месячные архивы будем помещать на магнитный барабан.

Собирающую задачи запишем в следующем виде:

Собирающая

		Подготовка	
РА	0	0	0
		Выделение	
		Поиск	
	и ν	0	$=0$
У1		*Я*+2	
		Запись	
РА	$<\overline{61}$		$\overline{1}$*
стоп			

*) Для простоты мы считаем все годы високосными, образуя для 29 февраля невисокосных лет «невозможную» синоптическую ситуацию.

В блоке *Подготовка* вводится во внутреннюю память сегодняшняя ситуация и производится формирование некоторых переменных ячеек блоков *Выделение*, *Поиск* и *Запись*. Далее работает цикл на 50 повторений, обрабатывающий информацию за 50 лет. В блоке *Выделение* вызывается с ленты очередной годичный архив и пересылаются на специальное рабочее поле из 124 ячеек синоптические ситуации для нужного месяца.

Блок *Поиск* сравнивает ситуации за выделенный месяц с сегодняшней и выдает на печать даты совпадающих ситуаций. Далее, этот блок вырабатывает признак u, который равен нулю, если в данном месяце нет ситуаций, совпадающих с сегодняшней, и равен (0, 0, 1), если такие ситуации имеются. Сравнение признака u с нулем в *Собирающей* вырабатывает сигнал $\omega = 0$ при наличии совпадающих ситуаций и $\omega = 1$ — в противном случае. В первом из этих случаев работает блок *Запись*, переносящий на магнитный барабан соответствующий месячный архив, во втором случае обращение к блоку *Запись* обходится.

Приступим теперь к написанию отдельных блоков.

Блок *Подготовка*, который мы составим позднее, вводит в семь последовательных ячеек внутренней памяти исходную ситуацию:

g (год), d (день месяца), M (месяц), T_0, H_0, T_1, H_1. По вводимой ситуации в этом блоке формируется ряд данных, необходимых для работы остальных блоков.

Составим теперь блок *Выделение*.

Прежде всего в этом блоке вызывается во внутреннюю память очередной из нужных годичных архивов и подготовляется вызов следующего архива. Это делается при известном номере $Q1$ начальной зоны ленты первого магнитофона следующим образом:

Выделение				$\Omega = конец$
	БВ	*Я*+1	*ЧМЛ*	Ω
τ_1	01	p_1	p_{1465}	Q_1
		τ_1+,	$(0, 0, 1) = \tau_1$	

Затем выбираем из поля $p_1 - p_{1465}$ и переносим на поле $m_1 - m_{124}$ архив месяца с номером s:

	БВ	*Я*+1	Перенос	Ω
τ_2		p_s	p_s+4n	m_1
конец			(н. п.)	

Здесь адреса p_s, p_{s+4n} должны быть сформированы в блоке *Подготовка*.

В блоке *Поиск* должен работать цикл из n повторений (n — число дней в выбранном месяце), при работе которого все суточные ситуации сравниваются с сегодняшней. Это сравнение и печать дат совпадающих

ситуаций организуются следующим образом:

Поиск			Ω	$=$*конец*
			g	$=b_0$
			0	$=b_1$
			M	$=b_2$
			0	$=u$
	РА	0*	0	*предконец*
Нач		b_1	$+$, (0,0,1)	$=b_1$
		m_1^*	$\nsim$ T_0	$=0$
	У0		Прод	
		m_2^*	$\nsim$ H_0	$=0$
	У0		Прод	
		m_3^*	$\nsim$ T_1	$=0$
	У0		Прод	
		m_4^*	$\nsim$ H_1	$=0$
	У0		Прод	
			(0,0,1)	$=u$
	БВ	*Я*$+1$	Печ. чисел	Ω
		b_0	b_2	0
Прод		b_1	$\nsim$ (0,0,n)	$=0$
	(0)	*РА* $\geqslant 0$	Нач	$\overline{4}$*
предконец			(н.п.)	
конец			(н.п.)	

В блоке *Запись* месячный архив, выбранный на поле $m_1 - m_{124}$, должен быть перенесен на очередное место на магнитный барабан и должна быть произведена переадресация, необходимая для переноса следующего архива. Заметим, что длина выбранного архива будет равна $4n$. Присоединим к этому архиву номер его года

$$p_1 = m_0$$

и будем переносить на третий магнитный барабан массив от m_0 до $m_1 + 4n - 1$. Блок *Запись* запишем в виде

Запись			Ω	$=$*конец*
			p_1	$= m_0$
	БВ	*Я*$+1$	*З М Б*	Ω
τ_3	03	m_0	m_0+4n	0
		τ_3 $+$,	(0,0,$4n+2$)	$=\tau_3$
		сч $+$,	(0,1,0)	$=$*сч*
конец			(н.п.)	

Здесь *сч* есть счетчик количества месячных архивов, занесенных на барабан. Переадресация строки информации τ_3 производится на величину $4n + 2$, так как на барабане для месячного архива нужно отвести $4n + 1$ ячейку и, кроме того, одну ячейку для записи контрольной суммы переносимого на барабан массива чисел.

После составления блоков

Выделение, *Поиск* и *Запись*

стали ясными функции блока

Подготовка.

Этот блок должен:

а) ввести с перфокарт и перевести «10 → 2» исходную синоптическую ситуацию,

б) по номеру текущего года g сформировать номер начальной зоны Q_1 (строка информации τ_1 блока *Выделение*), содержащей годичный архив полувековой давности,

в) по номеру текущего месяца сформировать строки информации τ_2, τ_3 (блоки *Выделение* и *Запись*) и константы переадресации (0, 0, $4n + 2$) и сравнения (0, 0, n).

Пункты а), б) реализуются следующими командами:

Подготовка			Ω	$=$*конец*
	В	$g_{(10)}$	$Я+2$	0
	стоп	0	0	0
	БВ	$Я+1$	Перевод чисел	Ω
		$g_{(10)}$	$H_{1(10)}$	g
		g,	—, g_0	$=R_1$
		τ_1^0	+, R_1	$=\tau_1$

Здесь $g_{(10)}$, $d_{(10)}$, $M_{(10)}$, $T_{0(10)}$, $H_{0(10)}$, $T_{1(10)}$, $H_{1(10)}$ — семь последовательных ячеек, в которые в десятичном виде вводится с перфокарты сегодняшняя ситуация; в ячейки g, d, M, T_0, H_0, T_1, H_1 эта ситуация перенесется после перевода в двоичную систему; в ячейке g_0 записано целое десятичное число, соответствующее первому году рассматриваемого полувекового периода, переведенное в двоичную систему, а τ_1^0 — константа (01; p_1, p_{1465}, 0).

Для того чтобы иметь возможность по номеру любого месяца M найти число его дней n, и место p_s, соответствующее положению нужного месячного архива в годичном архиве, составим таблицу из 13 последовательных ячеек t_1, t_2, ..., t_{12}, t_{13}. В III адресе каждой из этих ячеек укажем номер первого дня соответствующего месяца от начала года, в III адрес t_{13} поставим $366_{10} = 556_8$.

Таблица 1.39

задача Прогноз Собир. блок Подготовка стр. 1

	Собирающая				0400	*A*	1π
	Подготовка	0400	0	16	0401	0412	0007
	РА 0* 0 *ВР*	1	4	52	0000	0000	0410
	Выделение	2	0	16	0403	0500	0007
	Поиск	3	0	16	0404	0450	0007
	и $\nsim 0 = 0$	4	0	15	0574	0000	0000
	У1 *Я* +2	5	0	36	0000	0407	0000
	Запись	6	0	16	0407	0510	0007
	РА < $\overline{61}$ $\overline{1}$*	7	1	12	0061	0402	0001
ВР	(н. п.)	0410	0	77	0000	0000	0000
	стоп	11	0	77	0000	0000	0000

	Подготовка				0412	*A*	2π
	Ω = *конец*	0412	0	75	0000	0007	0442
	В $g_{(10)}$ *Я* + 2 0	3	0	30	0540	0415	0000
	стоп 0 0 0	4	0	77	0000	0000	0000
	БВ Я + 1 Пер. чисел Ω	5	0	16	0416	7034	0007
	$g_{(10)}$ $H_{1(10)}$ *g*	6	0	00	0540	0546	0550
	g —, g_0 = R_1	7	0	33	0550	0536	0571
	τ_1^0 +, R_1 = τ_1	0420	0	13	0443	0571	0502
	0 = *сч*	1	0	75	0000	0000	0575
	$\overline{114}$ → *M* = R_1	2	0	54	0114	0552	0571
	R_1 —, (0, 1, 0) = R_1	3	0	33	0571	7722	0571

Таблица 2.39

задача Прогноз блок Подготовка стр. 2

					0424	A	3π
	$[PA]$ 0* R_1 *предконец*	0424	4	72	0000	0571	0441
	t_2^* —, $t_1^* = R_1$	5	6	33	0522	0521	0571
	R_1 +, $0 = (0, 0, n)$	6	0	13	0571	0000	0576
	102 →, $(0, 0, n) = R_0$	7	0	14	0102	0576	0570
	R_0 +, $(0, 0, 2) = (0, 0, 4n + 2)$	0430	0	13	0570	0446	0577
	$\overline{114}$ →, $R_0 = R_0$	1	0	14	0114	0570	0570
	τ_3^0 +, $R_0 = \tau_3$	2	0	13	0445	0570	0513
	t_1^* —, $(0, 0, 1) = R_2$	3	4	33	0521	7721	0572
	$\overline{114}$ →, $R_2 = R_2$	4	0	14	0114	0572	0572
	$\overline{114}$ →, $R_2 = R_1$	5	0	14	0114	0572	0571
	R_1 +, $R_2 = R_1$	6	0	13	0571	0572	0571

					0437	A	4π
	R_1 +, $R_0 = R_1$	0437	0	13	0571	0570	0571
	τ_2^0 +, $R_1 = \tau_2$	0440	0	13	0444	0571	0505
предконец	(н. п.)	1	0	77	0000	0000	0000
конец	(н. п.)	2	0	77	0000	0000	0000
τ_1^0	(1; p_1, p_{1465}, 0)	3	0	01	1400	2771	0000
τ_2^0	(0; p_1, p_1, m_1)	4	0	00	1400	1400	1010
τ_3^0	(3; m_0, m_0, 0)	5	0	03	1000	1000	0000
	(0, 0, 2)	6	0	00	0000	0000	0002

Таблица 3.39

задача Прогноз блок Поиск стр. 3

Поиск					0450	A	5п
	Ω = *конец*	0450	0	75	0000	0007	0475
	$g = b_0$	1	0	75	0000	0550	0560
	$0 = b_1$	2	0	75	0000	0000	0561
	$M = b_2$	3	0	75	0000	0552	0562
	$0 = u$	4	0	75	0000	0000	0574
Нач	*РА* 0* 0 *предконец*	5	4	52	0000	0000	0474
	b_1 +, (0, 0, 1) $= b_1$	6	0	13	0561	7721	0561
	m_1^* $\nsim T_0 = 0$	7	4	15	1001	0553	0000
	УО Прод	0460	0	76	0000	0471	0000
	m_2^* $\nsim H_0 = 0$	1	4	15	1002	0554	0000
	УО Прод	2	0	76	0000	0471	0000

					0463	A	6п
	$m_3^* \nsim$ $T_1 = 0$	0463	4	15	1003	0555	0000
	УО Прод	4	0	76	0000	0471	0000
	$m_1^* \nsim$ $H_1 = 0$	5	4	15	1004	0556	0000
	УО Прод	6	0	76	0000	0471	0000
	(0, 0, 1) $= u$	7	0	75	0000	7721	0574
	БВЯ + 1 Печ. чисел Ω	0470	0	16	0471	7026	0007
	b_0 b_2 0	1	0	00	0560	0562	0000
Прод	$b_1 \nsim$ (0, 0, n) $= 0$	2	0	15	0561	0576	0000
	(0) *РА* $\geqslant 0$ Нач+1 $\bar{4}$*	3	1	71	0000	0456	0004
предконец	(н. п.)	4	0	77	0000	0000	0000
конец	(н. п.)	5	0	77	0000	0000	0000

Таблица 4.39

задача Прогноз блок Выделение Запись стр. 4

Выделение					0500	A	7п
	Ω = *конец*	0500	0	75	0000	0007	0506
	БВ Я + 1 *ЧМЛ* Ω	1	0	16	0502	7012	0007
τ_1	(01; p_1, p_{1465}, Q_1) н. п.	2	0	77	0000	0000	0000
	τ_1 +, (0, 0, 1) = τ_1	3	0	13	0502	7721	0502
	БВ Я + 1 *Перенос* Ω	4	0	16	0505	7030	0007
τ_2	(p_S, p_{S+4n}, m_1) н. п.	5	0	77	0000	0000	0000
конец	(н. п.)	6	0	77	0000	0000	0000

Запись					0510	A	8п
	Ω = *конец*	0510	0	75	0000	0007	0516
	p_1 = m_0	1	0	75	0000	1400	1000
	БВ Я + 1 *ЗМБ* Ω	2	0	16	0513	7000	0007
τ_3	(03; m_0, m_0 + 4n, 0) н. п.	3	0	77	0000	0000	0000
	τ_3 +, (0, 0, 4n + 2) = τ_3	4	0	13	0513	0577	0513
	сч +, (0, 1, 0) = *сч*	5	0	13	0575	7722	0575
конец	(н. п.)	6	0	77	0000	0000	0000

Таблица 5.39

задача Прогноз блок Константы стр. 5

Константы месяцев					0521	A	9п
t_1	(0, 0, 0_{10}) 1 янв.	0521	0	00	0000	0000	0000
t_2	(0, 0, 31_{10}) 1 февр.	2	0	00	0000	0000	0037
t_3	(0, 0, 60_{10}) 1 марта	3	0	00	0000	0000	0074
t_4	(0, 0, 91_{10}) 1 апр.	4	0	00	0000	0000	0133
t_5	(0, 0, 121_{10}) 1 мая	5	0	00	0000	0000	0171
t_6	(0, 0, 152_{10}) 1 июня	6	0	00	0000	0000	0230
t_7	(0, 0, 182_{10}) 1 июля	7	0	00	0000	0000	0266
t_8	(0, 0, 213_{10}) 1 авг.	0530	0	00	0000	0000	0325
t_9	(0, 0, 244_{10}) 1 сент.	1	0	00	0000	0000	0364
t_{10}	(0, 0, 274_{10}) 1 окт.	2	0	00	0000	0000	0422
t_{11}	(0, 0, 305_{10}) 1 нояб.	3	0	00	0000	0000	0461

					0534	A	10п
t_{12}	(0, 0, 335_{10}) 1 дек.	0534	0	00	0000	0000	0517
t_{13}	(0, 0, 366_{10}) 1 янв.	5	0	00	0000	0000	0556
g_0	(0, 0, 1920_{10})	6	0	00	0000	0000	3600

Таблица 6.39

0400 Собир	0440	0500 Выдел	0540 $g_{(10)}$	0600	0640	0700	0740
0401	0441	0501	0541 $d_{(10)}$	0601	0641	0701	0741
0402	0442	0502 τ_1	0542 $M_{(10)}$	0602	0642	0702	0742
0403	0443 $\tau_1^?$	0503	0543 $T_{0\,(10)}$	0603	0643	0703	0743
0404	0444 $\tau_2^?$	0504	0544 $H_{0\,(10)}$	0604	0644	0704	0744
0405	0445 $\tau_3^?$	0505 τ_2	0545 $T_{1\,(10)}$	0605	0645	0705	0745
0406	0446 (0, 0, 2)	0506	0546 $H_{1\,(10)}$	0606	0646	0706	0746
0407	0447	0507	0547	0607	0647	0707	0747
0410 *ВР*	0450 Поиск	0510 Запись	0550 g	0610	0650	0710	0750
0411	0451	0511	0551 d	0611	0651	0711	0751
0412 Подг	0452	0512	0552 M	0612	0652	0712	0752
0413	0453	0513 τ_3	0553 T_0	0613	0653	0713	0753
0414	0454	0514	0554 H_0	0614	0654	0714	0754
0415	0455 Нач	0515	0555 T_1	0615	0655	0715	0755
0416	0456	0516	0556 H_1	0616	0656	0716	0756
0417	0457	0517	0557	0617	0657	0717	0757
0420	0460	0520	0560 b_0	0620	0660	0720	0760
0421	0461	0521 t_1	0561 b_1	0621	0661	0721	0761
0422	0462	0522 t_2	0562 b_2	0622	0662	0722	0762
0423	0463	0523 t_3	0563 b_3	0623	0663	0723	0763
0424	0464	0524 t_4	0564 b_4	0624	0664	0724	0764
0425	0465	0525 t_5	0565 b_5	0625	0665	0725	0765
0426	0466	0526 t_6	0566 b_6	0626	0666	0726	0766
0427	0467	0527 t_7	0567	0627	0667	0727	0767
0430	0470	0530 t_8	0570 R_0	0630	0670	0730	0770
0431	0471	0531 t_9	0571 R_1	0631	0671	0731	0771
0432	0472 Прод	0532 t_{10}	0572 R_2	0632	0672	0732	0772
0433	0473	0533 t_{11}	0573 R_3	0633	0673	0733	0773
0434	0474	0534 t_{12}	0574 u	0634	0674	0734	0774
0435	0475	0535 t_{13}	0575 *сч*	0635	0675	0735	0775
0436	0476	0536 1920_{10}	0576 (0, 0, n)	0636	0676	0736	0776
0437	0477	0537	0577 (0, 0, $4n+2$)	0637	0677	0737	0777

Годичный архив (1400—4270)

0000	0400	1000	1400	2000	2400	3000	3400	4000
0010	0410	1010	1410	2010	2410	3010	3410	4010
0020	0420	1020	1420	2020	2420	3020	3420	4020
0030	0430	1030	1430	2030	2430	3030	3430	4030
0040	0440	1040	1440	2040	2440	3040	3440	4040
0050	0450	1050	1450	2050	2450	3050	3450	4050
0060	0460	1060	1460	2060	2460	3060	3460	4060
0070	0470	1070	1470	2070	2470	3070	3470	4070
0100	0500	1100	1500	2100	2500	3100	3500	4100
0110	0510	1110	1510	2110	2510	3110	3510	4110
0120	0520	1120	1520	2120	2520	3120	3520	4120
0130	0530	1130	1530	2130	2530	3130	3530	4130
0140	0540	1140	1540	2140	2540	3140	3540	4140
0150	0550	1150	1550	2150	2550	3150	3550	4150
0160	0560	1160	1560	2160	2560	3160	3560	4160
0170	0570	1170	1570	2170	2570	3170	3570	4170
0200	0600	1200	1600	2200	2600	3200	3600	4200
0210	0610	1210	1610	2210	2610	3210	3610	4210
0220	0620	1220	1620	2220	2620	3220	3620	4220
0230	0630	1230	1630	2230	2630	3230	3630	4230
0240	0640	1240	1640	2240	2640	3240	3640	4240
0250	0650	1250	1650	2250	2650	3250	3650	4250
0260	0660	1260	1660	2260	2660	3260	3660	4260
0270	0670	1270	1670	2270	2670	3270	3670	4270
0300	0700	1300	1700	2300	2700	3300	3700	4300
0310	0710	1310	1710	2310	2710	3310	3710	4310
0320	0720	1320	1720	2320	2720	3320	3720	4320
0330	0730	1330	1730	2330	2730	3330	3730	4330
0340	0740	1340	1740	2340	2740	3340	3740	4340
0350	0750	1350	1750	2350	2750	3350	3750	4350
0360	0760	1360	1760	2360	2760	3360	3760	4360
0370	0770	1370	1770	2370	2770	3370	3770	4370

Программа

Месячный архив

Таблица 7.39

Библиотека стандартных подпрограмм

4400	5000	5400	6000	6400	7000	7400
4410	5010	5410	6010	6410	7010	7410
4420	5020	5420	6020	6420	7020	7420
4430	5030	5430	6030	6430	7030	7430
4440	5040	5440	6040	6440	7040	7440
4450	5050	5450	6050	6450	7050	7450
4460	5060	5460	6060	6460	7060	7460
4470	5070	5470	6070	6470	7070	7470
4500	5100	5500	6100	6500	7100	7500
4510	5110	5510	6110	6510	7110	7510
4520	5120	5520	6120	6520	7120	7520
4530	5130	5530	6130	6530	7130	7530
4540	5140	5540	6140	6540	7140	7540
4550	5150	5550	6150	6550	7150	7550
4560	5160	5560	6160	6560	7160	7560
4570	5170	5570	6170	6570	7170	7570
4600	5200	5600	6200	6600	7200	7600
4610	5210	5610	6210	6610	7210	7610
4620	5220	5620	6220	6620	7220	7620
4630	5230	5630	6230	6630	7230	7630
4640	5240	5640	6240	6640	7240	7640
4650	5250	5650	6250	6650	7250	7650
4660	5260	5660	6260	6660	7260	7660
4670	5270	5670	6270	6670	7270	7670
4700	5300	5700	6300	6700	7300	7700
4710	5310	5710	6310	6710	7310	7710
4720	5320	5720	6320	6720	7320	7720
4730	5330	5730	6330	6730	7330	7730
4740	5340	5740	6340	6740	7340	7740
4750	5350	5750	6350	6750	7350	7750
4760	5360	5760	6360	6760	7360	7760
4770	5370	5770	6370	6770	7370	7770

При помощи этой таблицы формирование нужных ячеек

$$(0,\ 0,\ n),\ (0,\ 0,\ 4n+2),\ \tau_2,\ \tau_3$$

осуществляется следующим образом:

		$\overline{114}$	$\longrightarrow$	M	$=$	R_1
	$[PA]$	0^*		R_1		*предконец*
		$t_1{}^*$	—,	$t_0{}^*$	$=$	$(0,0,n)$
		$\overline{102}$	$\longrightarrow$	$(0,0,n)$	$=$	R_0
		R_0	$+$,	$(0,0,2)$	$=$	$(0,0,4n+2)$
		$\overline{114}$	$\longrightarrow$,	R_0	$=$	R_0
		τ_3^0	$+$,	R_0	$=$	τ_3
		$t_0{}^*$	—,	$(0,0,1)$	$=$	R_2
		$\overline{114}$	$\longrightarrow$,	R_2	$=$	R_2
		$\overline{114}$	$\longrightarrow$,	R_2	$=$	R_1
		R_1	$+$,	R_2	$=$	R_1
		R_1	$+$,	R_0	$=$	R_1
		τ_2^0	$+$,	R_1	$=$	τ_2
предконец				(н.п.)		
конец				(н.п.)		
τ_1^0	(01;	p_1,		p_{1465},		0)
τ_2^0	(00;	p_1,		p_1,		m_1)
τ_3^0	(03;	m_0,		m_0,		0)
		(0,		0,		2)

Распределим память для этой программы и закодируем ее (см. табл. 1.39 — 7.39). Саму программу, ее рабочие ячейки и таблицу t_1 — t_{13} поместим на участке памяти (0400—0600). Согласно памятке в табл. 6.39 для поля годичного архива p_1 — p_{1465} выделим ячейки 1400—4270, а для месячного архива m_0 — m_{124} — ячейки 1000—1174. Ячейки 6000—7777 будем считать занятыми библиотекой стандартных подпрограмм.

Для наглядности и удобства отладки программ, работающих с большими массивами информации, иногда используют памятки на всю внутреннюю память.

В табл. 7.39 изображено расположение информации на такой памятке для рассматриваемой задачи.

§ 40. Ввод и вывод алфавитно-цифровой информации

В этом параграфе мы рассмотрим два способа ввода алфавитно-цифровой информации с перфокарт в машину и один способ ее вывода (печати на бумажную ленту).

Очень удобным носителем как числовой, так и алфавитно-цифровой информации для ввода ее в машину являются *десятичные карты.*

До сих пор мы использовали перфокарты с нанесенной на них отдельными пробивками двоичной (двоично-восьмеричной или двоично-десятичной) информацией; такие перфокарты назовем *двоичными.*

Двоичная информация наносится на перфокарты «широкой» стороной в 12 строк.

На перфокарты можно заносить информацию и узкой стороной, поколонно, в десятичном виде. Такие перфокарты называют *десятичными.* При этом способе использования перфокарты в ее 80 колонок может быть помещено до 80 десятичных цифр. Каждая цифра изображается пробивкой в колонке на месте, где напечатана соответствующая цифра. Знаки + и — можно изображать пробивками в 12-й и 11-й позициях (12-я позиция — самая верхняя, 11-я позиция — следующая, эти позиции не помечены цифрами).

На рис. 33 изображена перфокарта, в первых одиннадцати колонках которой пробито целое число

$$+9753102468.$$

Десятичные карты вводятся в машину узкой стороной при помощи специального устройства ввода. Материал с каждой карты поступает в 80 последовательных ячеек памяти так, что каждая «цифра» занимает в ячейке младшие 12 двоичных разрядов.

Каждое число, наносимое на десятичные карты, может содержать любое количество значащих цифр *) и может быть числом с плавающей или фиксированной запятой или целым числом. Поэтому на десятичные карты можно наносить информацию из обычных числовых таблиц, не переписывая содержимое таблиц на специальные бланки. В качестве примера рассмотрим таблицу 1.40, содержащую сведения о денежных затратах на научно-исследовательские работы. В этой таблице под числа в отдельных столбцах отводится соответственно 2, 3, 3, 9, 9, 9 и 6 колонок.

---

*) На двоичные карты, как правило, наносятся десятичные числа с плавающей запятой с 9 знаками в мантиссе.

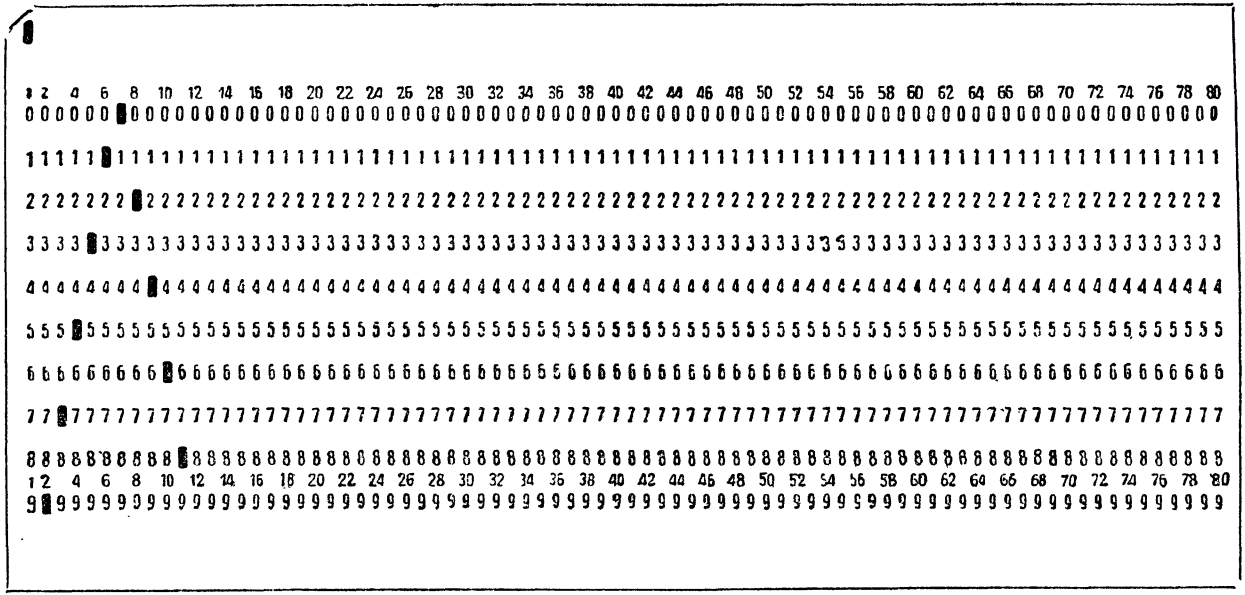


Рис. 33.

На рис. 34 изображена перфокарта, воспроизводящая первую строку таблицы. Такое размещение информации на перфокарте называется размещением по макету. Макет есть последовательность количеств колонок на перфокарте, отведенных под отдельные числа из таблицы. В соответствии с этим макетом информация наносится на 5 перфокарт, изображающих 5 строк данной таблицы. В данном примере

Таблица 1.40

Месяц	Отдел	Лаборат.	Номер темы	Затраты (в копейках)		
				материалы	зарплата	командировки
9	70	75	69 701 412	56 200	1 734 511	34 561
10	110	112	69 701 412	173 150	1 247 720	42 873
9	70	79	1 075 216	523 100	456 173	74 700
10	110	114	1 075 216	124 530	231 425	53 612
11	70	73	1 101 306	163 430	123 476	21 117

все числа неотрицательны, поэтому в макете под знаки чисел не отводится отдельных колонок.

Ввод числовых таблиц с десятичных карт и перевод чисел из десятичной системы в двоичную производится при помощи библиотечной подпрограммы «Ввод таблицы» со следующим обращением

$$\begin{array}{cccc} БВ & Я+1 & \text{Ввод таблицы} & \Omega \\ a_1 & & a_n & M \end{array}$$

где (a_1, a_n) — границы массива ячеек памяти, в который помещается двоичная таблица, а M — адрес первой из ячеек, содержащих макет отдельной строки таблицы. В ячейках M, $M+1$, ... по схеме, изображенной на рис. 35, в последовательных группах по 6 разрядов записывается макет $m_1, m_2, \ldots, m_k$ $(\Sigma m_i \leqslant 80)$; признаком конца макета является шестиразрядная группа

$$m_{k+1} = 0.$$

Нанесение алфавитно-цифровой информации на десятичные карты производится при помощи устройства перфорации, к которому подсоединена клавиатура, имеющая вид

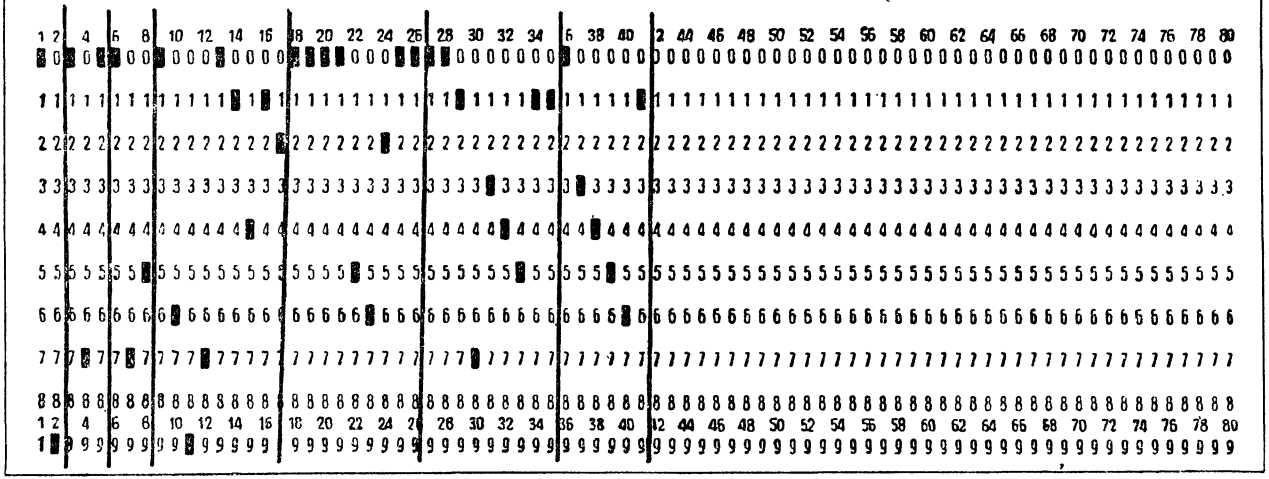


Рис. 34.

ячейка M

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
45 44 43	42 ... 38 37	36 ... 32 31	30 ... 26 25	24 ... 20 19	18 ... 14 13	12 ... 8 7	6 ... 2 1

номера разрядов ячейки

Рис. 35.

клавиатуры обычной пишущей машинки (см. рис. 36). При помощи этой клавиатуры в колонки перфокарты могут быть занесены эквиваленты цифр, русских букв и служебных знаков.

Рис. 36.

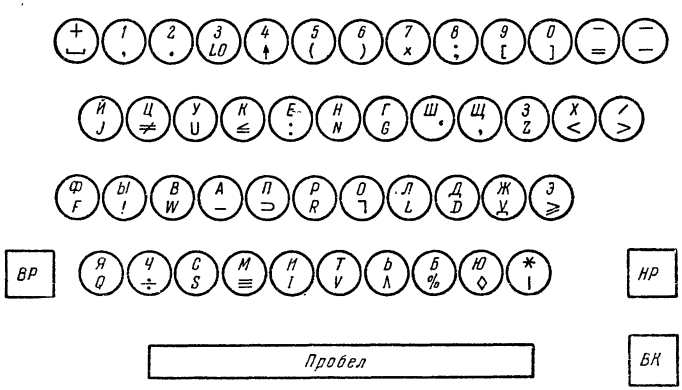


Рис. 37.

Каждая цифра, как и при нанесении числовой информации, изображается одной пробивкой в колонке, а каждая буква или служебный знак — комбинацией из двух пробивок, в соответствии с таблицей 2.40, содержащей 47 символов.

Во многих случаях указанного набора из 47 символов для работы явно не хватает, в этих случаях используется клавиатура, изображенная на рис. 37 и содержащая 92 символа. Не останавливаясь на вопросе о способе кодировки на десятичных картах этого набора символов, заметим, что алфавитно-цифровая информация может наноситься на перфокарты не по 80 колонкам, а по 12 строкам (двоичные карты). Для этого используется та же клавиатура (рис. 37), которая подсоединяется к перфоратору, пробивающему информацию на перфокарте не по колонкам, а по строкам. В этом случае под символ отводится по 7 двоичных разрядов (см. табл. 3.40). В строке перфокарты помещается 6 символов, эти 6 символов вводятся обычным образом с двоичной карты в одну сорокапятиразрядную ячейку памяти машины и располагаются в ячейке по схеме, изображенной на рис. 38.

	I символ	II символ	III символ	IV символ	V символ	VI символ
45 44 43	42 ... 37 36	35 ... 30 29	28 ... 23 22	21 ... 16 15	14 ... 9 8	7 ... 2 1
номера разрядов ячейки						

Рис. 38.

Для вывода из машины алфавитно-цифровой информации в настоящее время широко используется алфавитно-цифровое печатающее устройство АЦПУ-128. При помощи этого устройства можно печатать на широкой бумажной ленте любой набор из 92 графических символов табл. 3.40 *), причем в каждой строке может быть расположено до 128 символов.

Материал для печати одной строки занимает 22 ячейки внутренней памяти машины, в каждой ячейке по 6 семиразрядных символов — по схеме, изображенной на рис. 38. Кодировка символов приведена в табл. 3.40.

Обычно программист не формирует массивы из семиразрядных символов, предназначенных для печати на АЦПУ. Это делают специальные библиотечные подпро-

*) В ранних моделях АЦПУ имеется возможность печатать лишь первые 78 символов из табл. 3.40.

Таблица 2.40

Символ	Пробивки	Символ	Пробивки	Символ	Пробивки
0	0	№	6—9	Р	5—7
1	1	А	5—9	С	4—7
2	2	Б	4—9	Т	3—7
3	3	В	3—9	У	2—7
4	4	Г	2—9	Ф	1—7
5	5	Д	1—9	Х	0—7
6	6	Ж	9—11	Ц	7—11
7	7	З	3—12	Ч	4—12
8	8	И	5—8	Ш	5—6
9	9	Й	7—9	Щ	4—6
—	11	К	3—8	Ъ	3—6
.	4—8	Л	2—8	Ы	2—6
,	8—9	М	1—8	Ь	6—12
*	6—7	Н	0—8	Ю	0—6
◇	6—8	О	0—12	Я	6—11
%	7—8	П	8—11		

граммы, занимающиеся, кроме того, и редактированием числового и текстового материала, предназначенного для печати. Текстовой материал наносится на перфокарты при помощи клавиатуры, описанной ранее, вводится в машину и печатается в виде заголовков таблиц, пояснений и т. п. в нужные моменты работы программы.

Одной из простейших стандартных подпрограмм печати на АЦПУ является программа печати текста с обращением

$$\begin{array}{llll} \textit{БВ}\ \textit{Я}+1 & \text{Текст} & \Omega \\ a_1 & a_n & 0 \end{array}$$

По этой программе на широкой бумажной ленте АЦПУ печатают семиразрядные символы, расположенные в ячейках от a_1 до a_n внутренней памяти машины. Числовая

Таблица 3.40

Символ	Изображение		Символ	Изображение		Символ	Изображение	
	двоичное	восьмеричное		двоичное	восьмеричное		двоичное	восьмеричное
0	000000	0	:	011111	37	Я	111110	76
1	000001	1	А	100000	40	D	111111	77
2	000010	2	Б	100001	41	F	1000000	100
3	000011	3	В	100010	42	G	1000001	101
4	000100	4	Г	100011	43	I	1000010	102
5	000101	5	Д	100100	44	J	1000011	103
6	000110	6	Е	100101	45	L	1000100	104
7	000111	7	Ж	100110	46	N	1000101	105
8	001000	10	З	100111	47	Q	1000110	106
9	001001	11	И	101000	50	R	1000111	107
+	001010	12	Й	101001	51	S	1001000	110
—	001011	13	К	101010	52	U	1001001	111
/	001100	14	Л	101011	53	V	1001010	112
,	001101	15	М	101100	54	W	1001011	113
.	001110	16	Н	101101	55	Z	1001100	114
⌴	001111	17	О	101110	56	—	1001101	115
10	010000	20	П	101111	57	⩽	1001110	116
↑	010001	21	Р	110000	60	⩾	1001111	117
(	010010	22	С	110001	61	∨	1010000	120
)	010011	23	Т	110010	62	∧	1010001	121
×	010100	24	У	110011	63	⊃	1010010	122
=	010101	25	Ф	110100	64	¬	1010011	123
;	010110	26	Х	110101	65	÷	1010100	124
[	010111	27	Ц	110110	66	≡	1010101	125
]	011000	30	Ч	110111	67	%	1010110	126
*	011001	31	Ш	111000	70	◇	1010111	127
‘	011010	32	Щ	111001	71	|	1011000	130
’	011011	33	Ы	111010	72	—	1011001	131
≠	011100	34	Ь	111011	73	—	1011010	132
<	011101	35	Э	111100	74	!	1011011	133
>	011110	36	Ю	111101	75			

таблица, состоящая из целых двоичных чисел, может быть напечатана на АЦПУ при помощи библиотечной подпрограммы *Печать таблицы* с обращением

$$\begin{array}{lll} БВ\ Я+1 & \text{Печать таблицы} & \Omega \\ a_1 & a_n & M \end{array}$$

где (a_1, a_n) — границы таблицы в памяти, а M — адрес начала макета, образованного так же, как при обращении к программе *Ввод таблицы* (см. стр. 263).

По такой программе переводится «2 → 10» и печатается числовая таблица по k чисел в строке бумажной ленты, причем под числа таблицы отводятся соответственно m_1, m_2, ..., m_k позиций на бумаге.

Кроме рассмотренных простейших программ ввода и печати таблиц, в библиотеке стандартных подпрограмм должен быть большой набор обслуживающих программ ввода и вывода (разные типы ввода и вывода числовой информации, печать словарно-числовых таблиц, графиков и т. д.).

ГЛАВА VIII

ОТЛАДКА ПРОГРАММЫ

§ 41. Подготовка программы к отладке

Процесс решения задачи при помощи машины можно разбить на ряд этапов:

I) разработка алгоритма и составление программы в содержательных обозначениях;

II) распределение памяти и кодирование;

III) перфорация;

IV) отладка программы;

V) счет.

Математик, приступая к задаче, должен иметь в виду, что программа является одним из наиболее естественных способов записи алгоритма решения задачи. Поэтому мы и объединили в одном пункте создание алгоритма и написание программы. Математик, разрабатывая отдельные части алгоритма, записывает их в виде блоков программы. Эти отдельные части алгоритма, блоки, он объединяет вместе при помощи связующей части алгоритма — блок-программы.

На втором этапе прежде всего при помощи памятки производится распределение памяти. В тех случаях, когда программа не помещается во внутренней (оперативной) памяти, распределение материала между внутренней и внешней памятью становится существенной частью первого этапа решения задачи — разработки машинного алгоритма-программы.

На третьем этапе закодированную программу наносят на перфокарты. Полученную колоду перфокарт пропечатывают на бумажной ленте. Эту ленту для контроля считывают с правой частью программы. После третьего этапа работы алгоритм решения задачи оказывается превращенным в колоду перфокарт.

Казалось бы, теперь математику для решения задачи достаточно ввести программу в память машины, передать управление на начальную команду и подождать, пока на бумажной ленте напечатаются результаты счета. Однако на самом деле такую идеальную работу почти никогда не приходится видеть.

При первом выходе на машину никогда нельзя быть уверенным в том, что в программе нет ошибок. Поэтому счету, т. е. непосредственному решению задачи на машине, всегда предшествует отладка программы. *Отладка программы есть выявление и исправление ошибок в программе.*

Как мы уже говорили, при выходе на машину программа является колодой перфокарт. Ошибки в этой колоде могут появиться на первых трех этапах решения задачи. Отметим прежде всего, что кодирование и перфорация являются чисто техническим делом и при надлежащей организации работы ошибки на этих этапах могут быть почти полностью устранены. В тех случаях, когда программисту «не тесно» во внутренней памяти, в распределении памяти также обычно не бывает ошибок. *Поэтому отладка программы в основном состоит из выявления ошибок в алгоритме и в программе задачи.*

Перед выходом на машину каждый блок программы следует тщательно проверить путем детального просмотра. Однако такая проверка «на глаз» недостаточна для выявления всех ошибок.

К отладке следует детально подготовиться. *Основной составной частью такой подготовки является ручной расчет.*

Ручной расчет следует производить поблочно. Для этого берут сначала самые внутренние блоки программы, задают для блока определенные исходные данные и рассчитывают результаты.

Для проверяемого блока пишут отладочную программу, содержащую засылку исходных данных ручного расчета во входные ячейки блока, обращение к блоку и команды отладочной печати. Эти команды содержат обращения к библиотечной подпрограмме печати чисел, при помощи которых выводятся на печать исходные данные, промежуточные величины и результаты работы блока. Если, как мы рекомендовали выше, не совмещать ячейки для более или менее существенных промежуточных величин, то по такой

печати можно во всех деталях разобрать работу блока. Если в блоке имеются переменные команды, то полезно при помощи библиотечной подпрограммы *Печать программы* вывести на печать команды блока.

Заметим, что полный ручной расчет в силу громоздкости возможен не для всех блоков. Сплошь и рядом, например, встречаются блоки, содержащие циклы, работающие сотни, а то и тысячи раз. В этом случае проделывают ручной расчет для двух-трех шагов цикла, и на время отладки вносят изменение в проверяемый блок, заставляя его работать нужное число раз. В арифметическом блоке это можно сделать, меняя на время отладки соответствующим образом начальное (или конечное) значение счетчика или регистра адреса. В итерационном цикле для этого следует значительно увеличить допускаемую погрешность, тогда число итераций станет небольшим.

После ручного расчета для внутренних блоков проделывают расчеты и для блоков, использующих внутренние. Для этих блоков также пишут отладочные программы. При этом в отладочной программе, наряду с ячейками, характеризующими работу внешнего блока, полезно выдать на печать и содержимое ячеек, используемых в работе внутренних блоков, входящих во внешний. Если эти ячейки для разных внутренних блоков различны, мы по отладочной печати внешнего блока можем составить себе представление о совместной работе внешнего и внутренних блоков.

Для проверки программы всей задачи полезно сделать полный ручной расчет для одного из вариантов исходных данных *). Ручной расчет как для отдельных блоков, так и для задачи в целом следует делать в две руки и с большим числом знаков.

§ 42. Пульт машины

Пульт машины состоит из двух панелей: горизонтальной и вертикальной, на которых расположено большое количество кнопок, тумблеров, клавиш, переключателей и лампочек. Так как пульты рассматриваемых машин несколько

*) Организация такого расчета является довольно сложным делом, которое трудно стандартизировать.

различаются между собою, то мы дадим описание только тех устройств пульта, которые нужны программисту и имеются на всех машинах. Названия отдельных устройств и их размещение будут условными. В приложении будут сделаны необходимые уточнения.

На рис. 39 изображена схема вертикальной панели. В средней части вертикальной панели находятся несколько горизонтальных рядов из 45 неоновых лампочек. Горящая лампочка имеет значение двоичной единицы, негорящая — значение нуля. Нижний ряд из 45 лампочек называется *Регистр команд*. Выше этого регистра расположены еще три регистра — *РI*, *РII* и *РIII*, в которых горит содержимое ячеек по I, II и III адресам исполняющейся команды.

В правой части вертикальной панели расположены 12-разрядные ряды лампочек: *Счетчик команд* и *Регистр Адреса* (*РА*). В регистре команд горит команда, в счетчике команд — адрес команды в момент остановки машины, а в *РА* — значение регистра адреса в тот же момент. Кроме того, на вертикальной панели находятся лампочки — управляющий сигнал ω и сигнал аварийной остановки — *авост*.

Состояние описанных регистров полностью характеризует выполнение отдельной команды в машине. Поэтому их можно использовать для детальной, покомандной проверки программы.

На горизонтальной панели (рис. 40) находится ряд тумблеров, кнопок и переключателей, нужных для ручного управления работой машины.

1. Кнопка *Ввод*. При ее нажатии начинает работать устройство ввода, и материал с перфокарт, стоящих на этом устройстве ввода, поступает во внутреннюю память машины.

2. Кнопка *Пуск*, нажатие которой включает работу машины.

3. *Регистр команд пульта* — ряд из 45 клавиш; на них можно набрать команду, которую нужно с пульта исполнить.

4. *Переключатель режимов работы машины*. Он может занимать положения: *шаговый режим* *) и *автоматический режим*. При работе в шаговом режиме (*на шагах*) при нажатии кнопки *Пуск* машина выполняет очередную

*) На машинах типа М-20 шаговый режим назван *циклическим*.

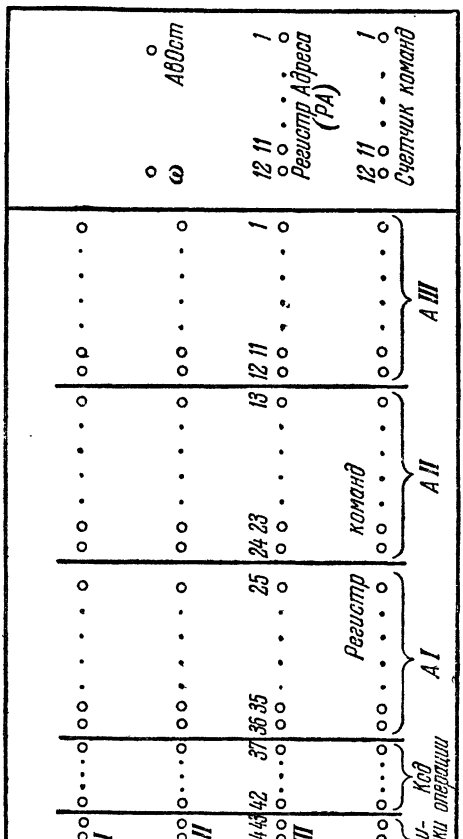

РI
РII
РIII
45 44 43 42 37 36 35 25 24 23 13 12 11 1
Признаки операции
Код операции
Регистр команд
A I
A II
A III
ω
A0Ост
12 11 1
Регистр Адреса (РА)
12 11 1
Счетчик команд

Рис. 39.

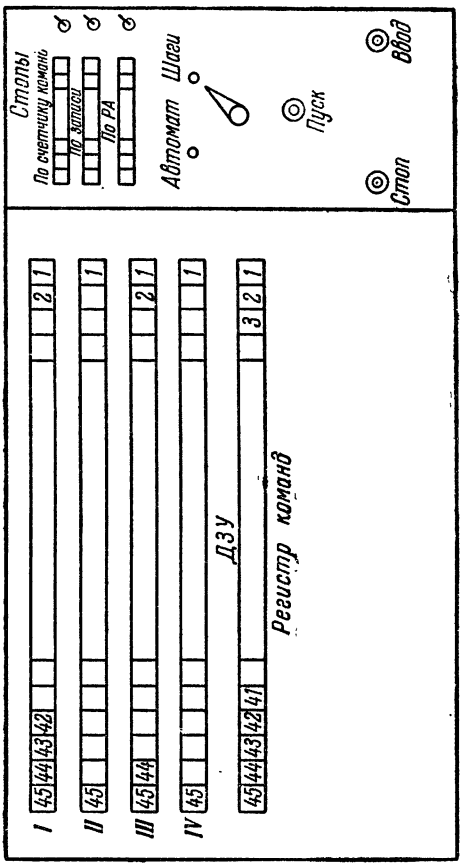

Столы
По счетчику команд
По адресу
По РА
Автомат Шаги
Пуск
Стоп
Ввод
I
II
III
IV
45 44 43 42 41
Регистр команд
ДЗУ

Рис. 40.

команду, после чего останавливается; чтобы выполнить следующую команду, нужно еще раз нажать кнопку *Пуск*. При положении переключателя *автоматический режим* машина после нажатия кнопки *Пуск* выполняет команды программы автоматически (*на автомате*), останавливаясь либо по команде *стоп*, либо в результате переполнения разрядной сетки машины (аварийная остановка).

5. Кнопка *Стоп*; служит для остановки работы машины или устройства ввода.

Указанных устройств пульта достаточно для того, чтобы ввести программу в машину и заставить ее работать. Для этого ставят на устройство ввода колоду перфокарт с программой, затем нажимают кнопку *Ввод*. Программа вводится во внутреннюю память машины. Устройство ввода прекращает работу, когда под его щетками окажется контрольное слово. Если в конце вводимого материала нет контрольного слова, то после того как все карты прошли через ввод, его работу нужно остановить кнопкой *Стоп*.

После этого на регистре команд пульта набирают команду передачи управления на начало программы:

$$16 \quad N_{\text{нач}},$$

где $N_{\text{нач}}$ — адрес начальной команды программы. Переключатель режимов ставят в положение *на автомате* и нажимают кнопку *Пуск*. На регистр команд переносится набранная команда *)

$$16 \quad N_{\text{нач}},$$

которая затем исполняется.

В счетчик команд заносится адрес $N_{\text{нач}}$ и машина начинает автоматически выполнять команды программы. При остановке машины по состоянию регистров вертикальной панели пульта можно определить характеристики последней исполнявшейся команды: на счетчике команд — ее адрес, на регистре команд — саму команду, на регистрах *РI*, *РII*, *РIII* — содержимое ячеек по I, II и III адресам, на *РА* — регистр адреса, на ω — значение управляющего сигнала.

*) Заметим, что эта команда равносильна 56 $N_{\text{нач}}$.

Помимо указанных устройств, на горизонтальной панели имеются еще устройства, используемые специально для отладки программы и оперативной работы за пультом.

6. Три *тумблера отладочных стопов*, включение которых дает возможность остановить работу машины в нужном месте. Возле каждого тумблера имеется ряд из двенадцати клавиш, в котором можно набрать двенадцатиразрядное двоичное слово.

Стоп по счетчику команд. При наборе на этом ряде клавиш адреса некоторой ячейки внутренней памяти машина останавливается перед исполнением команды, лежащей в этой ячейке. На счетчике команд в момент стопа горит указанный адрес.

Стоп по записи. При наборе здесь адреса ячейки памяти машина останавливается после исполнения команды, заносящей в эту ячейку какое-либо машинное слово.

Стоп по РА. Машина останавливается в тот момент, когда в регистр адреса заносится двенадцатиразрядное слово, равное набранному на этом ряду клавиш.

7. *Клавишные запоминающие устройства* (*КЗУ* или *ДЗУ*). Каждое из клавишных запоминающих устройств представляет собой ряд из 45 клавиш, в котором может быть набрано 45-разрядное двоичное слово. На пульте имеется четыре таких устройства и им присвоены адреса 7771, 7772, 7773 и 7774 ячеек памяти машины. Их обозначают *КЗУ*-1, *КЗУ*-2, *КЗУ*-3, *КЗУ*-4. Таким образом, у машины имеется еще одно запоминающее устройство из четырех ячеек, содержимое которых можно оперативно (вручную) менять на пульте машины. Машина в процессе выполнения команд программы может брать с *КЗУ* информацию, как из обычных ячеек памяти. Запись в эти ячейки памяти из машины невозможна *).

§ 43. Проверка работы программы на машине

Проверку работы программы на машине следует начинать с отладки самых внутренних блоков. Для этого, кроме основной программы, вводят в машину отладочные программы для отдельных блоков и передают управление на отладочную программу первого из проверяемых

*) О другом возможном режиме работы с КЗУ см. приложение, § 68.

блоков. Результаты работы блока, выданные на печать, сверяют с ручным расчетом. Если результаты совпадают, то блок считается отлаженным. В случае несовпадения каких-либо результатов сверяют напечатанные исходные данные и промежуточные величины с теми, которые были получены при ручном расчете. По несовпадающим промежуточным величинам обычно нетрудно найти участок блока, в котором имеется ошибка.

Если ошибку не удается обнаружить путем проверки блока «на глаз», можно поступить следующим образом. Пусть, например, участок блока от начальной команды K_1 до некоторой команды K_2 работает правильно (все промежуточные величины, вычисляемые на этом участке, выданы на печать такими же, какими они были при ручном расчете), а на участке от K_2 до K_3, по-видимому, имеется ошибка. Тогда составляют такую отладочную программу:

1)	*БВ*	*Я*+1	*Подготовка*	Ω
2)			K_3	= *Запас*
3)	*БВ*	*Я*+1	K_2	K_3
4)	*БВ*	*Я*+1	*Печать чисел*	Ω
5)		a_1	a_n	0
6)			*Запас*	$= K_3$
7)	*БВ*	*Я*+1	K_3	*Конец блока*
8)	*БВ*	*Я*+1	*Печать чисел*	Ω
9)		a_1	a_n	0
10)	*стоп*			

В этой программе команда 1) обращается к блоку, готовящему исходные данные для работы части отлаживаемого блока от команды K_2 до его конца. При помощи команд 2) и 6) выводится в запас и восстанавливается команда K_3, которая портится командой 3), обращающейся к участку $K_2 - K_3$ отлаживаемого блока*). Вывод на печать при помощи команд 4), 5), 8), 9) массива чисел $a_1 - a_n$ дает возможность проанализировать работу блока на двух участках $K_2 - K_3$ и K_3 — конец блока.

Ошибку в промежуточных или окончательных результатах работы блока иногда можно обнаружить еще следующим способом. Пусть нам нужно проверить образование ячейки a. Наберем ее адрес на клавишах отладочного *Стопа по записи* и включим тумблер этого стопа. Тогда при работе блока машина будет останавливаться после выполнения каждой команды, заносящей некоторое содержимое в ячейку a. По регистрам вертикальной панели пульта можно проанализировать изменение содержимого этой ячейки. При помощи *Стопа по записи* можно проследить, в частности, как изменяются переадресуемые команды.

Такими способами можно детально проверять работу всех внутренних блоков. При этом следует иметь в виду, что если в блоке имеются

*) Заметим, что команда K_3 при указанном способе отладки не должна быть переменной.

разветвления вычислительного процесса, то нужно проверить блок для нескольких вариантов исходных данных. Эти варианты должны составляться так, чтобы в программе проходились все возможные пути.

Найденные в блоках ошибки необходимо исправить в программных бланках и сразу же перебить и заменить соответствующие перфокарты.

После того как отлажены внутренние блоки, составляют отладочные программы для проверки внешних блоков. При несовпадении с ручным расчетом результатов работы внешнего блока (собирающей) на машине часто бывает полезно проверить правильность следования обращений к внутренним блокам. Для этого на клавишах отладочного *Стопа по счетчику команд* набирают адрес команды обращения к внутреннему блоку, включают тумблер этого стопа и передают управление на начало собирающей. Машина должна остановиться перед выполнением соответствующей команды обращения. Подобным же образом проверяется последовательность обращений и к другим внутренним блокам.

До сих пор мы предполагали, что машина проходит отлаживаемый блок до конца. Однако далеко не всегда так бывает. Здесь могут встретиться следующие случаи:

1. Машина продолжает вычисления без конца, не останавливаясь и совершая вычисления по некоторому замкнутому циклу (*зацикливание*).

2. Машина останавливается при переполнении разрядной сетки машины (*аварийный стоп*) *внутри блока*.

3. Машина останавливается по аварийному или обычному стопу *вне блока*.

К зацикливанию могут привести довольно разнообразные ошибки. Это может случиться при неверном изменении счетчика цикла. В этом случае можно обнаружить ошибку, набрав на клавишах *Стопа по записи* адрес этого счетчика и прослеживая его изменение при работе цикла. Если цикл меняется по регистру адреса, может помочь пультовый *Стоп по РА*. С его помощью можно проследить изменение *РА* в цикле. Часто бывает полезно использовать и *Стоп по счетчику команд*, набирая на его клавиатуре адрес команды, содержащей проверку условия окончания цикла.

Заметим, что эти рекомендации помогают только тогда, когда мы знаем, в каком блоке есть ошибки. Обнаружить это можно, проходя весь отлаживаемый блок или его части на шагах. Если блок является собирающей, то следует проходить собирающую на автомате, набирая на *Стопе по счетчику команд* адреса́ обращений к последовательным блокам. При зацикливании в одном из этих блоков машина не выйдет на следующий стоп. Если же все стопы «сработали», нужно искать зацикливание в собирающей.

Случай выхода на аварийный стоп, по существу, эквивалентен получению неверного результата. Неверное значение имеет та величина, адрес которой равен третьему адресу команды, обусловившей аварийный стоп. Для нахождения причины ошибки следует пройти на шагах отлаживаемый блок, от команды, в которой начинается вычисление соответствующей величины, и до команды, приводящей к аварийному стопу. При этом следует прослеживать по регистрам вертикальной панели пульта, как выполняется каждая команда. С третьим

случаем (стоп вне проверяемого блока) приходится сталкиваться тогда, когда имеются ошибки, приводящие к неверным передачам управления. При неверной передаче управления ячейке вне блока машина либо начинает выполнять «не те» команды, либо начинает воспринимать двоичные числа или константы, как команды. Ошибки такого рода обнаруживают при помощи пультового *Стопа по счетчику команд*, останавливая машину по очереди на последовательных командах передачи управления.

Встречаются ошибки, при которых передача управления происходит на ячейки, не занятые программой. Для обнаружения таких ошибок применяют следующий прием: перед вводом программы и началом работы производят не очистку внутренней памяти нулями, а заполнение всех ее ячеек машинным словом:

$$Щ = (777;\ F,\ F,\ F) = 777\ 7777\ 7777\ 7777.$$

Это делается при помощи специальной программы «*Роспись Щ*». После этого вводят свою программу.

Теперь, если при работе программы произойдет передача управления на ячейку, не занятую программой, в регистр команд поступит *Щ*. Это машинное слово воспринимается как команда *стоп*. Поэтому машина остановится, сигнализируя об ошибке в программе. Заметим, что указанный прием имеет еще одно применение. *Щ*, как двоичное число, есть наибольшее по модулю число, какое можно записать в машине. Поэтому, если арифметическая команда возьмет слово *Щ* в качестве числа, то в большинстве случаев произойдет аварийный стоп в результате переполнения разрядной сетки машины. Это обстоятельство дает возможность во многих случаях обнаружить ошибки в адресах арифметических команд.

При отладке программы за пультом машины сплошь и рядом программист оказывается в совершенно непредвиденной ситуации. Поэтому важно иметь удобные средства для оперативной работы за пультом. Так, например, очень полезно иметь возможность напечатать с пульта массив чисел или участок программы. Важно также уметь быстро и надежно исправлять с пульта ошибки в программе. При работе за пультом удобно пользоваться следующими тремя обслуживающими программами.

1. *Печать чисел с пульта*. Информацией для программы служат границы массива ячеек памяти, набираемые в первом и втором адресах первого клавишного запоминающего устройства пульта (*КЗУ*-1).

После передачи управления на начало этой программы числа из указанного на *КЗУ*-1 массива переводятся из двоичной системы в десятичную и печатаются. Напечатав последнее число, машина останавливается.

2. *Печать программы с пульта*. После передачи управления с пульта на начало этой программы содержимое ячеек массива, адреса начала и конца которого указаны в *КЗУ*-1, выдается на печать в командном виде. По окончании печати машина останавливается.

3. *Поправка команды с пульта*. Во втором адресе *КЗУ*-2 набирается адрес исправляемой команды, а в *КЗУ*-1 сама команда. После передачи управления на начало программы следует *стоп* и содержимое

КЗУ-1 и *КЗУ*-2 загорается в регистрах *РI* и *РII* вертикальной панели пульта. Проверив правильность набранной информации, следует нажать кнопку *Пуск*. Тогда произойдет исправление команды, исправленная команда напечатается, и машина остановится.

Этими тремя программами широко пользуются при пультовой отладке программ. Так, останавливаясь в «подозрительных» местах программы по пультовым отладочным стопам, выпечатывают при помощи указанных программ массивы рабочих ячеек или команд, после чего можно продолжать работу программы с того места, где она была остановлена.

Обнаружив ошибку, исправляют ее при помощи третьей из указанных программ. После отладки исправляют ошибки на перфокартах.

Заметим, что пультовая отладка является сложной работой, требующей высокой квалификации и не укладывающейся ни в какие твердые каноны. Эта работа зачастую занимает значительную долю времени работающей машины. Значительно облегчают отладку программ и сокращают время работы за пультом стандартные отладочные программы, краткие сведения о которых даны в следующем параграфе.

§ 44. Отладочные программы

Работу по отладке программ полностью автоматизировать, по-видимому, невозможно (по крайней мере, в настоящее время). Зачастую трудно отделить ошибку алгоритма решения задачи от ошибки программы. Машина для большинства современных программ работает лишь по заданному в программе алгоритму и потому не может сама менять этот алгоритм; поэтому и не удается заставить машину ни обнаруживать, ни устранять ошибки в программе.

Наиболее трудоемким этапом отладки является обнаружение ошибок. На это уходит львиная доля времени при пультовой отладке. Этот этап отладки можно частично автоматизировать, заставляя саму машину выдавать на печать достаточно обширную информацию о работе программы. Работу по обнаружению ошибок, пользуясь этой информацией, программист может провести уже «за столом», а не за пультом машины.

Выдачу нужной для обнаружения ошибок информации можно организовать при помощи специальных отладочных программ.

В процессе отладки программ для выдачи на печать нужной информации, вставки пропущенных команд и блоков используется так называемый *способ заплат*. По этому способу на место определенной команды программы ставится передача управления на свободное место памяти («заплата»), где пишутся операции печати нужной информации или вставляемые команды, затем пишется команда, затертая «заплатой», и операция передачи управления на команду, следующую за «заплатой».

В стандартной отладочной программе *Заплата* этот способ автоматизируется. В информации к этой программе указывается, какие действия (контрольные операции) следует произвести при постановке «заплат» в определенные места программы. Расстановку «заплат» и

выполнение контрольных операций в соответствии с записанной информацией производит программа *Заплата* *).

Программа *Заплата* может выполнять перед заданными командами контрольные операции следующих трех типов;

0) печать массива команд,

1) печать массива чисел,

2) выполнение блока команд.

Прежде чем отлаживаемая (основная) программа начнет работать вместе с программой *Заплата*, выполняя контрольные операции, необходимо в нужные места $\varkappa_1, \varkappa_2, \ldots, \varkappa_s$ расставить «заплаты». Для этого следует обратиться к подпрограмме *Расстановка заплат* следующим образом:

БВ	*Я*+1	*Расст. заплат*	Ω
$0N_1$	0	0	$\varkappa_1$
00	a_1	b_1	c_1
$0N_2$	0	0	$\varkappa_2$
00	a_2	b_2	c_2
. . . . .			
$0N_s$	0	0	$\varkappa_s$
00	a_s	b_s	c_s
00	0	0	0

Здесь $\varkappa_1, \varkappa_2, \ldots, \varkappa_s$ — адреса команд, на которые ставятся заплаты; $N_1, N_2, \ldots, N_s$ $(0 \leqslant N_i \leqslant 5)$ — номера соответствующих контрольных операций: $a_1, b_1, c_1, \ldots, a_s, b_s, c_s$ — информация к контрольным операциям.

После этого обращения программа *Заплата* расставляет «заплаты» на место команд $\varkappa_1, \varkappa_2, \ldots, \varkappa_s$, пересылая эти команды на свое рабочее поле. Затем машина останавливается в программе *Заплата*. Теперь для того чтобы в основной программе стали выполняться необходимые контрольные операции, следует передать управление на начало этой программы.

Опишем выполнение различных контрольных операций. Пусть команде $\varkappa$ соответствуют две строки в обращении к программе *Заплата*:

$0N$	0	0	$\varkappa$
00	a	b	c

При $N = 0, 1$ перед выполнением команды из ячейки $\varkappa$ основной программы выпечатывается массив от ячейки a до ячейки b основной программы.

При $N = 0$ этот массив выпечатывается в командном виде.

---

*) Из самого способа постановки «заплаты» следует, что та ячейка памяти машины, на которую ставится «заплата», должна быть постоянной командой. Действительно, если эта команда является формируемой или переадресуемой, то отлаживаемая программа «испортит» уход в программу *Заплата*.

Если $N = 1$, то содержимое ячеек массива от a до b, рассматриваемых как плавающие числа, переводится из двоичной системы в десятичную и печатается.

При $N = 2$ перед выполнением команды из ячейки $\varkappa$ выполняется блок, состоящий из команд, находящихся в ячейках от a до b.

Обычно при отладке критерием правильности работы программы является сравнение результатов работы программы с результатами ручного счета. Если результаты не совпали, следует искать ошибку в программе (либо в ручном счете). Часто для обнаружения ошибки полезно проделать сравнение не в конце решения задачи, а на каком-то промежуточном этапе решения задачи.

При помощи контрольных операций программы *Заплата* можно автоматизировать это сравнение.

В *Заплате* предусмотрено сравнение трех видов:

а) сравнение по абсолютной погрешности ε — два числа a и d считаются совпадающими, если $|a - d| < \varepsilon$;

б) сравнение по относительной погрешности ε — два числа a и d совпадают, если

$$\left|\frac{a-d}{d}\right| < \varepsilon;$$

в) сравнение машинных слов a и d по эталону ε: слова «совпадают», если в них совпадают все те разряды, которые помечены единицами в слове ε.

Для того чтобы произвести сравнение, в память машины вводят «эталонный» массив, который, в случае, если он числовой, переводят в двоичную форму. В рассматриваемом случае в информации к контрольной операции (a, b) — границы массива проверяемых ячеек памяти. В ячейках с адресами от c до $c + b - a + 1$ находится эталон сравнения ε и числа, с которыми сравниваются числа из массива (a, b).

В программе *Заплата* контрольная операция сравнение чисел по абсолютной погрешности имеет номер $N = 3$, для сравнения чисел по относительной погрешности $N = 4$ и для сравнения машинных слов $N = 5$. Результатом работы контрольных операций является печать несовпадающих слов.

Часто при отладке появляется необходимость получения детальной информации о ходе выполнения отлаживаемой (основной) программы машиной и изменении состояния памяти машины в процессе работы программы. Такого рода информацию можно получить при помощи отладочных программ, работающих по принципу *прокрутки*. Основной частью любой прокруточной программы является блок прокрутки. Рассмотрим коротко схему его работы.

Основной особенностью прокрутки является то, что команды основной программы выполняются не в ней, а в блоке прокрутки. Пусть $\varkappa$ — адрес очередной по выполнению команды основной программы.

Перед выполнением команды в специальных ячейках блока прокрутки, являющихся аналогами регистров ω и $РА$, находятся значения сигнала ω и регистра адреса ($РА$), выработанные предыдущей по исполнению командой, а в ячейке $СК$, моделирующей счетчик команд, — адрес $\varkappa$.

Блок прокрутки помещает команду x в специальную рабочую ячейку $РК$ (аналог регистра команд). Затем производится анализ кода операции команды. Если это не команда передачи управления, то к адресу ячейки $СК$ прибавляется 1. Если x — команда передачи управления, то $СК$ либо увеличивается на 1, либо в $СК$ заносится второй адрес этой команды, а в ячейку $РК$ заносится команда x со вторым адресом, замененным адресом определенной ячейки блока прокрутки. Затем запоминаются значения сигнала ω и $РА$, выполняется команда в ячейке $РК$, моделируется запоминание значения нового сигнала ω (в ячейке ω) и $РА$, выработанных этой командой. После этого при новом значении $СК$ блок прокрутки начинает работу сначала.

Таким образом, в блоке прокрутки полностью моделируется выполнение команд основной программы машиной. Этот блок является составной частью любой прокруточной программы. Режим прокрутки дает возможность разорвать во времени выполнение двух последовательных команд основной программы и в этом промежутке времени выполнить некоторый дополнительный анализ и выдачу нужной информации на печать.

Любая прокруточная программа содержит еще две составные части:

А) блок анализа заданной программе информации;

Б) блок печати.

Характер работы этих блоков определяется задачами, которые решает прокруточная программа, и видом задаваемой ей информации.

Характерной особенностью всех прокруточных программ является существенное замедление работы основной программы. Коэффициент замедления определяется:

1) средним количеством команд прокруточной программы, выполняемых при прокрутке одной команды основной программы (под прокруткой здесь понимается работа всех трех указанных выше блоков);

2) средним количеством информации, печатаемой прокруточной программой (опять же в расчете на одну команду основной программы).

Как показывает практика работы на машине, второй из этих факторов в большинстве задач является основным. Для смягчения его действия прокруточная программа строится так, чтобы программист имел возможность задать в информации к программе такие условия, которые максимально ограничивают объем печати. Ограничение объема выдаваемой на печать информации играет и другую, еще более важную роль. Если будет выдано на печать чрезмерное количество информации, то программист потонет в ней и не сумеет получить полезные сведения.

Режим прокрутки дает возможность покомандно анализировать работу основной программы. При этом можно ставить и решать самые разнообразные задачи как связанные, так и не связанные с отладкой. Так, например, можно произвести статистику использования в разнообразных программах команд различных типов с точки зрения того, сколько раз они выполняются при работе программы. Такого рода работа имеет большое значение для конструирования новых электронных вычислительных машин. Однако в основном прокруточные программы применяются для отладки программ.

Для обнаружения ошибок в основной программе может оказаться полезной следующая информация:

1) последовательность адресов команд, выполняемых основной программой;

2) содержимое ячеек памяти и регистров устройства управления (*РА*, ω, *РК*), используемых при последовательном выполнении команд основной программы.

Информацию первого и второго типа выдают, соответственно, описываемые ниже отладочные программы *Луч* и *Няня*.

На начальном этапе отладки очень полезно иметь информацию о порядке выполнения команд основной программы.

Для того чтобы иметь такого рода информацию, не нужно знать последовательность адресов всех команд, выполняющихся в программе в том порядке, в каком они расположены. Для этого нам достаточно иметь лишь адреса «начала» и «конца» такой последовательности. Последовательность команд, имеющих в этом смысле начало и конец, называется «лучом». Дадим точное определение.

Линейным участком («лучом») называется последовательность непосредственно друг за другом следующих в памяти и непосредственно друг за другом выполняющихся команд, обладающая двумя свойствами:

1) первая команда последовательности выполняется не после предшествующей ей в памяти команды или этой команде передается управление с пульта машины;

2) после последней команды выполняется не следующая за ней в памяти или на последней команде машина останавливается.

Очевидно, как правило, «луч» начинается с команды, выполняющейся после команды передачи управления, и заканчивается командой передачи управления. Если внутри «луча» есть команды передачи управления, то каждая из них передает управление на следующую за ней команду.

В процессе работы основной программы «образуется» ряд «лучей». «Лучи» во внутренних циклах проходятся много раз. Если «луч» проходится в программе n раз, то число n называется его кратностью.

Переход с одного «луча» на другой может происходить при безусловной или условной передаче управления. В последнем случае это бывает обусловлено значением сигнала ω или регистра адреса (*РА*) основной программы, вырабатываемых к моменту выполнения условной передачи управления.

Команды условной и безусловной передачи управления являются средством осуществления логических и структурных связей в программе. Поэтому сведения о всех «лучах», проходимых при работе программы, очень полезны для проверки правильности ее работы.

Получить такого рода сведения можно при помощи программы печати линейных участков (сокращенно *Луч*), работающей по принципу прокрутки.

В процессе прокрутки *Луч* моделирует работу основной программы, специально анализируя работу команд передачи управления.

Программа *Луч* формирует и выводит на печать:

а) начало и конец каждого «луча» $a_{\text{нач}}$, $a_{\text{кон}}$;

б) его кратность n;

в) состояние сигнала ω и регистра адреса *РА* перед передачей управления на следующий «луч».

Описанный режим работы *Луча* имеет следующий существенный недостаток. При работе основной программы в режиме прокрутки будут выпечатываться все «лучи», в том числе и те, которые принадлежат стандартным библиотечным подпрограммам и уже проверенным блокам основной программы. Поэтому программист после работы программы получит такое грандиозное количество «лучей», в котором он физически не сможет найти нужную ему информацию.

Можно частично устранить этот недостаток, введя другой режим работы программы *Луч*, при котором выпечатываются не все «лучи», а лишь «лучи» в данном интервале.

Рассмотрим последовательность команд основной программы,

$$\nu_0,\ \nu_{k_0},\ \nu_{k_0+1},\ \ldots,\ \nu_{k_0+s_0},\ \nu_{k_1},\ \nu_{k_1+1},\ \ldots$$
$$\ldots,\ \nu_{k_1+s_1},\ \ldots,\nu_{k_r},\ \nu_{k_r+1},\ \ldots,\ \nu_{k_r+s_r},\ \nu_l,$$

обладающую следующими свойствами:

а) все эти команды находятся в интервале $[m, M]$;

б) все участки $(\nu_{k_i}, \nu_{k_i+s_i})$ $(i = 0, 1, 2, \ldots, r)$ являются «лучами»;

в) все команды, выполняемые в программе от $\nu_{k_i+s_i}$ до $\nu_{k_{i+1}}$ $(i = 0, 1, 2, \ldots, r-1)$, лежат вне интервала $[m, M]$;

г) команда ν_0 выполняется непосредственно перед командой ν_{k_0}, а ν_l — после команды $\nu_{k_r+s_r}$.

Тогда участок $(\nu_{k_0}, \nu_{k_r+s_r})$ называется *«лучом» в заданном интервале* $[m, M]$.

Так как программа обычно отлаживается поблочно, то интервалы $[m, M]$ следует выбирать так, чтобы они охватывали проверяемые в данный момент блоки или несколько взаимосвязанных блоков.

Рассмотрим теперь порядок работы с *Лучом*.

Перед отладкой следует написать, закодировать и набить на перфокарте команды обращения к программе *Луч*:

БВ	*Я*+1	*Луч*	Ω
0	0	L	0
0	M_1	N_1	0
0	M_2	N_2	0
. . . . .			
0	M_s	N_s	0
0	0	0	0

Здесь (M_1, N_1), (M_2, N_2), ..., (M_s, N_s) — участки памяти, в которых расположены отлаживаемые блоки, а L — адрес начальной команды основной программы. Затем в память машины следует ввести основную программу, *Луч*, обращение к *Лучу* и передать с пульта управление на первую ячейку этого обращения. Тогда основная программа начнет работать с команды L в режиме прокрутки, выдавая на печать «лучи», проходимые программой в интервалах (M_1, N_1), (M_2, N_2),..., (M_s, N_s).

Программа *Луч* прослеживает порядок выполнения команд программы и помогает выявлению ошибок в логике и структуре основной программы. Однако, если даже все передачи управления выполняются

в соответствии с алгоритмом решения задачи, нельзя еще быть уверенным, что программа не содержит ошибок. В ряде случаев полезно проанализировать, как выполняются отдельные команды программы, притом, разумеется, не все команды, а лишь команды, удовлетворяющие определенным условиям.

Получить информацию обо всех или некоторых деталях выполнения команд основной программы можно при помощи отладочной прокруточной программы *Няня*. Эта программа, прокручивая очередную команду с номером N основной программы, формирует в семи ячейках памяти p_0, p_1, p_2, p_3, p_4, p_5, p_6 следующую информацию:

p_0	CK — номер команды N,
p_1	ω — значение управляющего сигнала ω перед исполнением команды N,
p_2	PA — значение регистра адреса перед исполнением команды N,
p_3	PK — команда N с действительными адресами,
p_4	PK' — команда N с исполнительными адресами,
p_5	A'_1 — содержимое ячеек по I, II, III исполнительным адресам
p_6	A'_2 (по I и II адресам — перед исполнением
p_7	A'_3 команды, по III адресу — после исполнения)

Обращение к программе *Няня* имеет вид:

БВ	*Я* + 1	*Няня*	Ω
0	0	K	0
ν_1	L_1	M_1	0
ν_2	L_2	M_2	0
. . . . .			
ν_s	L_s	M_s	0
0	0	0	0

Здесь (L_1, M_1), (L_2, M_2), ..., (L_s, M_s) — участки памяти, в которых расположены интересующие нас команды. По каждой из этих команд программа *Няня* выдает на печать некоторую информацию. K — адрес начальной команды отлаживаемой программы.

Пусть команда с адресом $\varkappa$ находится в интервале (L_i, M_i), т. е.

$$L_i \leqslant \varkappa \leqslant M_i.$$

В этом случае перед печатью программа *Няня* анализирует информацию, содержащуюся в разрядах кода операции ν_i строки:

$$\nu_i \quad L_i \quad M_i \quad 0.$$

Если в младшей триаде ν_i стоит 0, то содержимое ячеек p_4, p_5, p_6 печатается в виде команд, если в этой триаде 1, то содержимое ячеек p_4, p_5, p_6 переводится из двоичной системы в десятичную и печатается в виде десятичной мантиссы и порядка. Старшую триаду кода мы пока будем считать равной 4.

Рассмотренные условия дают возможность ограничить объем информации, выдаваемой на печать по данной команде, и обеспечить необходимую форму выдачи.

Так, например, если в информации к *Няне* задана строка

041 1200 1356 0,

то по каждой команде, находящейся в интервале (1200, 1356), будут выданы на печать ω, PA, PK, PK' и содержимое по A'_1, A'_2, A'_3 в десятичном виде.

Заметим, что количество информации, выдаваемой на печать *Няней* в расчете на каждую прокручиваемую команду, значительно больше, чем в *Луче*. В силу этого при работе *Няни* следует иметь более гибкие средства, ограничивающие объем печатаемой информации. Так, например, желательно иметь возможность печатать информацию лишь о командах, использующих содержимое заданного массива ячеек или меняющих содержимое ячеек другого массива. Для того чтобы ввести такие условия, рассмотрим *динамическую классификацию ячеек памяти*.

Распределяя память машины для кодирования, программист выделяет ячейки для команд программы, исходных данных, результатов счета и т. д. Распределение памяти и разбиение ячеек по указанным типам происходит статически, т. е. без учета того, как в действительности эти ячейки будут «работать» в машине в процессе выполнения программы. Между тем, четкую классификацию ячеек памяти машины можно дать, лишь рассматривая память в динамике, т. е. в процессе выполнения самой программы.

Дадим классификацию ячеек памяти по отношению к работе данной программы.

Если в процессе выполнения программы содержимое некоторой ячейки поступает на регистр команд (PK), то эта ячейка называется *командой*.

Если при фактическом выполнении некоторой команды программы содержимое некоторой ячейки памяти поступает в арифметическое устройство, то такая ячейка называется *аргументом*. Если же, наоборот, из арифметического устройства некоторое машинное слово поступает в ячейку памяти, то эта ячейка называется *результатом*.

В программе, как правило, имеются ячейки памяти, являющиеся и командами, и аргументами, и результатами. Это так называемые переменные команды.

Для того чтобы дать более четкую классификацию, охарактеризуем каждую ячейку памяти одной восьмеричной цифрой (триадой), т. е. тремя двоичными разрядами. Пусть старший разряд равен единице, если данная ячейка — команда, и нулю, если не команда; средний разряд — единица, если ячейка является результатом, и, наконец, младший разряд равен единице для ячейки-аргумента. Тогда мы приходим к следующей классификации ячеек памяти:

Триада	Восьмеричная цифра	Тип ячейки
000	0	Пустышка
001	1	Константа
010	2	Результат
011	3	Рабочая ячейка
100	4	Постоянная команда
101	5	Команда-аргумент
110	6	Формируемая команда
111	7	Переменная команда

Основная масса ячеек, используемых в программах (т. е. имеющих характеристику, не равную нулю), имеет характеристики 1 (константа), 3 (рабочие ячейки), 4 (постоянные команды).

Переменных команд (характеристика равна 7) в программе обычно бывает мало, так как переадресацию команд в циклах часто удается организовать при помощи *РА*.

Формируемые команды (характеристика равна 6) встречаются в основном в стандартных программах со строками информации.

Вернемся теперь к заданию информации к программе *Няня*. Рассмотрим опять строку информации:

$$v_i \quad L_i \quad M_i \quad 0.$$

В старшей триаде кода v_i задается характеристика массива ячеек (L_i, M_i).

До сих пор мы считали эту характеристику равной 4, что соответствует выдаче информации по всем командам интервала (L_i, M_i).

Если характеристика равна 2, то выдача будет происходить лишь по командам, которые изменяют содержимое ячеек массива (L_i, M_i).

При характеристике 6 печатается информация о командах из интервала (L_i, M_i) и командах, формирующих команды из этого интервала. Случаи характеристик, равных 3, 5, 7, может разобрать сам читатель.

При работе *Няни* каждая прокручиваемая команда анализируется по всем интервалам (L_1, M_1), (L_2, M_2), ..., (L_s, M_s), заданным в строках информации. Если удовлетворяется условие попадания (по соответствующим характеристикам) в один или несколько интервалов, то производится печать информации о выполнении команды.

ЧАСТЬ ТРЕТЬЯ

АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

ГЛАВА IX

ЯЗЫК «АВТОКОД 1 : 1»

§ 45. Автоматическое кодирование

При изучении программирования в содержательных обозначениях мы видели, что процесс получения программы, которую можно набить на перфокартах и ввести в машину для выполнения, состоит из двух частей — написания программы в содержательных обозначениях и ее кодирования. Кодирование представляет собой перевод программы, написанной в содержательных обозначениях, на язык машины. Иногда о закодированной программе говорят, что она написана в кодах машины. При этом подразумевается, что каждая команда записана в виде восьмеричного числа, отдельные части которого означают код операции и адреса команды.

Если программа в содержательных обозначениях уже написана и для нее произведено распределение памяти, т. е. составлена памятка, то кодирование является совершенно формальной работой, совершаемой по определенным правилам. Ее можно передать другому человеку и она требует лишь выполнения немногих стандартных и легко описываемых правил. Естественно, возникает мысль о возможности передачи кодирования самой машине. Но для этого нужно, прежде всего, ввести в память программу, написанную в содержательных обозначениях. Можно ли это сделать и как?

На клавиатуре стандартного перфоратора находятся цифры, восьмеричные или десятичные, которые переводятся перфоратором в соответствующую триаду или тетраду и пробиваются в нужных позициях перфокарты Поэтому стандартные перфораторы могут перфорировать лишь уже закодированную правую часть программы.

Представим себе теперь, что у нас имеется перфоратор с более обширной клавиатурой, содержащей кроме цифр еще и буквы алфавита, на котором пишется программа в содержательных обозначениях, и все знаки, которые при этом используются (например, знаки арифметических операций, стрелки для указания сдвигов и пр.). Для того чтобы различным знакам, имеющимся на клавиатуре такого перфоратора, соответствовали различные пробивки, необходимо, чтобы часть машинного слова, соответствующая каждому знаку, имела достаточно большое число разрядов. Обычно их берут шесть или семь.

Такой алфавитно-цифровой перфоратор переводит каждый знак своей клавиатуры в шестиразрядное (или семиразрядное) двоичное число и с его помощью мы можем набить на картах, а затем и ввести в машину программу, написанную в содержательных обозначениях. Например, если принять соответствие

A	0100000
B	0100010
C	0110001
$+$	0001010
,	0001101
$=$	0010101

то команда

$$A+, B=C$$

изобразится следующим 45-разрядным машинным словом

000 0100000 0001010 0001101 0100010 0010101 0110001

Можно ли непосредственно выполнить такую команду, после того, как она введена в память в приведенном виде? Разумеется, нет, потому что мы только заменили названия знаков их двоичными эквивалентами, не приводя структуру записи данной команды к тому виду, который она должна иметь в машинном коде. Однако теперь мы имеем возможность произвести такое преобразование структуры команды с помощью специальной преобразующей программы.

При этом следует предположить, что вместе с программой в содержательных обозначениях, введенной в машину

описанным выше способом, таким же способом введена в машину также и памятка программы — таблица распределения памяти. Впрочем, можно пойти и еще дальше, поручив и распределение памяти самой машине или, точнее говоря, той же преобразующей (кодирующей) программе.

Способ кодирования, при котором кодирование осуществляется в машине специальной программой, называют *автоматическим кодированием*. Программа, выполняющая кодирование, т. е. перевод программы, записанной в содержательных обозначениях, в машинные коды, называется *автокодировщиком* или *транслятором*. Процесс автоматического кодирования мы будем также называть *трансляцией*.

Транслятор должен быть написан так, чтобы он был пригоден для трансляции любой программы. Поэтому необходимо заранее стандартизировать и унифицировать правила записи программы в содержательных обозначениях, обозначения для всех команд, применяемые обозначения для величин, возможные специальные знаки и т. п.

Весь упомянутый набор знаков и правил образует некоторый язык программирования, который является более гибким и разнообразным, чем язык машины, но по необходимости более формализованным, чем язык содержательных обозначений. Поскольку этот язык служит для автоматического кодирования, мы будем называть его «*автокод*». Программу, написанную с соблюдением всех правил автокода, будем называть *автокодной* или *программой на автокоде*. Далее, так как каждой отдельной команде автокодной программы всегда соответствует точно одна команда программы в коде машины, то принято говорить об «Автокоде 1 : 1» (один к одному).

§ 46. Алфавит языка. Команды и числа

Алфавитом языка называют полный набор употребляемых в нем символов. Алфавит автокода содержит знаки: русские и латинские строчные (малые) и прописные (большие или заглавные) буквы, цифры 0, 1, ..., 9, двоеточие, точка, запятая, открывающаяся и закрывающаяся круглая и квадратная скобка и знаки арифметических и логических операций.

Уже здесь выясняется необходимость уточнений. В содержательных обозначениях для умножения мы пользовались и точкой, и знаком $\times$. В автокоде мы должны будем употреблять лишь знак $\times$, поскольку точка имеет здесь другой смысл — она используется как разделитель при записи чисел (см. ниже). Аналогично, для деления нельзя использовать двоеточие. В автокоде деление обозначается косой чертой: $a/b = c$.

Команды в автокоде пишутся так же, как и в содержательных обозначениях. Для транслятора существенно, что все команды автокода по структуре их записи делятся на две группы: команды со средним расположением кода операции и команды с левым расположением кода операции.

Команды со средним расположением кода операции имеют вид:

адрес, знак операции, адрес, знак равенства, адрес.

Такая структура используется для записи арифметических, фиксированных и циклических операций, действий с порядками, логических операций и сдвигов. Команды с левым расположением кода записываются в виде:

обозначение операции, адрес, адрес, адрес

и применяются для операций передачи управления, работы с регистром адреса и с внешней памятью.

В автокоде допускается запись любого кода операции в коде машины. В этом случае команду необходимо записывать в форме с левым расположением кода. Например, команду $a \times b = c$ можно записать 05, a, b, c.

Для изображения специальных констант типа (F, 0, 0), (0, 1, 0) и т. д. используется форма записи (адрес, адрес, адрес).

Числа в автокодной программе могут быть написаны в любой строке десятичной системе. Они используются как константы, входные данные, параметры и т. д. Десятичное число записывается в обычной форме со знаком + или — впереди (знак + можно опускать), только целая часть отделяется от дробной точкой, а не запятой. Если целая часть числа равна нулю, то ее можно опустить, оставив лишь разделяющую точку.

Например, можно писать

125.37, —12.5, +3791.2, +0.137, +.137, .137, причем последние три записи означают одно и то же число.

Можно пользоваться и записью числа с плавающей запятой. Для этой цели используется знак $_{10}$ (опущенная десятка), после которого следует помещать порядок числа (показатель степени). Так, число $0{,}125 \cdot 10^{-3}$ в автокодной программе запишется в виде 0.125_{10}—3. Показатель степени, идущий после опущенной десятки, должен быть непременно целым, знак + перед показателем не обязателен. Мантисса числа при записи с плавающей запятой может и не быть нормализованной, т. е. можно писать и $25_{10}4$, и $25_{10}2$, что означает одно и то же число 2500.

Ограничение при записи чисел в автокодной программе связано с тем, что при трансляции число будет переведено в двоичную систему и записано в ячейке памяти машины. Поэтому оно должно лежать в диапазоне $(2^{-64}, 2^{63})$ по абсолютной величине и иметь не более девяти значащих цифр мантиссы.

Адреса в командах автокодной программы могут писаться несколькими различными способами. Чаще всего для обозначения адреса используется идентификатор.

Идентификатором называется обозначение, выбранное для встречающейся в программе величины. В автокоде роль идентификатора может играть любая последовательность букв и цифр, начинающаяся с буквы. Примеры идентификаторов

a, *x*, *R*1, корень, номер 125, alpha, *pi*.

Команды с идентификаторными адресами имеют вид:

$$a + b = c,$$

БВ, *вход* 1, *заплата*, *возврат*,
скорость × *время* = *путь*,

$$\sqrt{R1} = x2.$$

Если какой-либо из адресов команды переадресуется по *РА*, то вслед за соответствующим идентификатором ставится звездочка (*), как это делалось и при записи в содержательных обозначениях.

Естественным ограничением на выбор идентификатора является запрещение использовать в качестве идентификатора сочетания букв, принятые для обозначения операций,

например *БВ*, *РА*, *У1*, *У0*. Специальный смысл имеет также идентификатор *Я*. Он обозначает адрес ячейки, в которой записана команда, содержащая адрес *Я*. Ограничения на длину идентификатора не накладываются.

Другим способом записи адреса является восьмеричный адрес. Так мы будем называть целое восьмеричное число без знака, взятое из диапазона адресов оперативной памяти машины. Восьмеричный адрес можно использовать для указания в команде действительного адреса ячейки памяти машины; например, команда

$$4500/a1 = C$$

означает, что делимое лежит в ячейке 4500 оперативной памяти.

Особенно удобно использование восьмеричного адреса в том случае, когда адрес команды является условным числом, как это бывает в командах сдвига по адресу, работы с регистром и т. п. Отличие использования восьмеричного адреса в автокодной программе от его использования в содержательных обозначениях состоит лишь в том, что восьмеричный адрес теперь пишется без черточки наверху.

В качестве адреса можно употреблять также десятичное число в любой форме записи, заключив его в круглые скобки. Такая запись обладает большой наглядностью при использовании десятичного числа в качестве аргумента операции. Фактически мы уже пользовались такой записью в содержательных обозначениях, заключая десятичное число в кавычки. Так, мы писали

$$x + «0,\ 125» = x,$$

тогда как в автокодной программе следует писать

$$x + (0.\ 125) = x.$$

Кроме десятичного числа в качестве адреса может быть взята также команда, заключенная в круглые скобки. Эта форма задания очень удобна при формировании команд. Например, можно писать

$$(D \vee 0 = 0) +, R1 = A.$$

Наконец, в ряде случаев бывает удобно использовать о т н о с и т е л ь н ы й а д р е с, который представляет собою выражение вида идентификатор $\pm$ восьмеричный адрес, взятое в круглые скобки. Например,

БВ, *Вход* 1, *сигма*, (*вход* 1 + 15).

Такой способ записи адреса использовался нами при обращении к стандартным программам, например:

БВ *Я* + 1 *интеграл* Ω.

Здесь первый адрес команды является относительным. Некоторые другие формы представления адресов будут рассмотрены нами в следующих параграфах.

М е т к и для обозначения команд при программировании в содержательных обозначениях могли быть самыми разнообразными. Для этой цели мы использовали любые значки, цифры и т. п. В автокодных программах в качестве меток будут применяться только идентификаторы, т. е. последовательность букв и цифр, начинающаяся с буквы.

Метка всегда пишется перед автокодной строкой и отделяется от нее закрывающейся круглой скобкой. В адресах команд метка используется в форме идентификатора, в соответствии с правилами записи команд.

§ 47. Описание величин. Массивы

Как мы видели в предыдущем параграфе, автокодная программа практически не отличается от программы, написанной в содержательных обозначениях. Если, кроме нее, ввести в машину распределение памяти для этой программы, то транслятор будет иметь полную информацию, для того чтобы транслировать программу. В каждой автокодной строке транслятор выделит символ операции, заменит его машинным кодом, который поместит на нужное место, а каждый адрес заменит фактическим в соответствии с заданной таблицей-памяткой.

Если же от транслятора требуется автоматическое распределение памяти, то одной автокодной программы для этого недостаточно. Для того чтобы транслятор правильно перевел автокодную программу и автоматически распреде-

лил память, все встречающиеся в программе величины должны быть описаны.

В автокодной программе различаются несколько классов используемых величин: простые переменные, массивы, десятичные числа, специальные константы, метки. Существует две формы описания величин этих классов.

При первой форме за описание величины принимается ее появление в автокодной программе, в соответствии с правилами записи. Ясно, что по этой форме можно описывать такие величины, сам способ изображения которых дает возможность определить, к какому классу они относятся. Это могут быть десятичные числа, специальные константы, метки.

Простые переменные и массивы по первой форме описаны быть не могут. Для описания величин по второй форме вводится слово-описатель **ДАННЫЕ**. Описатель воспринимается как единый символ, не зависящий от букв, из которых он состоит. Описание простых переменных начинается с описателя **ДАННЫЕ**, после чего выписываются разделенные запятыми идентификаторы всех простых переменных, которые будут использованы в программе.

Например, если автокодная программа содержит запись:

ДАННЫЕ *сумма, среднее, x, y, z, R,*

то в программе идентификаторы *сумма, среднее, x, y, z, R* будут восприняты как простые переменные. Каждому идентификатору, упомянутому в описании, транслятор ставит в соответствие одну ячейку памяти. Всюду, где будут в автокодной программе встречаться эти идентификаторы, они будут заменены адресами соответствующих ячеек памяти.

Однако ограничиться только простыми переменными часто не удается. Как мы видели во второй части книги, при решении многих задач приходится иметь дело с конечными числовыми последовательностями. При программировании мы обозначали различные элементы такой последовательности одной и той же буквой с различными индексами, например, $x_1, x_2, \ldots, x_n$ и при распределении памяти располагали их в последовательных ячейках.

В автокоде для описания числовых последовательностей вводится понятие м а с с и в а. Под массивом понимается упорядоченная последовательность числовых величин, обозначенная каким-либо одним идентификатором с индексами. Нужно только иметь в виду, что обычное написание индексов внизу буквы под строкой в автокодной программе не разрешается, так как на перфораторе нет соответствующих знаков. Индексы в автокоде пишутся рядом с идентификатором в одной строке и заключаются в квадратные скобки, например, вместо R_0, R_1 пишется R [0], R [1].

Массив характеризуется своей р а з м е р н о с т ь ю — количеством индексов, определяющих любой элемент массива. Числовая последовательность x_1, x_2, ..., x_n, или в автокодном написании x [1], x [2] ...x [n], является примером о д н о м е р н о г о массива. Если последовательность представляет собою м а т р и ц у, каждый элемент которой определяется двумя индексами — номерами строки и столбца, — то массив будет двумерным. Возможны и более сложные структуры, образующие многомерные массивы.

Описание массива для транслятора требует не только указания количества занимаемых им ячеек памяти, но и организации соответствия между индексами и элементами последовательности. Для этого необходимо указывать размерность массива и границы изменения его индексов.

В автокодной программе применяется следующая форма описания массива:

$$\textit{идентификатор}\ [N_1{:}M_1,\ N_2{:}M_2,\ \ldots,\ N_k{:}M_k].$$

Пара $N_i : M_i$ называется граничной парой по i-му измерению, ее элементы N_i, M_i, соответственно, нижней и верхней границей по i-му измерению. Они представляют собою целые десятичные числа, показывающие диапазон изменения i-го индекса. Число k граничных пар определяет размерность массива.

Число ячеек памяти, требуемых для записи массива, определяется транслятором по очевидной формуле

$$S=(M_1-N_1+1)\times(M_2-N_2+1)\times\ldots\times(M_k-N_k+1).$$

Например, матрица

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix},$$

которую мы назовем *Табл* 1, должна быть описана в автокоде следующим образом:

Табл 1 [1:3, 1:4].

Из этой записи следует, что *Табл* 1 представляет собой двумерный массив, первый индекс которого пробегает значения 1, 2, 3, а второй — значения 1, 2, 3, 4 и что для ее записи в памяти необходимо выделить $(3 - 1 + 1) \times (4 - 1 + 1) = 12$ последовательных ячеек.

Описание массивов нет нужды записывать отдельно от описания простых переменных. Они записываются вместе, образуя полное описание программы.

Например:

ДАННЫЕ *a*, *b*, *выход* 1, *x* [1:5, 2:3], *c*, *результат* [1:5, 2:3].

Для обозначения конкретного элемента последовательности вводится понятие переменной с индексом. Переменная с индексом обозначается идентификатором, соответствующим данному массиву; значения индексов записываются через запятую и заключаются в квадратные скобки. Переменные с индексом могут быть использованы в качестве адресов команд. Например, команда

$$a \times b\ [2,\ 3] = c$$

выполняет операцию умножения простой переменной a и фиксированного элемента массива b.

Как и любой адрес, переменная с индексом может иметь признак переадресации по *РА* (*).

§ 48. Блоки программы. Управляющие строки

Автокодная программа представляет собою последовательность строк, каждая из которых может быть либо командой, либо числом (десятичным или восьмеричным). Кроме таких строк, в программе должны быть и иные,

содержащие информацию, необходимую для работы транслятора. Такие строки, не являющиеся ни числами, ни командами, мы будем называть *управляющими строками*. К ним относятся, например, рассмотренные в предыдущем параграфе строки описания данных, но имеются и другие, с которыми мы сейчас познакомимся.

Как уже говорилось в § 45, задачей транслятора является трансляция программы, т. е. составление по программе в содержательных обозначениях программы в кодах машины. Однако, в конечном счете, такую программу надо не только получить, но и выполнить.

Автокод допускает возможность разделить или совместить по времени трансляцию программы и ее выполнение. Результат трансляции — рабочую программу в кодах машины — можно получить отпечатанным на алфавитно-цифровом печатающем устройстве (АЦПУ), вместе с исходной автокодной программой или без нее, а также записать полученную рабочую программу во внешнюю память или отперфорировать ее на выходном перфораторе. Если же требуется не только трансляция программы, но и ее немедленное выполнение, то после окончания трансляции нужно передать управление началу рабочей программы.

Информация о режиме работы задается транслятору с помощью нескольких различных управляющих строк. Выполнение программы непосредственно после трансляции задается с помощью управляющей строки

ВЫПОЛНИТЬ *m*,

где **ВЫПОЛНИТЬ** понимается как стандартный указатель, имеющий смысл единого символа, а *m* — метка команды программы, с которой должно начаться выполнение автокодной программы.

Если автокодную программу следует только транслировать и полученную рабочую программу записать во внешнюю память, то используется управляющая строка

ЗАГРУЗИТЬ 1

для случая записи оттранслированной программы в кодах машины на магнитный барабан или

ЗАГРУЗИТЬ 2

для записи оттранслированной программы на магнитную ленту.

Для перфорации на карты используется управляющая строка

ПЕРФОРАЦИЯ,

а для выдачи на печать через алфавитно-цифровое печатающее устройство (АЦПУ) — строка:

ПЕЧАТЬ.

Автокод допускает возможность исполнения всех этих процессов в различных сочетаниях. Например, последовательность управляющих строк

ПЕРФОРАЦИЯ
ПЕЧАТЬ
ЗАГРУЗИТЬ 2
ВЫПОЛНИТЬ *m*

вызовет следующую деятельность транслятора: программа будет оттранслирована, отперфорирована на картах через выходной перфоратор, напечатана в содержательных обозначениях и в кодах машины на алфавитно-цифровом печатающем устройстве, затем записана на магнитную ленту и, наконец, выполнена, начиная с метки *m*.

Описанные управляющие строки записываются в конце программы. Все связанные с ними операции будут выполняться в той последовательности, в которой они записаны. В частности, управляющая строка ВЫПОЛНИТЬ должна быть в последовательности управляющих строк последней.

Еще один тип управляющей строки, с которым мы сейчас познакомимся, связан с разбиением программы на блоки. Преимущества блочной структуры программы были достаточно подробно выяснены еще в гл. VII. Для автокодных программ к ним добавляется еще и то, что блочная структура программы позволяет совмещать автоматическое распределение памяти транслятором для одних блоков с «ручным» распределением самим программистом — для других. При этом в автокодной программе блоком принято называть участок программы, которому в рабочей программе соответствует участок последовательных ячеек памяти.

Приведенное определение блока несколько уже, чем определение, которое было дано в § 34, так что формально один блок в смысле определения § 34 может распадаться на два или несколько в смысле нынешнего определения. Однако это не играет никакой роли, тем более, что практически такого «распадения» обычно и не происходит.

Разбиение программы на блоки (иногда говорят с е г м е н т а ц и я) производится с помощью управляющей строки

БЛОК *Название блока, К*

Здесь **БЛОК** относится к числу стандартных указателей, *Название блока* есть идентификатор, а *К* — восьмеричное число из диапазона оперативной памяти машины. Появление этой управляющей строки в автокодной программе означает, что часть программы, расположенная после нее до следующей такой же строки, будет в рабочей программе занимать последовательные ячейки памяти, начиная с адреса *К*.

Например,

БЛОК *Подготовка*, 2000
. . . . . } 12 команд
. . . . . }
БЛОК *Результат*, 3200

означает, что часть программы, названная «*Подготовка*» и содержащая 12 команд, будет расположена в рабочей программе в ячейках от 2000 до 2013; часть программы, названная «*Результат*», будет расположена в ячейках, начиная с 3200 и т. д.

В управляющей строке **БЛОК** указание адреса (*К*) может отсутствовать. Такая управляющая строка считается информацией для транслятора о переходе на автоматическое распределение памяти. Например,

БЛОК *Первичный*
. . . . . .
. . . . . .
БЛОК *Решение*, 3000

означает, что участок программы, названный «*Первичный*», будет размещен в области автоматического распределения памяти, а участок «*Решение*» — начиная с ячейки 3000.

Название блока играет роль метки первой строки блока. Поэтому к участку программы, следующему за управляющей строкой **БЛОК**, можно обратиться с помощью команды передачи управления строке с названием блока.

При ручном распределении памяти для простых переменных и массивов в описание данных может быть вставлена строка вида

восьмеричный адрес: *список идентификаторов.*

Для идентификаторов, указанных в списке, будут отведены ячейки памяти, начиная с заданного восьмеричного адреса. Описание может выглядеть следующим образом:

ДАННЫЕ 3700: *a*, *c*

В этом случае под переменные *a*, *c* будут выделены ячейки памяти 3700 и 3701.

При описании

ДАННЫЕ 3700: *a*, *b*, 0200: *c*, *R*[1:5]

для переменных *a*, *b* будут отведены ячейки 3700, 3701, а для переменных *c* и массива *R* — ячейки 0200, 0201 — 0205.

При отсутствии адреса в управляющей строке или отсутствии соответствующих управляющих строк в программе, память для рабочей программы будет распределяться транслятором автоматически.

Рассмотрение языка Автокод 1 : 1 закончим примером автокодной программы.

Пример 1.48. Пусть задана функция

$$U(X) = 1 + \sum_{K=1}^{20} A_K \sin(KX + \pi/6),$$

где A_1, A_2, ..., A_{20} — известные числовые коэффициенты.

Предположим, что эта функция получена теоретически как описание некоторого физического процесса. Пусть из эксперимента найдены значения Y_0, Y_1, Y_2, ..., Y_{50} функции $U(X)$ для 51 значения аргумента X_0, X_1, X_2, ..., X_{50}, взятых с постоянным шагом H:

$$X_0,\ X_1 = X_0 + H,\ X_2 = X_1 + H,\ \ldots,\ X_{50} = X_{49} + H.$$

Степень близости «экспериментальной кривой» к теоретической можно охарактеризовать величиной максимальнсго отклонения:

$$M = \max_{0 \leqslant s \leqslant 50} |Y_s - U(X_s)|,$$

т. е. максимальным из отклонений экспериментальных точек от теоретических.

Составим программу для нахождения величины M.

Будем считать, что исходные величины для счета имеют вид трех колод перфокарт с числовым материалом и с контрольными суммами; в первой колоде — массив коэффициентов A_1, A_2, ..., A_{20}, во второй — величины Y_0, Y_1, ..., Y_{50}, в третьей — X_0, H.

При написании программы эти величины в десятичном виде будем обозначать *АД* [1], *АД* [2], ..., *АД* [20], *УД* [0], *УД* [1], ..., *УД* [50], *ХД* 0, *НД*. Те же величины, переведенные в двоичную систему счисления, будем записывать в виде: *A* [1], *A* [2], ..., *A* [20], *Y* [0], *Y* [1], ..., *Y* [50], *X* 0, *H*.

Программу начнем с описателя **ДАННЫЕ** и собирающей.

ДАННЫЕ 2500: *АД* [1 : 20], *A* [1 : 20], *УД* [0 : 50],

Y [0 : 50], *ХД*0, *НД*, *X*0, *H*, *X*

U, *M*, *K*, *R*1, *R*2

БЛОК СОБИРАЮЩАЯ

НАЧ)	*B*,	*АД*[1],	*Q*,	0
	B,	*УД*[0],	*Q*,	0
	B,	*ХД*0,	*Q*,	0
	БВ,	(*Я* + 1),	ПЕРЕВОД,	*W*
	0,	*АД*[1],	*АД*[20],	*A*[1]
	БВ,	(*Я* + 1),	ПЕРЕВОД,	*W*
	0,	*УД*[0],	*УД*[50],	*Y*[0]
	БВ,	(*Я* + 1),	ПЕРЕВОД,	*W*
	0,	*ХД*0,	*НД*,	*X*0
	БВ,	(*Я* + 1),	СЧ *M*,	*W*
	БВ,	(*Я* + 1),	ПЕЧ ЧИС,	*W*
	0,	*M*,	*M*,	0
	СТОП,	0,	0,	0
Q)	СТОП,	(*F*, 0, *F*),	0,	0

Здесь в описателе ДАННЫЕ описаны исходные массивы в десятичном и в двоичном виде, а также простые переменные X, U, M, K, $R1$, $R2$, играющие роль промежуточных величин и окончательных результатов счета.

В *собирающей* организуется ввод исходных данных с перфокарт, перевод их в двоичную форму, счет искомой величины (обращение к блоку СЧЁТ M) и печать результата M. Отметим, что в случае несовпадения контрольных сумм при вводе с перфокарт управление передается на аварийный стоп с меткой Q.

Блок *собирающая* выполняет чисто административную роль. Вычисление искомого отклонения M производится в блоке СЧЕТ M, который пишется следующим образом:

БЛОК СЧЕТ M

```
СЧ M)                W      =   KCM
                     0      =   M
        PA,  C*,     0,         (KCM-1)
        Б,   X0,     (Я+2),     X
   T1)       X      + H    =   X
   БВ, (Я+1),        СЧ U,      W
        Y[0]*        - U    =   R1
        R1 |         - |M   =   0
   У1,  0,           (Я+2),     0
        R1 |         - | 0    =   M
        PA<62,       T1,        1*
        Б,   0,      (КСM-1),   0
```

В этом блоке, являющемся обычным арифметическим циклом с переадресацией, считаются последовательно величины

$$|Y_0-U(X_0)|,\ |Y_1-U(X_1)|,\ \ldots,\ |Y_{50}-U(X_{50})|$$

и находится максимальная из них M. Расчет величины $U(X)$ производится в цикле при помощи обращения к

блоку СЧЕТ U, записываемому в виде:

```
      БЛОК СЧЕТ                          U
СЧ U)                          W    =   KCU
                               (1)  =   U
                               0    =   K
                  ПИ  /        (6)  =   C
          РА,     0*,          0,       (KCU-1)
   Т2)            K  +         (1)  =   K
                  K  ×         X    =   R1
                  R1 +         C    =   АРГ
          БВ,     (Я+1),       SIN,     W
                  A[1]*×       РЕЗ  =   R2
                  U  +         R2   =   U
          РА<23,               T2,      1*
          Б,      0,           (KCU-1), 0
ПИ)                            3.14159265
```

Автокодную программу закончим тремя управляющими словами:

ПЕРФОРАЦИЯ
ЗАГРУЗИТЬ 1
ВЫПОЛНИТЬ НАЧ

По первому из этих слов автокодная программа выводится на перфокарты, по второму — эта программа записывается на магнитный барабан, по третьему — передается управление на начало собирающей.

Отметим, что транслятор, согласно управляющему слову **ДАННЫЕ**, выделяет память для массивов и простых переменных, начиная с ячейки 2500. В блоках программы мы не указываем их место в памяти, поэтому для них память выделяет сам транслятор.

ГЛАВА X
ЯЗЫК АЛГОЛ-60

§ 49. Машинно-ориентированные и проблемно-ориентированные алгоритмические языки

Во второй части книги и в предыдущей главе третьей части речь шла о программировании для конкретной машины, хотя мы имели дело с тремя различными языками программирования.

Непосредственно выполняемая машиной рабочая программа должна быть написана на языке машины, или, как мы иначе говорили, в кодах машины. При этом программа записывается в виде последовательности элементарных операций машины, а информация о каждой элементарной операции — команда — имеет числовой вид: код операции и адреса ячеек памяти представляются в виде восьмеричных чисел, записываемых в определенной последовательности.

К сожалению, язык машины сложен и неудобен для человека, так как очень сильно отличается не только от обычного разговорного языка, на котором мы говорим, но и от общепринятого языка математических символов и формул. Это высказывание бесспорно для всякого программиста, которому приходилось разбирать чужую программу; зачастую это оказывается сложнее, чем написать ту же программу заново.

Машинный язык, в отличие от математического языка общения, обладает строгой однозначностью. Каждое слово машинного языка может пониматься единственным образом; на языке формул, а тем более, на разговорном языке, это не так. Например, запись $A \cdot B$ может означать и произведение величин (чисел), и произведение матриц, и

пересечение множеств, и совмещение событий. Что́ именно имеется в виду, должно следовать из дополнительной информации. Такая неоднозначность затрудняет перевод записанного формулами алгоритма на язык машины и, в особенности, обратный перевод.

К сказанному необходимо добавить, что каждый тип электронных вычислительных машин имеет свой набор элементарных операций, свою систему команд и, следовательно, свой машинный язык, так что программы, написанные на языке одной машины, не могут выполняться на машине другого типа.

Промежуточными языками между «символьно-формульным» математическим и машинным являются языки содержательных обозначений и автокод.

Пользование общепринятыми знаками математических действий, условные обозначения для специфически машинных операций, вроде передач управления или сдвигов, выбор идентификаторов для обозначения используемых в программе величин по возможности близкими к формульным значительно облегчают чтение программ. Вместе с тем и язык содержательных обозначений, и автокод обладают однозначностью машинного языка: команда, записанная в содержательных обозначениях или на автокоде, однозначно соответствует определенной команде машинного языка.

Собственно говоря, язык содержательных обозначений и язык автокод 1 : 1, описанный в предыдущей главе, принципиально не различаются между собою. В содержательных обозначениях допускается лишь бо́льшая свобода в выборе меток и идентификаторов (можно пользоваться произвольными символами и значками с любыми индексами снизу или сверху, подчеркиваниями и пр.) и в способе записи некоторых команд (замена меток в командах передачи управления стрелками и т. п.). Благодаря сделанным в автокоде дополнительным мелким ограничениям, программу оказывается возможным набить на алфавитно-цифровом перфораторе, ввести в машину и, тем самым, передать кодирование транслятору. Других различий между содержательными обозначениями и автокодом нет.

Ввиду указанной однозначной связи команды в содержательных обозначениях или автокодной команды с командой на машинном языке, при переходе к другой машине, с другой системой команд, и содержательные обозначения, и автокод 1 : 1 придется заменять. Эти языки ориенти-

руются на в п о л н е о п р е д е л е н н у ю машину. Поэтому их называют *машинно-ориентированными*.

Так как существует немало типов различных электронных вычислительных машин, то машинно-ориентированные языки не удовлетворяют нас вследствие трудностей перехода от одного типа машин к другому. Естественно желание создать такой язык программирования, который не требовал бы приспособления к языку конкретной машины. Такой язык не может находиться в отношении 1 : 1 к машинному, но это как раз и хорошо, так как позволит, в частности, избавить программиста от такой почти механической работы, как расписывание формулы по командам (см. § 11), и передать эту работу транслятору.

В настоящее время существует несколько сотен разных языков программирования. Уже сам этот факт показывает, что единого, удобного для всех областей применения вычислительных машин, универсального языка программирования все еще нет. Работа в области создания языков программирования идет, в основном, в направлении создания языков, хорошо приспособленных для задач из определенной области применения, т. е. задач определенного типа. Такие языки называют *проблемно-ориентированными*.

Нас интересуют, главным образом, задачи вычислительной математики. Из проблемно-ориентированных языков программирования, предназначенных для решения задач вычислительной математики, наиболее распространенным в СССР является международный язык алгол-60 *). Его изложению и посвящены следующие параграфы настоящей главы.

§ 50. Основные символы алгола

Все символы алгола разбиты на четыре группы: буквы, цифры, логические значения, ограничители.

В набор б у к в алгола входят строчные и прописные буквы латинского алфавита и строчные русские буквы.

*) Название «алгол-60» происходит от сокращения слов «алгоритмический язык» («ALGOrithmic Language») и указания года (1960), когда был утвержден основной свод правил этого языка. Нередко его называют универсальным языком программирования, понимая при этом под универсальностью независимость от языка конкретной машины.

В качестве ц и ф р алгола взяты арабские цифры от 0 до 9. Для обозначения л о г и ч е с к и х з н а ч е н и й используются два символа **true** (*да*) и **false** (*нет*). Эти символы являются самостоятельными знаками, не имеющими отношения к буквам, которыми они написаны.

Все символы алгола, которые изображаются последовательностью букв, при печатании набираются жирным шрифтом, а при письме от руки — подчеркиваются.

О г р а н и ч и т е л и представляются набором символов, которые обычным образом используются как знаки операций, скобки и для других целей. Этот набор символов мы будем вводить по мере дальнейшего изучения языка.

Итак, любой алгольный текст (программа, написанная на алголе) представляет собой последовательность алгольных символов. Алгольная программа набивается на алфавитно-цифровом клавишном устройстве, где каждая клавиша обозначает алгольный символ и перфорирует его на перфокарте (или перфоленте) в двоичном виде.

В алголе употребляются два типа чисел: *целые* и *действительные* (или *вещественные*). Основное различие между ними состоит в том, что действительные числа представляются в машине в форме с плавающей запятой и, следовательно, являются приближенными числами (погрешность представления зависит от конкретной машины и способа перевода), в то время как целые числа должны быть представлены в машине точно.

Целые числа изображаются последовательностью цифр, которым может предшествовать знак + или —. Никаких других алгольных символов в изображении целого числа не должно быть.

Примеры целых чисел:

170 0 356951 —100 +352 543125987000

Алгол не накладывает никаких ограничений на количество цифр в целом числе. Однако при построении трансляторов иногда вводятся ограничения на число цифр. Например, многие трансляторы требуют, чтобы целое число изображалось не более, чем двенадцатью цифрами.

Действительное число может быть написано в любой привычной для нас форме с тем лишь различием, что для

отделения целой части от дробной вместо запятой употребляется точка.

Использование точки оказывается более удобным, например, если требуется записать несколько десятичных чисел подряд. Запись

325,4 146,3 0,003

не может быть правильно прочитана, поскольку пробелы в алголе не принимаются во внимание. Наиболее естественным разделителем здесь является запятая. Поэтому за запятой оставлена роль р а з д е л и т е л я членов списка, а ее роль в изображении числа играет точка. Тот же пример теперь будет записан так:

325.4, 146.3, 0.003

Числа, у которых целая часть является нулем (правильная десятичная дробь), можно писать как с нулем впереди, так и без него; например,

0.143 и .143,
0.0017 и .0017

являются разными способами записи одних и тех же чисел.

Для записи числа в плавающей форме в алголе используется специальный символ — опущенная десятка ($_{10}$). Число 10^3 будет записываться как $_{10}3$, а число 10^{-5}, как $_{10}$—5. Перед таким числом может стоять знак, например, —$_{10}$—5, +$_{10}$—15 и т.д.

Употребительна запись

$15.6_{10}-3$ (что означает $15{,}6\times10^{-3}$),
$0.005_{10}2$, $.12_{10}+5$, $-15_{10}-2$

При записи чисел знак + можно опускать. Запрещается ставить точку в конце числа, например, записи 10. или —.0025. и $125_{10}-4.$ являются неправильными.

При описании вычислительных процессов приходится оперировать не только с числами, но и с другими объектами, такими как переменные, функции, массивы и т.д. Для обозначения таких объектов в алголе используются *идентификаторы*.

В предыдущей главе мы подробно рассмотрели вопрос о роли идентификаторов и правилах их построения. Напомним только, что идентификатором может служить любая последовательность букв и цифр (из числа букв и цифр алгола), всегда начинающаяся с буквы. Примеры идентификаторов:

summa, сумма, time, время, $x1$, $y25$.

§ 51. Арифметические выражения. Оператор присваивания

Понятие «арифметическое выражение» в алголе мало чем отличается от привычного понятия алгебраического выражения. Арифметическое выражение строится из чисел, переменных, знаков арифметических операций и круглых скобок и служит для вычисления некоторого числового значения.

Для обозначения операций употребляются следующие символы:

сложение	$+$
вычитание	$-$
умножение	$\times$
деление	$/$
возведение в степень	$\uparrow$.

Рассмотрим примеры:

обычная запись	*запись на алголе*
$a+b\cdot c$	$a+b\times c$
$x^2+2xy+0{,}5$	$x\uparrow 2+2\times x\times y+0.5$
$(x-\alpha)^m+k$	$(x-alpha)\uparrow m+k$
$\frac{a\cdot b}{c+d}$	$(a\times b)/(c+d)$.

Отдельное число или переменная тоже рассматриваются как арифметические выражения.

Порядок выполнения операций в арифметическом выражении обычный: сначала выполняются все возведения в степень, затем все умножения и деления и в конце —

сложения и вычитания. Действия одного старшинства выполняются в порядке следования слева направо, так что $a \times b/c \times d$ означает $((a \times b)/c) \times d$, а $a \uparrow x \uparrow 3$ в обычной записи означает $(a^x)^3$. Необходимый порядок выполнения действий в арифметическом выражении может быть установлен соответствующей расстановкой круглых скобок (в выражениях могут использоваться только круглые скобки). При расстановке скобок надо учитывать, что два знака операций не могут следовать один за другим. Выражение $a \times -b$ на алголе должно быть записано в виде $a \times (-b)$; a^{-3} не может быть изображено в виде $a \uparrow -3$, а должно быть записано как $a \uparrow (-3)$. Вместе с тем записи $-b \times a$ и $-a \uparrow 3$ являются правильными.

Значение выражения, получающееся подстановкой числовых значений вместо идентификаторов переменных и выполнения соответствующих операций, может быть либо целым, либо действительным. Тип результата зависит как от типа переменных, так и от операций, выполняемых над этими переменными.

Операции $+$, $-$, $\times$ дают результат типа *целый*, когда оба аргумента имеют тип *целый*. В противном случае тип результата *действительный*. Деление / всегда дает результат типа *действительный* независимо от типа аргументов. Так, выражение 10/5 дает действительное значение 2.0 (а не целое 2), в то время как 2×5 дает целое значение 10, а $2 + 5$ дает целое значение 7.

Возведение в степень $\uparrow$ с показателем типа *целый* понимается как умножение соответствующего числа нужных сомножителей. Например, $2.5 \uparrow (-3)$ понимается как $(1/2.5) \times (1/2.5) \times (1/2.5)$. Отсюда следует, что результат возведения в степень бывает типа *целый* только в случае целого основания и целого неотрицательного показателя степени. В остальных случаях результат будет типа *действительный*.

В двух случаях результат возведения в степень является неопределенным *): не определено возведение нуля в нулевую степень, $0 \uparrow 0$, и возведение отрицательного числа в действительную степень, например, $-3 \uparrow 2.5$.

*) Под неопределенным мы будем понимать результат, который определяется транслятором и, следовательно, может оказаться различным для разных трансляторов.

Рассмотренные выражения являются простейшими арифметическими выражениями. На практике часто приходится иметь дело с выражениями, которые содержат не только числа и переменные, но и элементарные функции. Для использования этой возможности в алголе имеется набор стандартных функций, которые могут входить в качестве аргументов в арифметические выражения.

Стандартная функция записывается в виде стандартного идентификатора, и аргумента, заключенного в круглые скобки. В алголе имеется следующий набор стандартных функций:

abs (E) — абсолютное значение E,

sign (E) — знак E: $\operatorname{sign}(E) = \begin{cases} 1 & \text{при } E > 0, \\ 0 & \text{при } E = 0, \\ -1 & \text{при } E < 0, \end{cases}$

sqrt (E) — квадратный корень из E,

sin (E) — синус (аргумент в радианах),

cos (E) — косинус (аргумент в радианах),

arctan (E) — главное значение (от $-\pi/2$ до $+\pi/2$),

ln (E) — натуральный логарифм,

exp (E) — показательная функция с основанием e или e^E

entier (E) — целая часть E.

Функции entier и sign дают результат типа *целый*.

Аргументом функции может быть любое арифметическое выражение. Например,

$$\sin(u+v+\cos(u-v)), \quad \operatorname{sqrt}(a\uparrow 2+b\uparrow 2/\exp(t)).$$

Приведем еще несколько примеров арифметических выражений.

Обычная запись	*Запись на алголе*
$\lvert 1-e^{4\cos x}\rvert$	$abs(1-\exp(4\times\cos(x)))$
$\ln x+y^k x$	$\ln(x)+y\uparrow k\times x$
$\dfrac{\sin x+\cos x}{\sqrt{a^2+1}}$	$(\sin(x)+\cos(x))/\operatorname{sqrt}(a\uparrow 2+1)$

В предыдущей главе для обозначения элемента массива было введено понятие переменной с индексом. Она пред-

ставлялась идентификатором, за которым в квадратных скобках шел список индексов, причем в качестве индексов можно было писать только целые числа. Например, *a* [15] означает 15-й элемент последовательности *a*, *сумма* [2,5] означает элемент второй строки и 5-го столбца матрицы *сумма* и т. д.

В алголе понятие переменной с индексом несколько расширено. Здесь в качестве индексов допускается использование арифметических выражений. Переменная с индексом может, например, иметь вид $x\,[i+1,\ j+2\times k]$ или $x\,[i]$. В свою очередь, переменная с индексом может входить в качестве компоненты арифметического выражения.

Приведем примеры арифметических выражений, содержащих переменные с индексами:

$$((x\times a\,[0]+a\,[1])\times x+a\,[2])\times x+a\,[3],$$
$$c\,[3\times k]+c\,[3\times k+1]\times x+c\,[3\times k+2]\times x\uparrow 2,$$
$$((3.75\times a\times b+c)/(p\,[i,\ j]+q\,[k]))\times m/n.$$

Если в состав арифметического выражения входят переменные с индексами, то вычисление выражения начинается с вычисления значений соответствующих индексных выражений. Как правило, в качестве индексов используются целые числа или выражения, принимающие лишь целые значения. Однако бывают случаи, когда индексы задаются выражениями, значения которых — действительные числа. В таких случаях полученное действительное число округляется до ближайшего целого. Например, x [2.3, 4.8, 3.8] задает тот же элемент трехмерного массива x, что и x [2, 5, 4]. Если в выражении $x\,[2\times k+1,\ 2\times p+4]$ переменные k и p принимают действительные значения, то результатом вычисления индексных выражений будут также действительные числа и, например, для $k=1.1$, $p=3.3$ получим x [3.2, 10.6], что равносильно x [3,11].

Основная роль арифметических выражений состоит в том, что вычисляемые по ним значения используются для получения или изменения значений других переменных. Чтобы указать, что какая-то переменная должна принять значение, полученное в результате вычисления арифметического выражения, используется *оператор присваивания*.

Оператор присваивания в простейшем случае имеет вид: ⟨переменная⟩: = ⟨арифметическое выражение⟩.

Эта запись означает: вычислить выражение, стоящее справа, и полученное значение присвоить переменной, стоящей слева. Приведем примеры операторов присваивания:

x: = 5 означает: присвоить переменной x значение 5;

u: = v означает: присвоить переменной u значение, которое имеет в данный момент переменная v;

sum: = $4 \times a + 2 \times b + (c-l) \uparrow 2$ означает: по значениям a, b, c, l вычислить значение выражения (получится число) и это значение присвоить переменной sum;

k: = $k + 1$ означает: увеличить значение k на единицу.

Хотя оператор присваивания похож на привычную запись уравнения, он вовсе не означает равенства левой и правой части (см. последний пример), а указывает на определенное действие: по выражению, написанному справа, определить число, которое будет значением переменной, стоящей слева. Таким образом, оператор присваивания дает возможность просто и естественно записать программу вычисления по формуле.

Пример 1.51. Вычислим значение переменной β по формуле

$$\beta = \frac{-2}{5x} + \frac{c^3}{4x^2}.$$

На алголе вычисление по этой формуле будет записано так:

beta : = — 2/(5 × x) + c ↑ 3/(4 × x ↑ 2)

Пример 2.51. Формула, по которой вычисляется значение r, имеет следующий вид:

$$r = \frac{1}{2\sqrt{-d}} \ln \frac{\sqrt{-d} - b - a}{\sqrt{-d} + b + c}.$$

На алголе требуемые вычисления будут записаны так:

r: = 1/(2 × sqrt (— d))
× ln ((sqrt (— d) — b — a)/(sqrt (—d) + b + c))

Обратите внимание на перенос части выражения на новую строку. На алголе такие переносы, как и пробелы между символами, не влияют на смысл написанного. Надо только учесть, что при переносе на новую строку знак операции повторять нельзя. Например, запись

$$s: = (k + l) \times m + \\ + \rho \uparrow (1 + n)$$ является неверной!

Алгол не накладывает никаких ограничений на тип переменной, стоящей слева, и тип значения выражения, стоящего справа. Если тип значения выражения не совпадает с типом переменной в левой части, то это значение выражения преобразуется в тип переменной, стоящей слева.

Это означает, что в процессе трансляции (или в процессе выполнения рабочей программы) всякий раз проверяется совпадение типов левой и правой части и в случае несовпадения автоматически выполняется преобразование в тип левой части. При записи операторов присваивания мы можем об этих преобразованиях не беспокоиться.

Заметим, что в алгольной программе можно написать подряд несколько операторов присваивания. В этом случае один оператор от другого отделяется точкой с запятой. Так, например, формулы для решения квадратного уравнения с действительными корнями на алголе можно записать в виде:

$$u: = \mathrm{sqrt}\,(b \uparrow 2 - 4 \times a \times c);$$
$$x1: = (-b + u)/(2 \times a);$$
$$x2: = (-b - u)/(2 \times a);$$

В случае, когда нескольким переменным $u1$, $u2$, $u3$ присваивается значение одного и того же арифметического выражения A, последовательность нескольких операторов присваивания:

$$u1: = A; \quad u2: = A; \quad u3: = A;$$

может быть заменена одним оператором присваивания:

$$u1: = u2: = u3: = A;$$

Такая форма оператора присваивания не только сокращает алгольную запись, но и, в ряде случаев, ускоряет выполнение протранслированной алгольной программы. В самом деле, в первом случае значение арифметического выражения вычисляется три раза, во втором — только один раз.

§ 52. Оператор перехода. Условные и составные операторы. Булевские выражения

При выполнении алгольной программы предполагается, что после выполнения некоторого оператора начнет выполняться оператор, следующий в программе непосредственно за только что выполненным.

Однако при выполнении программы очень часто приходится нарушать эту естественную последовательность выполнения операторов. Если после данного оператора нужно выполнить некоторый другой оператор, который не следует непосредственно за ним, то надо иметь возможность сослаться на него в программе. Для этой цели служит метка *).

Метка — это идентификатор, который может предшествовать любому оператору и отделяется от него двоеточием (:). Оператор, перед которым стоит метка, называется *помеченным* оператором. Например,

$$\begin{aligned} & x: = 2\times a\uparrow 2+x; \\ & u: = i+1; \\ L: \ & z: = x\uparrow u/p; \\ & u: = u+p; \end{aligned}$$

Третий оператор помечен меткой L.

Сама метка не меняет последовательности выполнения операторов. В приведенном примере после первого оператора будет выполняться второй, а после второго — третий, хотя перед ним и стоит метка. Чтобы иметь возможность изменить последовательность выполнения операторов, вводится оператор перехода.

*) С понятием метки мы уже познакомились по написанию программ на автокоде. В алголе роль метки мало чем отличается от ее роли в автокоде.

Общий вид этого оператора

go to ⟨метка⟩,

где **go to** (*иди*) символ алгола.

Если после данного оператора должен выполняться оператор, помеченный меткой L, то за данным оператором должен следовать оператор перехода **go to** L.

Пример 1.52. Найдем сумму членов рекуррентной последовательности a_k следующего вида *):

$$a_0 = 1,\ a_k = (k+1)(1+a_{k-1}) \quad \text{при } k = 1, 2, 3, \ldots$$

Программа на алголе может быть записана следующим образом:

```
        k: =1;
        a: =1;
    сумма: =0;
s:      a: =(k+1)×(1+a);   k: =k+1;
    сумма: =a+сумма;
      go to s;
```

Заметим, что расположение по одному оператору в строке не обязательно. Эту же программу можно было бы расположить и так

```
k: =1; a: =1; сумма: =0; s: a: =(k+1)×(1+a);
    k: =k+1; сумма: =a+сумма; go to s;
```

Оператор присваивания и оператор перехода относятся к классу *безусловных* операторов. При описании вычислительных алгоритмов нам очень часто приходится сталкиваться с разветвляющимися процессами. Чтобы обеспечить возможность написания программы для этих случаев, вводятся *условные* операторы.

Условный оператор указывает, к какому из двух операторов нужно перейти, в зависимости от выполнения каких-либо логических условий. В качестве простого логического условия могут быть взяты операции отношения

*) Этот пример играет чисто иллюстративную роль (реализация этого алгоритма на вычислительной машине приведет к аварийной остановке).

между арифметическими выражениями. В алголе используются следующие операции отношения:

$=$ равно,
$\neq$ не равно,
$>$ больше,
$\geqslant$ больше или равно,
$<$ меньше,
$\leqslant$ меньше или равно.

Примеры отношений:

$$a=3,\ a\leqslant b,\ c\neq d,\ (a+3)<5,$$
$$(u\uparrow 2-(1+k)\uparrow 2)\leqslant(p\uparrow 2+u).$$

Такого рода выражения (арифметические выражения, связанные операцией отношения) в алголе называют отношением. Точно так же, как значениями арифметических выражений являются числа, значением отношений являются логические значения **true** (да) и **false** (нет). Значением отношения будет **true**, если значения арифметических выражений, входящих в его состав, таковы, что соответствующее отношение выполняется. В противном случае значением отношения будет **false**.

Условный оператор может иметь одну из двух различных форм (неполную и полную).

Первая (неполная) форма условного оператора имеет вид

if ⟨логическое условие⟩ **then** ⟨безусловный оператор⟩;

здесь **if** (*если*) и **then** (*то*) — основные знаки алфавита алгола.

Неполный условный оператор выполняется следующим образом. Если логическое условие выполняется, т. е. принимает значение **true** (*да*), то оператор, стоящий после **then**, выполняется. Если же это условие принимает значение **false** (*нет*), то оператор, стоящий после **then**, пропускается. Вслед за условным работает оператор, написанный после него (кроме того случая, когда происходит выход «наружу» из идущего после **then** безусловного оператора).

Под логическим условием мы пока будем понимать отношение.

Рассмотрим примеры применения условных операторов.

Пример 2.52. Найдем большее из двух чисел a и b. Алгольная программа имеет вид:

$$M: = a;\ \textbf{if}\ b > a\ \textbf{then}\ M: = b;$$

Если $b > a$, то выполняется оператор $M: = b$, иначе этот оператор пропускается и M остается равным значению a, присвоенному ранее.

Пример 3.52. Составим алгольную программу, в которой переменным M и m присваиваются соответственно значения большего и меньшего из двух чисел a и b.

Программа состоит из четырех операторов присваивания, одного оператора перехода и одного условного оператора, содержащего оператор перехода:

```
        if a ≥ b then go to K;
   M: = b;      m: = a; go to продолж;
K: M: = a;      m: = b;
продолж:
```

В этой программе, в зависимости от соблюдения (или несоблюдения) условия $a \geqslant b$, выполняются либо первый и второй из операторов присваивания, либо третий и четвертый; при этом обход выполнения той или иной из этих пар операторов осуществляется при помощи двух операторов перехода, первый из которых является составной частью условного оператора.

При помощи условного оператора можно программировать составление циклов.

Пример 4.52. Составим программу вычисления величины $N = n!$. Программа состоит из четырех операторов присваивания (два — для подготовки, два — рабочая часть цикла) и оператора условного перехода

```
k: = 0;  N: = 1;  раб ч: k: = k + 1;  N: = N × k;
                  if k < n then go to раб ч;
```

Неполный условный оператор дает возможность выполнить или пропустить некоторый оператор в зависимости от соблюдения (или несоблюдения) логического условия. Часто встречается необходимость такого разветвления вычислительного процесса, когда условный переход при-

водит к необходимости не пропуска некоторого оператора, а выполнения одного из двух операторов. Такое разветвление можно реализовать, вводя в неполный условный оператор обычный оператор перехода (см. пример 3.52).

Однако это разветвление можно осуществить без оператора перехода при помощи полного условного оператора. Этот оператор записывается в виде:

if ⟨логическое условие⟩ **then** ⟨безусловный оператор⟩ **else** ⟨оператор⟩;

где **else** (*иначе*) — алгольный символ.

Полный условный оператор производит следующие действия. Если логическое условие соблюдается (т. е. принимает значение **true**), то выполняется безусловный оператор, следующий за **then**, а оператор, следующий за **else**, пропускается; в противном случае (т. е. когда логическое условие принимает значение **false**) безусловный оператор, следующий за **then**, пропускается, а оператор, следующий за **else**, выполняется.

Пример 5.52. Составить программу вычисления напряженности E электростатического поля, создаваемого зарядом, равномерно распределенным по объему шара радиуса ρ.

Из курса физики известно, что величина E определяется формулами:

$$E=\begin{cases}\dfrac{k}{r^2}, & \text{если } r\geqslant\rho,\\[2ex] \dfrac{kr}{\rho^3}, & \text{если } r<\rho,\end{cases}$$

где k — коэффициент, определяемый величиной поверхностного заряда и принятыми единицами измерения, а r — расстояние точки поля от центра шара.

Алгольная программа для вычисления E записывается в виде одного полного условного оператора.

if $r\geqslant ro$ **then** $E:=k/r\uparrow 2$ **else** $E:=k\times r/ro\uparrow 3$;

Здесь при выполнении условия $r\geqslant ro$ выполняется первый оператор присваивания, а второй пропускается, при невыполнении — наоборот.

Отметим, что, как следует из определения, в полном условном операторе оператор, стоящий за **else**, может быть тоже условным (в отличие от оператора, следующего за **then**). Этим можно воспользоваться для составления программ с многими разветвлениями.

Пример 6.52. Составим программу вычисления значения функции

$$u=F(y)=y^3-y+5, \text{ где}$$

$$y=\begin{cases} x^2-2x+2 & \text{при } x>2, \\ 2 & \text{при } |x|\leqslant 2, \\ -x^2-x-4 & \text{при } x<-2. \end{cases}$$

Алгольная программа записывается в виде

```
M0: if x>2 then M1: y:=x↑2-2×x+2 else
                M2: if abs(x)≤2 then y:=2
                                else M: y:=-x↑2-x-4;
                M3: u:=y↑3-y+5;
```

Рассмотрим порядок выполнения этой программы, состоящей из полного условного оператора с меткой *M*0 *) и оператора присваивания с меткой *M*3.

Если $x>2$, то выполняется оператор присваивания с меткой *M*1, после чего пропускается выполнение полного условного оператора, помеченного меткой *M*2 (вторая и третья строчки нашей записи); если же $x\leqslant 2$, то оператор присваивания с меткой *M*1 пропускается и выполняется полный условный оператор с меткой *M*2.

Как в первом, так и во втором случае выполнение программы заканчивается оператором присваивания с меткой *M*3.

При выполнении условного оператора некоторый оператор пропускается при соблюдении (или несоблюдении) логического условия. Как быть, если нужно пропустить несколько операторов? Для этого их достаточно объединить в *составной оператор* при помощи операторных скобок (символов алгола) **begin** (*начало*) и **end** (*конец*).

*) Метки *M*0, *M*1, *M*2, *M*3, *M* введены здесь в алгольную запись лишь для удобства словесного описания программы.

Так, например, программа нахождения бо́льшего и меньшего из двух чисел (см. пример 3.52) запишется при помощи операторных скобок следующим образом

```
if a ⩾ b then M1: begin M: =a; m: =b end
          else M2: begin M: =b; m: =a end;
```

Здесь при $a \geqslant b$ выполняется составной оператор с меткой $M1$ (пара операторов присваивания) и пропускается составной оператор с меткой $M2$ (другая пара операторов присваивания), при $a < b$ — наоборот.

При помощи операторных скобок **begin** и **end** в составной оператор можно объединить любую последовательность алгольных операторов. В частности, любой из операторов, входящих в составной оператор, может быть также составным оператором.

Пример 7.52. Функция $I = F(x, k)$ определена следующим образом:

$$I = \begin{cases} -\dfrac{(x+1)^{k+1}}{k+1} + 1 & \text{при } k \neq -1, \\ \displaystyle\sum_{s=1}^{10} (-1)^s \frac{x^s}{s} & \text{при } k = -1. \end{cases}$$

Алгольная программа для вычисления I записывается в виде полного условного оператора:

```
u1: if k=—1 then s1: begin I: =0; s: =0; u: =—x; v: =1;
                  u2: if s⩾10 then Q: go to продолж
                      else s2: begin s: =s+1; v: =v × u;
                                     I: =I+v/s;
                                     go to u2
                               end
                  end
      else P: I: =1—((x+1)↑(k+1))/(k+1);
продолж:
```

Здесь между первым **then** и последним **end** написан внешний составной оператор с меткой $s1$, который, в свою очередь, состоит из четырех операторов присваивания и полного условного оператора с меткой $u2$, в этом внутреннем условном операторе после **else** написан внутренний составной оператор с меткой $s2$.

Рассмотрим, как выполняется эта алгольная программа. Если $k \neq -1$, то пропускается составной оператор $s1$ *) и выполняется лишь последний оператор присваивания, вычисляющий по формуле $I = 1 - \frac{(x+1)^{k+1}}{k+1}$.

Если же $k = -1$, то выполняется составной оператор $s1$; в этом операторе ряд I вычисляется по следующим рекуррентным соотношениям:

$$I_0 = 0, \quad u = -x, \quad v_0 = 1, \quad v_{s+1} = v_s \cdot u, \quad I_{s+1} = I_s + v_s / (s+1)$$
$$(s = 0, 1, 2, \ldots, 9),$$

причем подготовка цикла осуществляется четырьмя операторами присваивания, стоящими в начале оператора $s1$, а рабочую часть цикла, реализующую две последние формулы, образует внутренний составной оператор $s2$. Цикл организован при помощи условного оператора $u2$. В этом операторе рабочая часть цикла $s2$ выполняется до тех пор, пока не соблюдается условие $s \geqslant 10$, при $s = 10$ выполняется следующий за **then** оператор перехода, передающий управление изнутри составных операторов $u2$ и $u1$ на продолжение счета.

На этом примере видно, что при любом расположении составных операторов можно выйти из составного или условного оператора при помощи оператора перехода. Заметим, что тем же оператором перехода можно войти внутрь условного или составного оператора, для этого, разумеется, нужно поставить метку на место перехода.

Отношение, входящее в состав условного оператора, дает возможность осуществить проверку простейших логических условий. Более сложные логические условия проверяются в алголе при помощи булевских выражений, частным случаем которых является отношение. Рассмотрение структуры булевских выражений мы начнем с введения понятия булевской переменной.

Наряду с переменными действительного и целого типа в алголе употребляются переменные булевского типа, которые могут принимать значения **true** (*да*) или **false**

*) Для удобства изложения мы в дальнейшем иногда будем считать названиями операторов метки, которыми они помечены; так, оператор $s1$ — это оператор с меткой $s1$.

(*нет*). Булевские переменные обозначаются в алголе идентификаторами. Над булевскими числами, переменными и отношениями вводятся следующие булевские (логические) операции:

$\equiv$ *совпадает*
$\supset$ *влечет*
$\vee$ *или*
$\wedge$ *и*
$\neg$ *не*

Значения булевских операций даются следующей таблицей

a	true	true	false	false
b	true	false	true	false
$a \equiv b$	true	false	false	true
$a \supset b$	true	false	true	true
$a \vee b$	true	true	true	false
$a \wedge b$	true	false	false	false
$\neg a$	false	false	true	true

Две операции нам уже встречались (см. § 31):

$\vee$ *или* — логическое сложение,
$\wedge$ *и* — логическое умножение.

Операция $\not\equiv$ (отрицание равнозначности) равносильна сложной операции $\neg \equiv$ (не совпадает).

Подобно тому, как из действительных или целых переменных, чисел, знаков арифметических операций и арифметических выражений, взятых в круглые скобки, строились арифметические выражения, так из булевских переменных, булевских значений, отношений, знаков булевских операций строятся булевские выражения.

Приведем некоторые примеры булевских выражений, считая, что булевские переменные обозначены здесь заг-

лавными буквами, а вещественные и целые — строчными:

$$x+y\leqslant 1 \vee x\geqslant 0 \wedge y<0,$$
$$y-2\times x\uparrow 2>0\equiv C,$$
$$a+b<d \wedge \textbf{false} \vee B,$$
$$k+1<i\equiv A \wedge c<d.$$

Порядок старшинства булевских операций следующий: $\equiv$, $\supset$, $\vee$, $\wedge$, $\neg$. Любое булевское выражение может быть взято в круглые скобки и войти в состав другого булевского выражения.

Пример 8.52. Пользуясь таблицей значения операций, вычислим значения двух булевских выражений

$$(a>b) \wedge (c<d)\equiv \textbf{true},$$
$$(\neg (a=b)\equiv \textbf{true}) \vee ((1+k<d) \wedge D)$$

при значениях переменных, указанных в таблице:

a	2
b	3.5
c	0.01
d	5.8
k	3
D	false

$$\underbrace{\underbrace{(a>b)}_{\textbf{false}} \wedge \underbrace{(c<d)}_{\textbf{true}}}_{\textbf{false}}\equiv \textbf{true}$$

Значением выражения будет **false**.

$$\underbrace{\underbrace{\underbrace{\neg \underbrace{(a=b)}_{\textbf{false}}}_{\textbf{true}}}_{\textbf{true}} \vee \underbrace{\underbrace{(1+k<d)}_{\textbf{true}} \wedge \underbrace{D}_{\textbf{false}}}_{\textbf{false}}}_{\textbf{true}}$$

Значением выражения будет **true**.

Приведем пример составления булевского выражения по заданному условию.

Пример 9.52. Переменная x принимает значения вне интервала (0, 1). Напишем булевское выражение, принимающее значение **true** при выполнении указанного условия и **false** в противном случае.

Требуемое булевское выражение можно записать в виде

$$x\leqslant 0 \vee x\geqslant 1.$$

Как и для арифметических выражений, значение булевского выражения можно с помощью оператора присваивания присвоить булевской переменной. Например,

B:=**true**; C: =**false**; D: =$\neg$ ($a \vee b$);
F:=$a > b$; R: =$\neg$ (($a \neq b$) $\wedge$ D).

Пример 10.52. Напишем оператор, присваивающий булевской переменной Q значение **true** при условии, что целая переменная n делится на 2 или 3, и **false** в противном случае. Требуемый оператор пишется в виде

Q:=entier (n/2) $\times$ 2=n) $\vee$ (entier (n/3) $\times$ 3=n);

Использование булевских выражений в их общей форме в условных операторах (вместо отношений) дает возможность проверки более сложных логических условий.

Неполный и полный условные операторы в этом случае записываются соответственно в виде:

if B_2 **then** S_2;

и

if B_1 **then** S_1 **else** T_1;

Здесь B_1, B_2 — булевские выражения, S_1, S_2 — безусловные операторы, а T_1 — произвольный оператор.

Выполняются эти операторы так же, как было описано ранее, с тем лишь отличием, что логические условия B_1, B_2 могут иметь более общую форму булевских выражений.

Обратим внимание на тот случай, когда оператор T_1, стоящий в полном условном операторе после **else**, сам является условным.

Пусть буква S с индексом будет обозначать безусловный оператор; тогда полный условный оператор можно изобразить в общем виде следующим образом:

if B_1 **then** S_1 **else if** B_2 **then** S_2 **else** S_3;

или

if B_1 **then** S_1 **else if** B_2 **then** S_2 **else if** B_3 **then** S_3;

В первом случае полный условный оператор состоит из нескольких неполных условных операторов, разделенных разделителем **else** и одного безусловного оператора (S_3); во втором случае — из нескольких неполных условных операторов. Ясно, что число внутренних условных

операторов может быть произвольным. Как же выполняется в этом случае полный условный оператор?

Вычисляется булевское выражение B_1. Если его значение **false**, то вычисляется значение B_2 и так далее, пока не будет найдено B_i со значением **true**. После этого выполняется безусловный оператор, следующий за этим условием, а затем оператор, следующий за полным условным. Если же все B_i имеют значения **false**, то в первом случае выполняется последний безусловный оператор условного оператора, а затем оператор, следующий за полным условным, во втором случае ни один из внутренних операторов условного оператора не выполняется, а выполняется оператор, следующий за полным условным.

В заключение рассмотрим пример логической задачи, решаемой при помощи условных операторов.

Пример 11.52. Действительные переменные a, b, c принимают положительные значения. Присвоить булевской переменной T значение **true**, если из отрезков с длинами a, b, c можно построить треугольник, и **false** — в противном случае. В первом из этих случаев булевской переменной U присвоить значение **true** или **false** в зависимости от того, треугольник тупоугольный или нет, во втором случае значение U оставить неопределенным. Из геометрии известно, что из отрезков a, b, c можно построить треугольник, если каждое из чисел a, b, c меньше суммы двух других; полученный треугольник будет тупоугольным, если квадрат одной из сторон больше суммы квадратов двух других.

Используя для проверки этих логических условий алгольную конструкцию «отношение», запишем программу нахождения булевских переменных T и U в виде

```
T: =true;      if a≥b+c then T: =false
            else if b≥a+c then T: =false
            else if c≥a+b then T: =false;
if T=false then go to продолж
            else if a↑2>b↑2+c↑2 then U: =true
            else if b↑2>c↑2+a↑2 then U: =true
            else if c↑2>a↑2+b↑2 then U: =true
            else U: =false;
продолж:
```

Читателю предоставляется возможность разобрать эту программу при помощи только что рассмотренной схемы работы условных операторов.

Заметим, впрочем, что, если воспользоваться булевскими выражениями общего вида, то алгольная программа для нашей задачи запишется совсем просто:

T: $=(a<b+c) \wedge (b<c+a) \wedge (c<a+b)$;
if T $=$**true then** U: $=(a\uparrow 2>b\uparrow 2+c\uparrow 2) \vee$
$(b\uparrow 2>c\uparrow 2+a\uparrow 2) \vee (c\uparrow 2>a\uparrow 2+b\uparrow 2)$;
продолж:

§ 53. Оператор цикла

Как мы уже видели в предыдущих главах, наиболее часто встречаемой особенностью вычислительного процесса является его цикличность. Циклы в алголе можно организовать при помощи условного оператора. Однако более целесообразная организация циклических программ обеспечивается оператором цикла.

Оператор цикла имеет вид

for k: $=E_1, E_2, \ldots, E_n$ **do** S;

здесь **for** (*для*) и **do** (*цикл*) *) — знаки алгола,

k — переменная,

S — алгольный оператор,

E_i — элементы цикла, образующие список цикла E_1, $E_2, \ldots, E_n$.

Переменную k называют *параметром цикла*.

Оператор цикла заставляет выполниться оператор S нуль или более раз для каждого элемента цикла, в порядке следования этих элементов в списке. Элементы цикла бывают трех типов, которые называются *арифметическим*, *прогрессии* и *пересчета*.

Элементом цикла арифметического типа является арифметическое выражение.

Элемент типа прогрессии имеет вид

A **step** D **until** B,

*) Английское слово *do*, которое мы для удобства перевели здесь словом «*цикл*», буквально означает «*делай*».

где A, D, B — арифметические выражения, а **step** (*шаг*) и **until** (*до*) — символы алгола.

Элемент типа пересчета имеет вид A **while** V, где A — арифметическое выражение, V — булевское выражение, а **while** (*пока*) — символ алгола.

Приведем примеры записи операторов цикла

for i: $=p$, $k+1$, $l-2$ **do** $a[i]$: $=b[i]\uparrow p$;
for j: $=1$ **step** n **until** 50 **do** $x[j]$: $=y[j]$;
for x: $=k+x$ **while** $x<c$ **do** s: $=s+x\uparrow 3$;

В нижеследующих примерах список элементов цикла состоит из элементов различного типа:

for i: $\underbrace{=p}_{E_1}$, $\underbrace{k+1}_{E_2}$, $\underbrace{0\ \textbf{step}\ 1\ \textbf{until}\ 10}_{E_3}$, $\underbrace{r\uparrow 2}_{E_4}$, $\underbrace{12\ \textbf{step}\ 1\ \textbf{until}\ 20}_{E_5}$ **do**
$x[i]:=a\times x[i]\times \mathrm{sqrt}\,(x[i])$;

for l: $=\underbrace{z\ \textbf{step}\ 1\ \textbf{until}\ n}$, $\underbrace{a}$, $\underbrace{a+z}$, $\underbrace{l+p\ \textbf{while}\ p<a\uparrow 2}$ **do**
p: $=p+l$;

Разберем порядок выполнения оператора цикла в зависимости от типа элемента цикла. Для каждого *элемента арифметического типа* происходит присваивание параметру цикла значения арифметического выражения и после каждого из этих присвоений выполняется оператор S.

Рассмотрим элементарный пример.

Пример 1.53.

Пусть нам нужно вычислить произведение

$$s=f(-3.4)\times f(-2)\times f(0)\times f(4.9),$$

где

$$f(x)=\frac{x^4}{4}-\frac{a^2}{2}\sin(3x-1).$$

Здесь следует сначала присвоить s значение 1, а затем выполнить оператор присваивания

$$s:=s\times(x\uparrow 4/4-a\uparrow 2/2\times\sin(3\times x-1))$$

четыре раза для значений x, равных —3.4, —2, 0, 4.9. Алгольная программа имеет вид:

s: $=1$; **for** x: $=-3.4$, -2, 0, 4.9 **do**
s: $=s\times(x\uparrow 4/4-a\uparrow 2/2\times\sin(3\times x-1))$;

Значительно бо́льшие возможности для составления циклических программ дает использование *элемента цикла типа арифметической прогрессии.*

В случае, если список цикла состоит из одного такого элемента, оператор цикла имеет вид:

$$\textbf{for } k\colon = A \textbf{ step } D \textbf{ until } B \textbf{ do } S;$$

Динамику выполнения оператора S в этом случае можно описать следующим образом.

Параметру цикла присваивается значение A и выполняется оператор S. Затем значение параметра увеличивается на D и выполняется S. Затем снова значение параметра увеличивается на D и выполняется S, и так до тех пор, пока значение параметра не выйдет за границу значения B.

Пример 2.53. Пусть нам нужно получить компоненты вектора $\boldsymbol{c}$, равные сумме двух n-мерных векторов $\boldsymbol{a}$ и $\boldsymbol{b}$. Мы имеем

$$c_1 = a_1 + b_1,\ c_2 = a_2 + b_2,\ \ldots,\ c_n = a_n + b_n.$$

Пусть $n = 8$. Тогда вектор $\boldsymbol{c}$ находится при помощи следующего оператора цикла:

$$\textbf{for } i\colon = 1 \textbf{ step } 1 \textbf{ until } 8 \textbf{ do } c[i]\colon = a[i] + b[i];$$

Здесь оператор присваивания

$$c[i]\colon = a[i] + b[i]$$

выполняется 8 раз для значений i, равных 1, 2, 3, 4, 5, 6, 7, 8.

Значения параметра цикла, которые он принимает в процессе своего изменения, могут никогда не совпадать с указанным в программе последним значением. Например, в цикле

$$\textbf{for } i\colon = 1 \textbf{ step } 3 \textbf{ until } 9 \textbf{ do } S;$$

переменная i принимает значения 1, 4, 7, 10 и никогда не равна 9. Поэтому необходимо уточнить, как происходит проверка окончания цикла.

Для цикла

$$\textbf{for } K\colon = A \textbf{ step } D \textbf{ until } B \textbf{ do } S;$$

каждый раз проверяется условие sign $(B - K) \times D \geqslant 0$ и оператор выполняется лишь тогда, когда указанное выражение имеет значение **true**. В частности, в приведенном выше примере оператор S будет выполняться три раза, при $i = 1, 4, 7$.

Таким образом, последовательность операторов, соответствующая действию элемента типа арифметической прогрессии, **for** $K := A$ **step** D **until** B **do** S будет следующей:

$K := A$;
L: **if** sign $(B - K) \times D < 0$ **then go to** *выход*;
S;
$K := K + D$;
go to L;
выход:

здесь *выход* — это метка оператора, следующего за оператором S.

Скажем, в примере

for $i := 4$ **step** -1 **until** 5 **do** $a[i] := x \times b[i]$;

оператор $a[i] := x \times b[i]$ не выполнится ни разу, так как

$$\text{sign}\,(5 - 4) \times (-1) < 0.$$

Еще один пример,

for $i := 6$ **step** 1 **until** 6 **do** $a[i] := b$;

Здесь оператор $a[i] := b$ выполнится один раз для $i := 6$, так как sign $(6-6) \times 1 = 0$, а далее sign $(6-7) \times 1 < 0$ и оператор не выполнится.

Следует предостеречь читателя от впечатления, что при выполнении цикла с элементом типа прогрессии параметр цикла непременно принимает значения, образующие арифметическую прогрессию. Так будет лишь в том случае, когда шаг цикла является п о с т о я н н ы м. Однако на самом деле это совсем не обязательно: шаг цикла может меняться при каждом выполнении оператора S.

Например, оператор

$s := 0$; **for** $i := 1$ **step** i **until** 16 **do** $s := s + i$;

сосчитает сумму $s = 1 + 2 + 4 + 8 + 16 = 31$.

В примере

$x := 1$; **for** $n := 1$ **step** x **until** 100 **do** $x := x \times n - x + n$;

параметр цикла будет принимать последовательно значения $n = 1$, 2, 5, 22 и после окончания цикла мы получим $x = 379$. Рекомендуем читателю самостоятельно убедиться в правильности этих утверждений.

Для *элемента типа пересчета* **for** $k := A$ **while** V **do** S; выполнение цикла состоит в следующем: параметру цикла k присваивается значение выражения A и выполняется оператор S. Затем снова параметру присваивается значение A и выполняется оператор S. И так до тех пор, пока булевское выражение V имеет значение **true**. Когда же значение булевского выражения станет **false**, то оператор S не выполнится, а выполнится оператор, следующий за ним. Выполнение оператора цикла с элементом типа пересчета эквивалентно выполнению следующей последовательности операторов:

L: k: $= A$;
if V **then begin** S; **go to** L **end**;

Из этого определения видно, что оператор S может не выполниться ни разу, если V с самого начала имеет значение **false**. Отметим, что значение выражения A может изменяться оператором S, и тогда параметр цикла k будет каждый раз принимать новые значения.

Элемент типа пересчета иногда называют элементом итеративного типа, так как оператор цикла с таким элементом очень удобен при программировании итерационных процессов.

Пример 3.53. Найти с точностью 0,001 кубический корень y из числа x, зная его приближенное значение y_0.

В § 14 для решения этой задачи была составлена программа в содержательных обозначениях, основанная на формуле

$$y_{n+1} = \frac{1}{3}\frac{x}{y_n^2} + \frac{2}{3}y_n$$

$$(n = 0,\ 1,\ 2,\ \ldots).$$

Составим алгольную программу для нахождения последовательных приближений y_1, y_2, ... по этой формуле, используя в качестве условия окончания счета соотношение

$$|y_{n+1} - y_n| < 0{,}0001.$$

Алгольная программа имеет вид:

$y:=y0$; **for** $u:=x/(3\times y\uparrow 2)+2\times y/3$
while $abs\,(u-y)\geqslant_{10}-4$ **do** $y:=u$;

Здесь на каждом шаге цикла вычисляется новое приближение u для $\sqrt[3]{x}$ при помощи оператора, следующего после **for**, затем, если не достигнута нужная точность (логическое условие после **while**), искомой величине y присваивается значение u, если же логическое условие выполняется, то происходит выход из цикла.

При использовании оператора цикла надо иметь в виду следующие особенности:

1) В цикле выполняется лишь один оператор, стоящий после символа **do**. Если требуется выполнить в цикле несколько операторов, то из них следует образовать один составной оператор с помощью операторных скобок **begin** и **end**.

Пример 4.53 Найти сумму бесконечного ряда

$$z=1-\frac{1}{1\cdot 2\cdot 3}+\frac{1}{1\cdot 2\cdot 3\cdot 4\cdot 5}-\ldots+\frac{(-1)^k}{1\cdot 2\cdot 3\ldots 2k\,(2k+1)}+\ldots$$

с точностью ε.

Целесообразно вычислять по рекуррентным формулам

$$z_1=1,\quad z_{k+1}=z_k+u_{k+1},$$
$$u_1=1,\quad u_{k+1}=-\frac{u_k}{2k\,(2k+1)}$$
$$(k=1,\ 2,\ 3,\ \ldots),$$

причем вычисления следует окончить при выполнении условия *)

$$|u_{k+1}|<\varepsilon.$$

Алгольная программа для вычисления z записывается в виде двух операторов присваивания и оператора цикла, в котором содержится составной оператор:

$z:=u:=1$; $k:=0$; **for** $k:=k+1$ **while** $abs\,(u)\geqslant eps$ **do**
begin $u:=-u/(2\times k\times(2\times k+1))$; $z:=z+u$ **end**;

*) Как известно, ошибка от замены суммы этого ряда частичной суммой не превосходит (по модулю) величины первого отбрасываемого члена.

2) Оператор S, выполняющийся в цикле, может, в свою очередь, быть оператором цикла. Таким образом, мы получаем возможность записывать на алголе программы, содержащие многократные циклы.

Пример 5.53. Напишем программу нахождения произведения C двух прямоугольных матриц A и B. Пусть матрица A имеет размеры $l \times m$, а матрица B размеры $m \times n$. Тогда элементы матрицы C определяются формулами

$$c_{ik} = \sum_{j=1}^{m} a_{ij} b_{jk}$$
$$(i = 1, 2, \ldots, l;\ k = 1, 2, \ldots, n).$$

Программа вычисления по этим формулам имеет вид трехкратного цикла:

```
стр: for i: =1 step 1 until l do
     столб: for k: =1 step 1 until n do
            begin c[i, k]: =0;
            элем: for j: =1 step 1 until m do
                  c[i, k]: =c[i, k]+a[i, j]×b[j, k]
            end;
```

Здесь вычисление элемента c_{ik} производится в составном операторе, состоящем из оператора присваивания $c\,[i, k]:=0$ и оператора цикла с меткой *элем*, «накапливающего» скалярное произведение соответствующей строки и столбца матрицы. Внешний оператор цикла с меткой *стр* организует цикл по строкам (параметр цикла i) и в качестве своего оператора содержит оператор цикла с меткой *столб*, организующий цикл по столбцам; этот последний оператор содержит описанный ранее составной оператор с меткой *элем*, вычисляющий отдельный элемент матрицы C.

3) К оператору S, выполняющемуся в цикле, нельзя обратиться из другого оператора, не находящегося в этом же цикле. Другими словами, в цикл можно войти только через его начало.

Например, неверна программа

```
L: for j: =0 step 1 until 10 do
   begin M: B[j]: =B[j]+k; if a<10 then
            begin k: =k+1; a: =a+k; go to M end;
   end; go to M;
```

Здесь в цикле выполняется составной оператор. Внутри этого оператора переход к метке M разрешается. Оператором **go to** M, находящимся вне составного оператора, к оператору $B[j]: = B[j] + k$ обратиться нельзя. В цикл можно войти только через метку L.

Для приведенного примера заметим, что при выполнении условий $a \geqslant 10$ и $j < 10$ мы переходим к новому шагу цикла, так как оператором, выполняющимся в цикле, является максимальный составной оператор — от первого **begin** до последнего **end**.

4) После выполнения оператора цикла параметр цикла имеет неопределенное значение.

Например, в программе

```
for i: =1 step 1 until N do
    a[i]: =sin(x[i]); b[i]: =0;
```

оператор $b[i]: = 0$, стоящий после цикла, не сможет выполниться, так как после окончания цикла параметр i не имеет предполагавшегося значения $N + 1$, а является неопределенным.

Если же мы выходим из оператора цикла не через его конец, а с помощью оператора перехода, то параметр цикла сохраняет свое значение.

Например,

```
for x: =A step 2 until B do
begin a[x]: =a[x]+T; if a[x]=TK then go to M end;
                M: Z: =x+p;
```

Когда выражение $a[x] = TK$ примет значение **true**, то произойдет выход из цикла, независимо от выполнения условия окончания цикла, и значение x параметра цикла сохранится.

5) Оператор цикла не относится к классу безусловных операторов и поэтому его нельзя писать в условном операторе после **then**.

Пример 6.53. Пусть нужно вычислить величину F, определяемую формулами:

$$F=\begin{cases}\sum\limits_{k=1}^{15} u_k(x), & \text{если } x\geqslant 0,\\ 0, & \text{если } x<0,\end{cases}$$

причем

$$u_k(x)=\begin{cases}\sin(kx), & \text{если } kx\geqslant 1,\\ \cos(kx), & \text{если } kx<1.\end{cases}$$

Неверно было бы написать программу в виде

```
F: =0; u1: if  x ≥0 then
       u2: for k: =1 step 1 until 15 do
           if  k × x≥1 then F: =F+sin (k × x)
           else F: =F+cos (k×x);
```

Действительно, такая программа неверна, так как из этой записи невозможно определить, к какому из двух условных операторов с метками $u1$ и $u2$ относится оператор, следующий за **else**. Правильную программу можно получить, если воспользоваться операторными скобками **begin** и **end**, так как в этом случае оператор цикла становится составным, а следовательно, и безусловным *).

```
F: =0; if x≥0 then begin for k: =1 step 1 until 15 do
                         if k × x≥1 then F: =F+sin (k × x)
                           else F: =F+cos (k × x)
                   end;
```

Теперь ясно, что оператор, следующий за **else**, относится к внутреннему условному оператору.

§ 54. Описание переменных и массивов. Блоки

Мы уже подробно останавливались на роли описаний при изучении автокода. Там с помощью описаний транслятор получал информацию, необходимую для распределения памяти. В алголе роль описаний еще более существенна. Здесь необходимо задать транслятору не только информацию для распределения памяти, но и указать характер

*) Заметим, что если в этом примере поставить **end** перед **else**, то получится неверная алгольная программа, ибо в последнем операторе присваивания значение k станет неопределенным.

и свойства описываемых величин. По этой информации транслятор проверяет правильность использования величин в разных алгольных конструкциях и принимает соответствующие решения.

В алголе используются числа, переменные, метки, стандартные функции. Числа бывают целые и дробные. Правила записи чисел дают возможность по их изображению понять, с каким числом мы имеем дело. В связи с этим нет необходимости их отдельно описывать. Переменные величины делятся на простые переменные и переменные с индексами, которые мы записываем в виде массивов. Эти величины могут быть действительными, целыми или булевскими. Так как любая из этих величин обозначается произвольным идентификатором, то в программе, где они используются, необходимо точно указать типы переменных, которые обозначаются каждым идентификатором.

В описаниях для указания *типов переменных* используются алгольные знаки (указатели типа)

real — для обозначения действительных переменных или массивов,
integer — для обозначения целых,
boolean — для обозначения булевских.

Идентификаторы простых переменных, относящиеся к одному типу, описываются следующим образом:

⟨указатель типа⟩ ⟨идентификатор⟩,

⟨идентификатор⟩, ... , ⟨идентификатор⟩;

Например,

real *x*, *y*, *z*; **integer** *i*, *k*, *p*;
boolean *B*, *D*, *A*;

Описание одного типа идентификаторов от другого типа отделяется точкой с запятой. Знаки указателей типа можно повторять для каждой переменной заново и в произвольном порядке. В частности, для приведенного выше примера описание можно приводить так:

real *x*; **integer** *i*, *k*; **real** *y*; **boolean** *B*;
integer *p*; **boolean** *D*, *A*, **real** *z*;

Для описания массива вводится алгольный знак **array** (*массив*). Описание массива имеет вид:

⟨указатель типа⟩ **array** ⟨идентификатор массива⟩ $[P_1 : R_1, \ldots, P_n : R_n]$, где $P_i : R_i$ — граничная пара, указывающая нижнюю и верхнюю границы индексов. Например,

real array *TABLE* 1 [0 : 5, 1 : 10, 1 : 10];
MATRIXAO [1 : 8, 1 : 8], *сумма* [1 : 10, 0 : 10];
integer array *I* [1 : 4], *Z* [1 : 5, 1 : 5], *K* [0 : 20];
boolean array *KOD* [— 1 : 4, — 2 : 0];

В граничных парах значения верхних границ всегда должны быть больше значений нижних, в противном случае описание считается неправильным. Указатель **real** перед **array** писать не обязательно.

Если несколько массивов имеют одинаковые граничные пары, то можно эти граничные пары писать один раз после перечисления идентификаторов массивов. Так, например, три прямоугольные матрицы A, B, C размера 15×20 с действительными элементами можно описать следующим образом

array *A*, *B*, *C* [1 : 15, 1 : 20];

что эквивалентно записи

array *A* [1 : 15, 1 : 20], *B* [1 : 15, 1 : 20], *C* [1 : 15, 1 : 20];

Для описания метки никаких дополнительных средств не требуется: под описанием метки понимается ее появление перед оператором, который она метит.

То же относится и к стандартным функциям: идентификаторы этих функций специально описывать не надо. Если будет написано sin (x), то это обозначение стандартной функции sin от аргумента x, если же будет написано sin x, то это идентификатор, ничего общего со стандартной функцией не имеющий.

Наряду со стандартными функциями введем еще два стандартных оператора ввода-вывода.

Первый из них — read (x, y, z, ..., w) — стандартный оператор ввода. Этот оператор производит следующие действия: вводит с читающего устройства числовые значения и присваивает их переменным, перечисленным в скобках, причем переменные могут быть: простые, с индексами, или массивы и относиться к типу целых или действительных.

Второй оператор — print (*a*, *b*, *c*, *p*) — стандартный оператор вывода *). В результате работы этого оператора на печатающем устройстве машины будут отпечатаны значения переменных, перечисленных в скобках. Для окончания вычислений вводится стандартный оператор stop.

Описания всегда следуют в начале составного оператора (между **begin** и первым оператором составного) и содержат описания идентификаторов, используемых в операторах составного. Порядок следования описаний может быть произвольным.

Составной оператор, в начале которого следуют описания идентификаторов, называется *блоком*.

Пример 1.54. Напишем программу для решения последовательности линейных систем двух уравнений с одними и теми же левыми, но различными правыми частями:

$$\begin{aligned} a_1x_1 + b_1x_2 &= c_i, \\ a_2x_1 + b_2x_2 &= d_i \end{aligned} \qquad (i = 1, 2, \ldots, 10).$$

Предполагается, что $\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \neq 0$. Условимся, что коэффициенты должны вводиться с читающего устройства, а каждое решение x, y должно печататься. Программу можно записать в виде блока так:

```
begin real a1, b1, a2, b2, D;
      array c[1 : 10], d[1 : 10], x1[1 : 10], x2[1 : 10];
      integer i;
      read (a1, b1, a2, b2, c, d);
      D: =a1 × b2 — a2 × b1;
      for i: =1 step 1 until 10 do
      begin
      x1 [i]: =(c[i]×b2 — d[i]×b1)/D;
      x2[i]: =(a1×d[i] — a2×c[i])/D;
            print (x1[i], x2[i])
      end;
      stop;
end;
```

*) По-английски *read* означает «читай», а *print* — «печатай».

Мы уже говорили, что описания дают транслятору информацию о свойствах величин, которые будут использованы в операторах блока. Кроме этого, *описания управляют распределением памяти*, которое должен произвести транслятор для использования значений описываемых величин. Суть этого управления состоит в следующем.

Каждому идентификатору переменной должна быть отведена ячейка памяти для хранения значения этой переменной. Каждому массиву отводится последовательность ячеек памяти для размещения значений элементов массива.

От начала блока и до его конца распределенная по описанию память остается зарезервированной для данного блока, и каждый идентификатор может быть использован только в соответствии со своим описанием. После выхода из блока вся память, которая была отведена для описанных в нем величин, освобождается, значения величин исчезают, описания идентификаторов перестают действовать. Выход из блока может произойти либо через его конец, либо с помощью оператора перехода.

Представим себе, что программа состоит из двух (или более) непересекающихся блоков, и переход от первого блока ко второму происходит через **end**. В этом случае схему программы можно представить, например, следующим образом:

```
begin real a, b, c; integer k, l;
      array M[1:5], N[0:10];
      S1
end;
begin array k[0:5], M[1:5]; integer a, i, j;
      real b, l;
      S2
end;
```

Рассмотрим, как используются переменные и массивы при работе этой программы. При входе в первый блок будет зарезервирована память: для трех действительных переменных a, b, c, для двух целых переменных k, l, для пяти элементов массива M и для одиннадцати элементов массива N. Внутри первого блока все описанные идентифика-

торы могут быть использованы только в соответствии со своими описаниями. Если в каком-то из операторов блока обнаружится несоответствие описанию, то транслятор прекратит свою работу и просигнализирует об этом (напечатает на печатающем устройстве машины тип ошибки). После выполнения всех операторов первого блока (последний оператор заканчивается **end**) мы входим во второй блок. Поскольку после выхода из первого блока описания использованных там идентификаторов теряют свою силу, то мы можем во втором блоке использовать те же идентификаторы для обозначения совсем иных величин. Так, идентификатор k в первом блоке был введен для обозначения целой переменной, а во втором мы используем его для обозначения действительного массива. Идентификатор массива M, несмотря на совпадение границ в первом и втором блоках, использован для обозначения элементов, быть может, обладающих разными свойствами. Один и тот же идентификатор b использован в первом и во втором блоках для обозначения различных действительных переменных и т. д. (просмотрите и сопоставьте все описания).

Другими словами, второй блок ввел свою систему обозначений, независимую от системы обозначений первого блока. Во втором блоке операторы могут содержать только те идентификаторы, которые описаны в его начале.

Что же произойдет, если во втором блоке в каком-то из его операторов встретится идентификатор c? Транслятор это воспримет как ошибку. Действительно, после окончания работы блока значения величин, которые были в нем описаны, исчезают. Значит, после работы первого блока значение переменной c исчезает и, следовательно, не сможет быть использовано вне этого блока.

Посмотрим теперь, как будет распределена память для описанных величин второго блока. После окончания работы первого блока вся резервированная там память освобождается, так что во втором блоке эту же память можно использовать для его величин. Так, ячейки, в первом блоке использованные для переменных a, b, c, k, l, могут быть здесь использованы для значений элементов массива k и т. д. Итак, мы видим, что блок дает возможность вводить произвольную систему обозначений величин и экономно распределять память.

Все величины, которые описаны в начале блока, называются локальными величинами.

Метка считается описанной в том месте, где она стоит перед оператором. Поэтому считается, что метки тоже локализуются внутри того блока, в котором они помещены. Это означает, что внутри блока не должно быть совпадающих меток, и из одного блока нельзя обратиться к метке внутри другого блока (так как во втором блоке описание метки теряет свою силу).

Например, в программе

```
begin real a, b;
  . . .
M : S;
  . . .
end;
begin real M, b;
  . . .
go to M;
  . . .
end;
```

оператор **go to** *M* во втором блоке ошибочен. Он не приведет к метке *M* первого блока (во втором блоке идентификатор *M* использован для обозначения простой действительной переменной).

Рассмотрим еще один пример,

```
begin integer i, k, l; real A;
  . . .
go to M;
  . . .
M : S;
end;
begin boolean B, D;
M :
  . .
go to M;
  . . .
end;
```

Здесь метка M встречается как в первом, так и во втором блоке. Оператор перехода в первом блоке ведет к оператору первого блока, помеченному меткой M, а оператор перехода во втором блоке ведет к оператору, помеченному меткой M во втором блоке. Как же в таком случае войти в блок? В соответствии с правилами алгола в блок можно войти только через начало (**begin**) *).

Блок в алголе, помимо тех функций, которые были рассмотрены, во всем остальном ничем не отличается от оператора и потому он может в себя включать другие операторы (а следовательно, и блоки) и сам может включаться в состав других операторов (или блоков). Как и любой оператор, он может быть помечен.

Рассмотрим следующую схему программы:

```
A { begin integer i, k;
    . . .
    B1 { begin integer k, r;
         S1
         end;
    . . .
    B2 { begin integer r, p;
         S2
         end;
    . . .
    end;
```

Эта программа состоит из трех блоков A, $B1$, $B2$. Блок A называется *внешним*, так как не содержится ни в каком другом блоке. Блок A непосредственно содержит блоки $B1$ и $B2$ (не существует блока, содержащегося в A и содержащего $B1$ или $B2$). Какова область действий описаний каждого из этих блоков?

Во внешнем блоке A, но вне его подблоков $B1$ и $B2$, действуют описания целых величин i и k, которые описаны в начале блока A.

*) Заметим, что внутрь составного оператора войти можно.

В подблоках $B1$ и $B2$ действуют описания идентификаторов, описанных в начале этих блоков. В блоке $B1$ локализуются k, r, в блоке $B2$ — r, p. Кроме того, в подблоках $B1$ и $B2$ действуют еще описания и тех идентификаторов внешнего блока A, которые не совпадают с идентификаторами, описанными в подблоках $B1$ и $B2$. Так, в блоке $B1$ действует описание идентификатора i блока A (но не k), в подблоке $B2$ действуют описания идентификаторов i и k блока A (но не r блока $B1$).

Распределение памяти и использование величин в процессе работы программы будет следующим. Во внешнем блоке A отводятся ячейки под описанные в его начале величины i, k и эти ячейки считаются занятыми до конца блока A (до последнего значения **end**). В подблоке $B1$ для величин k и r отводятся новые ячейки, так что в нем можно пользоваться величинами i, k, r, но k в этом блоке отлично от k в наружном. Во время работы блока $B1$ последнее недоступно; значение k из наружного блока хранится, но не может быть использовано, тогда как величина i может не только использоваться, но и изменяться.

После окончания работы $B1$ используемые в нем величины k и r исчезают и занятые ими ячейки освобождаются. Если здесь или в блоке $B2$ встретится величина k, то это не будет k из блока $B1$, а старое значение k из наружного блока A. При входе в блок $B2$ отводятся новые ячейки для переменных r и p, но значения i, k из блока A и отведенные им ячейки сохраняются и могут быть здесь использованы. При этом для величины r отводится, вообще говоря, уже не та ячейка, которая отводилась для r в блоке $B1$.

По окончании работы блока $B2$ величины r и p исчезают и остаются лишь значения i, k блока A. По окончании блока A и эти значения исчезают и вся память освобождается.

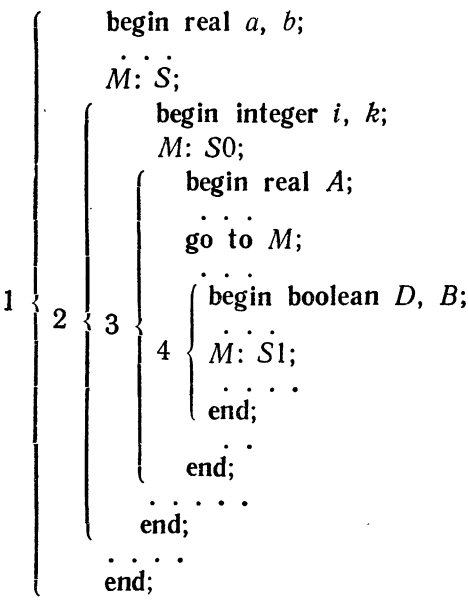
Величины, которые описаны во внешнем блоке и описание которых имеет силу в его подблоках, называются *глобальными*.

Таким образом, в каждом блоке имеет силу описание локальной величины, если же какой-либо идентификатор не является локальным (в начале блока нет его описания),

то в силу вступает описание глобальной величины (т. е. описание из внешнего блока).

Вернемся еще раз к вопросу о метках.

Рассмотрим следующую схему программы:



Здесь внешним является блок 1, его подблоком является блок 2. Подблоком блока 2 является блок 3, который является внешним для блока 4.

Метка M встречается в блоках 1, 2, 4. В блоке 3 есть оператор перехода к M. К какой из этих меток произойдет переход? Метка M в блоке 4 недоступна оператору блока 3, так как она здесь локальна. Описанная в блоке 1 метка M теряет свое описание в блоке 2, так как в блоке 2 есть свое описание метки M. В блоке 3 описание метки M отсутствует. Следовательно, здесь имеет силу описание минимального охватывающего блока. Но минимальным охватывающим блоком является блок 2. Следовательно, оператор **go to** M передаст управление оператору с меткой M в блоке 2, т. е. оператору $S0$.

Блочная структура предоставляет большие возможности для компоновки программы из отдельных блоков, написанных разными людьми. При написании различных блоков одной программы достаточно договориться только об обозначениях величин, которые будут использованы в обоих блоках одинаково. В остальном, в каждом блоке можно вводить свои наименования, которые могут даже совпадать; поскольку они локализуются в каждом блоке, то и использованы будут по-разному. Общие же для обоих блоков величины будут описаны во внешнем блоке и станут глобальными в обоих его подблоках.

Теперь можно дать определение программы, которое до сих пор мы понимали интуитивно: *Программа — это составной оператор или блок, не содержащийся ни в каком операторе и не обращающийся к операторам, не содержащимся в нем.*

При описании массивов в алголе разрешается в качестве границ индексов массива писать любые арифметические выражения *). Так, например, можно писать

$$\textbf{array}\ M[i\colon k+p,\ j+1\colon r\uparrow 2].$$

Такие массивы называются *динамическими*, поскольку их объемы изменяются с изменением величин, входящих в состав выражений для границ. При входе в блок для распределения памяти под динамические массивы значения переменных, участвующих в выражениях для границ, берутся в данный момент. Отсюда следует, что к моменту входа в блок все эти переменные должны были уже получить значения и эти значения должны сохраниться к этому моменту. В самом внешнем блоке описать динамический массив нельзя, так как предварительно должны быть получены значения переменных в выражениях для границ. Во внутренних же блоках идентификаторы этих переменных могут выступать как глобальные и их значения доступны для вычисления значений выражений границ.

---

*) При этом, как уже говорилось, если выражения для границ оказываются дробными, то они округляются до ближайшего целого.

Пример 2.54. Дана программа:

```
begin integer i, j, k, l;
      array M[1: 10], p[0: 4, 0: 4];
      read (i, j, k, l);
  W: begin array T[i+2: l↑2], T1[j: k×l+i];
      real k, p;
      . . . .
      i: =k×p; l: =2;
      . . . . . .
      end;
      go to W;
      . . . . .
end;
```

При первом входе в блок W границы для массивов T и $T1$ будут вычислены для значений i, l, k, j, введенных с читающего устройства. При втором обращении через оператор **go to** W эти границы вычисляются по значениям, которые были получены в блоке W для i и l, а для j и k остались прежними (величина k является глобальной и не зависит от изменения локальной величины k в блоке W).

Выше говорилось о том, что после выхода из блока все значения величин, локализованных в этом блоке, исчезают. Отсюда следует, что при возвращении в блок эти величины должны вновь приобрести значения (возможно, новые) при выполнении операторов данного блока. Это могут быть операторы ввода или присваивания. Но как быть в том случае, если значения некоторых локальных величин желательно запомнить, с тем чтобы при повторном входе в блок использовать их снова для дальнейших вычислений. Алгол дает такую возможность. Идентификаторы такого рода называются *собственными* и в начале описания такого идентификатора пишут алгольное слово **own** (*собственный*).

Например,

```
begin integer r;
      for r: =1 step 1 until 10 do
 begin own real S;
      if r=1 then S: =0; S: =S↑2+1/i;
      if r=10 then print (S)
 end;
end;
```

Мы видим, что оператором, выполняющимся в цикле, является блок. По окончании работы этого оператора (после каждого шага цикла) значения S не теряются, а используются при каждом новом выполнении блока.

Прежде чем переходить к рассмотрению полных алгольных программ, познакомимся еще с одним важным понятием алгола — понятием *переключателя*. До сих пор, имея дело с оператором перехода, общий вид которого, как говорилось в § 52,

go to ⟨именующее выражение⟩,

мы под именующим выражением понимали только м е т к у, которой помечен тот или иной оператор программы. В общем случае именующее выражение составляется не только из меток, но также и из переключателей.

Оператор перехода с переключателем имеет вид

go to $P[A]$;

где P — идентификатор данного переключателя, A — арифметическое выражение. Переключателю P соответствует его описание

switch $P := M, N, Q, \ldots, R$;

где **switch** (*переключатель*) — алгольный символ, P — идентификатор переключателя, о котором идет речь, а M, N, Q, ..., R — *переключательный список*, состоящий из именующих выражений (т. е. меток или других переключателей), разделенных запятыми.

Работа оператора перехода с переключателем

go to $P[A]$;

происходит следующим образом:

1) Вычисляется значение арифметического выражения A, которое затем округляется до ближайшего целого.

2) В переключательном списке данного переключателя, приведенном в его описании, находится соответствующее именующее выражение. Например, если округленное значение A есть 3, то в приведенном списке таким именующим выражением будет Q.

3) Выполняется оператор

go to Q,

Если значение арифметического выражения окажется неопределенным или меньшим половины или, наоборот, превзойдет число членов переключательного списка, то соответствующий оператор перехода считается неопределенным и пропускается.

Использование переключателя связано с передачей управления различным блокам, в зависимости от выполнения тех или иных условий, если такая передача производится не непосредственно после проверки условия, а после выполнения еще некоторых операторов.

Возвратимся, скажем, к примеру 2.35, в котором программа обращалась к блоку «По конусу» или «По цилиндру», в зависимости от знака разности Δv. Вычисление Δv было вынесено в собирающую. Если же писать эту программу на алголе, то здесь удобно воспользоваться переключателем и написать ее так:

```
real R, r, l, v, h, vk, lambda; integer n;
switch P: =по конусу, по цилиндру; read (R, r, l, v);
vk: =1/3×3.14159×(R↑2+r↑2+R×r); if v<vk then
                        n: =1 else n: =2;
go to P[n];
по конусу: begin h: =l×R/(R−r);
lambda: =(3×(v+vk)×h↑2)/(3.14159×r↑2)−h;
go to M;
          end;
по цилиндру:    begin
  lambda: =(v−vk)/(3.14159×R↑2)+l;
                  end;
  M: print (lambda); stop;
```

Этот пример был подробно рассмотрен в § 35 и поэтому написанная программа достаточно понятна и без дополнительных комментариев. Мы изменили только счет объема усеченного конуса. Однако в ряде случаев программа, написанная на алголе, может оказаться сложной для прочтения без дополнительных словесных комментариев.

Правила алгола допускают непосредственное включение комментариев в текст программы. Для этого перед коммен-

тариями ставится алгольный символ **comment** (*комментарий*) и весь текст, заключенный между этим символом и разделительным знаком (;), при работе транслятора пропускается. Обычно символ **comment** ставится после окончания какого-либо блока, т. е. после знака (;). Но правилами допускается и иное размещение комментариев. Имеются две следующие возможности.

Символ **comment** и комментарий, следующий за ним, можно помещать непосредственно в начале блока после символа **begin**. В этом случае текст комментария может состоять из любых знаков и символов алгола, кроме разделителя (;), который воспринимается как конец комментария. Разрешается также помещать символ **comment** с последующим текстом комментария после символа **end**, однако в этом случае в тексте комментария нельзя употреблять не только точки с запятой, но и символы **end** или **else**, которые также могут быть восприняты здесь как конец комментария. Комментарий после символа **end** можно помещать и непосредственно, опуская символ **comment**, но с соблюдением приведенных выше требований.

Познакомимся с применением перечисленных правил на конкретном примере.

Пример 3.54. Дана функция $y=x+e^{\frac{1}{x}}$, рассматриваемая для $x>0$. Непосредственно очевидно, что при $x\to+0$ и при $x\to+\infty$ функция y неограниченно возрастает. Отсюда следует, что она имеет минимум (строго говоря, хотя бы один, но нетрудно показать, что точно один). Напишем программу для нахождения абсциссы точки минимума с точностью до 10^{-6}.

Если функция имеет одну точку минимума, то, начиная от некоторого достаточно малого x она будет убывать до точки минимума, а затем начнет возрастать. Идя по x с постоянным шагом h до тех пор, пока функция не получит в данной точке большее значение, чем в предыдущей, мы найдем точку минимума с точностью до h. Поэтому естественно включить описанную часть программы в цикл типа пересчета, на каждом шаге которого минимум будет находиться указанным способом с точностью до h, а при переходе к следующему шагу h будет изменяться по величине и по направлению.

Указанную программу, вместе со всеми требуемыми пояснениями, можно написать следующим образом:
begin real x, $x1$, y, $y1$, h; **comment** x, $x1$ служат аргументами соответственно для внутреннего и внешнего цикла, y, $y1$ будут содержать предыдущее и последующее значения функции, h — шаг по x;

$$x1:=0.5; \quad y1:=100; \quad h:=0.5;$$

comment выбор начальных данных произведен с помощью графика;

```
begin comment начало наружного цикла;
      for h:=-h/10 while h>10-6 do
    begin comment начало внутреннего цикла; x:=x1;
                                счет: y:=y1;
  y1:=x+exp(1/x); if y1<y then
                  begin x:=x+h; go to счет;
                  end;
        x1:=x
    end конец внутреннего цикла
end конец наружного цикла; print (x); stop;
end;
```

§ 55. Процедуры

Использование ранее написанных программ существенно облегчает программирование. По этой причине большую роль играет библиотека стандартных подпрограмм, о которой мы достаточно подробно говорили в §§ 36—38. Однажды написанная, стандартная программа оказывается независимой от конкретных значений входных данных (даже таких, как порядок системы уравнений, не говоря уже о числовых значениях коэффициентов). При этом способ обращения к такой программе обязательно стандартизируется.

Алгольные программы, блоки, составные или простые операторы, играющие роль подпрограмм, называются в алголе *процедурами*. Процедура должна иметь наименование, для указания которого используется идентификатор.

Каждая процедура характеризуется совокупностью *формальных параметров*, для обозначения которых также применяются идентификаторы.

Текст вида

procedure ⟨идентификатор⟩ $(x_1, x_2, \ldots, x_n)$;

где **procedure** (*процедура*) — алгольный символ, а $x_1, \ldots, x_n$ — обозначения формальных параметров, называется *заголовком процедуры*. Оператор, блок или ссставной оператор Q, следующий непосредственно за заголовком процедуры, называется *телом процедуры*. Тело процедуры представляет собою описание вычислительного процесса над формальными параметрами. Примеры заголовков процедур

procedure *Корень* (*x*, *y*, *epsilon*);

procedure *Интеграл* (*F*, *H*, точность, *n*, *I*);

Заголовок процедуры и следующее за ним тело Q образуют описание процедуры. Как и любое другое описание, оно должно помещаться в начале блока, в котором эта процедура используется.

Пример 1.55. Блок, образующий тело процедуры нахождения модуля R n-мерного вектора $\boldsymbol{x}$ $(x_1, x_2, \ldots, x_n)$, может быть написан, например, так:

Модуль (*x*, *n*, *R*):

```
begin integer i; real s;
    s: =0;
    for i: =1 step 1 until n do
    s: =s+x[i]↑2;
    R: =sqrt (s);
end;
```

Описание процедуры само по себе никаких действий не вызывает и при выполнении программы пропускается целиком (вместе с телом процедуры Q). Для обращения к процедуре используется *оператор процедуры*, имеющий вид

⟨индентификатор процедуры⟩ $(a_1, a_2, \ldots, a_n)$;

где $a_1, a_2, \ldots, a_n$ — *фактические параметры* процедуры, число которых должно совпадать с числом формальных параметров, имеющихся в описании.

Выполнение оператора процедуры происходит следующим образом:

1. Отыскивается описание процедуры и формальные параметры в описании приводятся в соответствие с фактическими параметрами из оператора процедуры.

2. В теле процедуры Q формальные параметры заменяются соответствующими фактическими (получающийся после замены оператор $\tilde{Q}$ называют *модифицированным телом* процедуры).

3. Оператор $\tilde{Q}$ выполняется так, как будто он стоит на месте оператора процедуры. После него выполняется оператор, непосредственно следующий в программе за оператором процедуры.

Пример 2.55. Напишем программу для нахождения корней квадратного уравнения $x^2 + px + q = 0$.

```
begin real x, A, x1, x2, признак;
procedure Корень (p, q, u, v, B);
comment обозначения формальных параметров, p, q — коэффициенты,
        признак
    B = { 1, если корни действительные,
        { 0, если корни комплексные,
u, v — первый и второй корни, если они действительны; действитель-
      ная и мнимая части корней, если они комплексные;

begin real c; c: =(p/2) ↑ 2−q; v: =sqrt (abs (c));
    if c≥0 then begin B: =1; u: =(− p/2) −v; v: =(− p/2) +v;
                end
           else
                begin B: =0; u: =(− p/2)
                end

end     процедуры
comment до сих пор шли описания, только дальше начинаются опе-
        раторы программы, среди которых есть и обращение к на-
        писанной процедуре;

    read (A);
    . . . . .
    Корень (x ↑ 2, A, x1, x2, признак);
    print (x1, x2, признак)
    . . . . . .
end программы;
```

Еще раз обращаем внимание читателя на то, что сама программа начинается оператором read (A); весь текст до этого оператора относился к описаниям. При выполнении процедуры *Корень* по приведенному в описании алгоритму будет решаться квадратное уравнение с коэффициентами $p = x^2$ и $q = A$, признаку будет присвоено значение 1 при действительных корнях уравнения или 0 при комплексных, а сами значения корней или значения действительных и мнимых их частей будут присвоены переменным $x1$, $x2$. Иначе говоря, будет выполняться модифицированный оператор:

```
begin real c; c: =((x↑2)/2)↑2 — A; x2: =sqrt (abs (c));
    if c ≥ 0 then begin признак: =1; x1: =(— x↑2/2) — x2;
                                     x2: =(— x↑2/2) + x2;
                  end
             else begin признак: =0; x1: =(— x↑2/2)
                  end;
end;
```

Описанная работа с процедурами требует некоторых уточнений и дополнений.

Прежде всего, характер фактических параметров должен быть согласован с характером формальных таким образом, чтобы модифицированное тело процедуры Q не содержало неверных и бессмысленных операторов. Например, если тело процедуры с формальными параметрами x, y, z содержит оператор $z: = x + y$, то стоящий на третьем месте в описании оператора процедуры фактический параметр (который в модифицированном теле будет поставлен на место z) может быть лишь идентификатором переменной, так как в противном случае оператор присваивания не будет иметь смысла.

Чтобы избежать несоответствий, в описании процедуры приводится спецификация формальных параметров. Спецификация пишется между перечислением формальных параметров и телом процедуры и состоит в указании типа и класса объектов, которые могут быть использованы в качестве фактических параметров, заменяющих данные формальные. При этом используются, в основном, уже известные символы алгола:

integer — для спецификации формального параметра как целой арифметической переменной,
real — для действительной,
boolean — для булевской,
array — для массива,
procedure — для процедуры,

и только один новый символ

label (*метка*) — для спецификации формального параметра как метки.

Вместе со спецификациями описание процедуры будет иметь, например, следующий вид

procedure Переход (*m*, *n*, *x*, *y*, *B*, *D*, *L*); **real** *x*, *y*;
integer *m*, *n*; **array** *D*; **boolean** *B*, **label** *L*;

После приведенного текста должно идти тело процедуры.

Эти известные символы используются здесь, однако, по-иному, лишь сообщая информацию и не вызывая никаких действий. В частности, например, в спецификации массива у идентификатора массива отсутствует указание границ. Оно должно быть сделано при описании соответствующего фактического параметра по обычным правилам.

Фактические параметры процедуры, в зависимости от спецификации формальных параметров, должны удовлетворять следующим условиям:

1. Если формальный параметр специфицирован как целый, действительный или булевский, то в качестве фактического параметра может быть использовано любое арифметическое, соответственно булевское, выражение. При этом исключается случай, когда такое выражение может появиться в левой части оператора присваивания.

2. Если формальный параметр специфицирован как массив, то в качестве фактического параметра может быть взят только идентификатор массива.

3. Если формальный параметр специфицирован как метка или процедура, то роль фактического параметра может исполнять лишь идентификатор (соответствующего класса).

Некоторые формальные параметры процедуры могут быть объявлены *значениями*, что изменяет правила выполнения оператора процедуры. Чтобы пояснить, зачем и как это делается, обратимся к примеру.

Пример 3.55. Пусть требуется вычислить величину $S=\sum_{i=1}^{n}\frac{1}{x_i+y}$. Заголовок процедуры для вычисления S запишем в виде

procedure ПРИМ (X, Y, n, S); **real** Y, S, **integer** n, **array** X;

Тело Q, которое содержит цикл суммирования, можно записать так:

begin integer i; $S:=0$; **for** $i:=1$ **step** 1 **until** n **do**
$S:=S+1/(X[i]+Y)$;
end

Тогда при обращении к этой процедуре оператором

ПРИМ (A, $a\times(a-2)/k$, 12, B)

модифицированное тело процедуры $\tilde{Q}$ представится программой

begin integer i; $B:=0$; **for** $i:=1$ **step** 1 **until** 12 **do**
$B:=B+1/(A[i]+a\times(a-2)/k)$;
end

Как видно из этой программы, выражение $a\times(a-2)/k$ для второго фактического параметра будет вычисляться заново на каждом шаге цикла, что не вызывается необходимостью и замедляет счет. Объявление формального параметра Y процедуры ПРИМ *значением* позволяет избежать такого пересчета. В этом случае описание процедуры должно иметь следующий вид

procedure ПРИМ (X, Y, n, S); **value** Y; **real** Y, S,
integer n; **array** X;

Здесь **value** (*значение*) — алгольный символ.

Если какие-либо формальные параметры процедуры объявлены значениями, то оператор процедуры выполняется, грубо говоря, следующим образом: значения формальных параметров, объявленных **value**, вычисляются

один раз перед входом в тело процедуры, исходя из заданных значений фактических параметров.

Более точно выполнение оператора процедуры в этом случае описывается правилами:

1. Создается блок, охватывающий тело процедуры;

2. В начале этого блока описываются обозначения формальных параметров, попавших в список значений. Этим обозначениям присваиваются значения соответствующих фактических параметров. Такие присваивания выполняются один раз перед входом в тело.

3. При модификации тела формальные параметры, попавшие в список значений, сохраняют свои обозначения, т. е. не заменяются на фактические параметры.

Для нашего примера процедуры ПРИМ это означает, что модифицированное тело процедуры будет таким

begin integer i; $B:=0$; **for** i **step** 1 **until** 12 **do**
$B:=B+1/(A[i]+Y)$;

Перед этим циклом будет выполнено один раз присваивание $Y:=a\times(a—2)/k$.

Дальнейшие уточнения образования и выполнения модифицированного тела связаны с тем, что различные объекты программы могут оказаться обозначенными совпадающими идентификаторами и метками, как мы это видели в предыдущем параграфе. Поэтому необходимо согласовываться с описаниями, для чего следует дать точную классификацию объектов, встречающихся в теле процедуры.

Разобьем объекты тела процедуры на следующие классы:

1. *Локальные* объекты. Сюда относятся идентификаторы и метки, описанные в самом теле или помечающие операторы тела процедуры, которое всегда считается блоком.

2. *Формальные* объекты. Сюда относятся объекты тела, обозначенные идентификаторами формальных параметров процедуры, если только они не попали уже в число локальных.

3. Прочие объекты. Это те, которые не попали в две предыдущие группы. Поскольку они все равно должны быть ранее описаны, их описание должно находиться в объемлющем процедуру блоке, т. е. они являются здесь *глобальными* (см. § 54).

Пример 4.55. Рассмотрим классификацию параметров в процедуре

```
begin real a, b, c; integer i, k;
    procedure F(x, y); real x, y;
    begin real y, k;
        y: = x + a × k;
        k: = (i + 1) × b + x × y;
    end;
end;
```

Здесь в теле процедуры участвуют идентификаторы a, b, i, k, x, y. Из них идентификаторы y, k описаны в самом теле и потому являются *локальными* объектами. При этом не имеет никакого значения, что идентификатор y является формальным параметром, а k описан во внешнем блоке. Идентификатор x относится к числу формальных объектов, так как он является формальным параметром и не описан как локальный. Объекты a, b, i являются *прочими*, или *глобальными*, так как описаны во внешнем блоке, охватывающем процедуру.

При модификации тела процедуры для выполнения оператора транслятор руководствуется следующими правилами:

1. Формальные объекты тела заменяются фактическими параметрами. Объекты, вставленные на место формальных, понимаются в том смысле, который действует там, где расположен оператор процедуры.

2. Обозначения вставленного объекта могут совпасть с обозначениями локального объекта тела. Несмотря на это, их следует считать различными.

3. Прочие объекты понимаются в том смысле, который действует там, где расположено описание процедуры.

4. Обозначение прочего объекта может совпасть с обозначением объекта, действующего там, где расположен оператор процедуры. Однако если это различные объекты, то их следует считать различными, для чего необходимо временно сменить обозначение объекта, действующего в расположении оператора процедуры.

После этих преобразований модифицированное тело выполняется так, как если бы оно находилось на месте оператора процедуры.

В соответствии с приведенными правилами в примере 4.55 при обращении к процедуре F формальный параметр x будет при модификации заменен фактическим. Остальные переменные тела процедуры модификацией не затрагиваются.

Пример 5.55. Рассмотрим классификацию объектов оператора

```
begin real a, b, c;
    procedure R(u, v); real u, v;
    begin real a;
        a: = b×u + u↑2;
      begin integer u; real b;
        u: = 2;  b: = 0.7;
        v: = a×u + (b×u↑2)/(c — b);
      end блока
    end составного оператора (тела процедуры)
end;
```

и модификацию тела процедуры при обращении к нему оператором

$$R(x\uparrow 2 - y\uparrow 2,\ z).$$

Здесь тело процедуры является составным оператором. В первом операторе идентификатор u является формальным параметром и при модификации будет заменен. Объект a является локальным, и присвоение ему значения не затронет значения во внешнем блоке. Объект b — прочий, и его значение берется из внешнего блока.

Во втором операторе тела объекты u и b — локальные и при модификации не затрагиваются. Объект v — формальный. Объект a для данного блока является глобальным, но для тела процедуры — локальным и при вычислении v берется локальное значение a, вычисленное при присваивании в первом операторе тела, а не значение из внешней программы. Наоборот, объект c здесь прочий, и его значение берется из внешней программы.

Модифицированное тело при указанном обращении будет выглядеть так:

```
begin real a;
        a: =b× (x↑2−y↑2)+(x↑2−y↑2)↑2;
      begin integer u; real b;
        u: =2; b: =0.7;
        z: =a×u+(b×u↑2)/(c−b);
      end блока
end составного оператора тела;
```

Формальные параметры процедур дают возможность обращаться к процедурам с самыми разнообразными фактическими, что представляет значительные удобства при программировании. Но нередко случается, что к некоторой процедуре приходится на протяжении всего вычислительного процесса обращаться с одними и теми же фактическими параметрами, которые только изменяют свои числовые значения.

Пусть, например, в машину вводятся тройки положительных чисел a, b, c и требуется определить, можно ли из отрезков, длины которых выражаются этими числами, построить треугольник. Если написать процедуру проверки для одной такой тройки, то в общей программе обращение к процедуре всегда будет иметь один и тот же вид.

В таких случаях преимущество формальных параметров теряет силу и удобно пользоваться *процедурой без параметров*. Оператор процедуры без параметров имеет вид

⟨идентификатор процедуры⟩;

а ее описание

procedure ⟨идентификатор процедуры⟩; Q;

где Q — тело процедуры. Такая процедура не имеет фактических параметров, а ее описание не имеет формальных параметров. В теле процедуры Q действуют лишь локальные и прочие (глобальные) объекты.

Пример 6.55. Напишем программу для проверки возможности построения треугольника из N троек чисел

a, b, c, пользуясь процедурой без параметров. Это можно сделать с помощью программы

```
begin real a, b, c; boolean P;
    procedure Треугольник;  P: =(a<b+c)\/(b<c+a)
                                \/(c<a+b):
    begin integer i;
        for i: =1 step 1 until N do
        begin read (a, b, c); Треугольник; print (P);
        end;
    end; stop;
end;
```

Для процедур без параметров остаются в силе все правила модификации тела и выполнения процедуры, относящиеся к локальным и прочим объектам.

Разновидностью оператора процедуры является *оператор функции*, который отличается от оператора процедуры по виду описания и по использованию в программе.

В описании функции перед словом **procedure** пишется тип: **integer**, **real** или **boolean**, и это уже является полным описанием переменной с идентификатором функции. В теле функции обязательно должно выполняться одно или несколько присваиваний определенных значений идентификатору функции. Мы ограничимся рассмотрением случаев, когда идентификатор функции разрешается писать только слева от знака присваивания *).

Оператор функции может употребляться в любых арифметических или булевских выражениях. Когда в некотором выражении встречается функция, то программа находит соответствующее описание, вычисляет значение функции в теле описания и затем использует полученное значение. Аналогично процедуре функция может быть как с параметрами, так и без них.

Пример 7.55. Составим описание функции для суммирования элементов массива

$$S=\sum_{i=m}^{n} a_i,$$

*) Идентификатор функции может встречаться справа от знака присваивания при рекурсивном использовании процедур, которого мы рассматривать не будем.

где a — идентификатор массива

```
real procedure S (i, m, n, a);
    begin real s1; s1: =0;
        for i: =m step 1 until n do s1: =s1+a[i];
                                    S: =s1
    end;
```

В этой программе для суммирования в цикле нельзя было использовать сразу идентификатор S и обойтись без вспомогательного s1, так как иначе идентификатор S встречался бы справа от знака присваивания, чего мы не разрешаем.

Функция S может использоваться в различных операторах, например, в операторе присваивания

u: =exp (S (i, 1, 10, 1/i↑2)+sqrt (S (i, 2, 7, $x[i]\times y[i]$))),

который присвоит переменной u значение

$$e^{\sum_{i=1}^{10} \frac{1}{i^2} + \sqrt{\sum_{i=2}^{7} x_i y_i}}.$$

До сих пор мы рассматривали процедуры, тело которых было оператором, простым или составным. Правилами алгола допускается использование таких процедур, телом которых является программа, написанная не на алголе. Это дает возможность отдельные части алгольной программы писать, в случае надобности, на языке машины.

В приложении 2 приведены синтаксические таблицы, содержащие сводку определений основных понятий алгола.

ЧАСТЬ ЧЕТВЕРТАЯ

ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

ГЛАВА XI

СТРУКТУРА СОВРЕМЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

§ 56. Системы команд вычислительных машин

Как мы уже знаем, современная электронная вычислительная машина состоит из арифметического устройства, устройства управления и различных запоминающих устройств.

В этой главе мы рассмотрим функции и взаимодействие этих устройств в современных вычислительных системах.

Арифметическое устройство и устройство управления вместе образуют *процессор* вычислительной машины или вычислительной системы. Процессор предназначен для реализации программы, т. е. для выполнения отдельных команд в заданной последовательности. В этом параграфе мы рассмотрим системы команд, отражающие основные функции процессора.

Для *целей программирования* наиболее удобными являются *трехадресные машины*, команда в которых содержит три адреса. Это позволяет записывать команду в содержательных обозначениях наиболее удобным и естественным способом. Однако дальнейшее развитие трехадресных машин, по-видимому, будет сдерживаться следующим обстоятельством. Рассмотренная нами в предыдущих главах трехадресная машина имеет $4096 = 2^{12}$ ячеек памяти, поэтому для записи трех адресов нужно 36 разрядов. Кроме них, на запись кода отводится шесть разрядов и три — на запись признаков для работы с индексным регистром. Мы получаем 45 разрядов; этого вполне достаточно для записи числа. Если же увеличить оперативную память, то ячейка для записи трехадресной команды может оказаться

слишком длинной. Например, если оперативная память будет состоять из $32768 = 2^{15}$ ячеек, то уже для записи адресной части команды понадобится 45 разрядов. Если еще добавить, что в машине удобно иметь не один, а несколько индексных регистров, то ясно, что потребуется весьма длинная ячейка памяти; это слишком громоздко.

Выход из этого положения — в изменении структуры команды и уменьшении числа упоминаемых в ней адресов. Такие машины с меньшим числом адресов (двухадресные и одноадресные) широко применяются в настоящее время.

В *двухадресной машине* команда содержит код операции и два адреса. В случае арифметической операции вида

$$a+b=c$$

в двух адресах команды можно указать, например, адреса двух *операндов* — слагаемых a и b. Результат операции (сумма c) в двухадресной машине заносится в специальный регистр, называемый сумматором Σ, адрес которого в команде явно не указывается:

$$a+b \Rightarrow \Sigma.$$

Перенос результата операции сложения из сумматора Σ в ячейку c можно организовать при помощи команды переноса, совмещенной с безусловной передачей управления ячейке K.

$$Б,\ K,\ \Sigma \Rightarrow c.$$

Таким образом, арифметическая операция

$$a+b=c$$

в двухадресной машине выполняется в две команды

$$a+b \Rightarrow \Sigma$$
$$Б,\ Я+1,\ \Sigma \Rightarrow c$$

В двухадресных машинах имеется набор команд, аналогичных операциям трехадресной машины (арифметические, логические операции, операции условной и безусловной передачи управления и работы с индексным регистром).

Рассмотрим теперь структуру команды *одноадресной машины*. Здесь команда содержит лишь один адрес.

Поэтому не только результат операции, но и один из операндов должен браться из сумматора Σ. Вследствие этого, например, одна трехадресная команда

$$a+b=c$$

потребует для своей записи трех одноадресных, которые можно записать таким образом:

1) $\Sigma \Leftarrow a$
2) $\Sigma + b$
3) $\Sigma \Rightarrow c$

Эти команды означают:

1) послать число из ячейки a в сумматор Σ,

2) содержимое сумматора Σ сложить с числом из ячейки b, сумму направить в сумматор,

3) содержимое сумматора Σ переслать в ячейку c.

Не следует думать, что такое удлинение имеет место всегда. Во-первых, ряд команд является, по существу, одноадресными (например, команды передачи управления). Во-вторых, ряд промежуточных результатов не требуется отправлять на хранение в ячейки памяти. Они могут быть использованы тут же, и за счет этого программу можно значительно сократить.

Рассмотрим, например, как пишется программа вычисления кубического многочлена:

$$y=a_0x^3+a_1x^2+a_2x+a_3=((a_0x+a_1)\,x+a_2)\,x+a_3.$$

Для трехадресной машины эта программа требует шести команд:

1) $a_0 \times x = y$
2) $y + a_1 = y$
3) $y \times x = y$
4) $y + a_2 = y$
5) $y \times x = y$
6) $y + a_3 = y$

Для одноадресной машины здесь будет достаточно восьми команд:

1) $\Sigma \Leftarrow a_0$
2) $\Sigma \times x$
3) $\Sigma + a_1$
4) $\Sigma \times x$
5) $\Sigma + a_2$
6) $\Sigma \times x$
7) $\Sigma + a_3$
8) $\Sigma \Rightarrow y$

Удлинение, как мы видим, совсем не такое большое.

Быстродействие процессора вычислительной машины в значительной мере определяется структурой системы команд, их адресностью. В вычислительных машинах создалась в настоящее время диспропорция между скоростью работы арифметического устройства и временем обращения к памяти. Обычно в машине время выборки из памяти одного машинного слова значительно превышает время выполнения одной арифметической операции. Поэтому время выполнения отдельной операции почти полностью определяется количеством обращений (для выборки и записи) к памяти.

Трехадресная команда

$$\varkappa : a + b = c$$

требует для своего выполнения четырех обращений к памяти (выборки команды $\varkappa$, операндов a, b, запись результата c). Соответствующая группа из трех одноадресных команд:

$$\Sigma \Leftarrow a$$
$$\Sigma + b$$
$$\Sigma \Rightarrow c$$

требует уже шести обращений к памяти (выборка трех команд, двух операндов, запись результата). Таким образом, мы имеем вместо одной команды 3 (проигрыш в памяти для программы в 3 раза) и вместо четырех обращений к памяти — 6 (проигрыш в быстродействии в 1,5 раза). Однако фактически для одноадресной машины дело обстоит значительно лучше. Обычно в одноадресной машине ячейка памяти, предназначенная для хранения одного числа, бывает настолько длинной, что содержит не одну, а две команды, поэтому для выборки двух последовательных команд требуется лишь одно обращение к памяти.

Имея это в виду, сравним по быстродействию и памяти написанные выше две программы вычислений кубического многочлена для трехадресной и одноадресной машины. Первая программа занимает 6 ячеек памяти, вторая — 4 ячейки, первая требует $4 \times 6 = 24$ обращения к памяти, вторая $4 + 8 = 12$ обращений.

Из этого примера видно, что на одноадресной машине программы для некоторых задач оказываются более короткими и быстрыми, чем на трехадресной машине.

Таким образом, трехадресные машины обладают существенным преимуществом перед одно- и двухадресными — на них удобно программировать. Однако одноадресные машины обладают другими, не менее существенными преимуществами: они допускают при ограниченной длине ячейки прямую адресацию памяти большего объема, позволяют ввести большее количество индексных регистров, в ряде случаев дают возможность получать более быстрые и короткие программы *). Поэтому в последнее время наблюдается тенденция строить вычислительные машины одноадресными **).

По структуре системы команд, организации памяти и другим характеристикам среди одноадресных машин существует большое разнообразие.

Основные особенности таких машин мы проиллюстрируем на примере условной одноадресной машины.

В этом параграфе мы будем предполагать, что машина имеет память из 4096 двадцатичетырехразрядных ячеек, занумерованных от 0000_8 до 7777_8.

Содержимое одной ячейки памяти (24 двоичных разряда) называется неполным машинным словом. Содержимое двух последовательных ячеек памяти называется полным машинным словом (48 двоичных разрядов).

За одно обращение к памяти в арифметическое устройство или устройство управления может быть выбрано полное машинное слово.

Полное машинное слово может быть использовано в различных операциях машины следующими способами (рис. 41):

а) длинное машинное слово,

б) длинное целое число, 1 разряд — знак числа, 47 разрядов — значащая часть,

в) длинное число с плавающей запятой, 1 разряд — знак числа, 11 разрядов — машинный порядок, 36 разрядов — мантисса.

Неполное машинное слово используется в машине следующими способами (рис. 42):

а) короткое машинное слово,

б) короткое целое число,

в) короткое число с плавающей запятой,

г) команда.

В машине имеется 7 индексных регистров (*ИР*) с номерами 1, 2, 3, 4, 5, 6, 7, каждый из которых содержит 12 двоичных разрядов.

В разрядах 15-13 команды указывается номер k индексного регистра. Если $k \neq 0$, то исполнительный адрес команды образуется сложением

*) В меньшей степени теми же преимуществами обладают двухадресные машины.

**) Заметим, впрочем, что сейчас создаются машины с переменной длиной ячейки и с переменной адресностью команд.

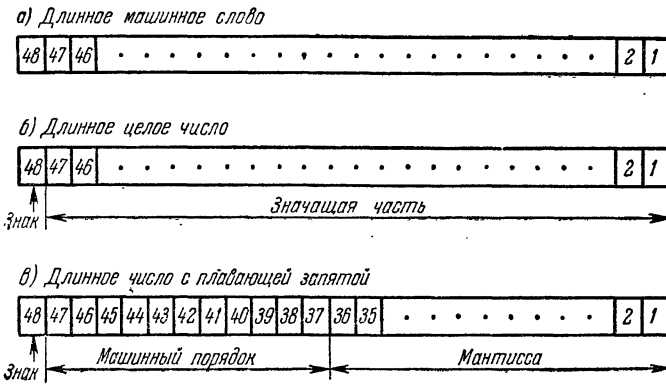

а) Длинное машинное слово
48 47 46 . . . . . 2 1
б) Длинное целое число
48 47 46 . . . . . 2 1
Знак
Значащая часть
в) Длинное число с плавающей запятой
48 47 46 45 44 43 42 41 40 39 38 37 36 35 . . . . . 2 1
Знак
Машинный порядок
Мантисса

Рис. 41.

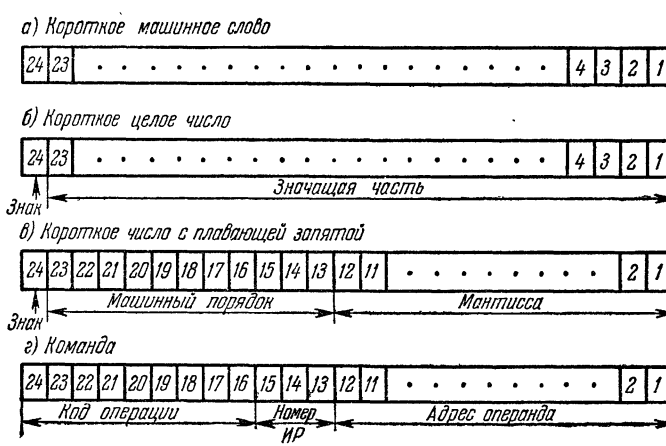

а) Короткое машинное слово
24 23 . . . . . 4 3 2 1
б) Короткое целое число
24 23 . . . . . 4 3 2 1
Знак
Значащая часть
в) Короткое число с плавающей запятой
24 23 22 21 20 19 18 17 16 15 14 13 12 11 . . . . . 2 1
Знак
Машинный порядок
Мантисса
г) Команда
24 23 22 21 20 19 18 17 16 15 14 13 12 11 . . . . . 2 1
Код операции
Номер ИР
Адрес операнда

Рис. 42.

с $ИР_k$. Таким образом, исполнительный адрес команды образуется по формуле:

$$A_{\text{исп}} = [A + ИР_k] \bmod (2^{12}),$$

где следует считать $ИР_0 = 0$.

Использование в команде нескольких индексных регистров создает большие удобства при программировании.

Большинство операций машины выполняется с участием сумматора Σ, являющегося 48-разрядным регистром. Левая половина сумматора участвует в операциях с неполными словами, весь сумматор — в операциях с полными словами.

В команде машины под код операций отводится 9 разрядов (рис. 42, *г*), поэтому возможное количество операций не превышает $2^9 = 512$. Однако различных по существу операций, как мы увидим, значительно меньше (около 40), большинство операций являются модификациями немногих основных.

Все элементарные операции машины делятся на восемь групп и описаны в таблицах 1.56—7.56. Номер группы определяется старшей восьмеричной цифрой (разряды 24-22) кода операции.

Описание операций мы начнем с группы 4-5 (таблица 5.56). В эту группу входят двухместные операции вида

$$a \quad \varkappa \quad b \Rightarrow c,$$

где $\varkappa$ — код операции, a и b — адреса операндов-аргументов, а c — адрес операнда результата. Всего в эту группу входит 16 основных операций, каждая из которых имеет по 7 модификаций.

Основные операции — это 12 арифметических операций над числами с плавающей запятой и над целыми числами и 4 логические операции.

13 из 16 основных операций выполняются по схеме:

$$\Sigma \quad \varkappa \quad A \Rightarrow \Sigma,$$

где A — исполнительный адрес команды, а 3 остальных (2 операции обратного вычитания и обратное деление) — по схеме:

$$A \quad \varkappa \quad \Sigma \Rightarrow \Sigma.$$

Порядок выполнения основных операций в арифметическом устройстве тот же, что и для подобных операций в трехадресной машине. Отметим лишь, что арифметические действия над целыми числами производятся по обычным правилам двоичной арифметики, а при операции сдвига — величина сдвига определяется величиной целого числа, расположенного в ячейке A, направление же сдвига — знаком этого числа (+ влево, — вправо).

Если в коде операции $19p = 1$, то операция производится по одной из схем:

$$\Sigma \quad \varkappa \quad A \Rightarrow A$$

или

$$A \quad \varkappa \quad \Sigma \Rightarrow A.$$

Если $20p = 1$, то операндами служат содержимое полной ячейки A и всего сумматора Σ, если $20p = 0$, то работа производится с неполной ячейкой A и левой половиной сумматора Σ.

Наконец, если $21p = 1$, то при образовании адреса операнда применяется способ косвенной адресации. Суть этого способа заключается в следующем. В качестве адреса A используется не сам исполнительный адрес команды, а адресная часть содержимого ячейки, имеющей этот адрес. Косвенная адресация оказывается очень удобным средством составления программ, содержащих блоки формирования *).

Рассмотрим теперь две группы одноместных операций (группы 1 и 2, табл. 2.56, 3.56).

При помощи одноместной операции группы 1 либо содержимое сумматора переносится в ячейку памяти ($19p = 1$), либо содержимое ячейки памяти переносится в сумматор ($19p = 0$). При этом происходит либо просто перенос машинного слова, либо перенос сопровождается преобразованиями, указанными в названии операции (первый столбец таблицы 2.56). Операции группы 2 обеспечивают либо занесение в какой-либо индексный регистр содержимого адресной части ячейки A, либо запоминание содержимого индексного регистра в адресной части ячейки A.

В особые группы выделены операции с операндом в виде адреса команды (группы 6-7, табл. 6.56, 7.56).

В эти группы входят двухместные операции одного из трех типов:

$$\Sigma \quad \varkappa \quad \overline{A} \Rightarrow \Sigma,$$

$$\overline{A} \quad \varkappa \quad \Sigma \Rightarrow \Sigma,$$

$$ИР \quad \varkappa \quad \overline{A} \Rightarrow ИР$$

и одноместных операций типа

$$\overline{A} \Rightarrow \textit{регистр}.$$

Отличие этих операций от аналогичных операций групп 2, 3, 4 состоит лишь в том, что в качестве операнда используется не содержимое ячейки A, а сам адрес A, при этом если операнд (в соответствии со смыслом операции) рассматривается как машинное слово, он дополняется слева нужным количеством нулей; в случае целого (или плавающего) числа старший разряд адреса A принимается за знак числа, младшие одиннадцать — за значащую (целую) часть.

Рассмотрим, наконец, группу операций передачи управления (группа 0, табл. 1.56). Основные операции этой группы являются операциями условной передачи управления команде A при выполнении условия:

$$\Sigma \quad \varkappa \quad 0,$$

где $\varkappa$ — обозначение, указанное во II столбце табл. 1.56. Например, операция $< A$ выполняется следующим образом: если $\Sigma < 0$, то происходит передача управления A, иначе — переход на следующую команду. При этом, если $19p = 1$, то передача управления сопровождается засылкой в специальный 24-разрядный регистр, называемый регистром возврата, команды безусловной передачи управления,

$$Б \; Я + 1,$$

где $Я$ — адрес исполняемой команды.

*) Применяется этот способ адресации и в других случаях, с одним из которых мы познакомимся в следующем параграфе.

Таблица 1.56

Группа 0. Операции передачи управления
($24p=0$, $23p=0$, $22p=0$)

Основная операция 18—16p			Модификации (21—19p)						Пояснения	
			19p		20p		21p			
название	обозначение	код	0	1	0	1	0	1	к выполнению операции	к содержательной записи
Остановка машины	Стоп	0	Без запоминания возврата		—		Прямая адресация			
Условные передачи управления	$\vee$ $\wedge$	1								
	$\vee$ $\wedge$	2								
	$\neq$	3								
	$=$	4								
	$\leqslant$	5								
	$\geqslant$	6								
Безусловная передача управления	Б	7	С запоминанием возврата		Анализ левой половины Σ		Косвенная адресация		Передача управления по исполнительному адресу, при выполнении условия $\Sigma \varkappa 0$, где $\varkappa$ — одно из отношений $<, >, \neq, =, \leqslant, \geqslant$	Если $19p=1$, то к коду операции приписывается буква B
			Обычная передача управления		Анализ всего Σ					
			Управление на команду обмена *)							

*) См. § 59.

Таблица 2.56

Группа 1. Одноместные операции с сумматором и операндом в ячейке памяти A
($24p=0$, $23p=0$, $22p=1$)

Основная операция $18-16p$		Модификации					
		$19p$		$20p$		$21p$	
название	код	0 в Σ	1 в ячейку	0	1	0	1
Перенос машинного слова	0	$\Leftarrow$	$\Rightarrow$	Операция с неполной ячейкой	Операция с полной ячейкой	Прямая адресация	Косвенная адресация
Перенос модуля числа	1	$\|\Leftarrow\|$	$\|\Rightarrow\|$				
Перевод целого в плавающее	2	$\Leftarrow$ цп	$\Rightarrow$ цп				
Перевод плавающего в целое	3	$\Leftarrow$ пц	$\Rightarrow$ пц				
Целая часть	4	$\Leftarrow$ цч	$\Rightarrow$ цч				
Дробная часть	5	$\Leftarrow$ дч	$\Rightarrow$ дч				
Подсчет единиц в машинном слове	6	$\Leftarrow$ се	$\Rightarrow$ се				
Номер младшей единицы в машинном слове	7	$\Leftarrow$ ме	$\Rightarrow$ ме				

Таблица 3.56

Группа 2. Одноместные операции с индексными регистрами и другими служебными регистрами (номер k индексного регистра указывается в $18-16p$, $24p=0$, $23p=1$, $22p=0$)

Основная операция $20p-19p$			Модификация ($21p$)		Пояснения к содержательной записи
название	код	обозначение	0	1	
Перенос из $ИР_k$ в ячейку	0	$ИР_k \Rightarrow$	Прямая адресация	Косвенная адресация	При $k=0$ вместо $ИР_0$ пишется $РВ$
Перенос из ячейки в индексный регистр $ИР_k$	1	$ИР_k \Leftarrow$			
Перенос служебного регистра в ячейку	2	$СР \Rightarrow$			В коде операции вместо $СР$ пишется обозначение служебного регистра
Перенос ячейки в служебный регистр	3	$СР \Leftarrow$			

Таблица 4.56

Группа 3. Двухместные операции с индексными регистрами (номер индексного регистра в $18-16p$, $24p=0$, $23p=1$, $22p=1$)

Основная операция		$20-19p$	Модификация $21p$		Пояснения	
название	обозначение	код	0	1	к исполнению	к содержательной записи
Сложение $ИР$ с ячейкой	$ИР_k+$	0	Прямая адресация	Косвенная адресация	Операции вида $ИР_k+A \Rightarrow ИР_k$ Результат, кроме того, направляется в левую половину Σ	При $k=0$ вместо $ИР_0$ пишется $РВ$
Вычитание из $ИР$ ячейки	$ИР_k-$	1				
Умножение $ИР$ на ячейку	$ИР_k\times$	2				
Вычитание из $ИР$ ячейки (без занесения в $ИР$)	$ИР_k-$б	3			Результат не направляется в $ИР$, а только в Σ	Буква б ставится справа от адреса

Таблица 5.56

Группа 4-5. Двухместные операции с операндом в ячейке памяти A ($24p=1$, $23p=0$)

Группа	Основная операция ($22p$, 18—$16p$)			Модификации							
	название	обозначение	код	$19p$		$20p$		$21p$			
				0	1	0	1	0	1		
Арифметические операции над числами с плавающей запятой	сложение	+	00	Результат в сумматоре	Результат в ячейке с адресом A	Операция с неполной ячейкой	Операция с полной ячейкой	Прямая адресация	Косвенная адресация		
	вычитание	—	01								
	умножение	×	02								
	деление	:	03								
	вычитание *) (обратное)	$-_0$	04								
	деление *) (обратное)	$:_0$	05								
Арифметические операции над целыми числами	сложение	+,	06								
	вычитание	—,	07								
	умножение	×,	10								
	частное	:,	11								
	вычитание *) (обратное)	$-_{,0}$	12								
	остаток	mod	13								
Поразрядные логические операции	логическое сложение	$\vee$	14								
	сверка	$\nsim$	15								
	логическое умножение	$\wedge$	16								
	сдвиг	→	17								

*) Первый операнд берется из ячейки A, второй из сумматора Σ.

Таблица 6.56

Группа 6. Операции с индексными (и служебными) регистрами и операндом в адресе команды (номер индексного или служебного регистра k в разрядах указывается в 18—16p, $24p=1$, $23p=1$, $22p=0$)

Основная операция 21—19p		
название	обозначение	код
—	—	0
Перенос адреса A в $ИР$	$ИР_k \Leftarrow$	1
—	—	2
Перенос адреса A в $СР$	$СР_k \Leftarrow$	3
Сложение $ИР$ с адресом A	$ИР_k +$	4
Вычитание из $ИР$ адреса A	$ИР_k -$	5
Умножение $ИР$ на адрес A	$ИР_k \times$	6
Вычитание из $ИР$ адреса A (без занесения в $ИР$)	$ИР_k - б$	7

Таблица 7.56

Группа 7. Двухместные операции с операндом в адресе команды ($24p=1$, $23p=1$, $22p=1$)

Основная операция 22p, 18—16p	Модификация 20p	
	0	1
Те же, что в группе 4-5	Операции с неполным Σ	Операции с полным Σ

Заметим, что используя операцию «Вычитание из *ИР* ячейки (без занесения в *ИР*)» (гр. 3, табл. 4.56) и операцию условной передачи управления, можно осуществлять разветвление вычислительного процесса по значению *ИР* (в частности, окончание цикла по *ИР*).

Заканчивая рассмотрение системы команд одноадресной машины, заметим, что для удобства программирования адреса группы из нескольких первых ячеек оперативной памяти выделены для регистров машины согласно табл. 8.56.

Таблица 8.56

№ ячейки	Содержимое	Обозначение	№ ячейки	Содержимое	Обозначение
0000—0001	машинный нуль	0	0010	регистр возврата	*РВ*
0002—0003	сумматор	Σ	0011	индексные регистры	*ИР*1
0004—0005	регистр защиты	*РЗ*	0012		*ИР*2
			0013		*ИР*3
0006	регистр прерывания	*РПР*	0014		*ИР*4
			0015		*ИР*5
0007	регистр маски	*РМ*	0016		*ИР*6
			0017		*ИР*7

В качестве примера применения системы команд одноадресной машины составим стандартную программу умножения прямоугольной матрицы A с m строками и n столбцами на n-мерный вектор $\boldsymbol{b}$. Результат умножения, m-мерный вектор $\boldsymbol{c}$ $(c_1, c_2, \ldots, c_m)$, определяется формулой

$$c_i = \sum_{s=1}^{n} a_{is} b_s \qquad (i = 1, 2, \ldots, m).$$

Обращение к программе запишем в виде

БВ,	*УМВ*	0, c_1
0	a_{11}	0, M
0	b_1	0, N

Здесь M, N — адреса неполных ячеек памяти, содержащих m и n; предполагается, что массивы $(a_{11}, \ldots, a_{mn})$, $(b_1, \ldots, b_n)$, $(c_1, c_2, \ldots, c_m)$ располагаются в последовательных полных ячейках памяти.

Таблица 9.56

УМВ	$РВ \Rightarrow \Sigma$			3200	200	0	0002
	$+ \quad \bar{4}$			1	700	0	0004
	$\Rightarrow$ к4 (конец *УМВ*)			2	110	0	0030
	$ИР1 \Rightarrow$ к3			3	201	0	0027
	$ИР2 \Rightarrow$ к2			4	202	0	0026
	$ИР3 \Rightarrow$ к1			5	203	0	0025
	$РВ \Rightarrow ИР1$			6	200	0	0011
	$ИР3 \Leftarrow 1^{*1}$		e_{11}	7	213	1	0001
	$ИР2 \Leftarrow 2^{*1}$		c_1	321 0	212	1	0002
	$\Leftarrow 3^{*1}$	k		1	160	1	0003
	$\longrightarrow \overline{1p}$			2	727	0	0001
	$\Rightarrow \mu$		(0,2 m)	3	110	0	0024
	$\Leftarrow 4^{*1}$	k		4	100	1	0004
	$\longleftarrow \overline{1p}$			5	727	0	0001
	$\Rightarrow \nu$		(0,2 n)	6	110	0	0023
	$ИР1 \Leftarrow 0^{*1}$		a_{11}	7	211	1	0006
	БВ Раб. ч.			3220	017	0	3222
	*Б*к1			1	007	0	0025
Раб. ч.	$РВ \Rightarrow$ к Р. ч.			2	200	0	3240
*Ц*1	$\Leftarrow \bar{0}$	n		3	120	0	0000
	$\Rightarrow 0^{*3}$	n	c_i	4	130	3	0000
*Ц*2	$\Leftarrow 0^{*1}$	n	a_{is}	5	120	1	0000
	$\times \; 0^{*2}$	n	b_s	6	422	2	0000
	$+ \; 0^{*3}$	n		7	420	3	0000
	$\Rightarrow 0^{*3}$	n	c_i	3230	130	3	0000
	$ИР1 +, \bar{2}$			1	641	0	0002
	$ИР1 -, \nu$	$б$		2	331	0	0023
	$< Ц2$			3	001	0	3225
	$ИР2 +, \nu$			4	302	0	0023
	$ИР3 +, \bar{2}$			5	643	0	0002
	$ИР3 -, \mu$	$б$		6	333	0	0024
	$< Ц1$			7	001	0	3223
к Р. ч.	0			3240	000	0	0000

Стандартная программа в содержательных обозначениях и в закодированном виде приведена в табл. 9.56. При содержательной записи команд мы помечали работу с полной ячейкой буквой *п*, косвенную адресацию — буквой *к*, блокировку записи в *ИР* — буквой *б*, а переадресацию по индексному регистру — звездочкой с номером *ИР*.

§ 57. Иерархия памяти

Наряду с арифметическим устройством важнейшими устройствами вычислительной машины являются *запоминающие устройства* (*устройства памяти*).

Типичное запоминающее устройство машины состоит из набора одинаковых запоминающих ячеек, каждая из которых может хранить машинное *слово* определенной *длины*. В разных запоминающих устройствах (в разных машинах) машинное слово содержит от 12 до 72 двоичных разрядов. За одно обращение к памяти из ячейки может быть выбрано (или в нее записано) одно машинное слово.

Время выборки (записи или чтения) одного машинного слова является одной из важнейших характеристик памяти— для различных видов памяти оно измеряется величинами от долей микросекунды (1 *мксек* $= 10^{-6}$ *сек*) до нескольких миллисекунд (1 *мсек* $= 10^{-3}$ *сек*).

Однако быстродействие памяти определяют не только временем выборки, но и *способом выборки*. По способу выборки запоминающие устройства делятся на 3 категории: устройства с произвольной, с периодической и последовательной выборкой. При произвольной выборке обеспечивается непосредственный доступ к любой ячейке памяти, и, следовательно, быстродействие памяти полностью определяется временем выборки. Типичное и наиболее распространенное запоминающее устройство с произвольной выборкой — это память на ферритовых сердечниках.

Примером устройства с периодической выборкой является магнитный барабан. За один оборот барабана могут быть последовательно выбраны машинные слова, расположенные по всей поверхности барабана.

Таким образом, максимальное время обращения к ячейке барабана равно периоду его полного оборота, а среднее время — половине оборота.

Таблица 1.57

Характеристики запоминающих устройств

Вид выборки	Тип запоминающего устройства	Время выборки одного слова (*мксек*)	Среднее время ожидания выборки (*сек*)	Емкость (количество машинных слов)
Произвольная	Регистры на триггерных ячейках	0,01—0,1	—	10—100
	Память на ферритовых сердечниках	0,1—10	—	10^4—10^6
Периодическая	Магнитные барабаны	10^2	0,01—0,1	10^5—10^7
	Магнитные диски	10^3	0,1	10^6—10^8
Последовательная	Магнитные ленты	10^4	10—100	10^6—10^9

Магнитная лента — наиболее типичное запоминающее устройство с последовательной выборкой. Для того чтобы получить доступ к машинному слову на магнитной ленте, следует перемотать ее от того места, над которым стоят считывающие головки, до требуемого слова. В некоторых случаях на это требуется несколько минут. В последнее время магнитные ленты постепенно вытесняются магнитными дисками, являющимися устройствами с периодической выборкой.

Наиболее быстрые устройства памяти с произвольной выборкой и малым временем выборки являются вместе с тем наиболее громоздкими и дорогими. Поэтому емкость (максимальное количество машинных слов или бит) наименьшая у быстрых устройств памяти и наибольшая у медленных устройств.

В таблице 1.57 приведены характеристики различных типичных запоминающих устройств вычислительных машин.

Из этой таблицы видно, что времена выборки и емкости различаются для разных устройств в миллионы раз.

Как мы видели, быстродействие процессора машины определяется, как правило, временем обращения к памяти. Поэтому в качестве оперативного (*внутреннего*) *запоминающего устройства*, непосредственно связанного с процессором, в машинах используют наиболее быстрые *устройства с произвольной выборкой*.

Устройства с периодической и последовательной выборкой используются в машине в качестве *внешней памяти*,

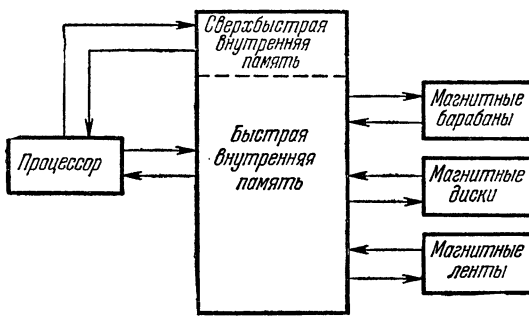


Рис. 43.

эти устройства не связаны с процессором, а лишь с внутренней памятью (рис. 43).

При времени обращения порядка 1-2 *мксек* для одноадресной машины быстродействие достигает 500 тыс. операций в секунду. Для того чтобы еще больше повысить быстродействие, часто несколько десятков ячеек оперативной памяти используют под индексные регистры и рабочие ячейки и делают их еще более быстродействующими. Эти сверхбыстрые ячейки изготовляют из тех же элементов, что и регистры арифметического устройства.

Таким образом, в современной вычислительной машине имеется следующая *иерархия памяти* — несколько десятков ячеек сверхбыстрой внутренней оперативной памяти; сотни тысяч быстрых ячеек внутренней (оперативной) памяти; миллионы ячеек медленной внешней памяти с циклической выборкой, применяемой для хранения, записи и чтения больших массивов информации, используемой

в процессе счета; миллиарды ячеек с более медленной внешней памятью с последовательной выборкой, используемой в основном в качестве архива для долговременного хранения информации.

Возможность обращения к конкретной ячейке в запоминающем устройстве обеспечивается с помощью *схемы адресации*. В наиболее мощных машинах объем внутренней памяти достигает сотен тысяч машинных слов. Для адресации такой памяти требуется иметь в команде 18-20 разрядный адрес. Такая длина адреса привела бы даже для одноадресной машины к чрезмерно длинной ячейке. Поэтому адресную часть команды делают обычно небольшой (от 8 до 12 разрядов), а для адресации любой ячейки памяти применяют различные искусственные приемы.

Рассмотрим некоторые из этих приемов на примере одноадресной машины.

Предположим, что одноадресная машина имеет внутреннюю память емкостью 2^{18} неполных машинных слов.

Будем теперь считать, что *индексные регистры* машины являются восемнадцатиразрядными и исполнительный адрес образуется по формуле:

$$A_{\text{исп}} = [A + ИР_k] \bmod (2^{18}).$$

Тогда при помощи индексации можно обеспечить доступ к любой из 2^{18} ячеек памяти. Этот способ обладает тем недостатком, что он исключает возможность использования индексного регистра для переадресации той же команды в цикле *) (см. рис. 44 *а*).

В другом способе полной адресации используются *косвенные адреса* (см. предыдущий параграф). Если в команде есть признак косвенной адресации, то за адрес операнда принимается адресная часть ячейки, адрес которой указан в команде.

---

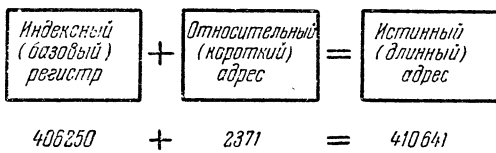
*) В некоторых машинах в операционной части команды часть разрядов выделяется под номер s базового регистра *БР*, а часть — под номер k обычного индексного регистра *ИР*. Тогда

$$A_{\text{исп}} = [A + БР_s + ИР_k] \bmod (2^{18}).$$

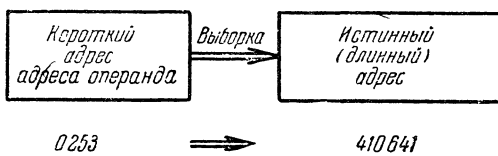
В таких машинах базовый регистр можно использовать для полной адресации памяти, а индексный регистр — для переадресации команд в цикле.

Если мы за адрес операнда примем не 12-разрядную адресную часть этой ячейки, а 18 ее младших разрядов, то тем самым обеспечим доступ к любой ячейке памяти (см. рис. 44 *б*). Способ косвенной адресации удобен, если адрес операнда формируется в программе, но неудобен, если он постоянен.

а) Индексная или базовая адресация



б) Косвенная адресация



в) Присоединенная адресация

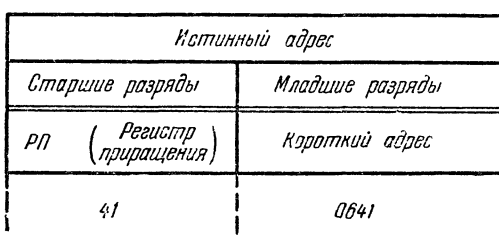


Рис. 44.

От этого недостатка свободен третий способ полной адресации, называемый *присоединенной адресацией*.

При этом способе адрес операнда состоит из двух частей— младшие 12 разрядов образуются из исполнительного адреса команды, старшие 6 разрядов берутся из специаль-

ного регистра, называемого регистром приращения адреса (*РПА*) (см. рис. 44в).

К 12-разрядной адресной части команды передачи управления при ее исполнении присоединяются 6 разрядов регистра приращения счетчика команд (*РПСК*). Таким образом, при помощи 12-разрядного регистра приращения *РП* (*РПА*, *РПСК*) удается как для команды, так и для операндов использовать всю оперативную память. Для образования нужных значений регистров приращений и их запоминания служат команды

$$РП \Leftarrow A$$
$$РП \Rightarrow A$$

По первой из них младшие 12 разрядов ячейки A заносятся в *РП*, по второй — содержимое *РП* запоминается в младших 12 разрядах ячейки A *).

Во внутренней памяти объемом в $2^{18} \approx 262$ тыс. ячеек можно одновременно держать много программ и выполнять их поочередно. При этом если какая-либо из этих программ содержит ошибку, она в процессе своей работы может «испортить» другую программу, которая в этой ошибке «не виновата». Особенно плохо, если этой программой будет библиотечная программа или диспетчер **). Для того чтобы было удобно распределять внутреннюю память между различными программами и избежать порчи одних программ другими, в современных машинах вводится *страничная структура памяти* и *защита памяти*.

Вся память объемом 2^n машинных слов разбивается на 2^s страниц, каждая объемом 2^k машинных слов ($s + k = n$). Каждой программе выделяется для пользования определенная совокупность из этих страниц.

Примем, что память нашей одноадресной машины состоит из 2^6 страниц (64 страницы), каждая емкостью 2^{12} слов (4096 слов), причем страница с номером i начинается с ячейки, старшие 6 разрядов адреса которой равны i, а младшие 12 разрядов нулевые.

---

*) Присоединенная адресация применяется в машинах БЭСМ-4, М-220, см. Приложение, § 69.

**) О диспетчере см. следующую главу.

При работе каждой данной программы ей выделяется определенная совокупность страниц, характеризуемая 64-разрядной двоичной шкалой (1 — страница занята данной программой, 0 — не занята).

Перед включением в работу данной программы эта шкала заносится на 64-разрядный регистр, называемый

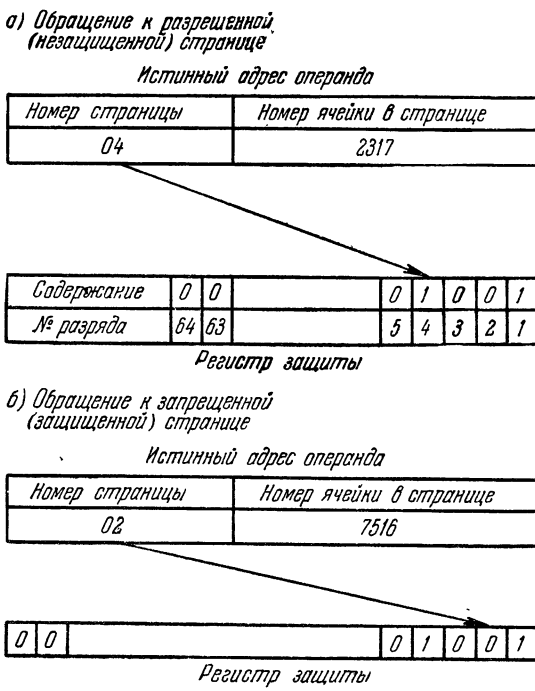


Рис. 45.

регистром защиты. Затем при исполнении каждой команды схема машины анализирует разряд регистра защиты, номер которого определяется шестью старшими разрядами 18-разрядного адреса операнда. Если в этом разряде 1, то команда обращается к разрешенной странице, если 0, то эта страница запрещена. В последнем случае выполнение данной программы прекращается, в первом — продолжается (см. рис. 45).

§ 58. Система прерывания

Процессоры современных вычислительных машин обладают громадным быстродействием, они выполняют от нескольких тысяч до нескольких миллионов элементарных операций в секунду. Поэтому любые простои в работе процессора, даже в несколько секунд, приводят к весьма неэффективному использованию вычислительной машины. При той организации работы на машине, которая описывалась ранее, такого рода простои неизбежны. Процессор машины простаивает, когда кончается выполнение одной программы (команда «стоп»), и вручную включается в работу другая программа, когда в результате ошибки в программе (или машине) происходит аварийная остановка (*авост*). Большие простои процессора связаны с различными манипуляциями за пультом управления машины (пультовая отладка, передача управления с пульта, ручная коррекция параметров для решаемой задачи и т. п.).

В этом параграфе мы опишем аппарат системы прерывания программ, дающий возможность свести к минимуму простои в работе процессора по указанным причинам.

Суть прерывания состоит в том, что по определенным сигналам (причинам прерывания) машина приостанавливает работу над текущей программой и передает управление другой программе.

Основной частью системы прерывания является специальный регистр (регистр прерывания), в котором на каждую причину остановки работы программы (причину прерывания) отводится один разряд. В частности, в этом регистре имеются разряды, характеризующие следующие причины (виды) прерывания:

аварийную остановку процессора (разряд «авост»),

обычную остановку процессора (разряд «стоп»),

обращение к защищенной странице памяти (разряд «защита памяти»).

Мы будем предполагать, что помимо данной (работающей) программы во внутренней памяти находится еще несколько программ, ждущих очереди для работы.

В случае возникновения одной из указанных причин прерывания целесообразно проанализировать и напечатать

сведения об этой причине и передать управление на начало другой программы.

Делается это следующим образом:

В определенной фиксированной ячейке памяти p заготовляется команда безусловной передачи управления с возвратом:

$$БВ \quad АП$$

где $АП$ — начало программного блока «Анализ прерываний», работу которого мы опишем дальше.

Пусть, например, произошел *авост* по команде с номером k работающей программы. Тогда в регистре прерывания устанавливается 1 в разряд «авост» и на регистр команд машины вызывается команда из ячейки p (счетчик команд при этом не меняется).

В результате выполнения этой команды в регистре возврата ($РВ$) запоминается $k+1$, т. е. увеличенный на единицу адрес команды, породившей *авост*, и передается управление на блок $АП$. В этом блоке производятся следующие действия:

а) в определенной ячейке s запоминается содержимое сумматора Σ машины,

б) при помощи специальной команды

$$РПР \Rightarrow a$$

в ячейку a засылается содержимое регистра прерывания,

в) по содержимому этой ячейки определяется причина прерывания (в данном случае *авост*),

г) на печать выдается причина прерывания, содержимое счетчика команд, регистра команд, сумматора и операнд команды в момент прерывания,

д) при помощи специальной операции $РПР \Leftarrow d$ заносится 0 в разряд «авост» регистра прерывания,

е) передается управление на начало другой программы *).

Аналогично организуется работа системы прерывания и при *стопе* или обращении программы к защищенной странице.

---

*) Здесь работа блока «Анализ прерываний» описана очень схематично, более подробно см. в § 62.

При описанной схемно-программной организации системы прерывания можно обеспечить непрерывный переход работы процессора с одной задачи на другую.

Кроме того, система прерывания дает возможность эффективно использовать процессор не только во время счета, но и в процессе отладки. В § 44 мы рассмотрели отладочные прокруточные программы, дающие возможность выдавать на печать многие детали выполнения отлаживаемой программы. При работе этих отладочных программ значительная доля машинного времени идет на прокрутку, т. е. на моделирование выполнения команд отлаживаемой программы. Это время можно свести к нулю, вводя еще одну причину прерывания: прерывание по каждой команде отлаживаемой программы. В этом случае после выполнения очередной команды отлаживаемой программы происходит прерывание, т. е. на регистр команд выбирается команда

БВ АП

и передается управление на блок *АП*. Однако при этом схемно следует запретить все прерывания (ибо если этого не сделать, то по каждой команде блока *АП* наступало бы прерывание). Блок *АП* в этом случае должен включить в работу блоки нужной прокруточной программы (например, программы «Няня»), а после окончания их работы передать управление следующей команде отлаживаемой программы, давая вместе с тем разрешение на прерывание.

Для работы с отладочной программой «Луч» (см. § 44) полезным оказывается еще одно прерывание, которое происходит не по каждой команде отлаживаемой программы, а лишь по командам передачи управления.

Проведенное рассмотрение показывает, что для правильной организации работы системы прерывания необходимо введение двух режимов работы процессора:

рабочий режим, при котором разрешены все (или некоторые) прерывания,

режим управления, при котором запрещены все прерывания.

В режиме управления работает блок *АП* *), в рабочем режиме — рабочие программы.

В вычислительной машине переход из рабочего режима в режим управления организуется специальной схемой в момент прерывания, обратный переход происходит по специальной команде, выполняемой в момент окончания работы блока *АП*.

Заметим, что рассмотренные прерывания по каждой команде или командам передачи управления полезны для

*) А также некоторые другие блоки операционной системы, которые будут описаны в следующей главе.

отлаживаемых рабочих программ и вредны для рабочих программ, по которым происходит счет (эти прерывания замедляют счет).

Для временного запрещения некоторых прерываний в машине имеется регистр маски. В этом регистре имеется несколько разрядов, соответствующих некоторым разрядам регистра прерывания.

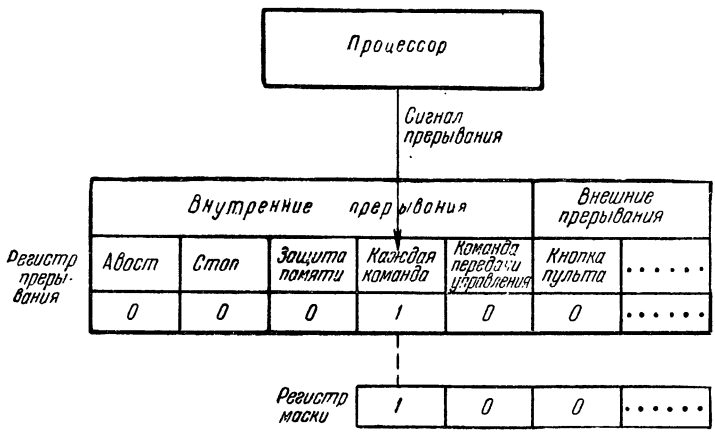


Рис. 46.

Если в определенном разряде регистра маски находится 1, то говорят, что соответствующее прерывание замаскировано. Если возникает причина замаскированного прерывания, то хотя в соответствующий разряд регистра прерывания заносится 1, однако самого прерывания не наступает (продолжается выполнение рабочей программы). При помощи специальной команды можно занести нули и единицы в разряды регистра маски, т. е. разрешить одни прерывания и запретить другие.

На рис. 46 изображено схематически состояние регистра маски и регистра прерывания в момент возникновения замаскированного прерывания по очередной команде рабочей программы, по которой производится счет *).

*) Заметим, что прерывания по *авосту* и *стопу* не имеет смысла маскировать, поэтому для них отсутствуют соответствующие разряды в регистре маски.

Рассмотренные пять видов прерывания (по авосту, стопу, защите памяти, каждой команде, командам передачи управления) вызываются работой процессора и поэтому называются внутренними прерываниями. Одно из внешних прерываний возникает при нажатии специальной кнопки на пульте управления машины. При помощи этого прерывания можно в процессе счета прерывать выполнение рабочей программы, сообщая при помощи регистров пульта управления те или иные указания машине.

§ 59. Управление обменом информации

При использовании вычислительной машины необходимо обеспечить обмен информации между внутренней (оперативной) памятью и внешним миром. Процессор машины может начать работать только после того, как во внутреннюю память введены исходные данные и программа. Во время работы программы может возникнуть необходимость обмена информацией между внутренней и внешней памятью. Результаты счета необходимо печатать (передавать информацию из внутренней памяти на печатающее устройство).

Устройства ввода-вывода, обеспечивающие указанный обмен информации, работают гораздо медленнее, чем процессор вычислительной машины. Чтобы понять, насколько медленно они работают, представим себе, что скорость работы процессора и устройств ввода-вывода современной вычислительной машины уменьшились в миллион раз. Тогда одна команда будет выполняться за 1 секунду, а одно машинное слово будет вводиться с перфокарты за одни сутки.

Пусть, например, нам нужно ввести во внутреннюю память и выполнить в цикле 10 раз программу из 30 команд. Тогда в нашей замедленной машине эта программа будет вводиться месяц, а выполняться пять минут, причем во время ввода программы процессор будет простаивать.

Таким образом, вычислительная машина используется весьма неэффективно, так как во время работы сравнительно медленных устройств ввода-вывода быстрый процессор машины не работает. Естественно было бы постараться увеличить производительность машины, сведя к минимуму простои процессора во время работы устройств ввода-вывода.

Покажем, как это можно сделать на примере ввода исходных данных с перфокарт. Пусть устройство ввода работает со скоростью 600 карт/*мин*, тогда за 1 *сек* вводится 10 карт, т. е. во внутреннюю память поступает 120 машинных слов. Если время обращения к памяти 1 *мксек* (10^{-6} *сек*), то в течение одной секунды лишь промежуток времени $10^{-6} \cdot 10^2 = 10^{-4}$ *сек* будет занят передачей информации в память, а остальное время ($1 - 10^{-4}$) *сек* пропадает впустую. Таким образом, эффективно используется лишь 0,01% машинного времени.

Оставшиеся 99,99% времени можно эффективно использовать, организуя «параллельную» работу процессора

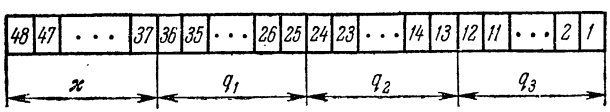


Рис. 47.

и внешнего устройства. Пусть нам нужно ввести с перфокарт массив информации из n машинных слов и поместить его в ячейки (a_1, a_n) внутренней памяти. При организации «параллельной» работы устройства ввода и процессора используется специальный 48-разрядный регистр обмена (*РО*), состоящий из 4 групп по 12 разрядов x, q_1, q_2, q_3 (рис. 47); величины q_1, q_2, q_3 мы назовем для краткости первым, вторым и третьим адресами *РО*.

Перед началом ввода следует поместить величины a_1 и a_n в адреса q_2, q_3 *РО* *), дать сигнал, запускающий устройство ввода, и начать выполнение программы.

В некоторый момент от устройства ввода поступит запрос на запись в память первого машинного слова; по этому запросу временно прекращается работа процессора и происходит передача в память машинного слова с перфокарты (в этом и заключается «параллельность» работы процессора и внешнего устройства).

*) Об использовании старших 24 разрядов *РО* будет сказано дальше.

Запись в память машинного слова с перфокарт организуется следующим образом. Из разрядов q_2 *РО* извлекается адрес a_1, по этому адресу происходит запись, затем сравниваются адреса q_2 и q_3 (a_1 и a_n), и если они не равны, то адрес q_2 увеличивается на 1. Таким образом, в q_2 оказывается величина a_2, что обеспечивает запись следующего машинного слова с перфокарты в ячейку a_2.

Если $q_2 = q_3$, то ввод с перфокарт прекращается (равенство $q_2 = q_3$ означает, что весь требуемый массив введен с перфокарт в память). Для того чтобы процессор «узнал», что ввод информации с перфокарт закончен и может приступить к ее обработке, при $q_2 = q_3$ подается сигнал прерывания. Это еще одно *внешнее прерывание*.

Описанный аппарат дает возможность производить запись в память лишь в те моменты времени, когда нужно очередное машинное слово с перфокарты передать в память, причем запись занимает только промежуток времени, требуемый для одного обращения к памяти.

Аналогично может быть организована «параллельная» работа процессора и какого-либо другого устройства ввода-вывода (ввод с перфоленты; вывод на перфоленту, перфокарты, печатающее устройство и т. п.).

Например, при выводе на перфокарты «параллельно» с работой процессора происходит чтение очередного машинного слова из памяти и перенос его в устройство вывода на перфокарты, в остальном организация работы та же, что и при вводе с перфокарт.

Аппарат «параллельной» работы используется не только при вводе-выводе, но и при обмене информации между внутренней и внешней памятью.

Устройство внешней памяти отличается от устройства ввода-вывода тем, что во внешней памяти информация адресуется (номера ячеек на *МБ*, зон на *МЛ*).

В связи с этим для организации обмена с внешней памятью следует задать, помимо границ массива (a_1, a_n) во внутренней памяти, его начало b_1 во внешней памяти.

Рассмотрим, например, как организуется в режиме «параллельной» работы запись на *МБ*, на поверхности которого рядом с содержимым каждой ячейки нанесен ее номер (адрес). Перед началом работы с *МБ* следует занести в адреса q_1, q_2, q_3 регистра обмена величины b_1, a_1, a_n

соответственно. Затем следует дать сигнал на начало работы с *МБ* и начать выполнение некоторой программы.

При прохождении под считывающими головками номера τ очередной ячейки *МБ* величина τ сравнивается с адресом q_1 регистра обмена (величина b_1); если эти величины не равны, то обмен не производится, если же $\tau = b_1$, то во время «параллельной» работы происходит запись содержимого начальной ячейки a_1 (адрес q_2) массива во внутренней памяти в ячейку b_1 (адрес q_1) *МБ*, после этого продолжается выполнение программы. Дальнейшее выполнение обмена в режиме «параллельной» работы происходит так же, как при работе с устройствами ввода-вывода.

При помощи аппарата «параллельной» работы и системы прерывания работу вычислительной машины можно организовать следующим образом.

Если при выполнении некоторой программы потребовалась работа с устройством ввода-вывода или внешней памятью, то из программы подается сигнал на запуск соответствующего внешнего устройства, а «границы» обмениваемого массива заносятся в *РО*. Тогда во время работы программы в режиме «параллельной» работы происходит требуемый обмен; по окончании обмена происходит внешнее прерывание, блок «Анализ прерываний» фиксирует причину прерывания и передает управление основной (прерванной) программе, которая может теперь уже воспользоваться обмененным массивом. При повторном запросе программы на обмен с этим же (или другим) внешним устройством работа организуется аналогично.

Описанные программные и схемные средства (устройство прерывания, аппарат «параллельной» работы, блок «Анализ прерываний») дают возможность организовать равноправный доступ к главной памяти процессора и внешнего устройства *).

§ 60. Современная вычислительная система

В предыдущих параграфах рассматривались различные компоненты вычислительной техники — процессор, запоминающие устройства (в том числе внутренняя оперативная память), устройства ввода-вывода, монопольный канал.

*) Устройство, организующее такую работу, иногда называют *монопольным* или *селекторным* каналом.

Структура, определенным образом связывающая эти компоненты в единое целое, называется вычислительной системой. Частным, простейшим случаем вычислительной системы является трехадресная вычислительная машина, рассмотренная нами во второй части книги. Обычно вычислительные системы содержат большее число компонент и обеспечивают более сложное взаимодействие между ними. В этом параграфе мы рассмотрим, как устроена типичная вычислительная система.

Основной частью системы является *главная* (оперативная) *память* большой емкости (порядка нескольких млн. бит). К этой памяти имеют доступ один или несколько процессоров и различные внешние устройства (устройства внешней памяти, устройства ввода-вывода, пульт, «часы» и т. д.). Процессоры и внешние устройства мы будем в дальнейшем называть абонентами главной памяти или просто *абонентами*.

Абоненты подсоединяются к главной памяти посредством устройства управления обменом информации, которое и организует доступ абонентов к памяти.

В предыдущем параграфе была рассмотрена такая организация работы, при которой поочередно обращаются к главной памяти два абонента — процессор и одно из внешних устройств. Это — так называемый монопольный режим работы.

Однако большей частью в вычислительной системе устройство управления обменом информации дает любому числу абонентов возможность параллельного обращения к главной памяти. Это — так называемый мультиплексный режим работы. Дальше мы увидим, что мультиплексный режим значительно расширяет возможности вычислительной системы и повышает ее производительность.

Рассмотрим вкратце функционирование системы в мультиплексном режиме, предполагая для простоты, что в системе имеется лишь один процессор.

Пусть этот процессор выполняет некоторую программу, и мы хотим, чтобы во время ее выполнения получили доступ к памяти еще N абонентов. Теперь, в отличие от монопольного режима, мы не можем «постоянно» (т. е. на все время обмена) хранить в регистре обмена ($РО$) границы обмениваемого массива, так как «параллельно» ведется обмен несколькими массивами.

Такой обмен может быть организован, например, следующим образом.

Каждому абоненту с номером k отводится в определенном массиве ячеек главной памяти (b_1, b_N) полная ячейка b_k. Перед включением в работу абонента k в эту ячейку помещается 48-разрядное машинное слово, имеющее структуру, указанную на рис. 47. Это машинное слово называется командой обмена, в коде операции этой команды указывается режим обмена (чтение, запись с контролем, без контроля и т. д.), в I и II адресах — начальные адреса массива у абонента и в главной памяти *), в III адресе — конечный адрес массива в главной памяти. По команде передачи управления

$$Я\colon\ Б\ k$$

(см. табл. 1.56) включается в работу абонент с номером k, а управление процессора переходит на команду $Я + 1$.

Обмен информацией между абонентом k и главной памятью организуется следующим образом.

В некоторый момент времени от абонента k в машину поступает запрос на память. Тогда на регистр обмена выбирается содержимое ячейки b_k, а по коду операции $\varkappa$ определяется режим обмена. Пусть, например, режим обмена — запись. Тогда содержимое ячейки памяти q_2 переписывается в память абонента по адресу q_1, адреса q_1 и q_2 увеличиваются на 1 и таким образом модифицированное содержимое $РО$ переписывается в ячейку b_k. При запросе от другого абонента с номером s обмен происходит аналогично при помощи аппарата, использующего в качестве информации команду обмена, лежащую в ячейке b_s.

Отметим, что так же, как и при монопольном режиме, в случае окончания обмена (при $q_2 = q_3$) возникает внешнее прерывание. Это прерывание свидетельствует об окончании текущего обмена с абонентом k и возможности нового обращения к этому абоненту.

Таким образом, запросы на обмен от абонентов обслуживаются в системе в порядке их поступления.

---

*) В случае неадресуемой памяти абонента первый адрес не используется.

Описанный мультиплексный режим работы называют иногда «параллельным» обслуживанием нескольких абонентов.

В наиболее мощных вычислительных системах число абонентов памяти достигает нескольких сот. В этом случае может создаться такое положение, когда один процессор захлебнется, будучи не в состоянии справиться с громадным потоком информации, идущей от абонентов, и не сможет выдать в достаточном объеме ответную информацию.

Выход из создавшегося положения состоит в подсоединении к главной памяти двух или более процессоров. Тогда один из процессоров может взять на себя часть работы по обработке информации. С другой стороны, повышается надежность системы — если один из процессоров выйдет из строя, другой может взять по крайней мере часть нагрузки на себя.

Важной компонентой вычислительной системы являются «электронные часы». Электронные часы могут, например, представлять устройство, состоящее из двоичного регистра t, в котором в каждый момент содержится показание астрономического времени с точностью 0,1 *сек* *), и датчика импульсов с частотой 0,1 *сек*.

В любой момент можно программным образом выбрать из регистра t двоичное показание астрономического времени. А датчик импульсов дает возможность организовать прерывание работы процессора через заданный промежуток времени T (относительное показание времени).

---

*) В разных «часах» этот промежуток различен.

ГЛАВА XII

ОПЕРАЦИОННАЯ СИСТЕМА

§ 61. Математическое обеспечение вычислительной системы

Неотъемлемой частью любой вычислительной машины является некоторый набор программ, составляющий математическое обеспечение. При этом нет четкого, приемлемого для всех машин, разделения функций машины и ее математического обеспечения. Так, например, для одних машин извлечение квадратного корня является элементарной операцией, в других машинах нет такой элементарной операции и для извлечения квадратного корня составляется специальная стандартная подпрограмма.

При этом для потребителя машины, составляющего рабочую программу, почти одинаково просто писать элементарную операцию или обращение к подпрограмме. Однако элементарная операция обычно выполняется в десятки раз быстрее, чем подпрограмма. Поэтому в систему команд машины включают обычно наиболее употребительные операции, оказывающие значительное влияние на быстроту выполнения программ. Так, например, во многих ранних вычислительных машинах действия над числами с плавающей запятой отсутствовали в системе команд и выполнялись по подпрограммам. Однако очень быстро выяснилось, что это приводит к громадным потерям в быстродействии при решении вычислительных задач. В настоящее время почти все вычислительные машины имеют в своих системах команд плавающие действия.

В больших вычислительных системах схемная и программные части еще теснее взаимодействуют между собой, чем в обычных, традиционных вычислительных машинах.

Так, например, в функционировании системы прерывания (см. § 58) одинаково важную роль играют и технические, и программные средства. Поэтому о работе современной вычислительной системы нельзя получить хотя бы приблизительного представления, не рассмотрев особенностей ее математического обеспечения.

Ту часть математического обеспечения системы, которая используется большинством потребителей машины, называют *общесистемным математическим обеспечением*.

В общесистемном математическом обеспечении следует особо выделить обслуживающие программы, которые не производят никакой работы по решению задач, а выполняют лишь вспомогательную организующую роль. Это — трансляторы с алгоритмических языков, организующие и редактирующие программы ввода-вывода и обмена, средства отладки и т. п. Значение таких обслуживающих программ тем больше, чем мощнее используемые в системе вычислительные средства и чем сложнее взаимодействие между ними.

Особенно большую роль обслуживающие программы играют в современных сложных вычислительных системах. Совокупность обслуживающих программ, организующих работу системы, образует *операционную систему*. Операционная система планирует решение задач, следит за их выполнением, выделяет задачам различные технические средства (участки внутренней и внешней памяти, время на одном из процессоров, устройства ввода-вывода), создает различные режимы решения задач (трансляция, отладка, счет), выявляет аварийные ситуации в процессе выполнения задач. Без операционной системы современная крупная вычислительная система практически бесполезна; без нее нельзя ни ввести в память ни одной программы задачи, ни организовать решение задач. В общесистемное обеспечение входит помимо операционной системы еще система программирования (трансляторы с алгоритмических языков; вспомогательные программы, организующие выявление и печать синтаксических и семантических ошибок и т. п.), библиотека стандартных подпрограмм (в том числе редактирующие программы ввода-вывода, программы наиболее употребительных численных методов), специализированные системы программ (например, набор программ для решения информационно-поисковых задач) и т. д.

§ 62. Организация многопрограммной работы

Одной из функций операционной системы является организация непрерывного (без промежуточных остановок процессора) решения потока задач. Блок операционной системы, выполняющий эту функцию, называют обычно *диспетчером*.

Предположим, что в главную память вычислительной системы одновременно помещены программы нескольких задач. Диспетчер может организовать решение задач в одном из следующих трех режимов:

а) последовательное обслуживание,

б) мультипрограммирование,

в) разделение времени.

В этом параграфе мы рассмотрим первые два режима.

В *режиме последовательного обслуживания* процессор системы выполняет программы задач последовательно; когда кончится решение одной задачи, включается в работу вторая, после второй — третья и т. д. В этом режиме каждой задаче на время ее решения могут быть предоставлены все технические средства системы, кроме главной памяти.

Главная память разделена между программами тех задач, которые в ней находятся в данное время. Для каждой задачи в распоряжении диспетчера имеется шкала защиты, в которой единицами помечены разрешенные для задачи страницы, нулями — запрещенные.

Перед включением в работу программы первой задачи диспетчер заносит ее шкалу защиты в регистр защиты и передает управление на начало программы. Сам диспетчер работает в режиме управления, а программы решаемых задач — в рабочем режиме. Поэтому во время работы программы задачи возможны прерывания. Если программа уже доработала до конца, то происходит прерывание по команде остановки, работает блок «Анализ прерываний», который передает управление на диспетчер. После этого диспетчер включает в работу вторую задачу.

Однако возможно возникновение и таких положений, когда в процессе выполнения программы возникает аварийная ситуация (авост или обращение к запрещенной стра-

нице). Тогда после прерывания диспетчер выдает на печать сведения об этой ситуации (номер задачи, адрес команды, содержимое операндов и т. д.) и включает в работу вторую задачу.

В процессе выполнения программы задачи может потребоваться работа с внешними устройствами (печать, ввод с перфокарт, обмен с внешней памятью и т. д.).

Запрос на такую работу происходит специальным обращением к диспетчеру. В этом обращении указываются границы выводимого массива, тип и номер внешнего устройства и другая дополнительная информация.

По этой информации диспетчер формирует команду обмена (см. § 59), защищает обмениваемые страницы, включает в работу внешнее устройство и передает управление на продолжение программы задачи. В дальнейшем может возникнуть одна из следующих двух ситуаций: процессор обратился к одной из ячеек обмениваемого массива до окончания обмена или обмен кончился раньше, чем произошло такое обращение.

Во втором из этих случаев произойдет прерывание, свидетельствующее о том, что внешнее устройство свободно, и по этому внешнему прерыванию диспетчер должен просто передать управление на команду, следующую за прерванной командой рабочей программы.

В первом случае произойдет прерывание по защите памяти, обозначающее, что не следует продолжать решение задачи до окончания обмена. Поскольку в нашем случае (режим последовательного обслуживания) к другой задаче переходить нельзя, то диспетчер вынужден организовать искусственное ожидание окончания обмена. Это делается путем выхода в рабочий режим и организации в этом режиме искусственного цикла (например, команда передачи управления на себя *Б*, *Я*). В таком случае некоторое время (до окончания обмена) процессор работает вхолостую. После прихода внешнего прерывания диспетчер организует продолжение выполнения программы задачи.

Целесообразно ускорить решение задачи и исключить холостую работу процессора. Для этого следует программу задачи организовать таким образом, чтобы требуемые обмены с внешними устройствами кончались раньше, чем программе потребуется обмениваемый массив.

Заметим, что это требование вносит большие затруднения в программирование, а иногда и вовсе невыполнимо.

Если сформулированное требование выполняется, то рассмотренный режим последовательного обслуживания дает возможность решить включенную в работу задачу за кратчайшее время (диспетчер «не отвлекается» на обслуживание других задач).

Поэтому в режиме последовательного обслуживания решают обычно наиболее срочные задачи.

В то же время рассмотренный режим для большинства задач приводит к большой неэффективности, обусловленной холостой работой процессора и малой загрузкой внешних устройств (параллельно с процессором работают лишь внешние устройства, требуемые для данной задачи, — как правило, одно устройство).

Главной целью *режима мультипрограммирования*, к рассмотрению которого мы переходим, является достижение максимальной производительности системы, т. е. выполнение совокупности требуемых задач за минимальное время.

В этом режиме диспетчер вначале включает в работу первую из задач, находящихся в главной памяти. Если эта задача потребовала для себя обмена с внешним устройством, то после включения в работу этого устройства управление передается на начало программы второй задачи (перед этим запоминается адрес возврата на программу первой задачи и организуется защита памяти). Таким образом, параллельно работает вторая задача и ведется обмен с внешним устройством для первой задачи. Если во время работы второй задачи произошло внешнее прерывание, свидетельствующее об окончании начатого обмена, то диспетчер организует продолжение работы первой задачи, запоминая адрес возврата во вторую задачу. Если же вторая задача выставила диспетчеру запрос на обмен с внешним устройством, то диспетчер включает в работу это устройство и переходит либо к продолжению выполнения первой задачи (если обмен для нее закончился), либо включает в работу третью задачу. Аналогично включаются в работу и остальные задачи, расположенные в главной памяти.

При описанной мультипрограммной обработке задач *), как правило, удается максимально загрузить внешние устройства (параллельно работают несколько устройств для разных задач) и исключить холостую работу процессора, т. е. эффективно использовать систему при решении совокупности задач.

§ 63. Загрузка системы

Одной из функций операционной системы является вызов в главную память программ решаемых задач. Такой вызов называется *загрузкой памяти*, а соответствующий блок операционной системы — *загрузчиком*. В этом параграфе мы опишем вкратце работу загрузчика.

Программа задачи, которая первый раз решается в системе, обычно имеет вид колоды перфокарт. В начале каждой колоды должна ставиться специальная перфокарта — паспорт задачи. В паспорте указывается номер задачи, примерное время решения, характер работы (отладка, пробный счет, вариантные расчеты и т. д.), требуемый объем главной и внешней памяти, адрес, с которого программа закодирована, перечень нужных внешних устройств и т. д.

Колоды перфокарт для нескольких задач, объединенных вместе, образуют *пакет* задач.

Ввод пакета задач в систему организуется блоком *начальной загрузки*. В этом блоке программы задач последовательно вводятся в главную память и помещаются во внешнюю память системы (в зоны магнитной ленты, на магнитные диски или барабаны). Затем некоторая совокупность из задач пакета вызывается в главную память. Вызов в главную память организуется таким образом, чтобы для совокупности всех вызываемых задач хватило ресурсов системы — главной памяти, внешней памяти, устройств ввода-вывода.

---

*) Режим мультипрограммирования, при котором *последовательно* выполняются части программ разных задач, следует отличать от режима *мультипроцессирования*. В этом последнем режиме программы нескольких задач выполняются параллельно на нескольких процессорах, присоединенных к главной памяти.

На этом кончается начальная загрузка и управление передается диспетчеру, который организует решение задач.

В тот момент, когда кончается выполнение какой-либо из задач, часть ресурсов системы освобождается и диспетчер передает управление загрузчику. Загрузчик осуществляет *текущую загрузку*, вызывая из внешней памяти одну из задач, ожидающих очереди на решение. Эта задача выбирается таким образом, чтобы в совокупности с теми задачами, которые находятся в процессе решения, было бы достаточно ресурсов системы. Затем управление передается диспетчеру, который включает в работу загруженную задачу.

С выполнением загрузки связано решение ряда непростых вопросов, некоторые из них мы сейчас рассмотрим.

Одним из этапов загрузки является выделение для очередной задачи необходимого количества страниц главной памяти. При начальной загрузке память для очередных задач выделяется последовательно — для первой задачи начальные страницы, для второй — следующие и т. д. К моменту текущей загрузки в главной памяти образуются свободные места — «дыры», в одну из которых и загружается очередная задача. При этом, как правило, программа задачи попадает не на те страницы, на которые она закодирована и, следовательно, непосредственно в загруженном виде работать не может. Для того чтобы привести программу к работающему виду, загрузчик, используя начальный адрес программы (адрес начальной страницы), указанный в паспорте задачи, перекодирует все команды со старых страниц на новые.

Заметим, что такой способ настройки программы по месту требует значительного машинного времени. Это время можно свести к минимуму в тех системах, в которых имеется базовая адресация (см. § 57). В этом случае программу пишут в адресах, начинающихся с нуля (относительные адреса), снабжая их признаком базовой адресации. Загрузчик перед включением программы в работу устанавливает базовый адрес на начальную страницу того участка главной памяти, на который помещена программа *).

*) В некоторых системах для тех же целей применяют схемный способ относительной адресации. В этом случае исполнительный адрес, помеченный признаком относительной адресации, образуется как сумма значения счетчика команд и действительного адреса.

Кроме главной памяти, каждая задача требует для своего выполнения определенного набора внешних устройств. При этом зачастую номера этих устройств для некоторых из загружаемых задач будут совпадать. Для того чтобы такие задачи могли работать в пакетном режиме, в операционной системе принимают специальные меры. Номера устройств, указанные в программе самой задачи, считаются математическими и могут не совпадать с действительными (физическими) номерами устройств, которые будут работать в процессе выполнения задачи. При загрузке задаче выделяются определенные внешние устройства и составляется таблица соответствия номеров математических и физических устройств (для данной задачи).

Во время выполнения задачи ее программа обращается к внешнему устройству через диспетчер. Диспетчер по таблице соответствия включает в работу то физическое устройство, номер которого соответствует заданному математическому.

Таким образом, в системе могут работать в пакетном режиме несколько задач, требующих устройств с одинаковыми номерами.

§ 64. Режим разделения времени

До сих пор мы рассматривали в основном *автоматическую* работу вычислительной системы, при которой вмешательство одного человека-оператора в процесс функционирования системы было минимальным. Это вмешательство сводилось в основном к организации ввода пакета задач (например, колоды перфокарт) в главную память.

Новые широкие возможности использования вычислительных систем дает *режим разделения времени*, при котором много людей — потребителей машины — могут оперативно вмешиваться в работу системы, выдавая заявки на решение требуемых задач. Для обеспечения такой работы к главной памяти системы должно быть подсоединено большое количество абонентов, являющихся пультами для работы людей-операторов.

Чаще всего в качестве такого рода пультов применяются специальные электрические пишущие машинки, иногда телетайпы, используемые в телеграфной аппаратуре. Основными частями пультов являются клавиатура

и печатающий механизм. С помощью клавиатуры, содержащей набор русских и латинских букв и служебных знаков, человек-оператор может вводить нужную информацию в главную память; вывод ответной информации из главной памяти на бумажную ленту производится с помощью печатающего механизма. К большой вычислительной системе подсоединяется до нескольких сотен пультов. Пульты могут располагаться как около самой вычислительной системы, так и находиться от нее на расстоянии от десятков до тысяч километров. В последнем случае пульты соединяются с системой телеграфными или телефонными каналами связи.

Рассмотрим, как функционирует вычислительная система в режиме разделения времени.

При работе в этом режиме имеется возможность как вводить с пульта оператора в систему программы решения задач, так и давать заявки на решения задач, уже загруженных в систему. Для простоты мы ограничимся лишь рассмотрением второй (наиболее часто используемой) из этих возможностей.

Будем предполагать, что во внешней памяти уже имеется полный архив из задач, заявки на решение которых могут поступить с пультов (задачи могут вводиться в систему при начальной загрузке с перфокарт).

Все заявки разделим по срочности на две категории; срочные — это те заявки, которые могут быть обслужены системой за время, меньшее некоторого промежутка T (величина порядка нескольких секунд) и несрочные — с большим временем обслуживания.

Операционная система для информации, поступающей с каждого пульта, выделяет свой участок памяти, границы которого записываются в ячейку, играющую роль команды обмена для этого абонента (см. § 59).

Оператор, находящийся за пультом, передает с помощью клавиатуры заявку (входное сообщение), заканчивая ее специальным служебным знаком — «конец сообщения». Входное сообщение поступает в главную память в режиме параллельной работы на выделенное операционной системой поле; при этом устройство управления обменом информации, расшифровывая знак «конец сообщения», формирует сигнал прерывания (еще одно внешнее прерывание).

По этому сигналу диспетчер ставит принятую заявку в конец очереди на обслуживание и возвращает процессор к выполнению прерванной программы.

Очередь на обслуживание состоит из двух очередей — очереди заявок первого приоритета и очереди заявок второго приоритета. В первую из них ставятся срочные заявки, во вторую несрочные.

Рассмотрим, как обслуживаются заявки абонентов. Пусть в некоторый момент времени кончилось решение задачи по одной из срочных заявок. Тогда диспетчер, получив прерывание по соответствующей команде остановки, включает в работу следующую по очереди срочную заявку. Для этого по информации, содержащейся в этой заявке, диспетчер при помощи загрузчика вызывает из внешней памяти нужную задачу и передает ей управление.

При этом диспетчер устанавливает в счетчике относительного времени значение T (см. § 60). Если решение задачи закончилось за время, не превышающее T, то диспетчер аналогичным образом начинает обслуживание следующей по очереди заявки. В противном случае через время T наступает прерывание и незаконченная задача ставится в начало очереди II приоритета. Затем обслуживается следующая заявка из очереди.

После того как обслужены все срочные заявки (I приоритет), диспетчер включает в обслуживание первую из несрочных заявок (II приоритет). Рассмотренный порядок работы диспетчера дает возможность достаточно быстро обслужить срочные заявки.

Обычно задача, решаемая по какой-либо из заявок, вырабатывает в процессе своей работы ответную информацию абоненту. Эта информация называется ответным сообщением.

По требованию от задачи диспетчер формирует команду обмена для соответствующего абонента, в которой указываются границы в главной памяти ответного сообщения. Это сообщение передается абоненту.

Как в режиме мультипрограммирования, так и в режиме разделения времени, диспетчер ведет протоколирование работы системы — на печатающие устройства абонентов и на специальное «протокольное» печатающее устройство выдаются справочные сведения о каждой из решенных

задач — шифр абонента, название задачи, дата, астрономические времена начала и конца выполнения заявки и т. д. При этом показания времени диспетчер снимает с электронных часов.

Нами был рассмотрен лишь один из возможных способов организации работы сложной вычислительной системы.

Отметим, что среди вычислительных систем существует большое разнообразие как по составу и взаимосвязи технических средств, так и по организации общесистемного математического обеспечения.

ПРИЛОЖЕНИЕ 1

ЭЛЕКТРОННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ ТИПА М-20

§ 65. Общая характеристика машин типа М-20

Машины типа М-20, программирование на которых рассматривалось в этой книге, принадлежат к вычислительным машинам средней мощности. Их быстродействие составляет в среднем около 20 тысяч операций в секунду.

Первой по времени создания из машин этого типа является ламповая машина М-20. Следующими представителями этого типа служат полупроводниковые машины БЭСМ-3, БЭСМ-4 и М-220, имеющие те же способы представления чисел и команд, почти ту же систему команд и примерно такое же быстродействие. Главное их отличие от машины М-20 в том, что они осуществлены на других физических элементах, имеют больший объем внутренней и внешней памяти и новые средства ввода-вывода. Сейчас выпускаются усовершенствованные модели машин этого типа.

Таблица 1.65 содержит характеристики основных устройств большинства машин рассматриваемого типа. В следующих параграфах будут рассмотрены некоторые особенности системы команд этих машин.

§ 66. Система команд

Во второй части книги рассмотрены основные элементарные операции машин типа М-20. В табл. 1.66—5.66 приведена сводка всех операций этих машин. Операции разбиты на пять основных групп, для каждой из которых приводится отдельная таблица. В ней даны наименование, обозначение и код каждой операции, условия выработки сигнала $\omega = 1$ или аварийной остановки (там, где они есть), а также указан параграф книги, в котором эта операция подробно описана.

Как обычно, машинное слово из ячейки с адресом c обозначается через c. В тех случаях, когда нужно рассматривать часть слова, используются обозначения:

$M(c)$ — мантисса числа или адресная часть машинного слова c (разряды 36-1);

$p(c)$ — условный (машинный) порядок числа c (разряды 43-37);

$П(c)$ — операционная (порядковая) часть машинного слова c (разряды 45-37).

Таблица 1.65

Основные устройства машин типа М-20

Характеристика устройства		М-20	БЭСМ-4	М-220	М-222
Число кубов оперативной (внутренней) памяти (1 куб = 4096 ячеек)		1	2	1—4	8
Число секций магнитных барабанов, (1 секция = 4096 ячеек)		3—4	16	8 или 16	16 или 32
Магнитофоны (магнитные ленты)	Количество магнитофонов	4	4	4—16	16
	Количество зон на ленте	256	2048	4096	4096
	Емкость одной ленты в ячейках	$75 \cdot 10^3$	10^6	10^6	10^6
Количество устройств цифровой печати (узкая бумажная лента)		1	1	—	—
Количество устройств алфавитно-цифровой печати (широкая лента)		—	1	1	1
Число ячеек буфера для выдачи результатов		512	512	1024	1024
Количество выходных перфораторов двоичных карт		1	1	1	1
Количество устройств ввода двоичных карт		1	1	1	—
Количество устройств ввода десятичных карт		—	—	—	1
Количество устройств вывода на перфоленту		—	—	—	1

Арифметические операции, описанные в табл. 1.66, обладают следующими двумя особенностями:

а) Если в результате выполнения какой-либо из этих операций получается

$$M(c) = 0,$$

то и разряды порядковой части, кроме метки (45 разряд), полагаются равными нулю.

б) Метка (45 разряд) результата любой арифметической операции образуется как логическая сумма меток двух чисел, участвующих в операции. В операциях сложения порядка с адресом и вычитания из порядка адреса (коды 06 и 46) метка c полагается равной метке b.

В табл. 6.66 приведены условные обозначения кодов всех элементарных операций. В левом столбце таблицы даны левые, а в верхней строке — правые цифры числовых кодов. Эта таблица является удобным вспомогательным средством при кодировании.

Из операций, общих для всех машин типа М-20, во второй части книги не были рассмотрены операции с кодами 00, 67, 20, 35.

Операция с кодом 00 *(пересылка)* записывается в виде

$$a \Rightarrow c.$$

Ее выполнение состоит в пересылке содержимого ячейки a в ячейку c. Она может быть заменена командой

$$a \vee 0 = c$$

или

$$0 \vee a = c,$$

которую мы записывали (см. § 31) в виде

$$a = c.$$

Операция *циклического сдвига* (код 67) записывается в виде

$$\rightleftarrows a = c$$

и используется обычно лишь для проверки работы машины в инженерных тестах. При ее выполнении содержимое ячейки a сдвигается на 24 разряда влево внутри 48-разрядной сетки с циклическим переносом разрядов, вышедших за пределы 48-разрядной сетки, в младшие разряды (с первого по двадцать первый). Результат такого сдвига заносится в ячейку c.

Обе описанные операции являются двухадресными; значение второго адреса на выполнение команд не влияет. Кроме того, эти команды не изменяют управляющего сигнала ω, выработанного в результате выполнения предыдущей операции.

Две другие команды будут описаны в следующем параграфе.

Машина М-20 отличается от остальных машин того же типа характером использования команд с кодами 40, 60, 17, 37, 57. Последние три операции на машине М-20 выполняются как команда *стоп* (код 77). Команды с этими кодами на машинах БЭСМ-4 и М-220 будут описаны ниже (см. §§ 69, 70). Команды с кодами 40, 60 на машине М-20 являются одноадресными командами засылки нуля по правому адресу; содержимое первого и второго адреса этих команд на их выполнение не влияет. На машинах БЭСМ-4 и М-220 эти коды используются для команд условной передачи управления по PA (см. § 25).

Таблица 1.66

Арифметические операции

Название операции	Разновидности	Обозначение	Код	Условия при которых		Где описана
				$\omega=1$	Авост	
Сложение	Обычное	$a+b=c$	01	$c<0$	$p(c)>177_8$	§ 11 § 21
	Без округления	$a+(БО)\ b=c$	21			
	Без нормализации	$a+(БН)\ b=c$	41			
	Без округления и нормализациии	$a+(БОН)\ b=c$	61			
Вычитание	Обычное	$a-b=c$	02	$c<0$	$p(c)>177_8$	
	Без округления	$a-(БО)\ b=c$	22			
	Без нормализации	$a-(БН)\ b=c$	42			
	Без округления и нормализации	$a-(БОН)\ b=c$	62			
Вычитание абсолютных величин	Обычное	$\|a\|-\|b\|=c$	03	$c<0$	—	
	Без округления	$\|a\|-(БО)\ \|b\|=c$	23			
	Без нормализации	$\|a\|-(БН)\ \|b\|=c$	43			
	Без округления и нормализации	$\|a\|-(БОН)\ \|b\|=c$	63			

Умножение	Обычное Без округления Без нормализации Без округления и нормализации	$a \cdot b = c$ $a \cdot (БО)\ b = c$ $a \cdot (БН)\ b = c$ $a \cdot (БОН)\ b = c$	05 25 45 65	$p(c) > 100_8$ и $M(c) \neq 0$	$p(c) > 177_8$	
Деление	Обычное Без округления	$a : b = c$ $a : (БО)\ b = c$	04 24	$p(c) > 100_8$	$p(c) > 177_8$ или $M(c) = 0$	§ 11 § 21
Извлечение квадратного корня	Обычное Без округления	$\sqrt{a} = c$ $\sqrt{a}\ (БО) = c$	44 64	$p(c) > 100_8$	$a < 0$	
Изменение порядка	Сложение адреса с порядком Сложение порядков Вычитание адреса из порядка Вычитание порядков	$\bar{a}\ (+)\ b = c$ $a\ [+]\ b = c$ $\bar{a}\ (-)\ b = c$ $a\ [-]\ b = c$	06 26 46 66	$p(c) > 100_8$	$p(c) > 177_8$	§ 28
Вывод младших разрядов произведения	—	мл. разр. $\Rightarrow c$	47	$M(c) = 0$	—	§ 21

Таблица 2.66

Фиксированные и логические операции

Название операции	Обозначение	Код	Условия, при которых $\omega = 1$	Где описана
Фиксированные действия				
Сложение мантисс	$a+,\ b=c$	13	$M(c) > 2^{36}$	§ 23
Вычитание мантисс	$a-,\ b=c$	33	$M(a) < M(b)$	
Сложение операционных частей	$a,+b=c$	53	$П(c) > 2^{9}$	
Вычитание операционных частей	$a,-b=c$	73	$П(a) < П(b)$	
Циклическое сложение	$a \oplus b=c$	07	$M(c) > 2^{36}$	§ 28
Циклическое вычитание	$a \ominus b=c$	27	$M(a) < M(b)$	
Сдвиги				
Сдвиг мантиссы по адресу	$\bar{a} \rightarrow,\ b=c$	14	$M(c)=0$	§ 30
Сдвиг мантиссы по порядку	$a[\rightarrow,]\ b=c$	34		
Сдвиг слова по адресу	$\bar{a} \rightarrow b=c$	54	$c=0$	
Сдвиг слова по порядку	$a[\dashrightarrow]\ b=c$	74		
Циклический сдвиг	$a \rightleftarrows c$	67	—	§ 65
Поразрядные логические операции				
Сверка	$a \curlywedge b=c$	15	$c=0$	§ 31
Сверка со стопом	$a \curlywedge_{\text{стоп}} b=c$	35		§ 67
Логическое умножение	$a \wedge b=c$	55		§ 31
Логическое сложение	$a \vee b=c$	75		

Таблица 3.66

Операции передачи управления

Название операции	Обозначение	Код	Где описана
Передачи управления с пересылкой			
Безусловная передача управления с возвратом	*БВ a b c*	16	§ 34
Условная передача управления при $\omega = 1$	*У1* $\overrightarrow{a\ b\ c}$	36	§ 12
Безусловная передача управления	*Б* $\overrightarrow{a\ b\ c}$	56	
Условная передача управления при $\omega = 0$	*УО* $\overrightarrow{a\ b\ c}$	76	
Условные передачи управления по *РА*			
Передача управления при $РА < \bar{a}$	$РА < \bar{a}\ b\ c$	12	§ 25
Передача управления при $РА \geqslant \bar{a}$	$РА \geqslant \bar{a}\ b\ c$	32	
Передача управления при $РА < [a]$	$[РА] < a\ b\ c$	40	
Передача управления при $РА \geqslant [a]$	$[РА] \geqslant a\ b\ c$	60	
Условные передачи управления по *РА* и ω			
Передача управления при $РА < \bar{a}$ и $\omega = 1$	(*1*) $РА < \bar{a}\ b\ c$	11	§ 25
Передача управления при $РА \geqslant \bar{a}$ и $\omega = 1$	(*1*) $РА \geqslant \bar{a}\ b\ c$	31	
Передача управления при $РА < \bar{a}$ и $\omega = 0$	(0) $РА < \bar{a}\ b\ c$	51	
Передача управления при $РА \geqslant \bar{a}$ и $\omega = 0$	(0) $РА \geqslant \bar{a}\ b\ c$	71	
Уход, возврат, стоп			
Уход	уход *m b c*	50...30	§ 69
Возврат	возврат *m b* 0	50...34	
Стоп	стоп *a b c*	77	§ 11 § 67

Таблица 4.66

Изменение *РА, РП* и пересылки

Название операции	Обозначение	Код	Где описана
Изменение *РА, РП*			
Занесение *РА* по адресу	*РА* $\bar{a}$ $\bar{b}$ *c*	52	§ 25
Занесение *РА* по слову	[*РА*] $\bar{a}$ *b c*	72	
Изменение регистров приращений	*ИРП* $\bar{a}$ $\bar{b}$ *c*	57	§ 69
Пересылки			
Пересылка обычная	$a \Rightarrow c$	00	§ 66
Выборка с *КЗУ*	*КЗУ* $a \Rightarrow c$	20	§ 67
Динамическая засылка	*ДЗ a b c*	37	§ 70
Динамическая выборка	*ДВ a b c*	17	

Таблица 5.66

Обращение к внешним устройствам

Название операции	Обозначение	Код	Где описана
Ввод с перфокарт (со стопом)	*ВС a b c*	10	§ 39
Ввод с перфокарт	*В a b c*	30	
Обмен с внешней памятью	*М* (*а*) $\overline{УЧ}$ b_1 a_n *М* (*б*) a_1 *c* Σ	50 70	§ 39, § 67

Таблица 6.66

Коды элементарных операций

	.0	.1	.2	.3	.4	.5	.6	.7
0.	$\Rightarrow$	+	−	|−|	:	·	(+)	$\oplus$
1.	*ВС*	(1) *РА* <	*РА* <	+,	→,	$\nearrow$	*БВ*	*ДВ*
2.	*КЗУ*	+(*БО*)	−(*БО*)	|−(*БО*)|	:(*БО*)	·(*БО*)	[+]	$\ominus$
3.	*В*	(1) *РА* $\geqslant$	*РА* $\geqslant$	−,	[→,]	$\nearrow_{\text{стоп}}$	*У1*	*ДЗ*
4.	[*РА*] <	+(*БН*)	−(*БН*)	|−(*БН*)|	$\sqrt{}$	·(*БН*)	(−)	мл. р.
5.	*М* (*а*)	(0) *РА* <	*РА*	,+	→	$\wedge$	*Б*	*ИРП*
6.	[*РА*] $\geqslant$	+(*БОН*)	−(*БОН*)	|−(*БОН*)|	$\sqrt{}$(*БО*)	·(*БОН*)	[−]	$\rightleftharpoons$
7.	*М* (*б*)	(0) *РА* $\geqslant$	[*РА*]	,−	[→]	$\vee$	*У0*	*стоп*

§ 67. Операции, связанные с работой на пульте

Описание основных регистров пульта управления машин типа М-20 было сделано в § 42. Отметим некоторые различия в наименованиях и обозначениях, а также в расположении регистров на пультах конкретных машин.

Регистры первого и второго адресов обозначены *Р*1 и *Р*2. Регистр третьего адреса носит название *регистра результата* и обозначен *РР*. На пультах всех машин типа М-20 он находится в центре вертикальной панели, являясь самой верхней строкой. Регистр команд (*РК*) является нижней строкой сорокапятиразрядных регистров. Кроме упомянутых регистров, на вертикальной панели пульта помещен еще сумматор (*СМ*). Порядок расположения сорокапятиразрядных регистров на вертикальной панели пультов всех машин (сверху вниз) такой: *РР*, *Р*1, *СМ*, *Р*2, *РК*.

Регистр ω находится слева, на машинах М-20 и М-220 на уровне *РК*, а на машине БЭСМ-4 — на уровне *СМ*. Регистр «Авост» находится на пульте М-20 над *РР* в центре, на пульте М-220 — над *РР* справа, а на БЭСМ-4 — под *РК* слева.

Счетчик команд на всех машинах типа М-20 называется *командным регистром адреса* и обозначен *КРА*. На пульте М-220 регистры *РА* и *КРА* расположены на вертикальной панели справа от основных регистров один под другим — *РА* на уровне *Р*2 и *КРА* на уровне *РК*. На пульте М-20 между *РА* и *КРА* находится сумматор адреса *СМА*, служащий для образования исполнительного адреса. Поэтому там *РА* находится на уровне *СМ*, а *СМА* — на уровне *Р*2. На пульте БЭСМ-4 эти

три регистра находятся в центре вертикальной панели, под регистром *РК* в таком порядке (сверху вниз): *КРА*, *СМА*, *РА*.

Наименование и размещение остальных регистров существенно только для инженеров, и мы на нем останавливаться не будем.

В команде *стоп* (код 77) мы обычно считали адреса команды нулевыми. Их использование связано с пультом управления машины. Если написать

$$стоп\ a\ b\ c,$$

то после остановки машины по команде *стоп* на пульт вызывается содержимое ячеек a и b, оно загорается в регистрах $P1$ и $P2$. Если снова нажать *пуск*, то управление передается в ячейку $Я + 1$ и при этом очищается ячейка c, т. е. в нее засылается нуль.

С работой за пультом связаны еще две команды, общие для всех машин типа М-20, о которых упоминалось в § 66. Из них в первую очередь следует отметить команду *сверки со стопом* (код 35), которая выполняется так же, как и команда *сверки* (код 15, см. § 31). Эту команду мы будем обозначать

$$a \not\sim_{стоп}\ b = c.$$

При ее выполнении производится поразрядная сверка машинных слов из ячеек с адресами a и b. В совпадающих разрядах записывается нуль, в различающихся — единица и полученное слово записывается в ячейке c. Если слова a и b не совпадают, то машина останавливается и оба слова загораются на пульте в регистрах $P1$ и $P2$. Эта команда обычно используется не программистами, а инженерами в различных тестах для проверки правильности работы машины.

Последняя из общих для всех машин и не описанных ранее команд связана с использованием *КЗУ*. Как указывалось в § 42, на горизонтальной панели пульта имеется четыре клавишных запоминающих устройства (*КЗУ*). Их называют еще *ДЗУ* (диодные запоминающие устройства) или же *РПУ* (регистры пульта управления).

Наряду с описанным в § 42 режимом работы с *КЗУ*, как с обычными ячейками памяти, возможен и другой режим, когда содержимое *КЗУ* переносится в обычную ячейку памяти с помощью специальной команды выборки из *КЗУ* (код 20). Эту команду мы будем обозначать

$$КЗУ\ k \Rightarrow c,$$

где k — номер *КЗУ*, указываемый при кодировке в младших разрядах первого адреса. Содержимое второго адреса команды на выполнение операции не влияет. Отметим еще, что на машине М-220 имеется не четыре, а шесть *КЗУ* с номерами *КЗУ*-1 — *КЗУ*-6.

§ 68. Операции для работы с внешними устройствами

Операция обмена с внешней памятью была коротко описана в § 39. Мы записывали ее в двух ячейках в виде

$$M\ (a)\ \overline{УЧ}\ b_1\ a_n$$
$$M\ (б)\ a_1\ c\ \Sigma.$$

Таблица 1.68

Условное число команды M (a)

№ разряда	Обозначение	Функции при содержимом разряда = 1
36	*БМ*	Блокируется запись во внутреннюю память или чтение из памяти (фиктивный обмен, блокировка МОЗУ)
35	*БК*	Не происходит сравнения контрольных сумм при чтении, нет передачи управления на c (блокировка контроля)
34	*ОН*	Магнитная лента движется в обратном направлении
33	*БО*	При несовпадении контрольных сумм во время чтения блокируется остановка машины (управление на c)
32	*ПФ*	Вывод информации на перфокарты
31	*ПЧ*	Вывод информации на узкую бумажную ленту (печать)
30	*РЛ*	Работа магнитофона в режиме разметки ленты
29	*МЛ*	Работа магнитной ленты
28	*МБ*	Работа магнитного барабана
27	*ЗП*	Запись (при $ЗП = 0$ — чтение)
26—25	k	k означает номер блока внешних устройств (магнитофон, барабан и т. п.)

Таблица 2.68

Условные числа команды $M(a)$ для основных операций обмена с внешней памятью (без разрядов 26, 25)

Вид устройства	Основная операция	Разновидности	Условное число
Магнитный барабан	Запись	с контролем	0014
		без контроля	2014
	Чтение	действительное с контролем	0010
		действительное с контролем и блокировкой остановки	0410
		действительное без контроля	2010
		фиктивное с контролем	4010
		фиктивное с контролем и блокировкой остановки	4410
Магнитофоны, магнитные ленты *)	Запись в прямом направлении		0024
	Запись в обратном направлении		1024
	Чтение в прямом направлении	действительное	0020
		действительное с блокировкой остановки	0420
		фиктивное	4020
		фиктивное с блокировкой остановки	4420
	Чтение в обратном направлении	действительное	1020
		действительное с блокировкой остановки	1420
		фиктивное	5020
		фиктивное с блокировкой остановки	5420
	Разметка ленты в прямом направлении	с записью	0040
		без записи	4040
	Разметка ленты в обратном направлении	с записью	1040
		без записи	5040

*) Все основные операции с магнитной лентой осуществляются с контролем.

Здесь (a_1, a_n) — границы обмениваемого массива оперативной памяти, b_1 — его начало во внешней памяти, Σ — адрес ячейки для записи накопленной при обмене в *арифметике* циклической суммы и c — адрес команды, которой передается управление, если, в случае чтения, сумма Σ не совпадает с суммой во внешней памяти.

Характер операции обмена определяется условным числом, занимающим первый адрес команды $M(a)$ (разряды 36—25). Отдельные разряды этого числа или их группы играют определенную функциональную роль.

В табл. 1.68 показано функциональное значение каждого из разрядов условного числа. Таблица 2.68 содержит условные числа основных операций обмена между внутренней (оперативной) и внешней (магнитные барабаны и ленты) памятью.

Таблица 3.68

Условные числа команды $M(a)$ для основных операций ввода-вывода

Устройство	Вид операции	Условное число
Буфер	Накопление	2300
Устройство цифровой печати (узкая бумажная лента)	Десятичная печать	2100
	Восьмеричная печать	2500
Алфавитно-цифровое печатающее устройство (широкая бумажная лента)	Алфавитно-цифровая печать	2140
Вывод на двоичные карты	Перфорация с выдачей контрольной суммы	0200
	Перфорация без контрольной суммы	2200
Ввод с десятичных карт	Старт-стопный ввод	3260
	Непрерывный ввод	2260
Ввод с перфоленты	Старт-стопный ввод	3240
	Непрерывный ввод	2240
Вывод на перфоленту	Перфорация	2244

Отметим, что кроме обычного считывания с барабана или ленты, для контроля правильности записанного во внешнюю память массива может производиться так называемое *фиктивное считывание*, при котором обмениваемый массив фактически в оперативную память не поступает, а происходит лишь его циклическое суммирование и сверка контрольной суммы.

Два младших разряда условного числа отведены под номер блока внешней памяти. Этого достаточно для четырех барабанов или лент, которые есть у машины М-20. У машин БЭСМ-4 и М-220, имеющих большую внешнюю память (см. табл. 1.65), для указания номера блока внешней памяти приходится пользоваться присоединенной адресацией (см. § 57), с помощью специальных трехразрядных регистров приращений *РПМБ* и *РПМЛ*. Работа с этими регистрами будет описана в следующем параграфе.

В таблице 3.68 приведены условные числа, используемые при выводе или вводе информации. У машин типа М-20 при выдаче информации на печать или перфорацию используется специальное буферное запоминающее устройство (емкость буфера — см. табл. 1.65); выдаваемый массив может либо по частям накапливаться на буфере, либо переноситься на буфер с непосредственной выдачей на перфорацию или печать. Во время занесения информации на буфер процессор машины не работает, но во время выдачи с буфера уже выполняются команды, следующие за командами *М* (*а*), *М* (*б*) обращения к буферу, т. е. происходит параллельная работа процессора и устройства вывода.

§ 69. Операции для работы с расширенной памятью

Структура команды машины М-20 отводит для записи адреса 12 двоичных разрядов, и тем самым допускает прямую адресацию $2^{12} = 4096$ ячеек внутренней (оперативной) памяти — один куб. У машины М-222 внутренняя память, в зависимости от комплектации, может иметь до 8 таких кубов, т. е. до 32768 ячеек. Для работы с такой расширенной памятью в этих машинах используется способ *присоединенной адресации*, описанный в § 57.

Адрес ячейки внутренней памяти считается пятнадцатиразрядным; старшие три разряда определяют номер куба, младшие двенадцать — адрес ячейки в этом кубе. В команде — или счетчике команд (*КРА*) — указывается адрес ячейки в кубе, а номер куба записывается в дополнительном регистре.

Выполнение команды в трехадресной машине требует четырех обращений к памяти:

а) выборка команды в соответствии со счетчиком команд (*КРА*);

б) выборка содержимого по первому исполнительному адресу (*АI*);

в) выборка содержимого по второму исполнительному адресу (*АII*);

г) засылка результата операции по третьему исполнительному адресу (*АIII*).

Для каждого из этих четырех обращений нужно указать номер куба, из которого выбирается ячейка, так что требуется четыре до-

полнительных трехразрядных регистра. Их называют *регистрами приращений* и обозначают соответственно символами *РП КРА*, *РП АI*, *РП АII*, *РП АIII*. Кроме них в машине имеется еще два регистра приращений для работы с большими номерами магнитного барабана и магнитной ленты. Их обозначают *РП МБ* и *РП МЛ*.

Таким образом, выборка команды происходит по пятнадцатиразрядному адресу, составленному из содержимого *РП КРА* — три разряда — и содержимого счетчика команд (*КРА*) — двенадцать разрядов. Аналогично происходит работа и с исполнительными адресами команды. Несколько иначе выполняются только команды передачи управления и обмена; о них речь будет идти ниже.

Запись или изменение содержимого регистров приращений происходит с помощью специальной команды *Изменение регистров приращений*, которую мы будем записывать в виде

$$ИРП\ \bar{a}\ \bar{b}\ c.$$

Здесь *ИРП* — обозначение команды (код команды — 57), c — адрес ячейки памяти, $\overline{a\,b}$ — 24-разрядное двоичное условное число, занимающее первый и второй адреса, структура которого приведена на рис. 48.

Шесть левых разрядов условного числа (36—31) содержат признаки, показывающие, нужно ли в один из регистров приращений вносить указанное справа содержимое. Остальные 18_{10} разрядов (30—13) состоят из шести триад — для каждого из регистров приращений. Регистры приращений, их признаки и соответствующие им триады считаются расположенными в таком порядке: *РП МБ*, *РП МЛ*, *РП КРА*, *РП АI*, *РП АII*, *РП АIII*. Так, признак, стоящий в 36 разряде, соответствует *РП МБ*, в 34 разряде — *РП КРА* и т. д. Аналогично, триада из 27-25 разрядов соответствует *РП МЛ*, а 18-16 — *РП АII*.

Команда изменения регистров приращений выполняется следующим образом. Если признак, соответствующий данному регистру приращений, равен 1, то в этот регистр заносится содержимое соответствующей триады. Если же признак равен нулю, то содержимое этого регистра приращений остается без изменения, независимо от содержимого соответствующей триады в команде. Например, если единицы стоят в 36, 34 и 32 разрядах признаков, а в 35, 33 и 31 стоят нули, то в регистры приращений *РП МБ*, *РП КРА* и *РП АII* вносятся соответственно содержимое триад из 30-28, 24-22 и 18-16 разрядов. При этом регистры *РП МЛ*, *РП АI* и *РП АIII* остаются без изменения, независимо от содержания остальных триад условного числа.

Одновременно в ячейку памяти c, указанную в третьем адресе *) команды *ИРП*, записывается команда с тем же кодом *ИРП*, нулевым правым адресом и условным числом, занимающим левый и средний адреса

$$ИРП\ \bar{d}\ \bar{e}\ 0,$$

---

*) Пятнадцатиразрядный адрес этой ячейки формируется из исполнительного адреса c присоединением содержимого *РП АIII*, которое было к моменту выполнения операции *ИРП*.

Разряд	36	35	34	33	32	31	30 – 28	27 – 25	24 – 22	21 – 19	18 – 16	15 – 13
Содержание	ПР МБ	ПР МЛ	ПР КРА	ПР А I	ПР А II	ПР А III	РП МБ	РП МЛ	РП КРА	РП А I	РП А II	РП А III

Рис. 48.

Разряд	36	35	34	33	32	31	30 – 28	27 – 25	24 – 22	21 – 19	18 – 16	15 – 13
Содержание	ПР МБ	ПР МЛ	ПР КРА	ПР А I	ПР А II	ПР А III	РП МБ′	РП МЛ′	РП КРА′	РП А I′	РП А II′	РП А III′

Рис. 49.

Разряд	45 – 44	43	42 – 40	39 – 37	36 – 34	33 – 31	30 – 28	27 – 25	24 – 13	12 – 1
Содержание	–	ω	РП МБ	РП МЛ	РП КРА	РП А I	РП А II	РП А III	КРА	РА

Рис. 50.

структура которого указана на рис. 49. Признаки в этом условном числе те же, что и в исходной команде *ИРП*, а на месте триад, заносимых в регистры приращений, стоит содержимое соответствующих регистров до начала выполнения операции. Иначе говоря, в ячейке *c* формируется команда восстановления прежнего состояния регистров приращения, подобно тому, как это делается с регистром адреса (см. § 27).

При выполнении команды передачи управления пятнадцатиразрядный второй адрес формируется присоединением к исполнительному второму адресу не содержимого *РП* A II, как при всех остальных операциях, а содержимого *РП КРА*.

Необходимо иметь в виду, что при изменении *РП КРА* следующая после *ИРП* команда выбирается из прежнего куба и лишь после этого начинает действовать новое содержимое *РП КРА*. Пусть, например, команда изменения регистров приращений помещается в *s*-м кубе в ячейке x, и в результате ее выполнения содержимое *РП КРА* становится равным *t*. Тогда после команды *ИРП* из ячейки x выполнится команда из ячейки $x + 1$ куба *s*, а затем — команда из ячейки $x + 2$ куба *t*. Если, в частности, после команды *ИРП* стоит команда передачи управления ячейке *b*, то порядок выполнения команд будет таким. При безусловной передаче управления или при условной передаче с выполненным условием (т. е. когда передача управления фактически происходит) будут выполняться команды: команда x куба *s* (*ИРП*), команда $x + 1$ из куба *s* (передача управления) и команда *b* из куба *t*. Содержимое *РП* A II при этом игнорируется. При невыполнении условия передачи управления будут выполняться команды: команда x из куба *s*, команда $x + 1$ из куба *s* и команда $x + 2$ из куба *t*, как это и было сказано выше.

В заключение укажем, что *РП* A III не воздействует на третий адрес команды *M* (*a*) *) при обмене с внешней памятью с помощью команд *M* (*a*), *M* (*б*). Номер куба внутренней памяти для обмениваемого массива определяется первым адресом команды *M* (*б*) с помощью *РП* A I.

Изменить содержимое регистров приращений можно и при помощи двух других операций, *Уход* и *Возврат*. Операция *Уход* состоит в передаче управления с запоминанием состояния машины; *Возврат* восстанавливает прежнее состояние машины. На машине с расширенной внутренней памятью эти операции играют роль, аналогичную операции безусловной передачи управления с возвратом для машин типа М-20. Рассмотрим эти операции подробнее.

Состояние машины в данный момент времени определяется содержимым основных регистров устройства управления. Для машины БЭСМ-4 и М-220 состояние полностью описывается 43-разрядным двоичным словом, разряды которого распределены по схеме рис. 50.

Код операции передачи управления с запоминанием состояния машины занимает не только 42-37 разряды, отведенные для записи кода всех остальных команд, но еще и шесть младших разрядов (30-25) первого адреса. Две триады старших разрядов первого адреса (36-31) играют роль условного числа. В содержательных обозначениях операцию

---

*) Если только младшие шесть разрядов условного числа команды *M* (*a*) не равны 30_8 или 34_8.

передачи управления с запоминанием состояния машины будем записывать так:

Уход d_1d_2 *b c.*

При выполнении этой команды в *РП КРА* заносится триада d_1, в *РП АIII* — триада d_2. Затем управление передается в ячейку *b* куба d_1, а в ячейке *c* куба d_2 запоминается по схеме рис. 50 состояние машины перед выполнением команды. При кодировке команды *Уход* на месте кода пишется 50_8, а в младших разрядах первого адреса — 30_8.

Операция восстановления состояния машины записывается в виде

Возврат d_1d_2 *b c.*

Код операции *Возврат*, как и предыдущей, занимает разряды 42-37 — здесь записывается по-прежнему 50_8 — и разряды 30-25 первого адреса, где, в отличие от *Ухода*, записывается 34_8. По этой команде содержимое ячейки *b* куба d_1 рассматривается как состояние машины, в соответствии со схемой рис. 50. При выполнении команды содержимое разрядов ячейки *b* куба d_1 рассылается в соответствующие регистры устройства управления машины. Составляющие d_2 и *c* на выполнение операции не влияют.

§ 70. Динамические пересылки

Как мы видели, память для массивов программ, написанных на алгоритмических языках, выделяется транслятором. Если массивы, используемые в программе, являются динамическими, или не помещаются во внутренней памяти, то полезно использовать динамическое распределение памяти. Принцип динамического распределения памяти состоит в следующем.

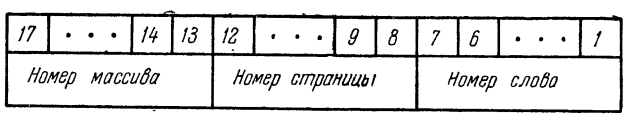


Рис. 51.

Для всех массивов транслируемой программы выделяется место во внешней памяти (магнитные барабаны и магнитные ленты). Вводится единая адресация поля внешней памяти, выделенной транслятором для массивов. Для этого поле памяти делится на 32 массива, каждый может содержать до 32 страниц, в каждой странице — 128 сорокапятиразрядных слов. Тогда можно организовать сплошную нумерацию ячеек внешней памяти при помощи 17-разрядных *математических адресов* по схеме, изображенной на рис. 51.

Для каждой страницы транслятор выделяет 128 последовательных ячеек внешней памяти. В машине хранится таблица распределения страниц, в которой каждому 10-разрядному *математическому номеру* страницы ставится в соответствие ее *физический номер* на магнитной

ленте или барабане (номер ленты или барабана и номер зоны или адрес начальной ячейки страницы).

Чтобы обеспечить доступ программе к содержимому отдельных ячеек массивов, применяется следующий алгоритм. Во внутренней памяти транслятором выделяется рабочее поле на несколько страниц. На это поле вызываются из внешней памяти страницы нужных массивов. С рабочего поля и происходит выборка величин, нужных для работы команд программы, и засылка результатов работы отдельных команд. Если же нужного массива на рабочем поле нет, то следует организовать чтение его из внешней памяти; это может привести к необходимости стирания на оперативном поле страниц другого массива. Эти страницы перед стиранием должны быть перенесены во внешнюю память.

Для реализации выборки 45-разрядных слов из ячеек рабочего поля, заданных математическими адресами, и засылки в ячейки этого поля в машинах БЭСМ-4 и М-220 введены операции динамических пересылок с кодами 17 и 37. Эти операции записываются в виде

$$ДВ\ a\ b\ c$$

и

$$ДЗ\ a\ b\ c,$$

где *ДВ* и *ДЗ* — коды операций *динамической выборки* и *динамической засылки*, а a, b, c — адреса трех ячеек внутренней памяти. Выполняются эти операции следующим образом.

Мантиссы слов из ячеек a и b складываются фиксированным образом и младшие 17 разрядов этой суммы принимаются за математический адрес. При помощи алгоритма, который будет описан ниже, по этому математическому адресу определяется физический адрес соответствующей ячейки рабочего поля (если соответствующая страница находится на рабочем поле). Затем по команде с кодом *ДВ* (17) происходит выборка содержимого этой ячейки рабочего поля и пересылка его в ячейку с адресом c. По команде с кодом *ДЗ* (37) содержимое ячейки с адресом c засылается в ячейку рабочего поля с ранее сформированным адресом. Если же нужной для выборки или засылки страницы нет на рабочем поле, то управление передается в ячейку 0021, с которой начинается программа, обеспечивающая чтение нужной страницы из внешней памяти на рабочее поле.

Алгоритм определения физического адреса ячейки на рабочем поле по ее математическому адресу использует две таблицы, хранящиеся во внутренней памяти: *таблица характеристик массивов* и *таблица характеристик страниц*.

Таблица характеристик массивов расположена в 32 ячейках с адресами 0140—0177. Массиву с номером M соответствует характеристика с адресом $0140 + M$, имеющая структуру, изображенную на рис. 52. В 32-разрядной шкале массива каждой странице соответствует один разряд, причем единицами помечены страницы, находящиеся в данный момент на оперативном поле (нумерация страниц от 0 до 31; им соответствуют разряды 13-44 на рис. 52). В III адресе характеристики массива содержится адрес начала таблицы характеристик страниц этого массива, находящихся на оперативном поле (длина этой таблицы равна числу единиц в шкале массива). При этом разрядам шкалы, содержащим

единицы, соответствуют характеристики, которые располагаются в идущих подряд ячейках памяти (в порядке возрастания номеров разрядов).

Характеристика страницы имеет вид, изображенный на рис. 53. Здесь в разрядах 1-12 располагается адрес начала страницы на рабочем поле, в разрядах 13-36 — счетчик числа выполнений динамических пересылок в программе («часы»), а в 45-м разряде — признак записи (1 соответствует записи в ячейку страницы, 0 — выборке).

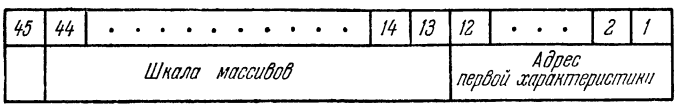


Рис. 52.

При помощи этих таблиц в машине выполнение команд динамических пересылок реализуется схемно следующим алгоритмом. Из 17-разрядного математического адреса, образуемого сложением младших разрядов машинных слов из ячеек a и b, выделяются старшие 5 разрядов, образующие номер массива M. По адресу $0140 + M$ выбирается характеристика массива с номером M. Затем из математического адреса выделяется номер страницы P (разряды 8-12).

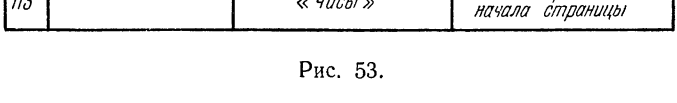


Рис. 53.

Отыскивается разряд с номером P в шкале массива. Если он равен 0, то управление передается в ячейку 0021, с которой начинается программа, реализующая вызов нужной страницы из внешней памяти на рабочее поле *). При этом в ячейке 0010 формируется команда

$$БВ\ 0\ \varkappa\ 0,$$

дающая возможность возвратиться к выполнению исходной команды с адресом $\varkappa$ обращения к динамической пересылке. В случае, если разряд с номером P в шкале массива равен 1, подсчитывается число s всех единичных разрядов шкалы, находящихся левее P-го. Сложением с младшими двенадцатью разрядами характеристики массива определяется адрес характеристики нужной страницы (напомним, что в рассматривае-

*) Краткие сведения об алгоритме работы этой программы, называемой административной системой, см. ниже.

мом случае страница находится на рабочем поле). Наконец, сумма младших двенадцати разрядов математического адреса и характеристики страницы дает искомый адрес ячейки на рабочем поле.

При выполнении команд динамических пересылок производятся следующие дополнительные действия, нужные для работы административной системы:

а) из ячейки с адресом 0020, называемой «часами», выбирается ее содержимое, прибавляется единица к 13-му разряду (операция фиксированного сложения) и результат сложения засылается обратно в ячейку 0020;

б) после выборки характеристики страницы в разряды 13-36 этой характеристики заносится показание часов, а в случае динамической засылки в 45-й разряд (признак занесения *ПЗ*) помещается 1 (в случае выборки содержимое 45-го разряда не меняется). Так преобразованная характеристика записывается в таблицу характеристик страниц.

Охарактеризуем теперь кратко работу программы «Административная система» (*АС*). Эта программа, просматривая таблицу характеристик страниц, находит ту страницу, у которой содержимое «часов» наименьшее (к этой странице дольше всего не обращались). Если в характеристике этой страницы признак занесения $ПЗ = 1$, то соответствующую страницу «административная система» (*АС*) записывает на ее место во внешней памяти, а на освободившееся место во внутренней памяти вызывает из внешней памяти нужную страницу. Если же $ПЗ = 0$, то сразу вызывается из внешней памяти нужная страница. При этом *АС* корректирует необходимым образом таблицы характеристик массивов и характеристик страниц. После этого управление передается на ячейку 0010.

СИНТАКСИЧЕСКИЕ ТАБЛИЦЫ АЛГОЛА

Таблица 1

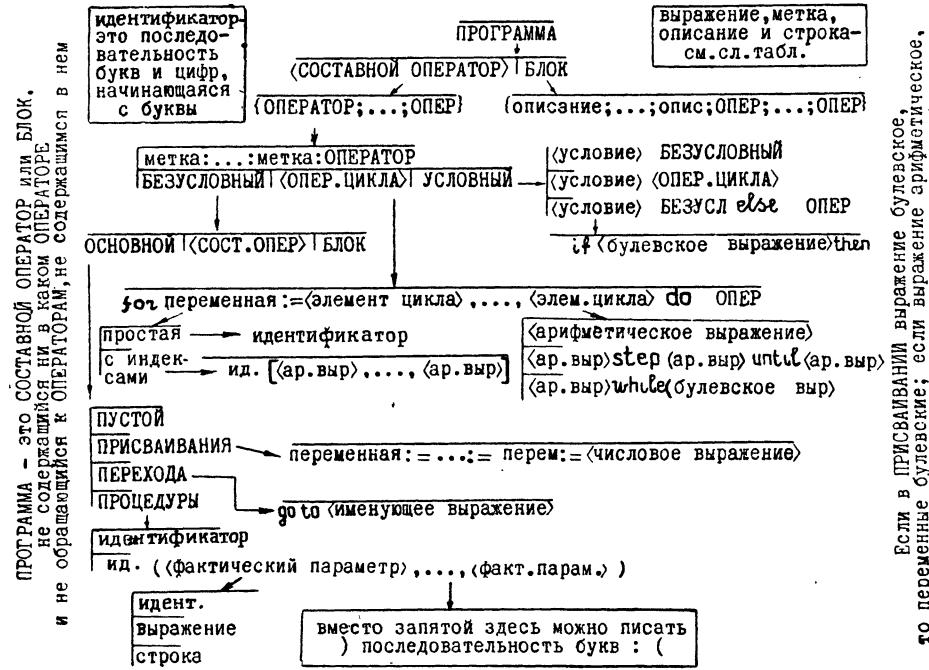


Таблица 2

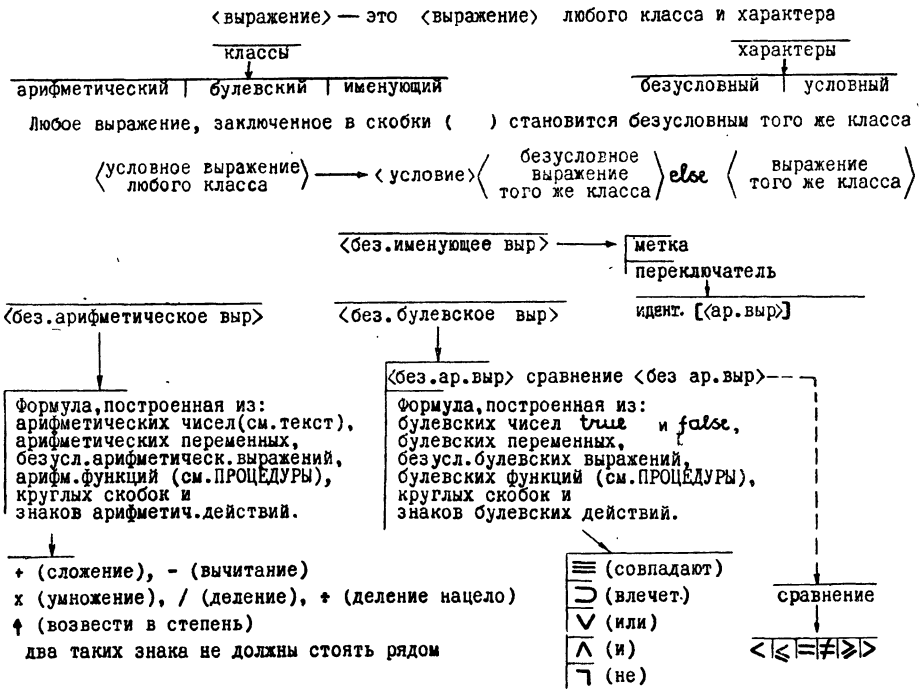

⟨выражение⟩ — это ⟨выражение⟩ любого класса и характера
классы
арифметический | булевский | именующий
характеры
безусловный | условный
Любое выражение, заключенное в скобки ( ) становится безусловным того же класса
⟨условное выражение любого класса⟩ ⟨условие⟩ ⟨безусловное выражение того же класса⟩ else ⟨выражение того же класса⟩
⟨без.именующее выр⟩
метка
переключатель
идент. [⟨ар.выр⟩]
⟨без.арифметическое выр⟩
⟨без.булевское выр⟩
Формула, построенная из:
арифметических чисел (см.текст),
арифметических переменных,
безусл.арифметическ.выражений,
арифм.функций (см.ПРОЦЕДУРЫ),
круглых скобок и
знаков арифметич.действий.
⟨без.ар.выр⟩ сравнение ⟨без ар.выр⟩
Формула, построенная из:
булевских чисел true и false,
булевских переменных,
безусл.булевских выражений,
булевских функций (см.ПРОЦЕДУРЫ),
круглых скобок и
знаков булевских действий.
+ (сложение), - (вычитание)
× (умножение), / (деление), ÷ (деление нацело)
↑ (возвести в степень)
два таких знака не должны стоять рядом
≡ (совпадают)
⊃ (влечет)
∨ (или)
∧ (и)
¬ (не)
сравнение
< ≤ = ≠ ≥ >

Таблица 3

Все идентификаторы (кроме меток и аргументов в описаниях процедур) должны быть описаны в начале блоков, где они употребляются. Перед описанием можно дополнительно поставить own. Обозначение метки считается описанным в начале минимального блока, заключающего метку-приемник.