А. Л. БРУДНО

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

ФИЗИКО-МАТЕМАТИЧЕСКАЯ БИБЛИОТЕКА ИНЖЕНЕРА

А. Л. БРУДНО

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

В СОДЕРЖАТЕЛЬНЫХ ОБОЗНАЧЕНИЯХ

ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА 1965

Б89
518
УДК 519.95

АННОТАЦИЯ

Книга написана автором метода содержательного программирования. Этот метод получил распространение, так как соединяет ряд достоинств программирования в адресах машины и программирования на алгоритмических языках.

Книга рассчитана на следующих читателей: научных работников и студентов, желающих оценить возможности электронных машин в своей тематике или самостоятельно научиться программировать; преподавателей программирования и учащихся, работающих под их руководством; наконец, математиков-программистов, пожелавших познакомиться с этим методом.

Изложение ведется так, чтобы читатель как можно меньше запоминал впрок и все время имел интересные задачи.

Александр Львович Брудно

Введение в программирование

М., 1965 г., 148 стр. с илл.

Редактор *М. М. Горячая*

Техн. редактор *К. Ф. Брудно* Корректор *О. А. Бутусова*

Сдано в набор 25/V 1965 г. Подписано к печати 24/IX 1965 г. Бумага 84 × 103 1/32. Физ. печ. л. 4,63. Условн. печ. л. 7,77. Уч.-изд. л. 6,67. Тираж 38000 экз. Т-10781. Цена книги 33 кол. Заказ № 1791.

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

Ленинградская типография № 1 «Печатный Двор» имени А. М. Горького «Главполиграфпрома» Государственного комитета Совета Министров СССР по печати, Гатчинская, 26.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Цифровые электронные машины (*ЦЭМ*) проникают во многие области человеческой деятельности: в технические расчеты, исследования теоретической физики, управление станками, организацию производства, военную тактику, физиологию, медицину, экономику, лингвистику ... Людям различных специальностей нужно знать возможности *ЦЭМ*, чтобы творчески оценить их применение в своей работе. Умение программировать становится элементом культуры. Кстати сказать, оно так же не является наукой, как правила умножения столбиком или пользования арифмометром или логарифмической линейкой. Оно должно быть таким же простым орудием в руках образованного человека, как эти правила и приборы.

Но по популярным книжкам программировать не научишься. Они излагают достижения (зачастую не делая различия между достигнутым и всего лишь принципиально возможным), опуская технику получения результата. Что же касается книг по программированию, то по ним нельзя научиться в короткий срок. На курсах учат программированию несколько месяцев.

Между тем наш опыт показывает, что при индивидуальном обучении математик может научиться программировать за два часа (в единичных случаях — за 15 минут).

Настоящая книга рассчитана на то, чтобы грамотный математик мог выучиться по ней за день. В ней опущено все, что не служит этой цели и что легче сообразить, чем запомнить.

У читателя формально не предполагается сведений, выходящих за границы школьного курса, но предъявляются высокие требования к его логической культуре. Поэтому нематематику для обучения понадобится несколько дней.

Порядок изложения выбран с таким расчетом, чтобы обучающемуся приходилось как можно меньше «запоминать для дальнейшего применения» и чтобы у него все время были интересные задачи. Но нужно предупредить, что эта книга не для легкого чтения перед сном — по ней можно выучиться быстро за счет напряженной работы.

Вначале, когда *ЦЭМ* было мало, каждый учился программировать для той машины, на которой имел возможность считать. Теперь выгоднее учиться программировать для простейшей машины и только по мере надобности усваивать особенности систем команд и пультов управления конкретных машин.

В этой книжке излагается *программирование в содержательных обозначениях*, когда команды и числа имеют вид, привычный для математика, но сохраняется возможность использовать все особенности системы команд и пульта конкретной машины. С 1954 года так программируют в *Институте электронных машин* и *Институте теоретической физики*. В дальнейшем на этот способ перешло еще несколько групп, связанных с нами.

Этот способ не был опубликован много раньше *) потому, что мне не приходило в голову, что можно программировать иначе. Только под давлением вещественных доказательств мы были вынуждены признать, что всюду программируют иначе,

---

*) В печати наш способ программирования излагали: Р. С. Гутер, Что такое машинная математика, «Преподавание математики в школе», вып. II, 1962, Киев (на украинском яз.); А. Л. Брудно, Содержательное программирование, ДАН СССР **154**, № 2, 1964; Р. С. Гутер, В. Л. Арлазаров, А. В. Усков, Практика программирования (справочник), Наука, 1965.

т. е. сразу же пишут программу в адресах ячеек с небольшими пояснениями.

Автор отдает себе отчет в том, что *содержательное* программирование снимает ряд трудностей обычного *адресного* программирования, стимулирующих развитие автоматического программирования, АЛГОЛа, компилирующих программ и т. д.

Я благодарен А. С. Кронроду, Г. М. Адельсон-Вельскому и А. Г. Пантелееву, которые первыми прошли этот курс. А. С. Кронрод представил мне свои «подтвержденные сталью и кровью» правила организации работы на *ЦЭМ*. Г. М. Адельсон-Вельский участвовал в написании второй главы. А. Г. Пантелеев жестко проводил канонизацию написания программ, когда необходимость ее была еще не всем ясна.

1. ЛОГИЧЕСКАЯ СХЕМА МАШИНЫ; КОМАНДЫ

1.1. Логическая схема машины. По логической схеме (рис. 1.1) все цифровые электронные машины (*ЦЭМ*) близки друг к другу. Научившийся считать на одной сумеет считать и на других. Но учиться надо на одной машине с точно определенными командами и возможностями. Мы будем ориентироваться на условную трехадресную машину с плавающей запятой без регистра переадресации*) и обозначать нашу машину через *ММ*.

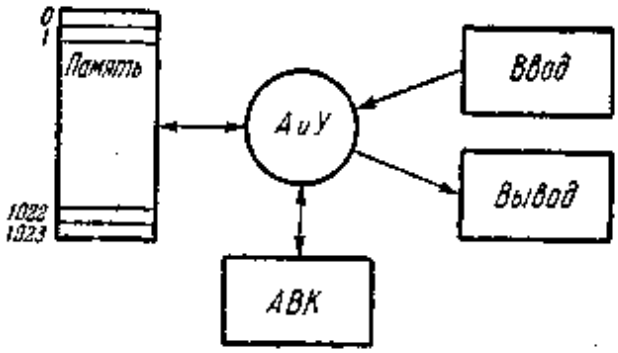


Рис. 1.1.

Память — это устройство, в которое можно *записывать* (помещать для хранения) и из которого можно *прочитывать* (передавать в другие блоки машины) различную информацию — *числа, команды*. Память состоит из *ячеек*. Каждая ячейка имеет номер 0, 1, 2, ... Номер ячейки называется ее *адресом* в машине. Информация, записанная в ячейке памяти, называется *словом*. Как правило, слово — это одно число или одна команда.

При записи слова в ячейку то, что в ней находилось, предварительно стирается. При прочитывании (извлечении) слова из ячейки памяти прочитанное по-прежнему сохраняется в ячейке.

Ввод и *вывод* — различные устройства для ввода информации извне в память и вывода из памяти машины вовне.

---

*) В настоящий момент обучающийся может не понимать этой фразы.

АиУ — устройство, осуществляющее арифметические действия и управление машиной.

АВК — адрес выполняемой команды — регистр, где находится номер ячейки с командой, которую машина выполняет в настоящий момент.

1.2. Работа машины.

1.2.1. В *ЦЭМ*, как правило, вводится (записывается в память) вся информация, нужная для решения задачи: программа вычислений, числа, готовые библиотечные подпрограммы (например, элементарных функций). Затем машину пускают и она автоматически производит действия, предписанные программой: ведет вычисления и печатает результаты.

В отдельных случаях в программе предусматривается дополнительный ввод информации в процессе вычислений. Но и он выполняется автоматически самой машиной, в соответствии с указаниями программы.

1.2.2. В *ЦЭМ употребляется адресный* принцип построения команд. В команде указываются не сами числа, над которыми нужно произвести арифметические действия, а адреса (номера) ячеек, в которых расположены эти числа. Так, например, в машине *ММ* нельзя написать команду

сложить числа 5 *и* 7

Для того, чтобы сделать это, нужно:

а) выделить четыре произвольные ячейки памяти, например, 142, 136, 111 и 115;

б) ввести в ячейку 142 число 5;

в) ввести в ячейку 136 число 7;

г) ввести в ячейку 111 команду:

сложить числа, находящиеся в ячейках 142 *и* 136, *и поместить сумму в ячейку* 115

Если теперь выполнить команду, находящуюся в ячейке 111, то в ячейку 115 поступит число 12, полученное при сложении чисел 5 и 7.

1.2.3. *ММ — трехадресная машина.* Это значит, что в одной команде указываются адреса трех ячеек памяти и код операции. Только из этих трех ячеек могут быть взяты числа для выполнения этой команды, только в них могут

быть отправлены результаты. Код операции указывает, что нужно сделать со словами, взятыми из упомянутых ячеек.

1.2.4. По слову, находящемуся в ячейке памяти, вообще говоря, нельзя судить, команда это или число, — машина этого не знает. Машина использует содержимое ячейки как команду или число в зависимости от указаний самой программы по следующему правилу:

Содержимое ячейки воспринимается как число, когда адрес этой ячейки встречается в выполняемой команде, например, в качестве адреса одного из слагаемых в команде сложения.

Если же адрес некоторой ячейки памяти появляется в регистре АВК (в этом случае говорят, что *управление передано такой-то ячейке*), *то слово из этой ячейки передается в устройство АиУ, расшифровывается в качестве команды* и машина совершает операции, предписанные этой командой.

1.2.5. *Всякая* команда *обязательно* включает явные или неявные указания — какой ячейке памяти передать управление дальше. Указания могут быть четырех типов:

1) стандартная передача управления;
2) безусловная передача управления;
3) условная передача управления;
4) стоп.

Вследствие этого различают четыре типа команд:

Команды со стандартной передачей управления. После выполнения этих команд в *АВК* добавляется единица. Таким образом, после выполнения команды со стандартной передачей управления машина автоматически переходит к выполнению следующей по адресу (по номеру ячейки) команды. В машине *ММ* стандартную передачу управления имеют *все команды, в которых не содержится явного указания на адрес следующей команды* (кроме команды *стоп*).

Команды с безусловной передачей управления. В такой команде (в одном из трех адресов) непосредственно указывается адрес следующей по исполнению команды. В результате выполнения рассматриваемой команды этот адрес заносится в *АВК*. Таким образом, после выполнения команды рассматриваемого типа управление может быть передано любой наперед заданной ячейке памяти, а не обязательно следующей по порядку номеров.

Команды с условной передачей управления. В этих командах машина проверяет некоторое условие. Если оно выполнено, то в *АВК* добавляется единица; если условие не выполнено, то в *АВК* заносится адрес, указанный в самой выполняемой команде.

Таким образом, после выполнения команды рассматриваемого типа машина, в зависимости от результата проверки условия, или переходит к следующей по порядку номеров команде или переходит к команде по указанному адресу.

Команда стоп. При выполнении этой команды машина останавливается.

1.2.6. Таким образом, работа машины состоит в автоматическом выполнении команд программы в нужном порядке: регистр *АВК* указывает, из какой ячейки взять очередную команду, а устройство *АиУ* выполняет указанную команду, т. е. производит нужное действие и заносит в *АВК* адрес следующей команды. Для этой цели каждая команда *ЦЭМ* содержит указание об адресе ячейки, из которой нужно взять следующую команду. Что касается адреса первой команды (с которой надо начать программу), то его перед пуском машины в регистр *АВК* заносит вручную оператор.

Пункт 1.2.5 рекомендуется прочесть очень внимательно. Обучающийся должен усвоить, что если в ячейках m и n написаны «одинаковые» команды со стандартной передачей управления, то выполняться они будут по-разному: в первом случае управление перейдет к ячейке $m+1$, а во втором — к ячейке $n+1$.

2. СОСТАВЛЕНИЕ ПРОСТЕЙШИХ ПРОГРАММ

2.1. Рассмотрим *команды со стандартной передачей управления* — четыре команды арифметических действий:

сложения, которая обозначается $a + b = c$,
вычитания, « « $a - b = c$,
умножения, « « $a \cdot b = c$,
деления, « « $a : b = c$.

Добавим к ним команду

стоп

и рассмотрим составление простейших программ с помощью этих пяти команд.

Вначале рассмотрим, как они выполняются. Пусть управление передано ячейке k, в которой расположена такая команда. Тогда машина прочтет слова из ячеек с номерами a и b (при этом слова в ячейках a и b сохранятся) и выполнит над ними указанное действие как над числами. Далее машина запишет результат в ячейку c (при этом пропадет слово, которое до этого было в ячейке c). Наконец, к *АВК* прибавится 1, и машина перейдет к команде в следующей ячейке — ячейке $k + 1$.

Приведем пример программы для вычисления значения многочлена

$$y = ax^4 + bx^3 + cx^2 + dx + e.$$

Для записи этой программы удобно ввести следующие обозначения:

ячейки, в которых будут храниться команды программы обозначим номерами 1, 2, 3, 4, ...,

ячейки, в которых будут храниться коэффициенты a, b, c, d, e, аргумент x и ячейку, в которой будет получено значение многочлена y, обозначим соответственными буквами.

При этом ячейки a, b, c, d, e, x, y — это любые ячейки памяти машины, а ячейки 1, 2, 3, 4, ... — это ячейки, номера которых обязательно идут подряд (но могут начинаться не с 1-й ячейки, а например, с 47-й).

Команды программы удобно записывать столбиком, одну под другой в левой части листа. Справа от такой записи, которая называется *содержательной*, помещают запись программы в *кодированной* форме, с которой мы познакомимся в § 4. Пока же мы поместим справа объяснения команд программы.

$$y = (((ax + b)x + c)x + d)x + e.$$

1) $a \cdot x = y$	По этой команде произведение ax будет записано в ячейку y (и управление перейдет к команде 2)
2) $y + b = y$	Будет вычислена сумма $y + b = a \cdot x + b$ и записана в ячейку y (старое значение $y = ax$ при этом пропадет)
3) $y \cdot x = y$	Произведение $y \cdot x = (ax + b)x = ax^2 + bx$ после вычисления запишется в ячейку y
4) $y + c = y$	Сумма $y + c = ax^2 + bx + c$ будет вычислена и записана в ячейку y
5) $y \cdot x = y$	В ячейке y образуется произведение $y \cdot x = (ax^2 + bx + c)x = ax^3 + bx^2 + cx$
6) $y + d = y$	В ячейке y образуется сумма $y + d = ax^3 + bx^2 + cx + d$
7) $y \cdot x = y$	В ячейке y образуется произведение $y \cdot x = ax^4 + bx^3 + cx^2 + dx$
8) $y + e = y$	В ячейке y образуется сумма $ax^4 + bx^3 + cx^2 + dx + e = y$
9) *стоп*	Машина остановится

Пусть в ячейки 1, 2, 3, 4, 5, 6, 7, 8, 9 введена эта программа, в ячейки x, a, b, c, d, e введены соответствующие числа, и управление передано ячейке 1. Тогда последовательно будут выполнены команды 1, 2, 3, 4, 5, 6, 7, 8, 9, в результате чего в ячейке y появится число $ax^4 + bx^3 + cx^2 + dx + e$, и машина остановится.

Приведем еще пример программы для вычисления выражения

$$y = (ax^3 + bx + c)/(dx^3 + ex + f).$$

В этом случае, кроме ячеек a, b, c, d, e, f, x, y, нужно использовать еще одну ячейку для хранения промежуточных результатов. Обозначим эту ячейку R. Кроме того, стоит написать краткую схему программы:

I — вычисление числителя в y;
II — вычисление знаменателя в R;
III — получение результата $y : R = y$.

Программа 2.1 приведена ниже.

Программа 2.1

1)	$a \cdot x = y$	6)	$R + e = R$
2)	$y + b = y$	7)	$R \cdot x = R$
3)	$y \cdot x = y$	8)	$R + f = R$
4)	$y + c = y$	9)	$y : R = y$
5)	$d \cdot x = R$	10)	*стоп*

Ячейки y и R, в которых хранятся числа, изменяющиеся в ходе выполнения программы, называются *рабочими ячейками*, ячейки a, b, c, d, e, f, где хранятся числа, не изменяющиеся в ходе выполнения программы, называются *ячейками констант*.

Упражнение. Пусть числа 1 и 10 находятся в ячейках, которые мы обозначим соответственно цифрами 1_n и 10_n (со значком n внизу, чтобы эти ячейки не путались с ячейками программы). Образуйте числа

2, 3/4, 1/4, 5/8, 0,05.

Решение дает программа 2.2 (см. левый столбик).

Программа 2.2

1)	$1_n + 1_n = 2_n$	1)	$1_n + 1_n = 2_n$
2)	$1_n : 2_n = (1/2)$	2)	$1_n : 2_n = R1$
3)	$(1/2) : 2_n = (1/4)$	3)	$R1 : 2_n = (1/4)$
4)	$1_n - (1/4) = (3/4)$	4)	$1_n - (1/4) = (3/4)$
5)	$(1/4) : 2_n = (1/8)$	5)	$(1/4) : 2_n = R2$
6)	$(1/2) + (1/8) = (5/8)$	6)	$R1 + R2 = (5/8)$
7)	$10_n + 10_n = 20_n$	7)	$10_n + 10_n = R2$
8)	$1_n : 20_n = 0{,}05$	8)	$1_n : R2 = 0{,}05$

Ячейки (1/2), (1/8) и $20_д$ — это, по сути дела, рабочие ячейки программы. Можно обойтись и без них, занося промежуточные результаты в те же ячейки, где будут помещены ответы. Такая программа приведена в правом столбике. В ней $R1$ и $R2$ — ячейки для чисел 5/8 и 0,05.

Задача 1. Составить программу для вычисления решений системы трех линейных уравнений с тремя неизвестными:

$$\begin{aligned} a_1x + b_1y + c_1z &= d_1, \\ a_2x + b_2y + c_2z &= d_2, \\ a_3x + b_3y + c_3z &= d_3. \end{aligned}$$

Задача 2. Для $-1 \leqslant x \leqslant 1$ с точностью до 10^{-7} справедлива формула

$$e^x = \left[1 + \frac{x}{8} + \frac{1}{2}\left(\frac{x}{8}\right)^2 + \frac{1}{6}\left(\frac{x}{8}\right)^3 + \frac{1}{24}\left(\frac{x}{8}\right)^4 + \frac{1}{120}\left(\frac{x}{8}\right)^5\right]^8.$$

Составить программу для вычисления e^x по этой формуле.

2.2. Введем еще две команды — команды с *безусловной передачей управления*:

печать числа, обозначаемая *печ* b; c
перенос, обозначаемый $a \longrightarrow b$; c

Выполняются эти команды так. По команде *печать числа* машина сначала печатает число из ячейки b (которое при этом сохраняется), а затем передает управление ячейке c. Значение команды печати (и других команд вывода, с которыми мы еще встретимся) в том, что только с их помощью получаются сведения о результатах, полученных машиной.

При команде *перенос* машина вначале переносит слово из ячейки a в ячейку b, а затем передает управление ячейке c. При этом слово в ячейке a сохраняется, а слово, находившееся ранее в ячейке b, пропадает.

В частности, при выполнении команды переноса

$$0 \longrightarrow 0;\ c$$

не произойдет ничего, кроме передачи управления ячейке c. В этом случае команду переноса называют *передачей управления*.

Таким образом, при помощи команд с безусловной передачей управления можно перейти от выполнения команд, расположенных в одном месте памяти, к выполнению команд, расположенных в произвольном другом месте. Можно осуществить и так называемую *челночную* программу, при которой

машина будет возвращаться к выполнению уже выполненных раньше команд.

Продолжим составление программ с помощью семи команд, введенных в п. 2.1 и п. 2.2.

Приведем, например, программу для вычисления значений многочлена

$$y = ax^4 + bx^3 + cx^2 + dx + e$$

при $x = 0,\ h,\ 2h,\ 3h,\ \ldots$

Схема программы будет выглядеть так:

I Возобновление состояния $x = 0$

II Увеличение x на h

III Вычисление y и печать x и y

Обозначим опять номерами 1, 2, 3, 4, ... ячейки, в которых помещаются команды программы. Ячейки, в которых будут храниться числа 0, a, b, c, d, e, x, y, h, обозначим соответствующими буквами. Программа может быть написана так (см. левый столбик программы 2.3):

Программа 2.3

1)	$0 \to x$; 3	$0 \to x$
2)	$x + h = x$	$x + h = x$
3)	$a \cdot x = y$	$a \cdot x = y$
4)	$y + b = y$	$y + b = y$
5)	$y \cdot x = y$	$y \cdot x = y$
6)	$y + c = y$	$y + c = y$
7)	$y \cdot x = y$	$y \cdot x = y$
8)	$y + d = y$	$y + d = y$
9)	$y \cdot x = y$	$y \cdot x = y$
10)	$y + e = y$	$y + e = y$
11)	*печать* x; 12	*печ* x
12)	*печать* y; 2	*печ* y

Когда управление перейдет к ячейке 12, машина отпечатает результат и перейдет к выполнению команды 2.

Таким образом, если в машину введена эта программа и все числа, а управление передано ячейке 1, то прежде всего в ячейку аргумента x будет записано число 0, а управление

перейдет к ячейке 3. Затем командами 3—10 будет вычислено и записано в ячейку y значение многочлена при $x=0$, после чего управление перейдет к ячейке 11. По команде, расположенной в ячейке 11, значение аргумента $x=0$ будет напечатано *), и управление перейдет к ячейке 12. Далее будет напечатано значение многочлена $y(0)$, и управление перейдет в ячейку 2, в которой стоит команда изменения значения x — увеличения этого значения на h. Затем, так как в ячейке 2 расположена команда со стандартной передачей управления, управление перейдет к ячейке 3. Теперь будет вычислено значение многочлена $y(x)$ при $x=h$, а новые значения x и y будут напечатаны. Затем будут вычислены и напечатаны значения аргумента и функции при $x=2h$ и т. д.

Заметим, что если заранее ввести в ячейку x число 0, то ту же программу можно написать без команды 1, которая выполняется только один раз. Однако такая программа не будет *возобновляющейся*. По ней невозможно повторить вычисления сначала. Это неудобно для работы и, в частности, для отладки программы (т. е. проверки на машине правильности программы, нахождения и исправления ошибок).

Программа называется *возобновляющейся*, если ее можно прервать на любой команде и начать сначала. Нужно приобрести привычку составлять такие программы, и мы во всех примерах и задачах будем приводить возобновляющиеся программы **).

При записи команд с передачами управления можно применять и другое обозначение, а именно — проводить стрелку от команды, передающей управление, к команде, которой передается управление. Наша программа в такой записи была приведена в правом столбике программы 2.3.

Разложение цепи операций для решения вычислительной задачи на отдельные команды не однозначно, при этом обычно можно также менять порядок некоторых операций, экономить ячейки и ускорять вычисления.

Задача 3. Составить программу вычисления e^x для $x=0$; 0,01; 0,02; 0,03 и т. д. по формуле задачи 2.

---

*) Здесь мы отвлекаемся от того, что машина считает в двоичной системе, и фактически, прежде чем печатать результат, его нужно перевести из двоичной системы в десятичную.

**) По необходимости изредка приходится применять и невозобновляющиеся программы.

2.3. Мы познакомились с программами двух типов. Программы из п. 2.1 выполняли команды по одному разу и затем останавливались. Одни эти программы не могут быть полезны, ибо легче выполнить арифметическое действие, чем написать команду.

Программы второго типа (из п. 2.2) использовали команды многократно. Но эти программы крутились без конца на одном месте. Без помощи человека они не могли закончить своих вычислений и перейти к другим частям задачи.

Выход из этого затруднения дают команды *с условной передачей управления*. В машине *ММ* их две:

сравнение, обозначаемое $a \geqslant b\ ;\ c$
сравнение по модулю $|a| \geqslant |b|\ ;\ c$

Пусть в ячейке с номером k расположена команда сравнения

$$a \geqslant b;\ c$$

и управление передано этой ячейке. Тогда машина прочтет слова из нужных ячеек и сравнит их как числа. Если окажется, что неравенство $a \geqslant b$ выполняется, то управление будет передано ячейке с номером $k+1$, следующим за номером ячейки, в которой стоит команда сравнения. Если же окажется, что неравенство не выполняется, то управление будет передано ячейке с номером c.

Сравнение по модулю выполняется аналогично, только сравниваются абсолютные значения чисел, находящихся в ячейках a и b.

При помощи команд условного перехода пишутся *циклические* программы, при которых машина несколько раз производит работу по одному и тому же циклу команд, а затем переходит к выполнению других команд.

Примером циклической программы является программа вычисления суммы

$$s = \frac{1}{1^2} + \frac{1}{2^2} + \cdots + \frac{1}{n^2}$$

при заданном числе n. При вычислении нам потребуются две рабочие ячейки: для номера k очередного члена и для промежуточных вычислений (см. программу 2.4).

Программа 2.4

1) $1_n \to k$	Единица заносится в ячейку, отведенную для номера k
2) $0 \to s$	Нуль заносится в s и машина переходит к команде 4
3) $k + 1_n = k$	Номер k увеличивается на единицу
4) $k \cdot k = R$	Вычисляется k^2
5) $1_n : R = R$	Вычисляется $1 : k^2$
6) $s + R = s$	Образуется сумма $\frac{1}{1^2} + \frac{1}{2^2} + \ldots + \frac{1}{k^2}$
7) $\lvert k \rvert \geqslant \lvert n \rvert$	Если $k = n$, то к команде 8, иначе к команде 3
8) *стоп*	

После команд 1 и 2 величина k станет равна единице, а s — нулю. После команд 4—6 величина s станет равна первому члену: $s = 1$. Если $n = 1$, то на этом вычисление прекратится. Если $n > 1$, то управление перейдет к команде 3, где k увеличится с 1 до 2, а команды 4—6 дадут $s = 1 + \frac{1}{4}$. Команда 7 прекратит вычисление, если $n = 2$, и продолжит их, если $n > 2$, и т. д.

При составлении программы мы считали, что у нас есть числа 0 и 1, находящиеся в некоторых ячейках, обозначенных цифрами 0 и 1_n.

Если нужно вычислить сумму бесконечного ряда

$$s = \frac{1}{1^2} + \frac{1}{2^2} + \ldots,$$

то можно воспользоваться программой 2.5 (левый столбик).

Программа 2.5

1) $1_n \to k$	1) $1_n + 1_n = k$
2) $0 \to s$	2) $1_n \to s$
3) $k + 1_n = k$	3) $k + 1_n = k$
4) $1_n : k = R$	4) $1_n : k = R$
5) $R : k = R$	5) $R : k = R$
6) $s + R = s$	6) $s + R = s$
7) $\lvert 0 \rvert \geqslant \lvert R \rvert$	7) $\lvert 0 \rvert \geqslant \lvert R \rvert$
8) *стоп*	8) *стоп*

С каждым новым циклом команд 3—7 к накопленной сумме s прибавляется следующий член ряда. Если бы вычисления в машине велись абсолютно точно, то при $x \neq 0$ неравенство 7 не выполнялось бы никогда, и машина никогда не кончила бы вычислений.

Однако при арифметических действиях машина заменяет нулем все результаты, меньшие по абсолютной величине некоторого минимума. Поэтому на некотором цикле команда 5 выдаст частное, равное нулю. Тогда неравенство 7 выполнится и машина закончит вычисления. Отброшенные члены будут иметь величину порядка точности машины, т. е. результат получится с той же точностью, что и по нециклической программе.

Приведенную программу можно ускорить, как это показано в правом столбике программы 2.5.

2.4. Каноническая схема циклической программы имеет вид:

ВОЗОБНОВЛЕНИЕ
ИЗМЕНЕНИЕ
РАБОТА
ПРОВЕРКА ОКОНЧАНИЯ
ДОСЧЕТ

Предыдущие циклические программы составлены по канонической схеме. Обучающийся должен педантично ее придерживаться по причинам, которые будут объяснены в дальнейшем. Расписывая программу по схеме, удобно начинать с «работы». Затем пишется «проверка окончания». Потом «изменение» и, наконец, «возобновление». В «досчете» выполняются вычисления, которые надо сделать после окончания цикла.

Пусть, например, надо вычислить натуральный логарифм по формуле

$$y = \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots \quad (|x| < 1).$$

Рабочую часть можно написать так:

$$(\pm x^k) : k = R$$
$$y + R = y$$

Проверку окончания естественно сделать такой:

$$|0| \geqslant |R|; \text{ (изменение)}$$

Теперь естественно дописать «изменение» — увеличение k на единицу, и этот кусок примет вид программы 2.6.

Программа 2.6

Изменение	$k + 1_n = k$
Работа	$(\pm x^k) \cdot (-x) = (\pm x^k)$ $(\pm x^k) : k = R$ $y + R = y$
Проверка	$\|0\| \geqslant \|R\|$

Остается приписать «возобновление». Положим вначале $y = x$. Тогда для вычисления суммы двух первых членов понадобится член $-x^3/2$. Поэтому вначале надо взять $k = 2$ и $\pm x^n = -x^3$. Кроме того, надо образовать число «$-x$», используемое в рабочей части. Получится программа 2.7:

Программа 2.7

Воз.	1) $0 - x = (-x)$	В ячейку, обозначаемую $(-x)$, заносится значение $-x$
	2) $1_n + 1_n = k$	В ячейку, обозначаемую k, заносится знаменатель второго члена, равный числу 2
	3) $(-x) \cdot x = \pm x^k$	Образуется числитель второго члена, равный $-x^3$
	4) $x \to y$	Заносится первая сумма, равная x, и машина переходит к команде 7
Изм.	5) $k + 1_n = k$	Знаменатель увеличивается на 1
Раб.	6) $(\pm x^k) \cdot (-x) = (\pm x^k)$	Степень $\pm x^n$ *множится на* $-x$ и образуется числитель следующего члена ряда
	7) $(\pm x^k) : k = R$	Образуется очередной член ряда
	8) $y + R = y$	Образуется очередная сумма
Пров.	9) $\|0\| \geqslant \|R\|$	Проверяется малость добавленного члена
	10) *стоп*	

Для вычисления $\ln(1+x)$ мы используем, кроме ячейки x, где хранится значение аргумента, и ячейки y, где получается значение $\ln(1+x)$, четыре вспомогательные рабочие ячейки $-x$, k, $\pm x^k$, R.

Рассмотрим еще один пример. При помощи команд условного перехода можно составить программу 2.8 вычисления

$$y=(ax_1^3+bx_1^2+cx_1+d)/(ax_2^3+bx_2^2+cx_2+d), \quad x_1 \neq x_2$$

с общим куском программы для вычисления числителя и знаменателя.

Сначала в ячейку x попадет значение x_1, машина сосчитает числитель, и сравнения 8 и 9 пропустят программу к выполнению команды 10. Числитель перенесется в ячейку $R1$, а в ячейку x поступит значение x_2, и машина перейдет к команде 2 для вычисления знаменателя. После команды 7, когда знаменатель будет вычислен, одно из сравнений 8 или 9 не выполнится, т. к. в ячейке будет находиться число x_2, не равное x_1, и машина перейдет к команде 12, где вычислит значение всей дроби.

Программа 2.8

1) $x_1 \to x$
2) $a \cdot x = y$
3) $y + b = y$
4) $y \cdot x = y$
5) $y + c = y$
6) $y \cdot x = y$
7) $y + d = y$
8) $x \geqslant x_1$
9) $x_1 \geqslant x$
10) $y \to R1$
11) $x_2 \to x$
12) $R1 : y = y$
13) *стоп*

Задача 4. Поправить предыдущую программу так, чтобы она давала верный ответ и в случае равенства $x_1 = x_2$.

2.5. Познакомимся еще с одной командой с безусловной передачей управления — с командой *возврата:*

$$a \leftarrow b;\ c$$

По этой команде в ячейку b заносится команда

$$0 \to 0;\ a$$

(т. е. команда безусловной *передачи управления* ячейке a), после чего управление передается ячейке c.

Команда возврата удобна, когда в основной программе возникает необходимость обратиться к подпрограмме, а затем вернуться для продолжения основной программы.

Пусть, например, требуется сосчитать для заданных x_1, x_2, x_3 значение функции

$$y = \varphi(x_1) \cdot \varphi(x_2)/\varphi(x_3), \quad \text{где} \quad \varphi(x) = (a + bx)/(c - dx^2).$$

В программе 2.9 команды $1 \div 10$ расположены подряд в одном месте, а команды $P \div Q$ — тоже подряд в произвольном другом. Ячейка Q в конце программы вычисления

Программа 2.9

1) $x_1 \to x$	P) $b \cdot x = \varphi$
2) $3 \leftarrow Q;\ p$	$P+1$) $a + \varphi = \varphi$
3) $\varphi \to y$	$P+2$) $x \cdot x = R$
4) $x_2 \to x$	$P+3$) $d \cdot R = R$
5) $6 \leftarrow Q;\ p$	$P+4$) $c - R = R$
6) $y \cdot \varphi = y$	$P+5$) $\varphi : R = \varphi$
7) $x_3 \to x$	$Q \equiv P+6$) ——
8) $9 \leftarrow Q;\ p$	
9) $y : \varphi = y$	
10) *стоп*	

$\varphi = \varphi(x)$ оставлена свободной. Она будет использована во время работы.

Рассмотрим работу программы.

Команда 1 заносит аргумент x_1 в ячейку x. Команда 2 образует команду

$$0 \to 0;\ 3$$

и заносит ее в ячейку Q, после чего передает управление ячейке P. Команды от P до $P+5$ вычисляют значение $\varphi(x_1)$, и управление попадает ячейке Q. Находящаяся здесь в этот момент команда

$$0 \to 0;\ 3$$

возвращает управление ячейке 3 основной программы, и т. д.

Ячейку Q называют *выходом* программы $\varphi = \varphi(x)$. Она играет роль рабочей ячейки, но в отличие от других рабочих ячеек должна располагаться в строго определенном месте — в конце программы $\varphi = \varphi(x)$.

Программист должен запомнить правило: *прежде чем передавать управление подпрограмме — позаботься о том, чем подпрограмма кончится.*

Приведем в табл. 2.1 список команд, с которыми мы познакомились в этом разделе.

Таблица 2.1

Номер	*Команда*	*Номер*	*Команда*
00	$a \rightarrow b;\ c$	05	$a - b = c$
01	$a \leftarrow b;\ c$	06	$a \cdot b = c$
02	$\|a\| \gtrless \|b\|;\ c$	07	$a : b = c$
03	$a \gtrless b;\ c$	10	*печ* $b;\ c$
04	$a + b = c$	—00	*стоп*

Странная нумерация команд в этой таблице будет разъяснена в дальнейшем. По важной причине, которая будет ясна из § 4, необходимо всегда писать аргументы, результаты и адреса передач управления в том порядке, в каком указано. Недопустимо, например, писать команду вычитания в таких видах:

$$c = a - b \qquad -a + b = c \qquad -b + a = c$$

или команды переноса и сравнения так:

$$a \leftarrow b;\ c \qquad c;\ a \rightarrow b$$
$$b \lessgtr a;\ c \qquad c;\ a \gtrless b$$

Равным образом запрещается пользоваться командами, которых нет в списке, придумывать новые команды — *машина их не выполнит.*

Задача 5. Составить программу для вычисления напряженности электростатического поля, создаваемого равномерно заряженным шаром радиуса r_0. Эта величина задается формулой

$$H = \begin{cases} Ar_0^3 : r^2 & \text{при } r \geqslant r_0, \\ Ar : r_0 & \text{при } r < r_0. \end{cases}$$

Сосчитать H для r, изменяющегося от 0 до 1 с интервалом Δr.

Задача 6. Составить программу для вычисления $y = e^x$ при помощи ряда

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots \quad (|x| < 1).$$

Задача 7. Написать программу для нахождения методом конечных разностей решения дифференциального уравнения

$$y' = \frac{x + y}{1 + y^2}$$

при $x = 0$, $y = y_0$. Найти y при $x = x_1$. Шаг h положить равным $(x_1 - x_0) : n$, где n — заданное число.

Рассмотреть случай формулы Эйлера

$$y(x+h)=y(x)+hy'(x)$$

и уточненной формулы Эйлера

$$y(x+h)=y(x-h)+2hy'(x),$$

причем во втором случае первый шаг сделать по обычной формуле Эйлера.

2.6. Цикл в цикле. Следующее, непринципиальное, усложнение представляют программы с двухкратным, трехкратным и т. д. циклом.

Пусть, например, требуется сосчитать сумму ряда *)

$$s=\frac{1}{1^2}\left(1+\frac{1}{1^2}\right)+\frac{1}{2^2}\left(1+\frac{1}{2^2}\right)\left(1+\frac{2}{2^2}\right)+$$
$$+\frac{1}{3^2}\left(1+\frac{1}{3^2}\right)\left(1+\frac{2}{3^2}\right)\left(1+\frac{3}{3^2}\right)+\ldots$$

Вычисление s будет осуществляться циклической программой — на каждом цикле к накопленной сумме будет прибавляться очередной член ряда. Эти циклы называются *наружными*. А внутри каждого наружного цикла будет работать

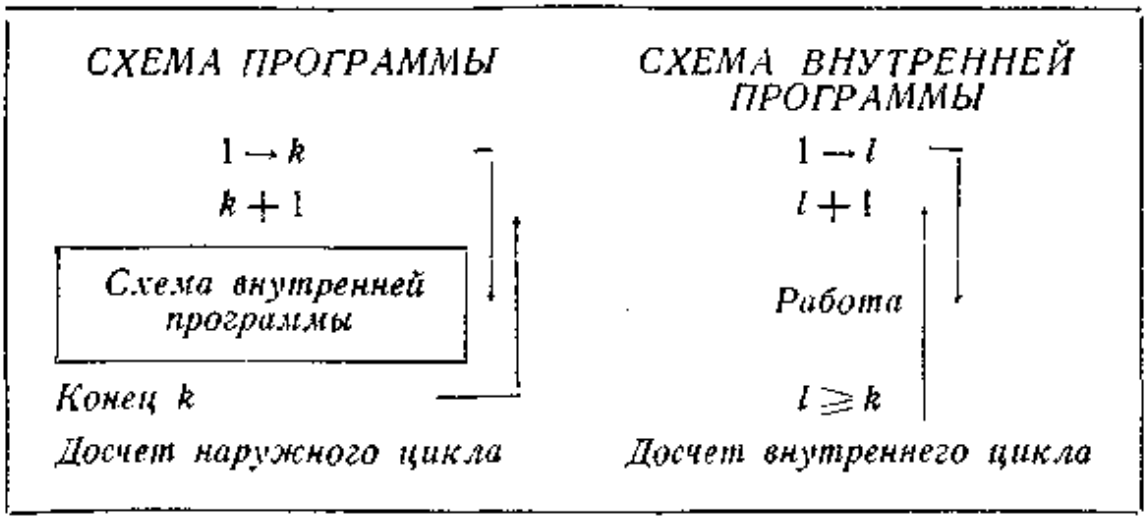


Рис. 2.1.

циклическая программа вычисления очередного члена. Ее цикл называют *внутренним*. Если написать

$$s=a_1+a_2+\ldots, \qquad a_k=\frac{1}{k^2}\left(1+\frac{1}{k^2}\right)\ldots\left(1+\frac{l}{k^2}\right)\ldots\left(1+\frac{k}{k^2}\right),$$

то внутренняя и наружная программы будут вычислять соответственно a_k и s (схемы этих программ приведены на рис. 2.1).

*) Этот ряд сходится. Его n-й член меньше, чем e/k^2.

Здесь «$1 \to k$» кратко означает «возобновление вычислений с начального значения k». Начальное значение может оказаться и не равным единице. В пункт «$1 \to k$» входят вообще все предварительные вычисления, засылки, очистки и т. п., нужные перед началом этого счета. То же относится к пункту «$1 \to l$». Пункты «$k+1$» и «$l+1$» означают соответственно подготовку для перехода к следующему наружному или внутреннему циклу. Пункт «$l \geqslant k$» — это проверка окончания цикла по l.

Расписывание программы, как всегда, удобно начать с самого внутреннего пункта работы по вычислению a — очередного члена ряда:

раб.) $l : k^2 = R$
$1_n + R = R$
$a \cdot R = a$
$l \geqslant k$; изм. l

Затем можно приписать увеличение l (одна команда: $l+1=l$). Займемся возобновлением цикла по l, т. е. пунктом «$1 \to l$». Внутренняя программа получает из внешней новое значение k. Ей нужно приготовить k^2 и отправить в ячейку a начальное значение, за которое можно принять $a = 1/k^2$. Программа внутреннего цикла принимает вид левого столбика программы 2.10.

Программа 2.10

ВНУТРЕННИЙ ЦИКЛ		*НАРУЖНЫЙ ЦИКЛ*	
$1 \to l)$	$k \cdot k = k^2$	$1 \to n)$	$0 \to s$
	$1_n : k^2 = a$		$1 \to k$
	$1_n \to l$	$n+1)$	$k+1=k$
$l+1)$	$l + 1_n = l$		*внутренний цикл*
раб.)	$l : k^2 = R$		$s + a = s$
	$1_n + R = R$		$0 \geqslant a$
	$a \cdot R = a$		*стоп*
	$l \geqslant k$		

Общий вид программы приведен в правом столбике.

Можно было бы вычисление нового значения k^2 (и даже $a = 1/k^2$) перенести из части «возобновление по l» в

«изменение l» и получать новое значение

$$k^2 = (k-1)^2 + (2k+1).$$

Но без нужды этого делать не следует. Желательно, чтобы наружный цикл как можно меньше вмешивался во внутренний, передавая ему минимум необходимой информации. Если вычислить величину наново не многим труднее, чем получить ее из предыдущего значения, то предпочтительнее прямое вычисление.

Таким образом, в двухкратном цикле наружный цикл пишется по обычной циклической схеме, у которой рабочая часть содержит внутреннюю программу. Внутренняя часть пишется как обычная циклическая программа.

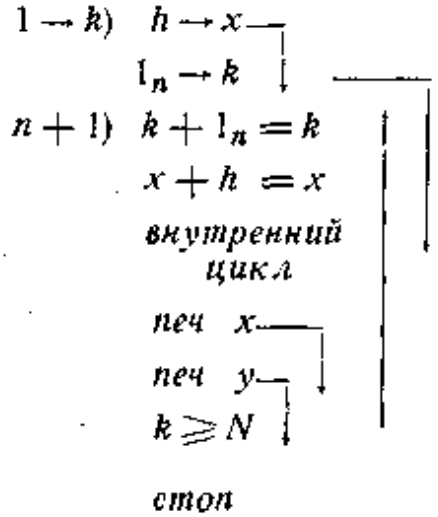
Задача 8. Составить программу, печатающую значения x и y для

$$y = \ln(1+x), \qquad x = h, 2h, \ldots, Nh,$$

используя программу 2.7. Числа h и N считать заданными.

Решение. В программе 2.7 заменить *стоп* печатями x и y и приписать внешний цикл (см. программу 2.11).

Программа 2.11



Задача 9. Составить программу, печатающую подряд числа сочетаний

$$C_1^1;\ C_2^1,\ C_2^2;\ C_3^1,\ C_3^2,\ C_3^3;\ \ldots$$

Напомним, что

$$C_k^m = \frac{k!}{m!\,(k-m)!}, \quad 0! = 1! = 1.$$

3. ПОЗИЦИОННЫЕ СИСТЕМЫ СЧИСЛЕНИЯ

3.1. Рассмотрим *десятичную систему,* знакомую нам с детства. В ней:

а) для каждого числа, меньшего десяти, введено отдельное изображение — цифра:

$$0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9;$$

б) число, большее или равное десяти, изображается строкой цифр, например, строка

$$583$$

изображает число

$$5E^2 + 8E + 3, \tag{3.1}$$

где E — число десять — основание десятичной системы счисления.

Эта схема записи маскируется в обычной речи тремя обстоятельствами:

1°. В обычной речи пропускаются слова *умножить* и *плюс*. Мы говорим *восемьдесят три* вместо *восемь умножить на десять и плюс три.*

2°. Для некоторых степеней десяти — E, E^2, E^3, E^6, ... приняты наименования: десяток, сотня, тысяча, миллион, ...

3°. Пользуясь наименованиями для некоторых степеней, мы в обычной речи вводим скобки и пропускаем нули в записи вида (3.1). Так, например, фраза *пятьдесят семь тысяч триста четыре* буквально значит

$$(5E + 7)E^3 + 3E^2 + 4 = 57304,$$

вместо неудобопроизносимой записи типа (3.1):

$$5E^4 + 7E^3 + 3E^2 + 0E + 4 = 57304,$$

Десятичные дроби записываются тоже строкой цифр, но с запятой, отделяющей целую часть. Так, например, строка

$$57{,}304$$

изображает число

$$5E + 7 + 3E^{-1} - 0E^{-2} + 4E^{-3},$$

где E — число десять.

Арифметические действия с десятичными числами (целыми и дробными) не представляют принципиальных трудностей по сравнению с действиями над однозначными числами-цифрами. Действия с однозначными числами мы выучиваем наизусть (таблицы сложения и умножения).

Резюмируем. *В десятичной системе строка*

$$a_n a_{n-1} \ldots a_1 a_0, \; a_{-1} a_{-2} \ldots a_{-k} \tag{3.2}$$

изображает число

$$a_n E^n + a_{n-1} E^{n-1} + \ldots + a_1 E + a_0 + \\ + a_{-1} E^{-1} + a_{-2} E^{-2} + \ldots + a_{-k} E^{-k}. \tag{3.3}$$

Здесь a_i — десятичные цифры из набора 0, 1, ..., 9, *а E — число десять — основание десятичной системы счисления.*

3.2. Изложенный принцип записи числа *строкой цифр* принадлежит к главным достижениям человеческой мысли. Без него невозможны арифметические вычисления *) и вряд ли могла бы существовать, а тем более возникнуть, современная техника. Но положенное в основание системы счисления число десять — «дар случайный — дар напрасный», — по-видимому, связано со счетом на пальцах рук, в настоящее время практически не используемым. Известно, что для повседневного употребления была бы удобнее двенадцатеричная система. Были предложены обозначения для недостающих цифр (десять и одиннадцать) и названия для «круглых» чисел (дюжина, гросс, ...). Но двенадцатеричная система не прививается, так как для нее пришлось бы всем переучиваться считать.

---

*) Сомневающемуся читателю предлагается поделить в римских цифрах MMCXXXVI на XCIV, не переходя к десятичным в промежуточных записях (1936 : 44).

Иначе обстоит дело с математическими вычислительными машинами. Выяснив, что для них удобнее двоичная система, стали строить машины, считающие в двоичной системе.

3.3. В основание *двоичной системы* положено число два, так что в ней всего две цифры: 0 и 1.

Строка цифр (3.2) расшифровывается по формуле (3.3), где E равно двойке, а цифры a_i принимают только значения 0 и 1. Вот первые числа натурального ряда, записанные в двоичной системе, и их названия в десятичной системе:

0; 1; 10; 11; 100; 101; 110; 111; 1000; 1001; 1010; 1011; ...

0 — *нуль;*

1 — *один;*

10 — *два;*

$11 = 1 \cdot 2 + 1$ — *три;*

$100 = 1 \cdot 2^2 + 0 \cdot 2 + 0$ — *четыре;*

$101 = 1 \cdot 2^2 + 0 \cdot 2 + 1$ — *пять;*

$110 = 1 \cdot 2^2 + 1 \cdot 2 + 0$ — *шесть;*

$111 = 1 \cdot 2^2 + 1 \cdot 2 + 1$ — *семь;*

$1000 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2 + 0$ — *восемь;*

$1001 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2 + 1$ — *девять;*

$1010 = 1 \cdot 2^3 + 0 \cdot 2 + 1 \cdot 2 + 0$ — *десять;*

$1011 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2 + 1$ — *одиннадцать.*

Таблица сложения для двоичной системы (см. табл. 3.1) очень коротка. В заголовках ее строк и столбцов стоят слагаемые, а на пересечении — сумма.

Таблица 3.1

Слагаемые / *Слагаемые*	0	1
0	0	1
1	1	10

Таблица показывает, как складываются *однозначные* числа. Но в позиционных системах не требуется дополнительных таблиц для сложения многозначных чисел. При сложении двух многозначных чисел фактически складываются только цифры: младший разряд результата записывается в сумму, а старшая единица (если она возникает) запоминается «в уме» и прибавляется к старшему разряду слагаемых.

Пример.

ДЕСЯТИЧНАЯ СИСТЕМА	*ДВОИЧНАЯ СИСТЕМА*
53	110101
+	+
25	11001
78	1001110

Складывать и вычитать в двоичной системе легко научиться на «двоичных счетах». Для этого нужно на обычных счетах отделить линейкой две левых косточки (рис. 3.1) и пользоваться во время счета только ими. Если в процессе сложения в каком-нибудь разряде будут отложены обе косточки, то их нужно сбросить и прибавить одну косточку в следующий разряд. Так же «занимают» при вычитании.

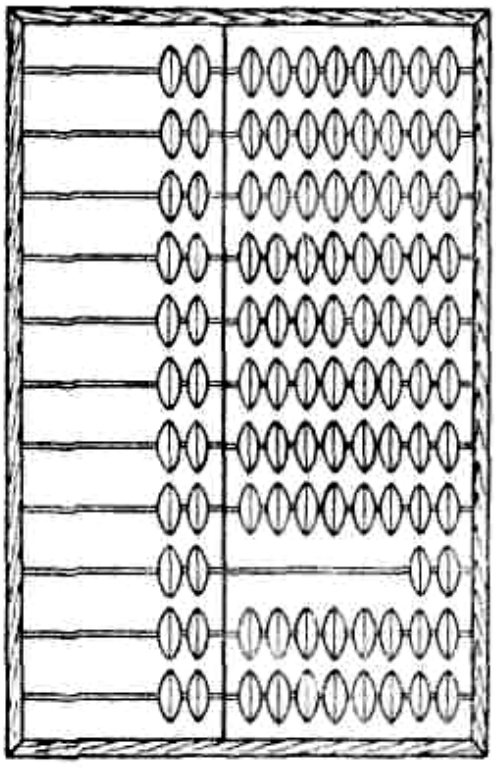


Рис. 3.1.

3.4. При программировании, кроме десятичной и двоичной системы, употребляется еще и *восьмеричная*. Цифры в этой системе обозначаются так же, как и в десятичной. Для перевода из двоичной системы в восьмеричную нужно влево и вправо от запятой отделить группы по три цифры и каждую группу заменить одной восьмеричной цифрой.

Пример записи числа в двоичной и восьмеричной системах:

двоичная	1	001	101	110	,	100	001	110		
восьмеричная	1	1	5	6	,	4	1	6		

3.5. Перевод чисел из десятичной системы в двоичную и обратно довольно трудоемок. К счастью, он почти не бывает нужен.

3.6. Добавление (только для программистов). Перевод из десятичной системы и в десятичную систему *).

3.6.1. Перевод из десятичной системы в систему с произвольным основанием E подсказывается самой формой записи (3.3).

Для перевода целой части A числа надо разделить A на E. Остаток даст младшую цифру числа A в E-ричной системе. Получившееся частное надо разделить на E. Остаток даст следующую цифру, и т. д.

Пример. Перевести 258 из десятичной системы в восьмеричную.

	ЧАСТНОЕ	*ОСТАТОК*
$258:8=$	32	2
$32:8=$	4	0
$4:8=$	0	4

$$(258)_{\text{десятичное}} = (402)_{\text{восьмеричное}}.$$

Для перевода дробной части B числа надо умножить B на E. Целая часть произведения даст первую цифру (после запятой) числа B в E-ричной системе. Дробную часть произведения надо умножить на E. Целая часть результата даст следующую цифру, и т. д.

Пример. Перевести 0,267 в восьмеричную систему.

$$\begin{array}{ll} 0{,}267 \cdot 8 = 2{,}136 & 0{,}704 \cdot 8 = 5{,}632 \\ 0{,}136 \cdot 8 = 1{,}088 & 0{,}632 \cdot 8 = 5{,}056 \\ 0{,}088 \cdot 8 = 0{,}704 & \ldots\ldots\ldots \end{array}$$

$$(0{,}267)_{\text{десятичное}} = (0{,}21055\ \ldots)_{\text{восьмеричное}}.$$

3.6.2. Для перевода числа в десятичную систему из E-ричной сначала переводят в десятичную систему E-ричные цифры и основание E, т. е. 1, 2, ..., E, а затем вычисляют результат по формуле (3.3). Если E меньше десяти, то цифры, разумеется, переводить нет надобности.

Пример. Перевести восьмеричное число 320,13 в десятичную систему.

Решение.

$$3 \cdot 8^2 + 2 \cdot 8 + 0 \cdot 8^0 + 1 \cdot 8^{-1} + 3 \cdot 8^{-2} = (208{,}171875)_{\text{десятичное}}.$$

Вычисления можно делать на арифмометре. Для облегчения счета удобно заготовить в десятичной системе степени 8^2, 8^{-1}, 8^{-2}, ..., т. е. числа 64; 0,125; 0,015625; ...

---

*) Десятичная система играет здесь роль системы, в которой мы (или наши вычислительные приборы) производим вычисления.

3.6.3. Двоичные числа длинны и действовать с ними неудобно. Поэтому для перевода чисел из десятичной системы в двоичную и обратно удобно сначала получить промежуточный результат в восьмеричной системе.

Пример. Перевести 258,267 из десятичной системы в двоичную.

Решение. Переводим отдельно целую и дробную части в восьмеричную систему. Результат собираем и переписываем в двоичной системе. Вычисления удобно выполнять на арифмометре.

ЦЕЛАЯ ЧАСТЬ			*ДРОБНАЯ ЧАСТЬ*
	частное	*остаток*	$0{,}267 \cdot 8 = 2{,}136$
$258 : 8 =$	32	2	$0{,}136 \cdot 8 = 1{,}088$
$32 : 8 =$	4	0	$0{,}088 \cdot 8 = 0{,}704$
$4 : 8 =$	0	4	$0{,}704 \cdot 8 = 5{,}632$
			$0{,}632 \cdot 8 = 5{,}056$
			. . . . .

$$(258{,}267)_{\text{десятичное}} = (402{,}21055\ldots)_{\text{восьмеричное}} =$$
$$= (100\ 000\ 010,\ 010\ 001\ 000\ 101\ 101\ \ldots)_{\text{двоичное}}.$$

Пример. Перевести двоичное число, полученное в предыдущем примере, в десятичную систему.

Указание. Сначала переписать число в восьмеричной системе.

В заключение отметим следующее: если читатель знаком только с десятичной системой, то беглое чтение текста 3.6 ничему его не выучит. Он должен читать внимательно и по каждому пункту делать много (2—5) примеров. Переходить к следующему пункту до усвоения предыдущего не следует.

4. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В МАШИНЕ И ПРОГРАММЕ

Рассмотрим запись информации (команд, чисел и их адресов) внутри *ЦЭМ* (в ячейках памяти, регистре *АВК*, ...). Рассмотреть это пора по двум причинам. Во-первых, любознательный читатель мог уже призадуматься, каким образом программа, написанная в содержательных обозначениях (числах, буквах — вплоть до картинок) попадет в конце концов в *ЦЭМ*, т. е. запишется в машине и будет пониматься (исполняться) машиной так, как это задумал программист. Вторая причина важнее. Мы собираемся преобразовывать команды с помощью самой машины. А для этого нужно знать точно, как они выглядят внутри машины.

Любое устройство *ЦЭМ*, предназначенное для помещения информации, состоит из определенного количества *разрядов*. Отдельный разряд представляется нам клеткой, в которую записана цифра 0 или 1. Таким образом, в рассматриваемом устройстве всегда оказывается записанной *строка из нулей и единиц*. Например, строка

001 101 010 (4.1)

Но длинную строку из нулей и единиц читать и писать неудобно. Поэтому ее цифры группируют (как правило) по три и записывают на бумаге восьмеричными цифрами. Так, например, находящаяся в машине строка (4.1), записывается на бумаге строкой

152 (4.2)

Строку (4.1) можно рассматривать как двоичное число (т. е. число в двоичной системе счисления), а строку (4.2) — как равное ему восьмеричное.

Упражнение. Переведите двоичное число (4.1) и восьмеричное (4.2) в десятичную систему.

Решение.

$$1\cdot 2^6+1\cdot 2^5+0\cdot 2^4+1\cdot 2^3+0\cdot 2^2+1\cdot 2^1+0\cdot 2^0=106,$$
$$1\cdot 8^2+5\cdot 8^1+2\cdot 8^0=106.$$

4.1. Представление адреса. Память машины *ММ* состоит из 512 ячеек *), имеющих адреса 0, 1, 2, ..., 511. В двоичной системе адреса ячеек суть числа 0, 1, 10, ..., 111 111 111. Поэтому внутренние устройства, предназначенные для помещения адреса, имеют девять разрядов.

Внутри машины *адрес представляется девятиразрядным двоичным числом* (см. правый столбик табл. 4.1)

Таблица 4.1

Восьмеричные адреса	*Двоичные адреса*
000	000 000 000
001	000 000 001
...	... ...
...	... ...
776	111 111 110
777	111 111 111

Двоичный девятиразрядный адрес мы разобьем на группы по три разряда и *всегда будем записывать трехзначным восьмеричным числом* (см. левый столбик табл. 4.1).

Примеры записи адресов в восьмеричной системе (на бумаге) и двоичной (в машине):

020	000 010 000
317	011 001 111
400	100 000 000
757	111 101 111

4.2. Представление слова. Ячейка памяти *ММ* состоит из 34 разрядов:

34-й	33-й	. . . .	2-й	1-й

*) Память современной машины содержит тысячи ячеек. Мы взяли машину с малой памятью, чтобы не писать многозначных чисел.

Таким образом, в ячейке всегда находится строка из 34-х нулей и единиц. Например,

0 101 001 001 111 011 110 101 000 010 100 011 (4.3)

Условимся (вообще говоря) записывать крайнюю *левую цифру знаком* $+$ *(если она 0) или* $-$ *(если она 1), а остальные собирать в группы по три и изображать восьмеричными цифрами.* Для легкости чтения между тройками цифр оставляют промежутки. В такой записи (она называется *словарной*) слово (4.3) примет вид

$$+51\ 173\ 650\ 243 \qquad (4.4)$$

(проверьте!). Знак $+$ впереди, как правило, не пишут.

Упражнение. Перепишите в двоичной форме слова, заданные в словарной форме:

$+10\ 251\ 674\ 321$ $\qquad -70\ 001\ 001\ 246$

Упражнение. Перепишите в словарной форме слова, заданные в двоичной форме:

0 000001 010011100 101100111 001000010
1 001001 001010001 011001100 001101110

4.3. Представление команды. Мы уже знаем, что когда адрес ячейки попадает в регистр *АВК*, то *ЦЭМ* извлекает из ячейки слово и воспринимает его как команду. *Слово, воспринимаемое как команда, расшифровывается так, как показано на рис.* 4.1.

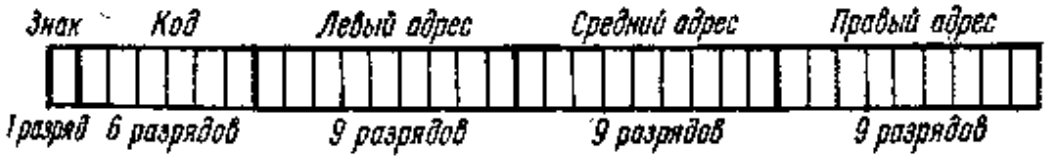


Рис. 4.1.

Из расшифровки команды видно, что *ММ* — трехадресная машина, ее команда содержит код операции и три адреса.

Команды записываются в словарной форме (см. п. 4.2), но знак $+$ впереди опускается потому, что все команды *ММ* (кроме *стоп*) начинаются с нуля (с $+$).

Таким образом, *команда изображается восьмеричными цифрами. Первые две обозначают код, а остальные*

группируются по три и дают восьмеричные номера адресов, упомянутых в команде.

Рассмотрим, например, команду вычитания. Описать ее можно так:

$$u)\ a - b = c$$

Здесь u — адрес ячейки, в которой записана команда, а a, b, c — адреса ячеек, указанных в команде. Вся запись означает, что в ячейке u находится команда

«из содержимого ячейки, указанной по левому адресу, вычти содержимое ячейки, указанной по среднему адресу, и полученную разность занеси в ячейку, указанную в правом адресе».

Пусть для определенности

$$u = \begin{matrix} 001 & 101 & 000 \\ 1 & 5 & 0 \end{matrix}$$

$$a = \begin{matrix} 001 & 100 & 010 \\ 1 & 4 & 2 \end{matrix}; \quad b = \begin{matrix} 000 & 010 & 110 \\ 0 & 2 & 6 \end{matrix}; \quad c = \begin{matrix} 111 & 001 & 011 \\ 7 & 1 & 3 \end{matrix}$$

Операция вычитания в *ММ* имеет код

$$\begin{matrix} 0 & 000 & 101 \\ + & 0 & 5 \end{matrix}$$

Тогда рассматриваемая команда запишется в ячейке u

001101000

в виде

вычитание	*a*	*b*	*c*
0	000101	001100010	000010110	111001011

На бумаге это удобно записать так:

150) 05 142 026 713

Вся запись означает, что в ячейке 150 содержится команда «из числа, находящегося в ячейке 142, вычти число, находящееся в ячейке 026, и запиши полученную разность в ячейку 713».

4.4. Кодирование. Во второй главе мы предупредили, что лист нашей программы будет делиться на левую — *содержательную* часть и правую — *кодированную*. В левой поло-

вине программа пишется в содержательных обозначениях, с которыми мы уже знакомы. Каждую строчку левой половины нужно *закодировать*, т. е. переписать в правую в том виде, какой она должна иметь внутри машины. Только вместо двоичных нулей и единиц пишутся восьмеричные цифры. В правой половине сперва пишется адрес ячейки, а затем слово, находящееся в ней. Таким образом, на правой половине листа пишется та же программа, что и на левой, но уже в том виде, который она должна иметь внутри *ЦЭМ*. Программы пишутся обязательно на стандартных бланках (см. рис. 4.2 на стр. 38). В дальнейшем кодированная программа будет отперфорирована на специальных картах и, с помощью устройства ввода, введена в машину.

Пример. Команда вычитания, рассмотренная в предыдущем примере, запишется в программе так:

Содержательная часть	*Кодированная*
и) $a - b = c$	150 | 05 142 026 713

Здесь, как и в дальнейшем, перед кодом опущен знак $+$ (0). Приходится различать адрес команды (150) от адреса в команде (142, 026 или 713 *)).

Теперь легко понять, почему в п. 2.5 мы настаивали, чтобы вычитание обозначалось так

$$a - b = c, \tag{4.5}$$

но ни в коем случае не как

$$c = a - b \text{ или } -b + a = c. \tag{4.6}$$

Адреса необходимо педантично писать в указанном нами порядке для того, чтобы они шли в содержательной и кодированной части в одном и том же порядке. В содержательной части дополнительно пишутся лишь некоторые значки ($=$, ;) и код операции передвигается на середину.

Это дает, с одной стороны, возможность писать в содержательной части операции, привычные математику, а с другой — облегчает кодирование. Само собой разумеется, что обозначения операций для содержательной части мы выбираем

*) Адрес команды предлагалось называть *прописка*, но название пока не привилось.

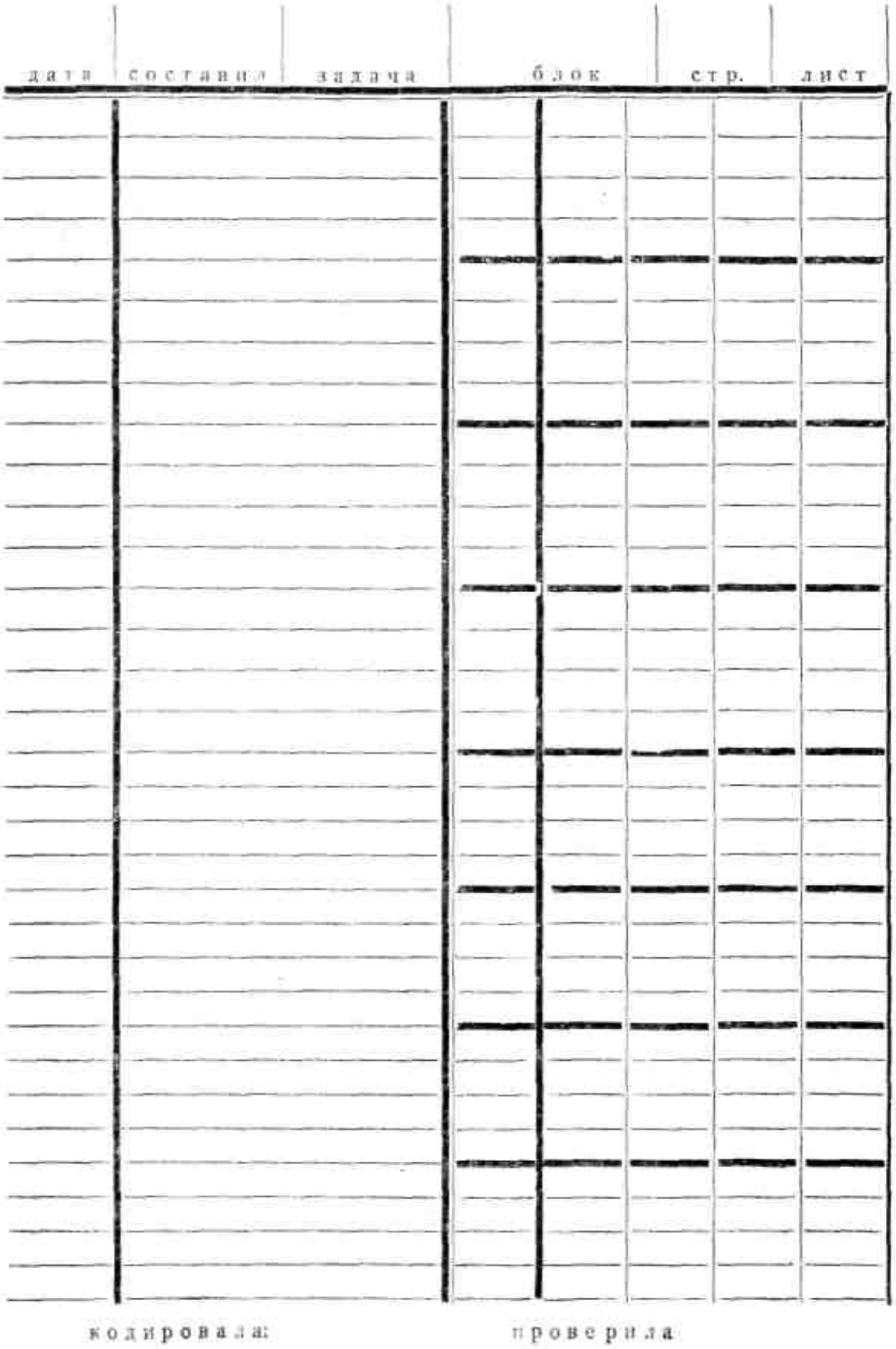


Рис. 4.2.

с учетом истинного порядка адресов в командах машины *ММ*.

В п. 2.5 был приведен список команд *ММ*, с которыми мы познакомились. Слева от команд были приведены их коды — двузначные восьмеричные числа (знак + впереди опущен). Выпишите себе на бумажку краткие обозначения команд и их коды в таком виде, как это показано в табл. 4.2.

Таблица 4.2

Команда	*Код*	*Команда*	*Код*	*Команда*	*Код*
$-$	00	$+$	04		
$\leftrightarrow$	01	$-$	05	*печ.*	10
$\|\geqslant\|$	02	$\cdot$	06	*стоп*	—00
$\geqslant$	03	:	07		

Команда *стоп* начинается со знака «минус» в отличие от всех остальных команд.

Упражнение. Закодируйте программу из п. 2.1 для вычисления многочлена

$$y = (((ax + b)\,x + c)\,x + d)\,x + e.$$

Примите, что первая ячейка программы имеет адрес 155:

$$1) \sim 155.$$

Ячейки, используемые в программе, пусть имеют следующие адреса:

$$a \sim 041, \quad d \sim 136, \quad x \sim 100,$$
$$b \sim 135, \quad e \sim 715, \quad y \sim 020.$$
$$c \sim 021,$$

Решение представлено программой 4.1.

Программа 4.1

1)	$a \cdot x = y$	155	06 041 100 020
2)	$y + b = y$	156	04 020 135 020
3)	$y \cdot x = y$	157	06 020 100 020
4)	$y + c = y$	160	04 020 021 020
5)	$y \cdot x = y$	161	06 020 100 020
6)	$y + d = y$	162	04 020 136 020
7)	$y \cdot x = y$	163	06 020 100 020
8)	$y + e = y$	164	04 020 715 020
9)	*стоп*	165	—00 000 000 000

Обратите внимание, что после команды 157 идет команда 160 (система восьмеричная).

Список адресов ячеек памяти для слов, употребляемых в содержательной части программы, называется *словарем.*

Упражнение. Закодируйте программу вычисления

$$y = \varphi(x_1)\varphi(x_2)/\varphi(x_3), \quad \varphi(x) = (a + bx)/(c - dx^2)$$

из п. 2.5 по словарю

1) ~ 250	x ~ 110	a ~ 221
p) ~ 175	y ~ 112	b ~ 222
x_1 ~ 101	φ ~ 111	c ~ 223
x_2 ~ 102	R ~ 100	d ~ 224
x_3 ~ 103		

Решение представлено программой 4.2.

Программа 4.2

1)	$x_1 \to x$	250	00 101 110 251
2)	$3 \leftrightarrow Q;\ P$	251	01 252 203 175
3)	$\varphi \to y$	252	00 111 112 253
4)	$x_2 \to x$	253	00 102 110 254
5)	$6 \leftrightarrow Q;\ P$	254	01 255 203 175
6)	$y \cdot \varphi = y$	255	06 112 111 112
7)	$x_3 \to x$	256	00 103 110 257
8)	$9 \leftrightarrow Q;\ P$	257	01 260 203 175
9)	$y : \varphi = y$	260	07 112 111 112
10)	*стоп*	261	— 00 000 000 000
P)	$b \cdot x = \varphi$	175	06 222 110 111
	$a + \varphi = \varphi$	176	04 221 111 111
	$x \cdot x = R$	177	06 110 110 100
	$d \cdot R = R$	200	06 224 100 100
	$c - R = R$	201	05 223 100 100
	$\varphi : R = R$	202	07 111 100 100
Q)	*выход*	203	— —

Ячейка 203 *в последнем упражнении, как и всякая рабочая ячейка, получает в кодированной части адрес* (чтобы ее не использовали при размещении программы), *но вместо кодированного слова в ней пишется прочерк,* ибо в нее ничего не нужно вводить до начала работы программы.

Составление словаря не является механической работой — нужно следить, чтобы разные части не налезали друг на друга и чтобы использовалось поменьше рабочих ячеек (в содержательной части они могли быть обозначены по-разному для удобства понимания). Наоборот, кодирование должно выполняться совершенно автоматически.

Упражнение. Составить словари и закодировать две задачи из § 2 (по выбору).

Задача 1. Рассмотрим следующий метод вычисления квадратного корня *). Пусть x_n есть приближенное значение $\sqrt{x}$. Тогда следующее приближение равняется

$$x_{n+1} = \left(\frac{x}{x_n} + x_n\right)/2.$$

Если $x_n \geqslant \sqrt{x}$, то $x_n \geqslant x_{n+1} \geqslant \sqrt{x}$. Если ε_n обозначает относительную ошибку приближения x_n, т. е. $\varepsilon_n = (x_n - \sqrt{x})/\sqrt{x}$, то при $x_n \geqslant \sqrt{x} > 0$

$$\varepsilon_{n+1} = \frac{\varepsilon_n^2}{2(1+\varepsilon_n)}$$

(проверить!) и, следовательно, $\varepsilon_{n+1} < \varepsilon_n/2$ всегда, а при $\varepsilon_n < 0{,}1$ число верных знаков на каждой итерации примерно удваивается (*если* $\varepsilon_n < 0{,}1$, *то* $\varepsilon_{n+1} < 0{,}01$; $\varepsilon_{n+2} < 0{,}0001$, ...).

Составьте и закодируйте самовозобновляющуюся программу вычисления корня $\sqrt{x}$ для любого $x > 0$, занимающую минимальное число ячеек.

Словарь:

$x \sim 020$,	$0 \sim 000$,	рабочие ячейки $\sim$ 022, 023, ...,
$\sqrt{x} \sim 021$,	$1_n \sim 014$,	начало программы $\sim$ 201.
	$1/2 \sim 015$,	

Указание. Программа займет 6 ячеек (не считая ячеек для x, $\sqrt{x}$, для чисел 0, 1, 1/2 и рабочих ячеек). В качестве первого приближения удобно взять $x_1 = (1+x)/2$, так что $x_1 > \sqrt{x}$ при любом $x > 0$ (проверить). Итерации будем продолжать до тех пор, пока (с машинной точностью) значения x_n будут убывать, т. е. пока $x_{n+1} < x_n$.

Решение легко уложить в 6 ячеек (см. программу 4.3), если (вопреки указанию) построить первое приближение $x_1 = (x+1)$.

---

*) Принадлежащий Герону Александрийскому — автору известной формулы для площади треугольника.

Программа 4.3

1)	$1_n + x = V$	201	04 014 020 021
2)	$V + 0 = R$	202	04 021 000 022
3)	$x : V = (x/V)$	203	07 020 021 023
4)	$(x/V) + V = (2V)$	204	04 023 021 023
5)	$(2V) \cdot (1/2) = V$	205	06 023 015 021
6)	$V \geqq R$	206	03 021 022 202

Но можно уложить в 6 ячеек и программу, в которой $x_1 = (1 + x)/2$ (см. программу 4.4).

Программа 4.4

1)	$1_n + x = (2V)$	201	04 014 020 022
2)	$(2V) + 0 = R$	202	04 022 000 023
3)	$(2V) \cdot (1/2) = V$	203	06 022 015 021
4)	$x : V = R1$	204	07 020 021 022
5)	$R1 + V = (2V)$	205	04 022 021 022
6)	$(2V) \geqq R$	206	03 022 023 202

4.5. Представление чисел. Мы уже знаем, что по виду слова нельзя судить, команда оно или число. Но и в качестве чисел слова могут расшифровываться и перерабатываться по-разному, различными командами. В *ЦЭМ* можно выделить четыре типа восприятия слов (рис. 4.3).

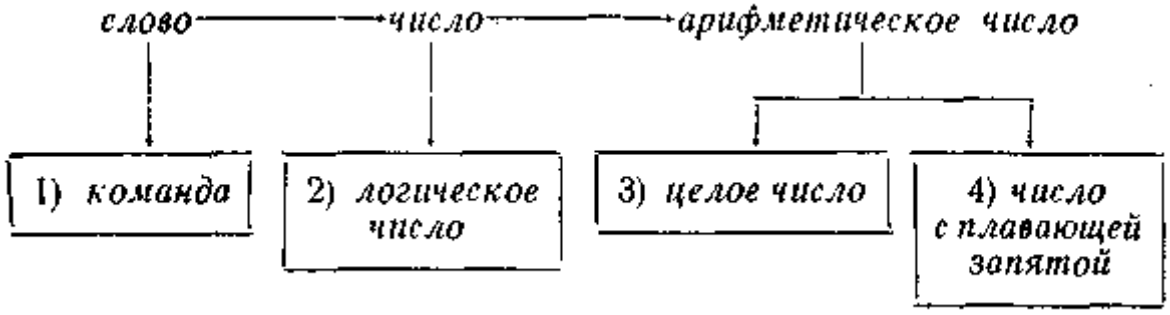


Рис. 4.3.

1. Если адрес некоторой ячейки памяти попадает в регистр *АВК*, то слово из этой ячейки расшифровывается машиной как *команда*.

При выполнении команды (например, команды вычитания $a - b = c$) машина вызывает слова из ячеек, указанных в команде (в нашем примере — слова a и b). *Слова, вызванные командой* (а не регистром *АВК*), *называются числами*. Таким образом,

любая команда, как и любое число, прежде всего *является словом,* ибо занимает отдельную ячейку памяти, и наоборот, всякое слово может выступить и в роли команды и в роли числа. В машине *ММ* они изображаются строками из 34 нулей и единиц, *а на бумаге записываются в словарной форме.*

2. По способу расшифровки вызываемых ими чисел команды машины *ММ* делятся всего на три группы. *Команды логической группы* обращаются с вызванными числами просто как со строками из 34 нулей и единиц. В этом случае числа называют *логическими числами,* или даже просто *словами.*

3. Арифметические команды расшифровывают вызванные ими числа *в двоичной системе.* Но эти числа можно считать целыми или дробными. *Арифметические команды группы Целых Чисел* (*ЦЧ*) расшифровывают одну часть слова как *целое число,* а другую не рассматривают.

4. Наконец, *арифметические команды группы плавающей запятой* (*ПЗ*) одну часть слова расшифровывают как двоичное число, а другую — как указатель положения запятой в этом числе. Таким образом, запятая может перемещаться по числу, от чего и происходит название группы *ПЗ.*

Сейчас мы перейдем к систематическому рассмотрению всех этих групп команд и типов восприятия слова. Наперед заметим лишь, что по виду слова нельзя судить, к какой группе оно относится. Однако, для краткости речи, мы будем говорить о *командах, логических числах, Целых Числах* и *числах с плавающей запятой,* имея в виду способ восприятия слов машиной или командами соответствующей группы.

4.6. Действия с плавающей запятой (*ПЗ*) — это знакомые нам четыре арифметические команды, приведенные в таблице 4.3.

Все вычисления в *ЦЭМ* производятся с помощью этих действий. Остальные команды лишь обслуживают вычислительный процесс.

Числа с плавающей запятой вводятся в машину в десятичной форме (как правило). Перед началом вычислений специальная программа переводит их в двоичную систему и записывает в форме *ПЗ.* Именно с этими *переведенными*

числами и производятся все вычисления. Если же число с *ПЗ* надо отпечатать, то другая программа переводит его в десятичную систему и печатает.

Таблица 4.3

Код	*Название*	*Обозначение*
04	сложение	$a + b = c$
05	вычитание	$a - b = c$
06	умножение	$a \cdot b = c$
07	деление	$a : b = c$

Наш опыт показывает, что для обучения программированию прикладнику не обязательно знать, как именно записываются числа *ПЗ* в ячейке памяти, достаточно знать твердо лишь нижеследующее:

1. Действия *ПЗ* машины *ММ* работают с числами, лежащими по абсолютной величине в интервале (примерно) *от* 10^{-9} *до* 10^{+9} (число 10 — десятичное).

2. Точно записать все эти числа невозможно, как невозможно точно записать число 1/3 десятичной дробью. Точность изображения числа с *ПЗ* в машине *ММ* примерно соответствует восьми значащим цифрам десятичной системы (относительная ошибка меньше 10^{-8}). С такой же примерно точностью выполняются арифметические действия с *ПЗ*. Число, меньшее 10^{-9}, в форме *ПЗ* машина заменяет нулем со знаком $+$.

3. Знак числа с *ПЗ* записывается в левом (34-м) разряде ячейки памяти.

4. В содержательной программе число с *ПЗ* пишется в любой десятичной форме. Около небольших целых чисел ставится значок $_n$, чтобы они не путались с номерами команд. Например,

$$0;\ 1_n;\ -2{,}53 \cdot 10^{-6};$$

$$0{,}03124;\ 31{,}248756 \cdot 10^{3}. \qquad (4.7)$$

5. Для ввода десятичного числа в машину договоримся представлять его в виде произведения

$$a = \pm 10^{\pm n} \cdot \alpha,$$

где n — целое, $0 \leqslant n \leqslant 9$ и $0 \leqslant \alpha < 1$, и записывать в кодированной части подряд:

знак числа; знак показателя; показатель; семь цифр α, начиная с первой за запятой.

Наша запись согласована с определенной программой перевода «10 — 2» и, разумеется, целиком обусловлена последней. Пример кодированной записи чисел:

0	$= + 10^{0} \cdot 0$	$+ + 0\ 0000000$
1	$= + 10^{1} \cdot 0{,}1$	$+ + 1\ 1000000$
—2,53	$= - 10^{-5} \cdot 0{,}253$	$- - 5\ 2530000$
0,03124	$= + 10^{-1} \cdot 0{,}3124$	$+ - 1\ 3124000$
31,248756	$= + 10^{5} \cdot 0{,}3124876$	$+ + 5\ 3124876$

Два знака перед числом покажут перфораторщице, *что далее идут десятичные цифры, а не восьмеричные* (как в словарной записи).

Первые четыре числа можно закодировать и в других формах. Например, число 1 можно было кодировать и так:

$$+ + 2\ 0100000 \text{ или } + + 7\ 0000001.$$

Но при иной кодировке число 31,248756 (если взять $n \neq 5$) изобразится с меньшей точностью.

Упражнение. Закодировать в десятичной форме числа

$$0{,}000179 \cdot 10^{12};\ -1234567;\ 1/3;\ \pi;\ 1/7.$$

Для примера: $1/3 = + 10^{0} \cdot 0{,}3333333 \sim + + 0\ 3333333$.

4.7. Действия с Целыми Числами (*ЦЧ*). Эта группа состоит из команд, приведенных в табл. 4.4. Они

Таблица 4.4

Код	*Название*	*Обозначение*
14	сложение *ЦЧ*	$a +, b = c$
15	вычитание *ЦЧ*	$a -, b = c$
16	умножение *ЦЧ*	$a \cdot, b = c$
17	деление *ЦЧ*	$a :, b = c$

действуют со словами как с целыми числами, записанными в двоичной системе счисления. При этом слова расшифровываются так:

Разряды 27—1 воспринимаются как целое число в двоичной записи, а 34-й разряд — как знак этого числа (рис. 4.4).

Рис. 4.4.

Кодовые разряды 33—28 не участвуют в действии и переносятся из слова *a* (упомянутого в левом адресе команды) в результат *c* без изменения.

Действия, которые производят команды, ясны из их названий. Добавить надо лишь следующее. Если результат слишком велик для записи (выражается числом, содержащим больше 27 разрядов), то разряды старше 27-го разряда пропадают. Результат, равный нулю, обязательно записывается со знаком $+$. Результатом деления (:,) является целая часть частного.

В содержательной программе Целые Числа (точнее — аргументы действий с целыми числами) пишутся либо в виде команд, либо как целые числа в восьмеричной системе. В последнем случае иногда ставят значок $_8$ внизу, чтобы избежать путаницы с номерами команд и числами в форме *ПЗ*.

Примеры записи Целых Чисел в содержательной и кодированной частях программы и внутри ячейки памяти приведены в табл. 4.5.

Таблица 4.5

0_8	+00 000 000 000	0 000000 000000000 000000000 000000000
1_8	+00 000 000 001	0 000000 000000000 000000000 000000001
-2_8	−00 000 000 002	1 000000 000000000 000000000 000000010
17025_8	+00 000 017 025	0 000000 000000000 000001111 000010101
2^6	+00 000 000 010	0 000000 000000000 000000000 000001000
-2^9	−00 000 001 000	1 000000 000000000 000000001 000000000
2^{28}	+00 400 000 000	0 000000 100000000 000000000 000000000
2^3-1	+00 000 000 007	0 000000 000000000 000000000 000000111
2^9-1	+00 000 000 777	0 000000 000000000 000000000 111111111
$2^{27}-1$	+00 777 777 777	0 000000 111111111 111111111 111111111

В вычислительных программах действия с Целыми Числами играют вспомогательную роль и служат для преобразования команд. Мы вернемся к ним в следующей главе.

Упражнение. Записать в словарной форме Целые Числа $2^{18} - 2^9 + 1$; десятичное число 512; 2^{24}.

Ответы:

+ 00 000 777 001; + 00 000 001 000; + 00 100 000 000.

Упражнение. Пусть

$a_1 = +41\ 250\ 127\ 357$ $\quad a_5 = +17\ 001\ 000\ 002$

$a_2 = +64\ 000\ 001\ 000$ $\quad a_6 = -15\ 010\ 005\ 001$

$a_3 = -32\ 000\ 000\ 002$ $\quad a_7 = +06\ 777\ 000\ 003$

$a_4 = -72\ 777\ 777\ 777$ $\quad a_8 = +00\ 070\ 001\ 777$

Тогда

$a_1 +, a_5 = +41\ 251\ 127\ 361$ $\quad a_6 +, a_1 = +15\ 240\ 125\ 356$

$a_1 -, a_5 = +41\ 247\ 127\ 355$ $\quad a_2 \cdot, a_3 = -64\ 000\ 002\ 000$

$a_5 +, a_1 = +17\ 251\ 127\ 361$ $\quad a_2 :, a_3 = -64\ 000\ 000\ 400$

$a_5 -, a_1 = -17\ 247\ 127\ 355$ $\quad a_1 \cdot, a_2 = +41\ 127\ 357\ 000$

$a_7 +, a_5 = +06\ 000\ 000\ 005$ $\quad a_1 :, a_2 = +41\ 000\ 250\ 127$

$a_1 +, a_7 = +41\ 247\ 127\ 362$ $\quad a_6 \cdot, a_2 = -15\ 000\ 010\ 005$

$a_1 +, a_8 = +41\ 340\ 131\ 356$ $\quad a_8 :, a_2 = +32\ 000\ 000\ 000$

$a_8 -, a_4 = +00\ 070\ 002\ 000$ $\quad a_8 +, a_4 = -00\ 707\ 776\ 000$

5. ПРЕОБРАЗОВАНИЕ КОМАНД

Для краткости записи договоримся обозначать через

$$(p,\ q,\ r) \qquad (0 \leqslant p,\ q,\ r \leqslant 777)$$

ячейку (и ее содержимое), имеющую нули в знаковом и кодовых разрядах и содержащую числа p, q, r в левом, среднем и правом адресах. Числа p, q и r в записи (p, q, r) будем писать в восьмеричной системе. Кроме того, будем обозначать одной буквой Ω число 777. Например,

(0, 0, 0)	+00 000 000 000
(0, 1, Ω)	+00 000 001 777
(35, Ω, 2)	+00 035 777 002

Задача 1. Чему равно слово (p, q, r) как *ЦЧ*.

Ответ. $(p, q, r) = p8^0 + q8^3 + r$.

Задача 2 (для программистов). Что произойдет при выполнении команды (p, q, r)?

Ответ. Содержимое ячейки (p, q, r) является командой

$$p \rightharpoonup q;\ r.$$

При ее выполнении слово из ячейки с номером p (но не число p) будет перенесено в ячейку с номером q и управление перейдет к ячейке с номером r.

Некоторые константы употребляются в большом числе программ. Естественно собрать их в *библиотечные константы* и считать всегда введенными на постоянные легко запоминаемые места в начале памяти. Здесь же будут находиться числа с плавающей запятой 1, 10, π, e и восемь рабочих ячеек α, $R1$, $R2$, ..., чтобы не приходилось всякий раз задавать их адреса. Библиотечные константы приведены в табл. 5.1.

Кроме того, мы будем считать, что ячейки от 027 до 177 заняты библиотекой стандартных подпрограмм (о которой речь пойдет в гл. 8) и будем писать свои программы, начиная с адреса не меньше 200.

Таблица 5.1

Обозначение	*Адрес*	*Содержимое ячейки*
(0, 0, 0) или 0	000	+ 00 000 000 000
(0, 0, 1) или $1_{Ц}$	001	+ 00 000 000 001
(0, 1, 0)	002	+ 00 000 001 000
(0, 1, 1)	003	+ 00 000 001 001
(1, 0, 0)	004	+ 00 001 000 000
(1, 0, 1)	005	+ 00 001 000 001
(1, 1, 0)	006	+ 00 001 001 000
(1, 1, 1)	007	+ 00 001 001 001
10_n	010	+ 04 500 000 000
(0, 0, Ω)	011	+ 00 000 000 777
(0, Ω, 0)	012	+ 00 000 777 000
(Ω, 0, 0)	013	+ 00 777 000 000
1_n	014	+ 01 400 000 000
$(1/2)_n$	015	+ 00 400 000 000
π_n	016	+ 02 622 077 325
e_n	017	+ 02 533 741 243
α	020	—
$R1$	021	—
$R2$	022	—
$R3$	023	—
$R4$	024	—
$R5$	025	—
$R6$	026	—
$R7$	027	—

5.1. Преобразование команд действиями *ЦЧ*. Начнем с простейших примеров.

1. Пусть в ячейке u находится команда

$u)$ $0 \dot{+} 0 = a$ | 225 | 04 000 000 313

Прибавим к ней (0, 0, 1) как Целое Число, т. е. выполним команду

$w)$ $n \dot{+}$, (0, 0, 1) $= v$ | 130 | 14 225 001 227

В результате этого в ячейке v образуется команда

$v)$ $0 \dot{+} 0 = (a \dot{+} 1)$ | 227 | 04 000 000 314

Таким образом, из команды u, записывающей число 0 в ячейку $a = 313$, мы получили команду v, которая может записать нуль в ячейку 314, следующую за a. Принципиальным здесь было то, что *мы получили команду v, не зная адреса* 313 *ячейки a* (ситуация отнюдь не надуманная).

Заметим, что уже для этого надо было знать точно, как записывается команда сложения $(0+0=a)$ в ячейке памяти машины *ММ*. Если бы сумма записывалась не по правому адресу команды $(0+0=a)$, а по левому $(a=0+0)$, то добавлять к u пришлось бы не (0, 0, 1), а (1, 0, 0).

2. Пусть в ячейке u находится команда

$u)$ $0+a=b$ | 225 | 04 000 313 300

Прибавим к ней (0, 1, 0) как Целое Число:

$w)$ $u+$, $(0, 1, 0)=v$ | 230 | 14 225 002 227

В результате этого в ячейке v образуется команда

$v)$ $0+(a+1)=b$ | 227 | 04 000 314 300

которая переносит в ячейку b содержимое ячейки, следующей за a.

3. Пусть выделены ячейка b и две группы последовательных ячеек

$$x_1, x_2, \ldots; y_1, y_2, \ldots$$

Пусть имеется команда

$u)$ $x_1 : b = y_1$ | 225 | 07 315 300 406

Прибавим к команде u число (1, 0, 1)

$w)$ $u+$, $(1, 0, 1)=v$ | 230 | 14 225 005 227

В результате этого в ячейке v образуется команда

$v)$ $x_2 : b = y_2$ | 227 | 07 316 300 407

Упражнение. Из команды

$$u)\ \Sigma + a = \Sigma$$

получить в ячейке v команду, прибавляющую к Σ число из ячейки, следующей за ячейкой a.

Ответ:

$$w)\ u+, (0, 1, 0) = v$$

Важная задача. Пусть в последовательных ячейках находятся числа $x_1, x_2, \ldots$, а в ячейке v команда

$$v)\ \Sigma + x_1 = \Sigma.$$

Поместить в ячейку Σ сумму четырех чисел $x_1 + x_2 + x_3 + x_4$, если адрес $v = 400$. Решение этой задачи дается программой 5.1.

Программа 5.1

s)	v +, $0 = (s+4)$	200	14 400 000	204
$s+1$)	$(s+4)+$, $(0, 1, 0) = (s+5)$	201	14 204 002	205
$s+2$)	$(s+5)+$, $(0, 1, 0) = (s+6)$	202	14 205 002	206
$s+3$)	$(s+6)+$, $(0, 1, 0) = (s+7)$	203	14 206 002	207
$s+4$)	—	204	—	
$s+5$)	—	205	—	
$s+6$)	—	206	—	
$s+7$)	—	207	—	
$s+8$)	*стоп*	210	—00 000 000	000

Если эту программу ввести в память (введутся команды s, $s+1$, $s+2$, $s+3$ и $s+8$, а остальные ячейки останутся как были) и начать с команды s, то в ячейке Σ образуется сумма $x_1 + x_2 + x_3 + x_4$.

Действительно, команда s сложит v с нулем и запишет сумму в $s+4$. От этого команда, находящаяся в v, перенесется в $s+4$, и в $s+4$ будет находиться

$$s+4) \quad \Sigma + x_1 = \Sigma$$

Затем команда $s+1$ добавит к команде $s+4$ число (0, 1, 0) и запишет сумму в $s+5$, где таким образом окажется команда

$$s+5) \quad \Sigma + x_2 = \Sigma$$

После этого команда $s+2$ образует в $s+6$ команду

$$s+6) \quad \Sigma + x_3 = \Sigma$$

а команда $s+3$ образует

$$s+7) \quad \Sigma + x_4 = \Sigma$$

После этого управление перейдет к $s+4$, которая сложит 0 (находящийся по условию в Σ) с x_1 и запишет в Σ. Таким образом, в Σ образуется число x_1. Команда $s+5$ добавит

к $\sum$ число x_2 и в $\sum$ образуется x_1+x_2 и т. д. После команды $s+7$ в $\sum$ образуется сумма $x_1+x_2+x_3+x_4$, управление передается $s+8$, и машина остановится.

Эта программа представляет исключительную важность для всего курса программирования. Поэтому обучающемуся предлагается выполнить последовательно команды s, $s+1$, $s+2$, $s+3$, выписывая каждый раз результат в соответствующую ячейку (т. е. в $s+4$, $s+5$, $s+6$, $s+7$) и убедиться, что в дальнейшем в $\sum$ соберется нужная сумма. Содержание ячейки v можно принять таким:

$v)$ $\sum + x_1 = \sum$ | 400 | 04 300 301 300

(тем самым приняты адреса 300 для $\sum$, 301 для x_1, 302 для x_2, ...).

Другое решение этой же задачи доставляется программой 5.2.

Программа 5.2

$v \to (я+1)$	200	00	400	201	201
—	201			—	
$(я-1)$ +, $(0, 1, 0) = (я+1)$	202	14	201	002	203
—	203			—	
$(я-1)$ +, $(0, 1, 0) = (я+1)$	204	14	203	002	205
—	205			—	
$(я-1)$ +, $(0, 1, 0) = (я+1)$	206	14	205	002	207
—	207			—	
стоп	210	—00	000	000	000

Здесь, как и в дальнейшем, в данной команде ее адрес обозначается буквой $я$, а адреса предыдущей и следующей обозначаются $я-1$ и $я+1$. Убедимся, что и эта программа решает задачу.

Упражнение. Какие положительные константы *C1*, *C2*, *C3* нужно прибавить как Целые Числа к командам *s*1, *s*2, *s*3 (или вычесть), чтобы из них получились команды *t*1, *t*2, *t*3, если

*s*1) $u+v=w$ *s*2) $u \to v;\ w$ *s*3) $3 \to v = w$

*t*1) $u_1+v=w_1$ *t*2) $u_2 \to v;\ w_1$ *t*3) $5 \to v = w$

а адреса ячеек имеют значения

$u=215$ $v=315$ $w=415$

$u_1=214$ $w_1=413$ $u_2=216$

Ответ:

вычесть (—,) *C1*) (1, 0, 2)
прибавить (+,) *C2*) (0, 2, 776)
прибавить (+,) *C3*) (2, 0, 0)

Обратить внимание на получение *t2* из *s2*!

5.2. Логические действия. Познакомимся с последней группой команд машины *ММ* — *логическими действиями*. Они приведены в табл. 5.2.

Таблица 5.2

Код	*Название*	*Обозначение*
12	логическое сложение	$a \vee b = c$
13	логическое умножение	$a \wedge b = c$
11	сдвиг	$a = b = c$

Они рассматривают вызванные ими числа просто как *строки* из 34-х нулей и единиц.

При *логическом сложении* слов *a* и *b* образуется слово *c*, у которого единицы находятся в тех разрядах, где имеются единицы хотя бы в одном из слов *a* или *b*. В остальных разрядах слова *c* образуются нули (т. е. в тех разрядах, где и у *a* и у *b* стоят нули).

При *логическом умножении* слов *a* и *b* образуется слово *c* с единицами в тех разрядах, где стоят единицы в обоих словах *a* и *b*. В остальных разрядах слова *c* образуются нули. Таким образом, разряды слова *c* получаются перемножением разрядов *a* и *b* с одинаковыми номерами.

Пример образования логической суммы и логического произведения:

$a = 0\ 111101\ 111000000\ 001110110\ 011100110 = +75\ 700\ 166\ 346$
$b = 1\ 111000\ 000001111\ 101101100\ 001100000 = -70\ 017\ 554\ 140$
$a \vee b = 1\ 111101\ 111001111\ 101111110\ 011100110 = -75\ 717\ 576\ 346$
$a \wedge b = 0\ 111000\ 000000000\ 001100100\ 001100000 = +70\ 000\ 144\ 140$

Чтобы выполнить логическое сложение или умножение над числами, заданными в словарной форме, надо представить каждую восьмеричную цифру в двоичной системе, сделать нужное действие и записать результат восьмеричными цифрами.

Упражнение. Проверьте табл. 5.3 логического сложения и умножения восьмеричных цифр (над диагональю — сумма, а под диагональю — произведение).

Таблица 5.3

b \ a	0	1	2	3	4	5	6	7	
0	0	1	2	3	4	5	6	7	
1	0	1	3	3	5	5	7	7	
2	0	0	2	3	6	7	6	7	
3	0	1	2	3	7	7	7	7	$a \vee b$
4	0	0	0	0	4	5	6	7	
5	0	1	0	1	4	5	7	7	
6	0	0	2	2	4	4	6	7	
7	0	1	2	3	4	5	6	7	
					$a \wedge b$				

Например:

$2 = 010$ $\qquad$ $2 = 010$

$3 = 011$ $\qquad$ $4 = 100$

$2 \vee 3 = 011 = 3$ $\qquad$ $2 \vee 4 = 110 = 6$

$2 \wedge 3 = 010 = 2$ $\qquad$ $2 \wedge 4 = 000 = 0$

Заметим, что при логическом умножении на 0 всегда получается 0. При логическом умножении любой цифры на 7 получается исходная цифра. При логическом сложении любой цифры с нулем получается исходная цифра. При логическом сложении с цифрой 7 всегда получается 7.

Упражнение. Сделайте следующие примеры (напомним, что + изображает цифру 0, а — изображает 1).

$a = +\,27\,456\,121\,103$ $\qquad$ $a = +\,27\,456\,121\,103$

$b = -\,77\,777\,777\,777$ $\qquad$ $b = -\,27\,456\,121\,103$

$a \vee b = -\,77\,777\,777\,777$ $\qquad$ $a \vee b = -\,27\,456\,121\,103$

$a \wedge b = +\,27\,456\,121\,103$ $\qquad$ $a \wedge b = +\,27\,456\,121\,103$

$a = +\,27\,456\,121\,103$ $\qquad$ $a = +\,05\,315\,000\,713$

$b = +\,00\,000\,777\,000$ $\qquad$ $b = +\,00\,000\,121\,000$

$a \vee b = +\,27\,456\,777\,103$ $\qquad$ $a \vee b = +\,05\,315\,121\,713$

$a \wedge b = +\,00\,000\,121\,000$ $\qquad$ $a \wedge b = +\,00\,000\,000\,000$

5.3. Преобразование команд логическим сложением и умножением. Числа, с которыми работают логические действия, зачастую записываются в содержательной программе в виде команд или в обозначениях (p, q, r). Договоримся еще обозначать через

$$(\pm s;\ p,\ q,\ r),\ \text{где}\ 0 \leqslant s \leqslant 77\ \text{и}\ 0 \leqslant p,\ q,\ r \leqslant 777,$$

слово (p, q, r) со знаком и кодом $\pm s$. Если в знаковом и всех кодовых разрядах стоят семерки, то мы будем писать одну букву Ω.

Так, например, $(+0;\ p,\ q,\ r) = (p,\ q,\ r)$

$(+0;\ 0,\ 0,\ 0)$	$+00\ 000\ 000\ 000$
$(-0;\ 0,\ 0,\ 0)$	$-00\ 000\ 000\ 000$
$(\Omega;\ 0,\ 0,\ 0)$	$-77\ 000\ 000\ 000$
$(-25;\ 35,\ 0,\ 200)$	$-25\ 035\ 000\ 200$
$(\Omega;\ \Omega,\ \Omega,\ \Omega)$	$-77\ 777\ 777\ 777$

Логическое умножение употребляется, чтобы вырезать часть слова, — например, определенный адрес в команде, — а остальные части погасить (нулями). Логическим сложением удобно «вставлять» нужную часть слова (вместо нулей). Пусть, например, имеются команды u, v, w:

$u)\ a + x = a$	300	04 315 325 315
$v)\ y + a = a$	400	04 326 315 315
$w)\ 0 + 0 = z$	500	04 000 000 327

Выполним следующую программу:

1) $u \wedge (0,\ \Omega,\ 0) = R1$ 3) $R1 \vee R2 = R3$
2) $v \wedge (\Omega,\ 0,\ 0) = R2$ 4) $w \vee R3 = R4$

В результате в ячейке $R4$ образуется команда

$$R4)\quad y + x = z$$

Действительно, после первой, второй и т. д. команд в ячейках $R1$, $R2$, $R3$, $R4$ будет

$R1)\ +00\ 000\ x\ 000$ $R3)\ +00\ y\ x\ 000$
$R2)\ +00\ y\ 000\ 000$ $R4)\ +04\ y\ x\ z$

Приведем для ясности выполнение первой команды в восьмеричной и двоичной системе:

1_8)
$$\begin{array}{r} u \sim +04\quad a\quad x\quad a \\ \wedge \\ (0,\ \mathfrak{Q},\ 0) \sim +00\ 000\ 777\ 000 \\ \hline (0,\ x,\ 0) \sim +00\ 000\ x\ 000 \end{array}$$

1_2)
$$\begin{array}{r} u \sim 0\ 000100\ 011001101\ 011010101\ 011001101 \\ \wedge \\ (0,\ \mathfrak{Q},\ 0) \sim 0\ 000000\ 000000000\ 111111111\ 000000000 \\ \hline (0,\ x,\ 0) \sim 0\ 000000\ 000000000\ 011010101\ 000000000 \end{array}$$

5.4. Команда сдвига обозначается двумя способами, в зависимости от направления сдвига

$$a \leftarrow b = c \text{ или } a \rightarrow b = c.$$

Она сдвигает слово b влево или вправо на определенное число разрядов и записывает результат в c. Вместо «сдвигаемых» разрядов вводятся нули. Из трех восьмеричных цифр левого адреса a команды сдвига левая цифра указывает направление сдвига: влево, если она нуль, и вправо, если она единица. Остальные две цифры дают в восьмеричной системе число разрядов, на которое сдвигается слово b. Таким образом, команда сдвига не вполне адресная (она работает с адресом a, а не со словом из ячейки a).

В содержательной части направление сдвига указывается стрелкой. Например,

команды	*кодируются так:*
$v)$ $11 \leftarrow b = c$	11; 011, b, c
$w)$ $11 \rightarrow b = d$	11; 111, b, d

и означают сдвиг слова b на 11_8 (т. е. девять в десятичной системе) разрядов влево и вправо. Если, например,

$$b = 0\ 000100\ 011001101\ 011010101\ 001001111 = \\ = (4;\ 315,\ 325,\ 117)$$

то после выполнения команды v

$$c = 1\ 001101\ 011010101\ 001001111\ 000000000 = \\ = (-15;\ 325,\ 117,\ 000)$$

а после выполнения команды w

$$d = 0\ 000000\ 000000100\ 011001101\ 011010101 = \\ = (0;\ 00,\ 4,\ 315,\ 325)$$

Команда сдвига нужна при перестановке адресов в имеющихся командах. Пусть, например,

$$u)\ x \dotplus 0 = z \\ v)\ y \dotplus 0 = 0$$

и нужно получить в $R3$ команду $x \dotplus y = z$. Для этого напишем и выполним программу

$$1)\ v \bigwedge (\Omega, 0, 0) = R1 \\ 2)\ 11 \rightarrow R1 \quad = R2 \\ 3)\ u \bigvee R2 \quad = R3$$

При работе этой программы последовательно образуются значения

$$R1)\ (y, 0, 0) \\ R2)\ (0, y, 0) \\ R3)\ x \dotplus y = z$$

Заметим, что если бы сдвигу (второй команде) не предшествовало логическое умножение, то в результате сдвига получилось бы $(4, y, 0)$, а не $(0, y, 0)$, ибо код команды u сдвинулся бы в левый адрес.

Упражнение. Из команды

$$u)\ x \dotplus y = z, \quad \text{где} \quad u = 300$$

получить в $R1$ команду $z \dot{-} y = x$.

Решение дается программой 5.3.

Программа 5.3

$u \bigwedge (\Omega, 0, 0) = (x, 0, 0)$	200	13 300 013 021
$22 \rightarrow (x, 0, 0) = (0, 0, x)$	201	11 122 021 021
$u \bigwedge (0, \Omega, 0) = (0, y, 0)$	202	13 300 012 022
$(0, 0, x) \bigvee (0, y, 0) = (0, y, x)$	203	12 021 022 021
$u \bigwedge (0, 0, \Omega) = (0, 0, z)$	204	13 300 011 022
$22 \leftarrow (0, 0, z) = (z, 0, 0)$	205	11 022 022 022
$(0, y, x) \bigvee (z, 0, 0) = (z, y, x)$	206	12 021 022 021
$(0 \dot{-} 0 = 0) \bigvee (z, y, x) = (z \dot{-} y = x)$	207	12 211 021 021
стоп	210	— 00 000 000 000
$0 \dot{-} 0 = 0$	211	05 000 000 000

Словарь:

$$(x,\ 0,\ 0) = R1, \qquad (0,\ 0,\ z) = R2,$$
$$(0,\ 0,\ x) = R1, \qquad (z,\ 0,\ 0) = R2,$$
$$(0,\ y,\ 0) = R2, \qquad (z,\ y,\ x) = R1,$$
$$(0,\ y,\ x) = R1 \qquad (z - y = x) = R1.$$

Задача (для программистов). Пусть в ячейке $u \sim 300$ находится слово

$$u)\ (0,\ 0\ r)$$

Выполнить арифметическую команду, находящуюся в ячейке r (адрес r — не известен), и остановиться.

Решение представлено программой 5.4.

Программа 5.4

	$22 \leftarrow u = R1$	201	11 022 300 021
	$(0 \vee 0 = \rho) \vee R1 = (я + 1)$	202	12 206 021 203
	$r \vee 0 = (я + 1)$	203	—
$\rho)$	r	204	—
	стоп	205	— 00 000 000 000
	$0 \vee 0 = \rho$	206	12 000 000 204

Команды в ячейках 203 и 204 не кодируются — их образует сама машина. При выполнении команды 201 в ячейке 021 образуется $(r,\ 0,\ 0)$. Команда 202 заносит в 203 команду

$$r \vee 0 = 204$$

Команда 203 заносит в 204 команду, находящуюся в ячейке r. Теперь команда 204 выполняет попавшую в нее команду r.

6. СИСТЕМА КОМАНД МАШИНЫ *ММ*

Теперь мы приведем *полный* список команд машины *ММ* и уточним их выполнение. Формально обучение программированию можно было начать прямо с такого списка, снабженного обстоятельными объяснениями. Предыдущие главы должны были лишь помочь читателю усвоить этот материал и научить его самостоятельно пользоваться новым аппаратом.

Система счисления двоичная; числа — целые и с плавающей запятой; память — 512 ячеек; в ячейке 34 разряда.

«+» («—») изображается числом 0 (1); ввод адресно-безадресный двоичных, восьмеричных и десятичных цифр; в нулевой ячейке — нестираемый нуль.

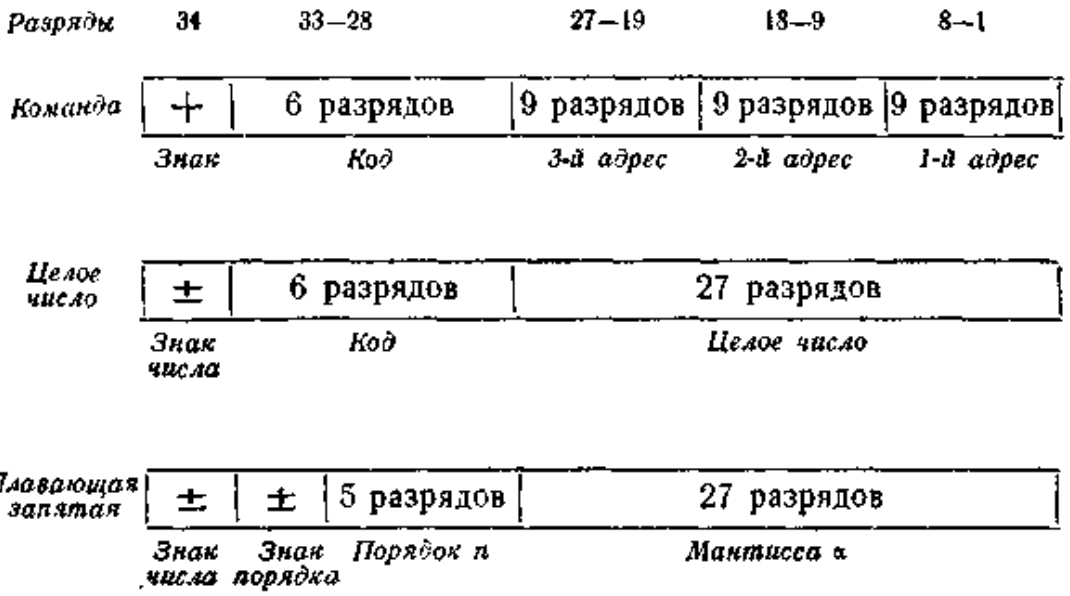


Рис. 6.1.

Форма представления команд и чисел в ячейках *ММ* показана на рис. 6.1. Система команд машины *ММ* приведена в табл. 6.1.

Таблица 6.1

Код	*Обозначение*	*Код*	*Обозначение*
	Логические		
00	$a \rightarrow b \;;\; c$	10	*внешн.* $b \;;\; c$
01	$a \leftarrow b \;;\; c$	11	$a = b = c$
02	$\|a\| \geqslant \|b\| \;;\; c$	12	$a \vee b = c$
03	$a \geqslant b \;;\; c$	13	$a \wedge b = c$
	Арифметические		
04	$a + b = c$ *плавающие*	14	$a +, b = c$ *целые*
05	$a - b = c$	15	$a -, b = c$
06	$a \cdot b = c$	16	$a \cdot, b = c$
07	$a : b = c$	17	$a :, b = c$
—00	*стоп*	знак + перед кодом опущен.	

6.1. Краткое описание команд *ММ.*

$a \rightarrow b;\ c$

Перенос слова из a в b и передача управления c.

$a \leftarrow b;\ c$

В ячейку b заносится слово +00; 000; 000; a, являющееся и командой передачи управления ячейке a, и Целым Числом a. Управление передается c.

$|a| \geqslant |b|;\ c$
$a \geqslant b;\ c$

Если для слов из ячеек a и b как для чисел с плавающей запятой выполняются написанные условия, то машина переходит к следующей команде; в противном случае управление передается ячейке c. *Эти же команды служат для сравнения команд и Целых Чисел с равными кодами*

$a + b = c$
$a - b = c$
$a \cdot b = c$
$a : b = c$

Арифметические действия над словами, как числами с плавающей запятой

внешн. b; c	Обращение к внешним устройствам, а после окончания их работы — передача управления c. В *ММ* имеются три обращения, различающиеся числами в левом адресе
ввод b; c (10; 000, b; c)	Ввод с ленты одного слова в ячейку b
пс b; c (10; 200, b, c)	Печать содержимого ячейки b в словарной форме
пч b, c (10; 400, b, c)	Десятичная печать слова b. Слева направо печатаются: 34-й и 33-й разряды знаками «+» и «—», далее 8 групп по 4 разряда печатаются восемью десятичными цифрами.
$a \vee b = c$	Логическое сложение
$a \wedge b = c$	Логическое умножение
$a \rightleftarrows b = c$	Сдвиг слова b с записью результата в ячейку c. Направление сдвига задается первой восьмеричной цифрой левого адреса (0 — влево, 1 — вправо), а число разрядов сдвига — двумя другими цифрами
$a +, b = c$ $a -, b = c$ $a \cdot, b = c$ $a :, b = c$	Арифметические действия над словами как Целыми Числами. Код переносится без изменения из слова a. При делении результатом служит целая часть частного
стоп	Останавливает машину

6.2. Сделаем необходимые пояснения к системе команд *ММ*.

6.2.1. Адресно-безадресный ввод направляет слова по тем адресам, которые указаны в кодированной части программы. Если же адрес слова не указан, то оно направится в следующую ячейку (следующую за той, в которую направилось предыдущее слово).

6.2.2. В ячейке 000 находится нестираемый нуль, т. е. слово

$$+\,00\ 0000000\ 000$$

Команды, включающие запись в эту ячейку, выполняются нормально, но записи не происходит. Сделано так потому, что число нуль требуется почти во всех программах, ячейка на него все равно должна тратиться и удобно, чтобы она имела постоянный, легко запоминаемый адрес и не могла быть случайно испорчена.

6.2.3. Обращения к внешним устройствам — это фактически три различные двухадресные команды, у которых роль кода (как признака команды) выполняют не только кодовые разряды, но и разряды левого адреса. Стоит еще сказать, что команда «ввода в элементарных программах» никогда не используется. Ею пользуются для ввода дополнительной информации, когда информации так много, что всю ее с самого начала нельзя было ввести. Печать *пс* (10, 200, *b*, *c*) служит для печати команд, а печать *пч* (10, 400, *b*, *c*) — для печати десятичных чисел, подготовленных программой перевода «2 — 10».

6.2.4. Сравнения

$$|a| \geqslant |b|; c \qquad \text{и} \qquad a \geqslant b; c$$

сравнивают слова *a* и *b*, которые в программе могут играть роль команд, чисел или даже просто слов. Это, разумеется, не мешает нам рассматривать сравниваемые слова как числа *ПЗ*, числа *ЦЧ* или команды.

При сравнении чисел *ПЗ* бóльшим признается большее. Неравные числа всегда признаются неравными (даже если они так мало отличаются, что их разность с плавающей запятой $a - b = c$ машина записывает нулем).

При сравнении Целых Чисел с равными кодами бóльшим признается большее Целое Число.

При сравнении команд с равными кодами большей окажется команда с большей адресной частью: с бóльшим левым адресом; если левые адреса равны — то с бóльшим средним адресом; если и средние адреса равны, — то с бóльшим правым адресом.

Наконец, при сравнении «$a \geqslant b$; *c*» слово (+00 000 000 000) признается бóльшим, чем (—00 000 000 000).

Сравнения целых чисел или команд *с не равными кодами* мы не определяем, будем считать, что результат может оказаться любым.

6.3. Добавление для программистов о числах и действиях с *ПЗ*.

6.3.1. Число с плавающей запятой представляется в ячейке памяти в нормальной форме в двоичной системе:

$$a = \pm 2^{\pm n} \cdot \alpha,$$

где

$$1/2 \leqslant \alpha < 1 \text{ при } a \neq 0,$$
$$n = \alpha = 0 \text{ при } a = 0.$$

Порядок n записан в разрядах $33 \div 28$, а мантисса α — в разрядах $27 \div 1$. Если $a \neq 0$, то старший разряд мантиссы (27-й разряд памяти) обязательно занят цифрой 1. Нулевой порядок ($n = 0$) обязательно имеет знак «+». Если $a = 0$, то и порядок и мантисса — нулевые и имеют знаки «+». Выполнять плавающие действия над другими словами запрещается — машине позволено давать в этих случаях любые ответы.

Примеры. Единица в форме *ПЗ* записана в ячейке памяти так:

$$1 = +2^{+1} \cdot 0{,}1_2 = (+\,01\ 400\ 000\ 000) =$$
$$= (00\ 00001\ 100000000\ 000000000\ 000000000).$$

Аналогично

$$2 = +2^{+2} \cdot 0{,}1_2 = (+\,02\ 400\ 000\ 000),$$
$$-1 = -2^{+1} \cdot 0{,}1_2 = (-\,01\ 400\ 000\ 000),$$
$$1/2 = +2^{+0} \cdot 0{,}1_2 = (+\,00\ 400\ 000\ 000),$$
$$1/4 = +2^{-1} \cdot 0{,}1_2 = (+\,41\ 400\ 000\ 000) =$$
$$= (01\ 00001\ 100000000\ 000000000\ 000000000),$$
$$10_{10} = +2^{+4} \cdot 0{,}101_2 = (+\,04\ 500\ 000\ 000),$$
$$1/8 = +2^{-1} \cdot 0{,}1010\ldots_2 = (+\,41\ 525\ 252\ 525).$$

Самое большое число с плавающей запятой равно

$$M = +\,+\,11111\ 11111111\ 11111111\ 11111111$$

или в словарной форме

$$M = +\,37\ 777\ 777\ 777.$$

Оно равно

$$M = +\,2^{+16+8+4+2+1}(2^{-1} + 2^{-2} + \ldots + 2^{-27}) = 2^{31}(1 - 2^{-27}) \approx 10^9.$$

Самое маленькое положительное число с плавающей запятой все же должно иметь мантиссу $\alpha \geqslant 1/2$. Оно равно

$$m = +\,-\,11111\ 100000000\ 000000000\ 000000000,$$
$$m = (+\,77\ 400\ 000\ 000) = 2^{-31} \cdot 1/2 = 2^{-32} \approx 10^{-9}.$$

6.3.2. Чтобы знать точно результат действия с плавающей запятой, нужно знать

Правила округления при действиях с плавающей запятой. Можно считать, что машина вначале определяет алгебраический знак результата и выясняет, какое действие ей нужно произвести над абсолютными значениями заданных чисел. Если это действие будет вычитанием, то из большего будет вычитаться меньшее. После выполнения действия результат будет записан с плавающей запятой с отбрасыванием не поместившихся (младших) разрядов мантиссы. Если результат меньше, чем m (по абсолютной величине), то записывается нулем.

Сложение, вычитание, умножение и деление абсолютных величин выполняются так, как будто вначале они были выполнены точно, а затем записаны в форме *ПЗ* с отбрасыванием непоместившихся разрядов мантиссы.

7. ПРОГРАММИРОВАНИЕ

В программировании имеется всего два принципиально новых обстоятельства. Именно их изобретение создало программирование и привело к небывалой широте использования *ЦЭМ*.

С *первым* мы уже познакомились — это *адресный принцип построения команд*. Считая на арифмометре, мы задаем ему сами числа, над которыми нужно произвести действие. Наоборот, в команде *ЦЭМ* пишется *не информация, над которой выполняется действие* (не сами числа), *а адреса ячеек, где эта информация расположена*. Этот принцип был подготовлен еще при ручном счете, когда в заголовке таблицы стали писать, *какие действия надо произвести над числами из строк с такими-то номерами*, чтобы получить число в *такой-то строке*. Но это была программа для работы живого вычислителя, а в *ЦЭМ* она стала восприниматься машиной.

Дальнейшим развитием адресного принципа являются команды, содержащие лишь адреса ячеек, где находятся адреса чисел, над которыми производится действие. В нашей простейшей машине *ММ* таких команд еще нет. С другой стороны в *ЦЭМ* сохранились и некоторые второстепенные команды, построенные на старом, не кодовом принципе. Так, например, в команде сдвига *ММ* прямо указывается, на сколько разрядов сдвинуть слово (а не адрес ячейки, где лежит число, указывающее размер сдвига).

Второй принцип программирования заключается в том, чтобы машина получила в первоначальной программе *короткую информацию* о задаче и затем уже сама образовывала программу счета по мере надобности, подобно тому как танк едет по рельсам, которые сам же выкладывает перед собой в понадобившемся направлении. Этот принцип подготовлен

адресным характером команд и тем, что сама команда может восприниматься как число — над командами можно производить арифметические и логические действия. В сложных задачах число различных команд очень велико и выписывание их человеком сделало бы нерентабельным применение *ЦЭМ*. Поэтому в рабочих программах прежде чем выполнить некоторые команды, машина сама же их и готовит.

7.1. Переадресация команд. Начнем с типичной задачи:

Пусть в $n = 64_{10}$ последовательных ячейках

$$a_1 = 501, \quad a_2 = 502, \ \ldots, \quad a_{64} = 600$$

(проверить последний адрес!) находятся 64 числа

$$x_1, \ x_2, \ \ldots, \ x_{64}.$$

Сложить их и поместить сумму в ячейку $\Sigma = 500$.

Решение 1. Эту задачу можно решить программой 7.1 из 66 команд:

Программа 7.1

Содержательная часть		*Кодированная*	*Результат*
$0 \to \Sigma$	200	00 000 500 201	0
$x_1 + \Sigma = \Sigma$	201	04 501 500 500	x_1
$x_2 + \Sigma = \Sigma$	202	04 502 500 500	$x_1 + x_2$
. . . . .	. . . . .		
$x_{64} + \Sigma = \Sigma$	300	04 600 500 500	$x_1 + \ldots + x_{64}$
стоп	301	—00 000 000 000	

Вместо трех первых команд можно написать одну:

$$x_1 + x_2 = \Sigma.$$

Но все равно программа слишком длинна и команды в ней разные. Заметим наперед, что *ускорить по существу* эту программу нельзя, ибо 63 сложения сделать необходимо. Таким образом, наша цель состоит в том, чтобы сократить *свою* работу по написанию программы, хотя бы за счет большей работы машины.

Решение 2. Удлинив программу, можно упростить написание содержательной части. Получим программу 7.2.

Программа 7.2

Содержательная часть		*Кодированная*	*Результат*
$0 \to \Sigma$	200	00 000 500 201	$\Sigma)\ 0$
$x_1 + \Sigma = \Sigma$	201	04 501 500 500	$\Sigma)\ x_1$
$(я - 1) +, (1, 0, 0) = я + 1$	202	14 201 004 203	$203)\ a_2 + \Sigma = \Sigma$
—	203	—	$\Sigma)\ x_1 + x_2$
$(я - 1) +, (1, 0, 0) = я + 1$	204	14 203 004 205	$205)\ a_3 + \Sigma = \Sigma$
—	205	—	$\Sigma)\ x_1 + x_2 + x_3$
. . . . .			
$(я - 1) +, (1, 0, 0) = я + 1$	376	14 375 004 377	$377)\ a_{64} + \Sigma = \Sigma$
—	377	—	$\Sigma)\ x_1 + \ldots + x_{64}$
стоп	400	—00 000 000 000	

Здесь команды 200 и 201 занесут первое слагаемое x_1 в ячейку Σ. Команда 202 изготовит и запишет в 203 команду $x_2 + \Sigma = \Sigma$ и т. д.

Эта программа подобно предыдущей является условием нашей задачи, написанным на языке машины. Но и она удлиняется с ростом числа слагаемых. Между тем условие задачи, написанное на «человеческом» языке, было коротким: оно не удлинялось с ростом числа слагаемых. Это наводит на мысль, что перевод задачи на машинный язык сделан плохо, и побуждает искать программу, не удлиняющуюся с ростом числа слагаемых.

Естественно попробовать получить и выполнить все команды 201, 203, 205, ..., 377 на одном месте 201. Тогда и для переделки этих команд понадобится единственная команда типа 202, 204, ..., 376, выполняемая многократно.

Решение 3 (неверное) дается программой 7.3.

Программа 7.3

1) $0 \to \Sigma$	200	00 000 500 202
2) $(я + 1) +, (1, 0, 0) = (я + 1)$	201	14 202 004 202
3) $x_1 + \Sigma = \Sigma$	202	04 501 500 500
4)	203	00 000 000 201

Здесь команда 200 очистит Σ и передаст управление ячейке 202. Команда 202

3) $x_1 + \Sigma = \Sigma$ | 202 | 04 501 500 500

сложит x_1 с нулем, в Σ появится число x_1 и управление перейдет к 203. Команда 203 передаст управление 201. Команда 201 прибавит к команде 202 число (1, 0, 0) и запишет результат в 202. От этого в 202 образуется команда

3_2) $x_2 + \Sigma = \Sigma$ | 202 | 04 502 500 500

и управление перейдет к команде 202. Теперь ячейка 202 (ее содержимое см. в 3_2) добавит к Σ число x_2. От этого в Σ появится сумма $x_1 + x_2$ и управление перейдет сначала к 203, а потом вернется к 201. Команда 201 переработает команду 202, находящуюся в состоянии 3_2), и запишет в 202 команду

3_3) $x_3 + \Sigma = \Sigma$ | 202 | 04 503 500 500

Управление перейдет к 202, которая добавит x_3 к Σ, от чего в Σ накопится сумма $x_1 + x_2 + x_3$ и т. д.

После определенного числа циклов (состоящих из команд 202, 203, 201) команда в 202 примет вид:

3_{64}) $x_{64} + \Sigma = \Sigma$ | 202 | 04 600 500 500

и получит управление. Она добавит к Σ *последнее* нужное нам слагаемое x_{64} и в Σ образуется искомая сумма $x_1 + \ldots + x_{64}$. Управление перейдет к 203.

Если бы мы могли в этот момент остановить машину, то задача была бы решена. Но машина будет продолжать работу по циклу 203, 201, 202, 203, ..., добавляя к Σ не нужные нам и нелепые слагаемые, которые испортят уже полученный результат.

Решение 4. Прекратить работу программы в нужный момент можно так. Заведем рабочую ячейку k, в которой будем считать *число слагаемых*, уже занесенных в Σ. И прежде чем передавать командой 203 управление 201 (см. решение 3), будем проверять, не вся ли уже сумма подсчитана, т. е. не достигло ли k нужной величины. Для этого мы заведем еще ячейку n, где напишем *число нужных слагаемых*, и будем сравнивать k с n. Если $k < n$, то пойдем из 203 к 201. Если k уже достигло n, то остановим машину. Ячейку n на-

зывают *эталоном* или *эталоном окончания*. Число k занесенных слагаемых проще всего считать в Целых Числах. Число $n = 64$ нам придется собственными руками перевести в двоичную систему счисления и полученное число 100_8 внести в машину в форму *ЦЧ* — ведь оно должно сравниваться с числом k, которое машина считает в двоичной системе в форме *ЦЧ*. Таким образом, получится не возобновляемая программа 7.4.

Программа 7.4

$0+,\ 0=\Sigma$	200	14 000 000 500
$1ц \to k$	201	00 001 021 205
$n)$ 64ц. дес	202	+00 000 000 000
$k+,\ (0,\ 0,\ 1)=k$	203	14 021 001 021
$(я+1)+,\ (1,\ 0,\ 0)=я+1$	204	14 205 004 205
$x_1+\Sigma=\Sigma$	205	04 501 500 500
$k \geqslant n$	206	02 021 202 203
стоп	207	—00 000 000 000

Вначале после команд 200, 201 и 205 в счетчике k будет 1, в Σ поступит первое слагаемое x_1 и управление перейдет сравнению 206, которое отошлет к 203, ибо $k = 1$ меньше, чем $n = 64$. На втором цикле команд 203 ÷ 206 в k будет число 2, в Σ будет $x_1 + x_2$ и сравнение 206 опять отошлет к 203. На последнем цикле, после команд 203—205 в k накопится число 64_{10}

$$k)\ +00\ 000\ 000\ 100$$

в Σ накопится сумма $x_1 + \ldots + x_{64}$ и управление, как и раньше, перейдет к 206. Но на этот раз сравнение 206 впервые выполнится, пропустит на 207 (а не отошлет на 203) и машина остановится.

Остается сделать эту программу возобновляющейся и переписать по нашей обычной схеме (рис. 7.1). Получим программу 7.5.

Ячейки 200—204 относятся к «возобновлению», ячейки 205 и 206 — к «изменению», ячейка 207 — к «работе», а 210 — к «проверке окончания». Ячейка $\rho = 207$, с переменной командой, в содержательной части пишется обязательно для

удобства чтения программы. Но она не кодируется и не вводится в машину. В начале работы программы команда 201 возобновляет 207, занося в нее слово из 202.

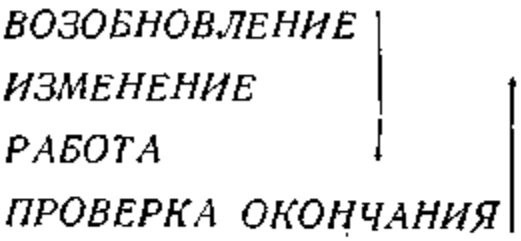


Рис. 7.1.

Важное замечание. Предыдущие шесть глав по существу были только подготовкой для § 7, а § 7, в котором

Программа 7.5

воз.)	$0+,\ 0=\Sigma$	200	14 000 000 500
	ρ	201	00 202 207 203
	$x_1+\Sigma=\Sigma$	202	04 501 500 500
	$1\rightarrow k$	203	00 001 021 207
n)	64ц. дес	204	00 000 000 000
изм.)	$k+,\ (0,\ 0,\ 1)=k$	205	04 021 001 021
	$\rho+,\ (1,\ 0,\ 0)=\rho$	206	04 207 004 207
ρ)	$x_k+\Sigma=\Sigma$	207	—
	$k\geqslant n$	210	02 021 204 205
	стоп	211	−00 000 000 000

началось и окончится элементарное программирование, заключается, строго говоря, в одной-единственной программе, которую мы сейчас привели. Обучающийся должен на ней научиться программировать. Поэтому мы рекомендуем не двигаться дальше, а разобрать эту программу со всей возможной тщательностью. Для этого желательно построить свои собственные варианты решения разобранной задачи.

7.2. Сейчас мы канонизируем нашу обычную схему одноцикловой программы (рис. 7.2) и порядок команд в ее частях. Зачем это надо и почему выбрана такая схема, будет подробно разбираться в п. 9.1.

Схема одноцикловой программы, изображенная на рис. 7.2, может быть записана и строкой:

$$(i=1)\ \overline{(i+1)}\ A(i)\ (i\geqslant n)$$

Возобновление начинается с очистки рабочих ячеек вне программы. Затем возобновляются рабочие ячейки вне программы. Наконец возобновляются команды программы и

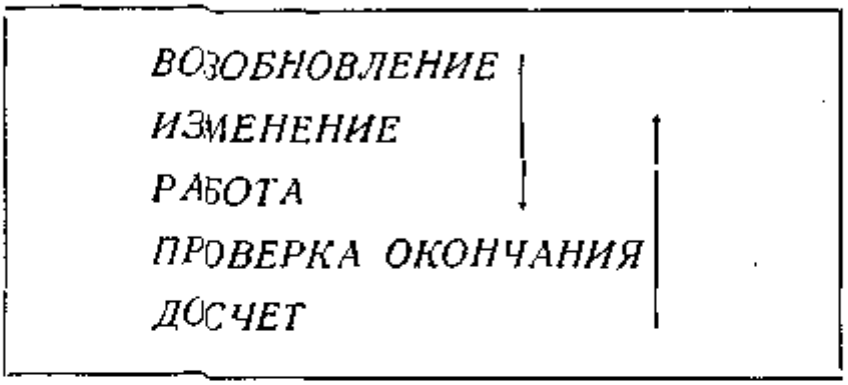


Рис. 7.2.

последним возобновляется счетчик циклов. Если слово, нужное для возобновления, уже имеется где-нибудь, то засылка

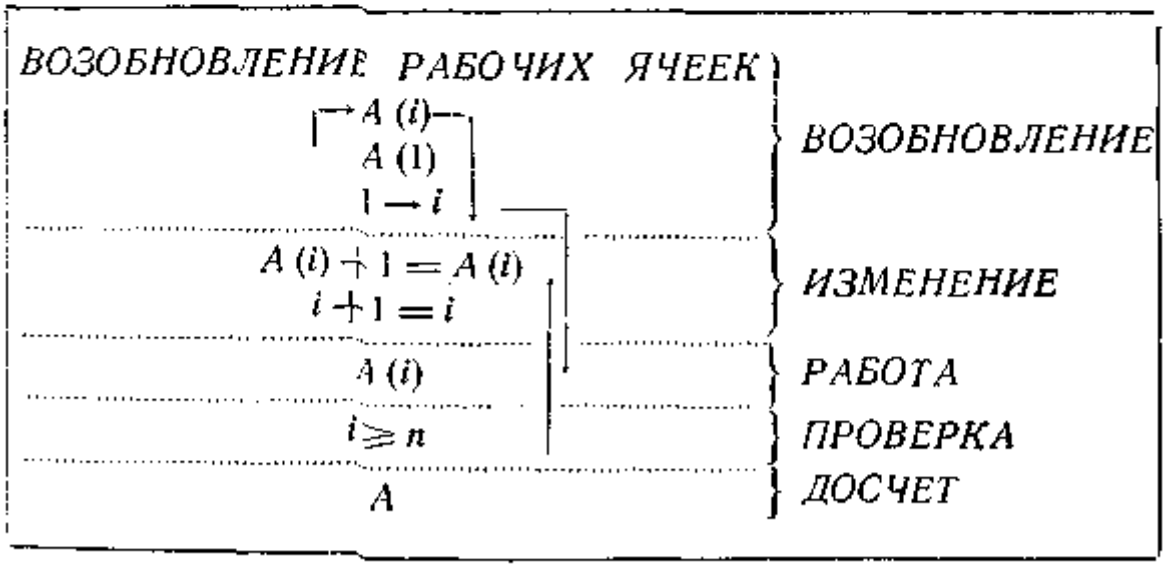


Рис. 7.3.

делается добавкой $(+,)$ нуля к этому слову, а не переносом (чтобы иметь меньше адресов, зависящих от местоположения

программы). Но если засылку нужно сочетать с передачей управления, то она делается переносом (—). Засылка положительных целых чисел, не превосходящих 777, делается командой (←—) и засылаемое число пишется в левом адресе этой команды. Если нужного возобновителя еще нет, то он засылается переносом и пишется под командой переноса, чтобы они были видны одновременно. *Возобновители команд пишутся в таком виде, в котором команды выполняются в первый раз.* Число циклов считается как *ЦЧ*. Проверка окончания производится сравнением счетчика с эталоном.

Более подробно схему цикла можно изобразить в виде рис. 7.3.

7.3. Решим несколько задач с помощью одноцикловых программ.

Задача 1. Сложить n чисел $x_1, x_2, \ldots, x_n$, расположенных в последовательных ячейках, если известен адрес первого слагаемого x_1 и известен адрес n ячейки, где в форме *ЦЧ* задано число слагаемых.

Решение мало отличается от решения задачи 7.1 и представлено программой 7.6.

Программа 7.6

$$
\begin{array}{l}
0+,\ 0=\Sigma \\
\rho \\
\Sigma + x_1 = \Sigma \\
1\text{Ц} - Сч \\
Сч +,\ 1\text{Ц} = Сч \\
\rho +,\ (0,\ 1,\ 0) = \rho \\
\rho)\ \Sigma + x_k = \Sigma \\
Сч \geqslant n
\end{array}
$$

Задача 2. Сосчитать скалярное произведение

$$\Sigma = \sum_{k=1}^{n} x_k y_k,$$

если ячейки чисел $x_1, x_2, \ldots$ и $y_1, y_2, \ldots$ расположены последовательно, а число n задано.

Решение представлено программой 7.7.

Программа 7.7

```
    0 +, 0 = Σ
┌→  p
│  x1 · y1 = R1
└  I11 → Cч
n)    n
   Cч +, 1 = Cч
   p +, (1, 1, 0) = p
p) xk · yk = R1
   Σ + R1 = Σ
   Cч ⩾ n
```

Задача 3. Сосчитать

$$\Sigma = x_1y_n + x_2y_{n-1} + \ldots + x_ny_1.$$

Решение представлено программой 7.8.

Программа 7.8

```
    0 +, 0 = Σ
┌→  p
│  x1 · yn = R1
└  I11 → Cч
   Cч +, 1 = Cч
   p +, (0, Ω, 0) = p
p) xk · yn-k = R1
   Σ + R1 = Σ
   Cч ⩾ n
```

8. ПОДПРОГРАММЫ

8.1. Библиотечные подпрограммы. Мы уже заметили, что некоторые константы встречаются во многих программах, и выделили для этих констант постоянные места. Легко предвидеть, что и некоторые подпрограммы будут встречаться во многих программах. Это переводы «$10 \to 2$» и «$2 \to 10$», извлечение квадратного корня, вычисление элементарных функций и т. д. Они вместе с часто употребляемыми константами, составляют *библиотеку подпрограмм машины*. Если программисту понадобится такая подпрограмма, то он не станет ни писать ее, ни входить в детали ее осуществления. В нужном месте он просто обратится к нужной подпрограмме *тем способом*, который указан в инструкции к библиотеке.

Составление библиотечных подпрограмм — дело исключительной сложности. Они готовятся особенно тщательно и должны удовлетворять ряду условий. Мы будем говорить об этом в § 14. Даже профессионалу-программисту незачем знать библиотечные подпрограммы машины, на которой он считает. Но всякий считающий на машине должен твердо усвоить правила обращения к библиотеке. В нашей библиотеке будет два типа обращений: одиночные (см. 8.1.1) и групповые (см. 8.1.2).

8.1.1. Одиночное обращение к библиотеке употребляется для вычисления одного значения функции, имеющейся в библиотеке (например, логарифма, корня и т. д. — см. далее «каталог библиотеки» от 031 до 037). Строится обращение так:

1. Перед обращением нужно переслать аргумент в ячейку $a = 020$ (или позаботиться, чтобы он там оказался в результате предыдущих вычислений).

2. В содержательной части обращение обычно указывается одним названием нужной функции. Например, для вычисления логарифма пишут

ln,

что *по раз навсегда заведенному правилу* кодируется как команда

$$(я + 1) \longleftarrow Ω;\ \text{ln}.$$

Если после обращения к подпрограмме нужно передать обращение не к $я + 1$, а к произвольной ячейке Q, то в содержательной части полностью пишут:

$$Q \longleftarrow Ω;\ \text{ln}.$$

3. Полученный результат подпрограмма заносит в

$$R1 = 021$$

и передает управление ячейке Q. Если программа дает два результата, то они заносятся в

$$R1 = 021$$

и

$$R2 = 022$$

4. При обращении к библиотеке аргумент в $α$ сохраняется, но старое содержание ячеек $Ω$, $R1$ (а иногда и $R2$) оказывается замененным новыми значениями.

Каталог библиотеки подпрограмм для машины *ММ* приведен в табл. 8.1.

Таблица 8.1

Код	*Операция*	*Пояснения*
030	групповая и «10 — 2»	
031	«2 — 10»	Перевод в десятичную систему и печать
032	$\sqrt{\ }$	
033	exp	$e^{α}$
034	cos-sin	
035	ch-sh	Гиперболические косинус и синус
036	ln	Натуральный логарифм
037	arccos	

Это значит, например, что программа $\ln\alpha$ начинается с ячейки 036. Программа cos-sin дает два ответа: в ячейку $R1$ поступает $\cos\alpha$, а в ячейку $R2$ поступает $\sin\alpha$. То же относится к программе гиперболического косинуса-синуса. Остальные программы дают по одному ответу (в ячейке $R1$).

Примеры пользования библиотекой:

1. В ячейке $x=300$ находится аргумент, направить в ячейку $y=425$ число $\ln x$.

Решение дается программой 8.1.

Программа 8.1

$x+,\ 0=\alpha$	200	14 300 000 020
ln	201	01 202 777 036
$R1+,\ 0=y$	202	14 021 000 425

Рассмотрим выполнение этой программы машиной. Команда 200 отправит значение x в ячейку α. Команда 201 занесет в ячейку Ω слово

$$\Omega)\ +00\ 000\ 000\ 202$$

и передаст управление 036 — началу программы натурального логарифма. Программа логарифма вычислит $\ln\alpha$, поместит этот результат в $R1$ и отдаст управление Ω. Но в это время в Ω находится передача управления ячейке 202. Таким образом, управление перейдет к 202. Команда 202 перенесет найденное значение $\ln x$ из $R1$ в ячейку y.

2. Найти $z=x^y$. Решение ищем по формуле

$$z=x^y=e^{y\ln x}.$$

Решение доставляется программой 8.2.

Программа 8.2

$$x+,\ 0=\alpha$$
$$\ln$$
$$R1\cdot y=\alpha$$
$$\exp$$
$$R1+,\ 0=z$$

3. Найти $y = \sqrt{x} + \operatorname{tg} x$. *Решение дается программой* 8.3.

Программа 8.3

$x +, 0 = \alpha$
$\sqrt{}$
$R1 +, 0 = y$
cos-sin
$R2 : R1 = \operatorname{tg} x$
$y + \operatorname{tg} x = y$

4. Вычислить $y = \sum_{k=0}^{n} \operatorname{arctg}\left(a \sin \frac{ek}{d+e}\right)$.

Задача 1. В треугольнике ABC заданы a, A, B. Найти остальные элементы (адреса A, B, C, a, b, c, S суть 301 ÷ 307).

Программа 8.4

$A +, 0 = \alpha$	200	14 301 000 020
cos-sin	201	01 202 777 033
$R1 +, 0 = \cos A$	202	14 021 000 023
$R2 +, 0 = \sin A$	203	14 022 000 024
$B +, 0 = \alpha$	204	14 302 000 020
cos-sin	205	01 206 777 033
$(\sin A) \cdot R1 = R1$	206	06 024 021 021
$(\cos A) \cdot R2 = R3$	207	06 023 022 023
$R1 + R3 = \sin C$	210	04 021 023 025
$a \cdot R2 = R3$	211	06 304 022 023
$R3 : \sin A = b$	212	07 023 311 305
$a \cdot \sin C = R3$	213	06 304 025 023
$R3 : \sin A = c$	214	07 023 311 306
$R3 \cdot b = R3$	215	06 023 305 023
$R3 \cdot (1/2) = S$	216	06 023 015 307
$\pi - A = R1$	217	05 016 301 021
$R1 - B = C$	220	05 021 302 303
стоп	221	— 00 000 000 000

Решение. По известным формулам

$$C = \pi - A - B, \qquad b = \frac{a \sin B}{\sin A},$$

$$c = \frac{a \sin C}{\sin A}, \qquad S = \frac{ab}{2} \sin C.$$

Чтобы не вычислять лишний раз тригонометрических функций, заметим, что

$$\sin C = \sin A \cdot \cos B + \cos A \cdot \sin B.$$

Решение дается программой 8.4.

Словарь: $\cos A \sim R3$, $\sin A \sim R4$, $\sin C \sim R5$.

8.1.2. Групповое обращение к библиотеке занимает в основной программе две ячейки. Зато при нем нет надобности заботиться о направлении аргумента в ячейку α и пересылке результата из $R1$. Употребляется групповое обращение в тех случаях, когда надо вычислить несколько значений библиотечной функции f от аргументов $x_1, x_2, \ldots, x_n$, расположенных в последовательных ячейках. При этом полученные значения $f(x_k)$ будут занесены в последовательные ячейки $y_1, y_2, \ldots, y_n$, тоже указанные в основной программе.

Групповое обращение осуществляется командами

$u)$	$(v+1) \leftrightarrow \Omega$; *групп*
$v)$	$(\pm f;\ x_1,\ y_1,\ n)$
$v+1)$	. . . . .

Здесь u и v — произвольные ячейки основной программы; f — название функции (см. далее); x_1 и y_1 — адреса первого аргумента и первого результата; n — число аргументов (в восьмеричной системе), или адрес ячейки, где находится n в форме *ЦЧ* (в зависимости от знака $\pm$ перед f).

Дойдя в основной программе до команды u, машина занесет в Ω слово (00; 0, 0, $v+1$) и передаст управление в 030 — групповой программе. Групповая программа сосчитает значения $f(x_k)$, положит их в ячейки y_k для $k = 1, 2, \ldots, n$ и вернет управление основной программе в ячейку $v+1$. Таким образом, u и v — любые ячейки, но v и $v+1$ — ячейки последовательные. Впрочем, как правило, все три ячейки u, v, $v+1$ пишут в программе подряд.

Для «названия функции» f в ячейке v взяты последние две цифры адреса, с которого начинается функция f в библио-

теке. Это требует двух дополнений: 1) если взять $f=30$, то будет выполняться групповой перевод «$10 \to 2$», таким образом, этот перевод может быть только групповым, 2) если $f=34$ (программа cos-sin), то по каждому значению аргумента будут вычисляться (и записываться подряд) два значения результата: косинус и синус. Таким образом, ответов получится не n, а $2n$. Это же относится и к $f=35$.

Перейдем к примерам.

1. В $n=5$ ячейках, начиная с 315, находятся десятичные числа $x_1, x_2, \ldots, x_5$. Перевести их в двоичную систему с *ПЗ* и расположить в ячейках $y_1, y_2, \ldots, y_5$, начинающихся с $y_1 \sim 432$.

Решение доставляется программой 8.5 ($u \sim 201$).

Программа 8.5

u)	$(я+2) \leftarrow ⍰$; *групп*	201	01	203	777	030
$u+1$)	$[(10-2);\ x_1,\ y_1,\ 5]$	202	30	315	432	005
$u+2$)		203				

2. Пусть в условиях предыдущего примера адреса x_1 и y_1 совпадают и равны 315, а число $n=25_{10}$.

Для решения нужно сначала перевести в восьмеричную систему десятичное число $n=25_{10}$. Получаем $n=31_8$. После этого пишем программу 8.6.

Программа 8.6

u)	$(я+2) \leftarrow ⍰$; *групп*	201	01	203	777	030
$u+1$)	$[(10 \to 2);\ x_1,\ y_1,\ 31]$	202	30	315	315	031

3. В $n=15_{10}$ ячейках, начиная с 217, лежат значения аргументов $x_1, x_2, \ldots, x_5$. Найти значения $\sqrt{x_k}$ и записать их, начиная с $\sqrt{x_1} \sim 311$.

Решение доставляется программой 8.7 (начатой с 250).

Программа 8.7

$(я+2) \leftrightarrow ⍰$; *групп*	250	01	252	777	030
$(\sqrt{\ }; x_1, \sqrt{x_1}, 17)$	251	32	217	311	017

Упражнение. В решении задачи 1 первые четыре команды можно заменить двумя (убедитесь в этом):

$$(я + 2) \leftrightarrow Ω;\ групп$$
$$[(\cos\text{-}\sin);\ A,\ \cos A,\ 1]$$

8.2. Подпрограммы. Внутри программы, которую мы пишем, может понадобиться вычисление функции, для которой уже есть готовая подпрограмма. Тогда естественно обратиться к нужной подпрограмме, не вникая в ее устройство, и затем вернуться в свою программу. Может также случиться, что в разных частях нашей программы требуется длинное вычисление одной и той же функции. Тогда естественно оформить это вычисление в виде одной самостоятельной подпрограммы и обращаться к ней во всех нужных случаях.

Подпрограммы оформляются как самостоятельные возобновляющиеся программы. Они (в частности) не должны портить передаваемой им информации, например, аргумента, по которому вычисляют функцию. По окончании своей работы подпрограмма должна передавать управление свободной ячейке выхода. При каждом обращении к подпрограмме основная программа заносит в эту ячейку адрес, к которому должно вернуться управление. Само обращение осуществляется командой основной программы:

$$и)\ Q \leftarrow (выход);\ (начало\ подпрогр.)$$

Здесь $и$ и Q — ячейки основной программы. Команда $и$ передает управление началу подпрограммы, а Q получает его из ячейки выхода по окончании работы подпрограммы. Если (как это часто бывает) надо вернуться к ячейке $и + 1$, то $Q = я + 1$.

При выборе местоположения ячейки «выхода» бытуют следующие два приема:

1) *Ячейка выхода располагается в конце подпрограммы.* Обращение к такой подпрограмме имеет вид

$$и)\ Q \leftarrow (конец\ подпр.);\ (начало\ подпр.)$$

Этот прием — самый старый и экономный. Употребляется он в простейших случаях. Его основной недостаток в излишней информации: приходится указывать (иногда это значит запоминать) не только начало, но и конец подпрограммы. При переделке подпрограммы ее начало удается сохранить

(как например «начала-заголовки» в нашей библиотеке), но конец обычно съезжает и приходится вносить исправления в рабочие программы.

2) Наиболее употребительны подпрограммы *со стандартной ячейкой выхода* Ω, одной для всех подпрограмм. Так построены и наши библиотечные подпрограммы. Обращение к ним имеет вид

$$n)\ Q \leftrightarrow \Omega; \quad \textit{(начало подпр.)}$$

Недостаток этих подпрограмм в следующем. Пусть A и B — подпрограммы этого типа и внутри подпрограммы A требуется обратиться к подпрограмме B. Тогда перед обращением подпрограмма A должна перенести слово из Ω в свою собственную ячейку свободного выхода. Тем не менее этот прием оказался лучшим, и мы рекомендуем привыкать к нему.

Сложнее обстоит дело с указанием адресов для аргументов и результатов. По этому признаку подпрограммы делятся на следующие три типа (возможны и промежуточные случаи):

1. *Со стандартными ячейками.* Прежде чем обращаться к такой подпрограмме, нужно позаботиться, чтобы аргументы находились в стандартных ячейках — ячейках, адреса которых известны подпрограмме. Результаты получаются тоже в стандартных ячейках подпрограммы. Примерами таких подпрограмм являются наши библиотечные подпрограммы (кроме групповой).

2. *Со стандартной информацией.* В стандартные, известные подпрограмме ячейки предварительно направляется информация об адресах, где находятся аргументы и куда должны попасть результаты.

Задача 2. Написать подпрограмму, вычисляющую в ячейке $R1$ скалярное произведение

$$\sum_{k=1}^{n} x_k y_k = R1$$

по информации, находящейся в стандартной ячейке

$$\alpha \sim (x_1, y_1, n).$$

Предполагается, что $x_1, x_2, \ldots$ и $y_1, y_2, \ldots$ расположены в последовательных ячейках памяти.

Решение дается программой 8.8.

Программа 8.8

```
    α ∧ (0, 0, Ω) = эт
    α —, эт = R1
    (0 · 0 = R2) +, R1 = ρ
    (0, 0, 1) → Сч
    Сч +, 1 = Сч
    ρ +, (1, 1, 0) = ρ
ρ)  x_k · y_k = R2
    Σ + R2 = Σ
    Сч ⩾ эт
    выход
```

Словарь: $эт \sim R2$, $Сч \sim R3$, $\Sigma \sim R1$.

3. *С сопутствующей информацией.* Обращение к таким подпрограммам осуществляется командой:

```
        u)  Q ← (выход); (начало подпрограммы)
    Q — s)  . . . . . .
Q — s + 1)  . . . . . .
. . . . . .
    Q — 1)  . . . . . .
        Q)  . . . . . .
```

Получив управление, подпрограмма извлекает из ячейки выхода (как правило, это Ω) адрес Q, и по нему находит информацию, расположенную в ячейках от $Q - s$ до $Q - 1$. По окончании работы подпрограмма передает управление ячейке Q. Зачастую s информационных ячеек пишут вслед за командой обращения u и тогда $Q = я + s + 1$. Примером подпрограммы с сопутствующей информацией может служить групповая программа нашей библиотеки — в ней одна ($s = 1$) информационная ячейка.

9. ОДНОЦИКЛОВЫЕ ПРОГРАММЫ

9.1. Каноническое написание одноцикловой программы. Расходы на программирование превосходят все остальные расходы по эксплуатации *ЦЭМ*, включая арендную плату, т. е. и цену самих машин (по данным США). Но программирование требует такой аккуратности, что до него мы даже не подозревали о существовании работ, где она не была бы уже излишней и вредной. Ведь *всякая* описка, *любая* неверная цифра делает программу *вконец* непригодной. Программирование требует напряженного внимания, программист быстро утомляется и начинает ошибаться. Поэтому выгодно приучиться писать программы стандартным образом и без необходимости не отходить от стандарта. С одной стороны, это облегчает чтение и проверку программ (в первую очередь — чужих, но и своих тоже), с другой — это дает возможность писать некоторые части программы с меньшим напряжением, почти механически, по привычному стандарту. Неверно по существу возражение, что в оригинальных частях программы ошибки редки. Утомившись на трудном месте, мы «отдыхаем» за ним *).

Чтобы программы писались по единой схеме, нужно, чтобы избранная схема была пригодна во всех случаях. Наша схема этому условию удовлетворяет. Но легко предложить схемы, которые будут лучше нашей в одних случаях и окажутся непригодны для других случаев. Чтобы сократить число таких «открытий», мы приведем непригодные, вообще говоря, схемы одноцикловых программ.

1. Если схему 9.1 осуществить в решении задачи на сложение чисел $x_1 + \ldots + x_n$ (см. программу 7.5), то

*) Для шахматиста это убедительно иллюстрируется партией Бронштейна с Ларсеном на Амстердамском межзональном турнире.

возобновитель 202 будет иметь «предпервый» вид:

$$x_0 + \Sigma = \Sigma \qquad 04\ 500\ 500\ 500$$

и станет нужным после первого увеличения.

Схема 9.1 не универсальна потому, что в некоторых задачах предпервое состояние логически не определено, например, может не иметь смысла. Этого достаточно, чтобы она была отброшена. Другой недостаток схемы (9.1) в том, что «предпервое» состояние приходится вычислять, и порою с трудом.

ВОЗОБНОВЛЕНИЕ ПРЕДПЕРВОЕ
ИЗМЕНЕНИЕ
РАБОТА
ПРОВЕРКА

Схема 9.1.

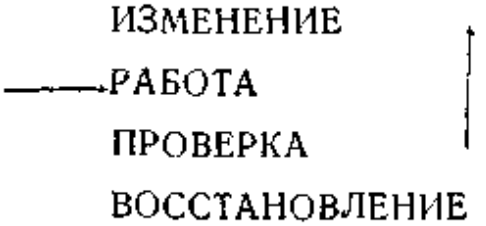


Схема 9.2.

2. Схема 9.2 начинается с работы, а оканчивается не возобновлением, а именно «восстановлением» первоначального вида изменяющихся команд *из их же последнего состояния. Так как ряд команд программы меняется одинаково,* то восстановить их можно с помощью (чаще всего вычитания) одной общей константы. Поэтому схема 9.2 часто приводит к сокращению программы.

Схема 9.2 должна быть отвергнута как непригодная для откладки программы за пультом машины. Цикл, написанный по схеме 9.2, нельзя возобновить, если машина станет в середине его, не дойдя до конца. Тогда придется заново

вводить программу, что приведет к потере времени. А стать программа рассматриваемого цикла может из-за ошибки в другой части программы, приведшей в данном цикле, например, к делению на нуль (или еще к чему-нибудь невыполнимому). Поэтому цикл должен возобновляться вначале, даже если он остановлен посередине, как мы уже указывали в § 2.

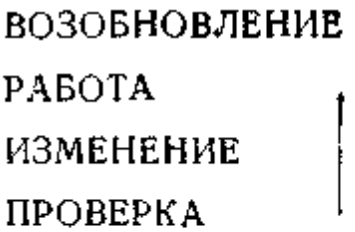


Схема 9.3.

3. Схема 9.3 выгодна в двухцикловых программах, где последнее (излишнее) увеличение во внутреннем цикле может придать команде вид, нужный для начала следующей серии (к этому случаю мы еще вернемся). Но схема 9.3 тоже не может быть канонизирована. «Запоследнее» состояние может оказаться логически не определимым, подобно «предпервому».

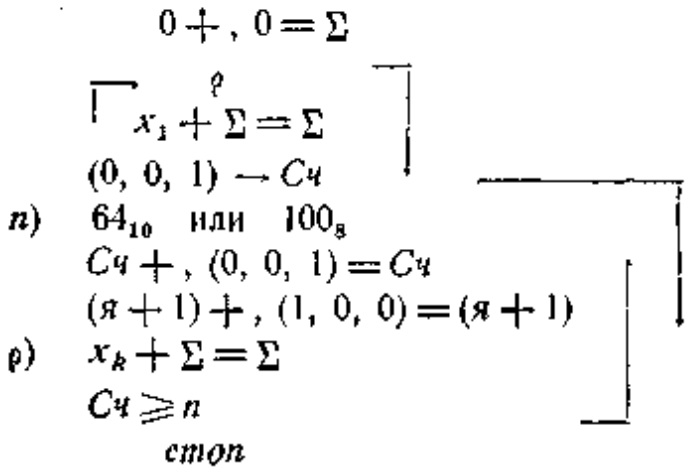
Все же в некоторых случаях программисты позволяют себе небольшие отклонения от канонической схемы. Сейчас мы их приведем.

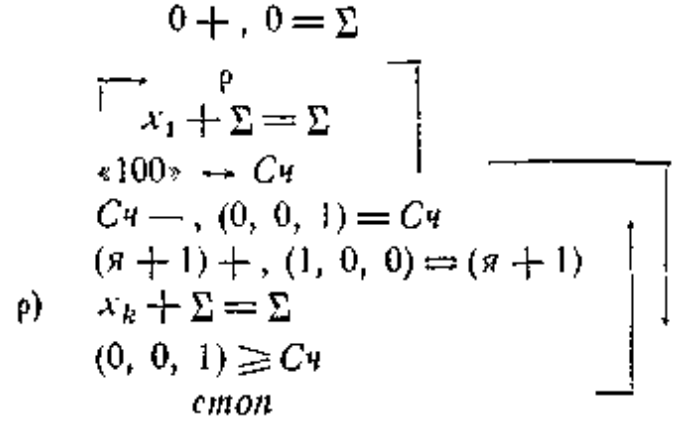
9.2. Обратный счет циклов. Для счета циклов мы тратим в группе «возобновление» две ячейки: одну для посылки 1 в счетчик, другую для числа циклов n (см., например, программу 7.5). Но если число циклов $n \leqslant 777$, то можно обойтись одной командой. Для этого надо командой передачи «↔» занести в счетчик циклов число n (из левого адреса самой команды), а затем на каждом цикле *вычитать* из счетчика по единице. В этом случае левый адрес команды ↔ пишется в кавычках и переносится в кодированную часть без изменения. Прекратить повторение цикла нужно тогда, когда в счетчике появится единица. Вот для сравнения программа 7.5 (первый столбик программы 9.1) и аналогичная программа с обратным счетом циклов (второй столбик).

Прием этот экономит мало, а откладку за пультом машины замедляет. Пользоваться им в подобных программах не следует.

Упражнение. Укоротите две программы (по выбору) из § 7 с помощью обратного счета циклов.

Программа 9.1





9.3. Эталон-команда. Рассмотрим программу 7.5. В ее цикле участвуют 4 команды: увеличение счетчика, увеличение рабочей команды, рабочая команда, проверка окончания. В процессе счета рабочая команда изменяется. Ее состояния от первого до последнего приведены в табл. 9.1.

Таблица 9.1

1) $x_1 + \Sigma = \Sigma$	201	04 501 500 500
2) $x_2 + \Sigma = \Sigma$	201	04 502 500 500
. . . . .		
63) $x_{n-1} + \Sigma = \Sigma$	201	04 577 500 500
64) $x_n + \Sigma = \Sigma$	201	04 600 500 500

Это наводит на мысль *сообразить заранее вид изменяемой команды во время последнего исполнения цикла, написать его в качестве эталона, сравнивать на каждом цикле изменяющуюся команду с этим эталоном и счи-*

тать вычисления законченными, когда команда сравняется с эталоном.

Тогда программа 7.5 преобразуется в программу 9.2.

Программа 9.2

	$0+,\ 0=\Sigma$	200	14 000 000 500
воз.)	ρ	201	00 202 205 205
в-ль)	$x_1+\Sigma=\Sigma$	202	04 501 500 500
эт)	$x_n+\Sigma=\Sigma$	203	04 600 500 500
	$(я+1)+,\ (1,\ 0,\ 0)=(я+1)$	204	14 205 004 205
ρ)	$x_k+\Sigma=\Sigma$	205	—
	$\rho \geqslant$ эт	206	02 205 203 204
	стоп	207	—00 000 000 000

Разберем подробно, как работает эта программа. После команд 200 и 201 ячейка $\rho \sim 205$ примет вид

$\rho)\ x_1+\Sigma=\Sigma$ | 205 | 04 501 500 500

и занесет в Σ первое слагаемое x_1. Затем сравнение 206 сравнит ячейку 205 с эталоном 203

эт) $x_n+\Sigma=\Sigma$ | 203 | 04 600 500 500

Так как кодовые части обеих команд совпадают и имеют знак $+$, то большей считается та, у которой больше адресная часть. Следовательно, содержимое ячейки ρ будет сочтено меньшим, чем содержимое ячейки эт (так как $501 < 600$), условие 206 *не выполнится* и управление вернется в 204.

Команда 204 придаст команде ρ вид:

$\rho)\ x_2+\Sigma=\Sigma$ | 205 | 04 502 500 500

Команда ρ добавит x_2 к Σ, сравнение 206 снова отдаст управление 204 и т. д.

На предпоследнем цикле команда ρ будем иметь вид:

$x_{n-1}+\Sigma=\Sigma$ | 205 | 04 577 500 500

Она добавит x_{63} к Σ и сравнение 206 снова не выполнится, ибо

$$577\ 500\ 500 < 600\ 500\ 500.$$

Управление перейдет к 204. Эта команда придаст команде ρ вид:

$\rho)\ x_n+\Sigma=\Sigma$ | 205 | 04 600 500 500

Команда ρ занесет в Σ слагаемое x_n и сравнение 206 впервые выполнится — ибо впервые команда ρ «доросла» до эталона — управление перейдет к 207 и вычисления закончатся. Цикл вычислений сократился с 4-х команд до 3-х, и задача стала считаться на 25% быстрее.

Чем больше в цикле команд, тем меньше доля ускорения от применения эталон-команды. Следует также иметь в виду, что единственная ошибка приведет к такой потере времени, которая сведет на нет выигрыш ото всех эталон-команд во всей программе.

Другое затруднение в том, что в двойных циклах эталон-команда внутреннего цикла становится переменной (как мы увидим в § 11), что усложняет и удлиняет программу. Поэтому обучающемуся не рекомендуется пользоваться эталон-командами без нужды.

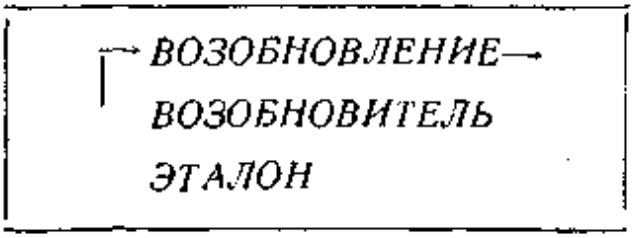


Схема 9.4.

Канонизируем порядок написания программы с эталон-командой, приведенный на схеме 9.4.

Первое состояние команды (возобновитель) и последнее (эталон) пишутся рядом потому, что они похожи — это уменьшает число описок.

Задача 1. Занести 0 в ячейки от 300 до 356. Решение дано программой 9.3.

Задача 2. Сосчитать $\Sigma = \sum_{k=1}^{n} x_k y_k$ для заданного числа n. Решением является программа 9.4.

Программа 9.3

```
      ρ
   0 +, 0 = 300
эт) 0 +, 0 = 356
   ρ +, (0, 0, 1) = ρ
ρ) 0 +, 0 = a_k
   ρ ≧ эт
   Стоп
```

Программа 9.4

```
   0 +, 0 = Σ
      ρ
   x_1 · y_1 = R1
эт) x_n · y_n = R1
   ρ +, (1, 1, 0) = ρ
ρ) x_k · y_k = R1
   R1 + Σ = Σ
   ρ ≧ эт
   стоп
```

Задача 3. Сосчитать и занести в последовательные ячейки $x_1, x_2, \ldots, x_n$ числа $2^n, 2^{n-1}, \ldots, 2$. Решением является программа 9.5. Так как переменная команда ρ «уменьшается», то проверка окончания имеет вид

$$эт \geqslant \rho$$

(а не $\rho \geqslant$ эт).

Программа 9.5

$2_n +, 0 = x_n$

ρ

$x_n \cdot 2_n = x_{n-1}$

эт) $x_2 \cdot 2_n = x_1$

$\rho -, (1, 0, 1) = \rho$

ρ) $x_k \cdot 2_n = x_{k-1}$

эт $\geqslant \rho$

стоп

Эталон-команду (так же как и эталон в счетчике циклов) не обязательно брать совпадающим с последним состоянием контрольной команды. Его можно брать промежуточным между ее предпоследним и последним видами. Посмотрим этот прием на следующих задачах.

Задача 4. Сосчитать с эталон-командой суммы

4.1) $\Sigma = x_1 y_n + x_2 y_{n-1} + \ldots + x_{n-1} y_2 + x_n y_1$,

4.2) $\Sigma = x_n y_1 + x_{n-1} y_2 + \ldots + x_2 y_{n-1} + x_1 y_n$,

если адреса ячеек $x_1, x_2, \ldots, x_n$ и $y_1, y_2, \ldots, y_n$ идут подряд и в задаче 4.1 заданы адреса x_1, x_n, y_n, а в задаче 4.2 — адреса x_1, x_n, y_n.

Решение. Нет нужды вычислять адрес y_1 в первом случае и y_n во втором. Можно выбрать эталоны и так, как это сделано соответственно в программах 9.6 и 9.7.

Программа 9.6

$0 +, 0 = \Sigma$

ρ

$x_1 \cdot y_n = R1$

эт) $x_n \cdot 0 = R1$

$\rho +, (0, \Omega, 0) = \rho$

ρ) $x_k \cdot y_{n-k+1} = R1$

$\Sigma + R1 = \Sigma$

$\rho \geqslant$ эт

Программа 9.7

$0 +, 0 = \Sigma$

ρ

$x_n \cdot y_1 = R1$

эт) $x_1 \cdot \Omega = R1$

$\rho -, (0, \Omega, 0) = \rho$

ρ) $x_{n-k+1} \cdot y_k = R1$

$\Sigma + R1 = \Sigma$

эт $\geqslant \rho$

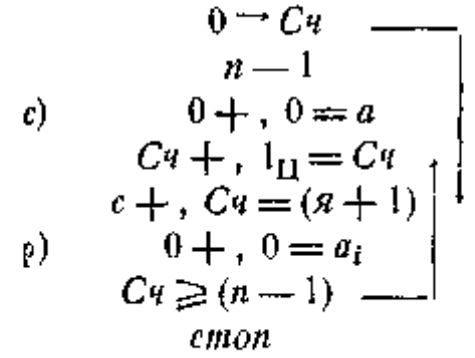
Действительно, в задаче 4.1), например, команда ρ будет меньше эталона при $k = 1, 2, \ldots, n-1$, а при $k = n$ она заведомо превзойдет эталон.

9.4. Индексное программирование. В некоторых задачах новое состояние переменной команды легко образуется из ее *первоначального* состояния и счетчика циклов, *минуя предыдущее состояние этой команды.* Такие команды следует получать из начального состояния и счетчика циклов. Но при этом весь счет циклов удобно вести, начиная с 0, а не с 1, что сказывается на эталоне счетчика.

Задача 5. Занести число 0 в n ячеек, начиная с ячейки a.

Решение дает программа 9.8. Не перфорируемая команда ρ на каждом цикле заново получается из счетчика *Сч* и постоянного слова c.

Программа 9.8



9.5. Получение изменяющихся команд друг из друга. Если несколько команд цикла меняются синхронно, так что их разность постоянна, то достаточно изменять одну, а другие образовывать из нее. Это сокращает программу, так как «образуемые» команды не нужно возобновлять. Впрочем, злоупотреблять образованием команд друг из друга не следует — это усложняет программирование.

Вот примеры удачного применения рассматриваемого приема.

Задача 6. Пусть идут подряд ячейки со значениями

$$x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_n, y_n, z_n.$$

Сосчитать

$$\Sigma = \sum_{k=1}^{n} f_k, \quad \text{где} \quad f_k = \frac{x_k^3 + a x_k^2 y_k + b x_k y_k^2 + y_k^3}{z_k^2 + c z_k + d},$$

Решение. Чтобы не переадресовывать команд вычисления f_k, естественно написать одну программу вычисления f по x, y, z, находящимся в ячейках $R1$, $R2$, $R3$. Тогда в эти ячейки придется переносить текущие значения x_k, y_k, z_k и программе можно будет придать вид, приведенный в программе 9.9. Здесь команда ρ переадресовывается, а команды $\rho 1$ и $\rho 2$ получаются из нее.

Программа 9.9

$0 +, 0 = \Sigma$

ρ

$x_1 +, 0 = R1$

эт) $x_n +, 0 = R1$

$\rho +, (0, 0, 3) = \rho$

ρ) $x_k +, 0 = R1$

$(я - 1) +, (1, 0, 1) = (я + 1)$

ρ1) $y_k +, 0 = R2$

$(я - 1) +, (1, 0, 1) = (я + 1)$

ρ2) $z_k +, 0 = R3$

$f(R1, R2, R3) = f$

$\Sigma + f = \Sigma$

$\rho \geqslant$ эт

Задача 7. В последовательных ячейках от 317 до 563 находятся числа

$$x_1, x_2, \ldots, x_n.$$

Поместить в ячейку $x \sim 311$

$$x = \max_{1 \leqslant k \leqslant n} x_k.$$

Программу сделать возможно короткой.

Указание. Обязательно сделайте эту задачу сами и только потом разбирайте программу 9.10, использующую восемь ячеек.

Программа 9.10

	$\rho; (\rho + 1)$	200	00 201 204 205
	$x_1 \geqslant x$	201	02 317 311 207
эт)	$x_n \geqslant x$	202	02 563 311 207
	$(я + 1) +, (1, 0, 0) = (я + 1)$	203	14 204 004 204
ρ)	$x_k \geqslant x$	204	—
	$0 +, (я - 1) = (я + 1)$	205	14 000 204 206
	$x_k \to x$	206	—
	$\rho \geqslant$ эт	207	02 204 202 203

9.6. Стандартизация начала. Программу 7.5 легко ускорить, занося $x_1 + x_2$ в Σ вместо нуля, тогда она примет вид программы 9.11.

Программа 9.11

```
      x1 + x2 = Σ
   ┌──  ρ  ──┐
   │ x3 + Σ = Σ │
   │ «3» ← k    │
n) │ 64 Ц. дес. │
   │ k + , 1Ц = k │
   │ (я + 1) + , (1, 0, 0) = я + 1 │
ρ) │ xk + Σ = Σ │
   └ k ≧ n ─────┘
```

Ускорение это мало́ и отнюдь не обязательно. Говорить мы стали о нем потому, что иногда первому циклу невозможно придать стандартный вид без специальных усложнений. Это ясно видно на следующей задаче.

Сосчитать

$$\Sigma = \sum_{k=1}^{15} f(x_k),$$

если заданы x_1 и Δ_1, а

$$x_k = \varphi(x_{k-1}, \Delta_{k-1}), \quad \Delta_k = \psi(\Delta_{k-1}).$$

Функции можно считать заданными подпрограммами

$$y = f(x), \quad x = \varphi(x, \Delta), \quad \Delta = \psi(\Delta),$$

которые по аргументам, находящимся в ячейках x и Δ, дают значения функций в ячейках y, x, Δ. Эти подпрограммы начинаются соответственно с ячеек f, φ, ψ и кончаются свободной ячейкой Ω. Функции определены *только* для нужных значений аргументов.

Предостережем обучающегося от неверного решения, приведенного в программе 9.12.

Программа 9.12 (неверная)

$$
\begin{array}{l}
0+,\ 0=\Sigma \\
\Delta_1+,\ 0=\Delta \\
x_1+,\ 0=x \\
1_{Ц}\to Сч \\
15_{Ц} \\
Сч+,\ 1_{Ц}=Сч \\
\varphi(x,\Delta)=x \\
\psi(\Delta)=\Delta \\
f(x)=y \\
\Sigma+y=\Sigma \\
Сч+,\ 1_{Ц}=Сч \\
Сч\geqslant 15
\end{array}
$$

Ошибка программы 9.12 в том, что функция $\psi(\Delta)$ считается для $\Delta=\Delta_1, \Delta_2, \ldots, \Delta_{15}$. Между тем $\psi(\Delta_{15})$ может оказаться просто неопределенной (как $\arcsin x$ для $x>1$), и программа станет.

Вот два верных решения. В программе 9.13 не стандартное начало: функция f один раз вычисляется вне цикла. Это решение не годится, если число циклов сделать равным единице, а не пятнадцати.

В программе 9.14 в цикле стоит лишняя проверка, замедляющая программу.

Программа 9.13

$$
\begin{array}{ll}
\text{Воз.)} & \Delta_1+,\ 0=\Delta \\
 & x_1+,\ 0=x \\
 & f(x)=y \\
 & y+,\ 0=\Sigma \\
 & (0,\ 0,\ 1)\to Сч \\
 & 15_{Ц} \\
\text{Изм.)} & Сч+,\ 1=Сч \\
 & \psi(\Delta)=\Delta \\
\text{Раб.)} & \varphi(x,\Delta)=x \\
 & f(x)=y \\
 & \Sigma+,\ y=\Sigma \\
\text{Пров.)} & Сч\geqslant 15_{Ц}
\end{array}
$$

Программа 9.14

$$
\begin{array}{l}
0+,\ 0=\Sigma \\
\Delta_1+,\ 0=\Delta \\
x_1+,\ 0=x \\
(0,\ 0,\ 1)\to Сч \\
15_{Ц} \\
Сч+,\ 1=Сч \\
Сч\geqslant(0,\ 0,\ 3) \\
\psi(\Delta)=\Delta \\
\varphi(x,\Delta)=x \\
f(x)=y \\
\Sigma+,\ y=\Sigma \\
Сч\geqslant 15_{Ц}
\end{array}
$$

По современным воззрениям следует предпочесть второе решение, у которого внутри цикла идет проверка

$$[Сч \geqslant (0,\ 0,\ 3)]$$

необходимости вычисления

$$\psi(\Delta) = \Delta.$$

Дело в том, что в более сложных случаях может оказаться несколько различных проверок *такого типа* и их не удастся выполнить одним не стандартным началом.

10. МНОГОКРАТНЫЕ ЦИКЛЫ

10.1. С двухкратным циклом мы уже познакомились в конце второй главы. Теперь, мы научимся писать двухкратный цикл с переменными командами, которых не было во второй главе.

Начнем с простейшего примера. Пусть по значениям

$$a_1, a_2, \ldots, a_n$$

требуется вычислить и занести в память значения

$$c_{ij} = a_i - a_j, \quad i = 1, 2, \ldots, n; \quad j = 1, 2. \ldots, n,$$

если ячейки

$$a_1, a_2, \ldots, a_n \quad \text{и} \quad c_{11}, c_{12}, \ldots, c_{1n}; \quad c_{21}, c_{22}, \ldots, c_{2n}; \ldots$$

идут подряд.

Разберемся в такой задаче. При фиксированном i нужно вычислять и заносить в последовательные ячейки значения $c_{ij} = a_i - a_j$ для всех $j = 1, 2, \ldots, n$. Это должен делать внутренний цикл. По окончании внутреннего цикла нужно

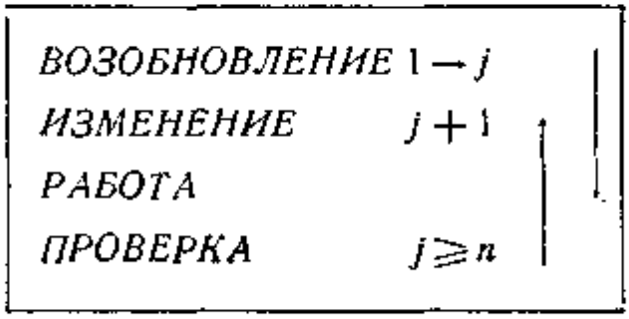
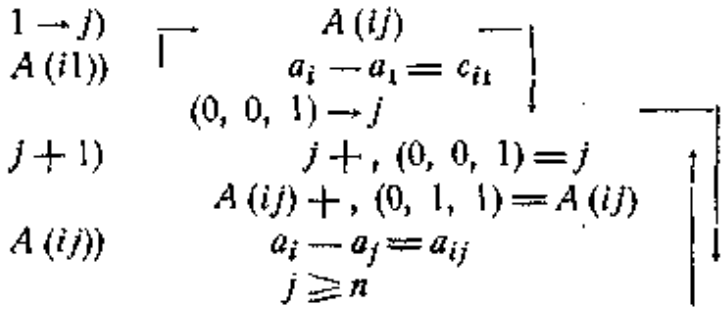


Рис. 10.1.

увеличить значение i и опять провести внутренний цикл. Изменение i от 1 до n составит наружный цикл. Писать программу удобно, начиная с внутреннего цикла.

Напишем внутренний цикл (рис. 10.1) по канонической схеме циклической программы так, как будто он нужен для одного единственного и *заданного* значения i.

Программа 10.1



Если значение i задано, то мы можем определить адреса a_i и c_{i1}, написать возобновитель $A(i1)$ и осуществить приведенную программу 10.1.

Теперь напишем наружный цикл тоже по канонической схеме циклической программы (см. рис. 10.2 и программу 10.2).

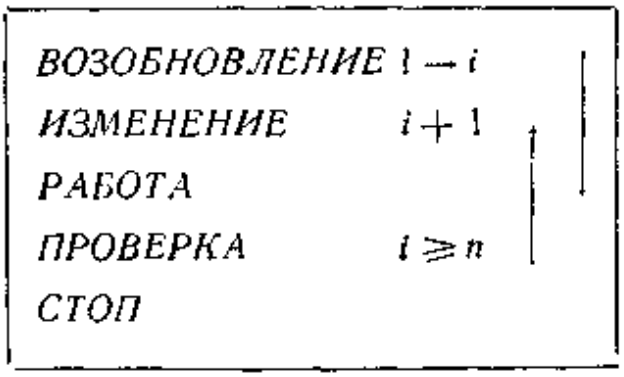


Рис. 10.2.

Его возобновление $(1 \rightarrow i)$ должно содержать возобновление слова $A(i1)$ (употребляемого во внутреннем цикле) в самом первом состоянии: $A(11)$. Группа увеличения $(i + 1)$ должна изменять слово $A(i1)$ так, как оно меняется при увеличении i на единицу. Что касается рабочей части наружного цикла, то она (в нашей простой задаче) состоит только из внутреннего цикла.

Новой для нас в программе 10.2 является команда

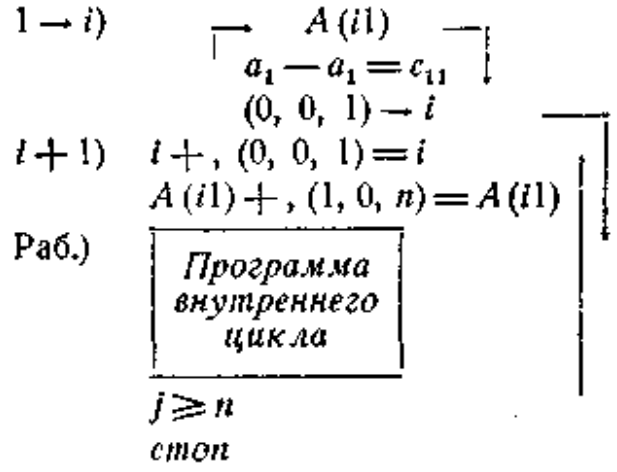
$$A(ij) \equiv (a_i - a_j = c_{ij}),$$

которая должна была бы изменяться в обоих циклах — и во внутреннем и в наружном. Для нее введена *промежуточная* команда

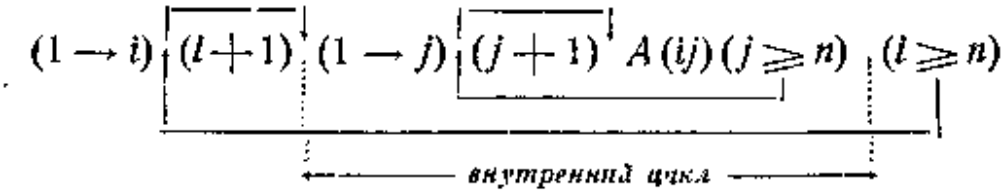
$$A(i1) \equiv (a_i - a_1 = c_{i1}),$$

которая во внутреннем цикле служит восстановителем, а изменяется в наружном.

Программа 10.2



Схему двухкратного цикла (в нашем *простейшем* случае) можно записать так:



В более общем случае внутренняя часть может состоять не из одной команды $A(ij)$, а из серии вычислений, которые удобно обозначить оператором $A(ij)$, зависящим от обоих индексов i и j. Число внешних циклов n может не совпадать с числом внутренних циклов. Последнее может даже изменяться в зависимости от номера наружного цикла. После внутренних циклов может, разумеется, идти «досчет $A(i)$», а после наружных — «досчет A». Впрочем часть «досчета $A(i)$» может оказаться в рабочей части наружного цикла.

10.2. Каноническая схема двухкратного цикла. Наружный цикл нужно писать по канонической схеме циклической программы. Его рабочая часть должна

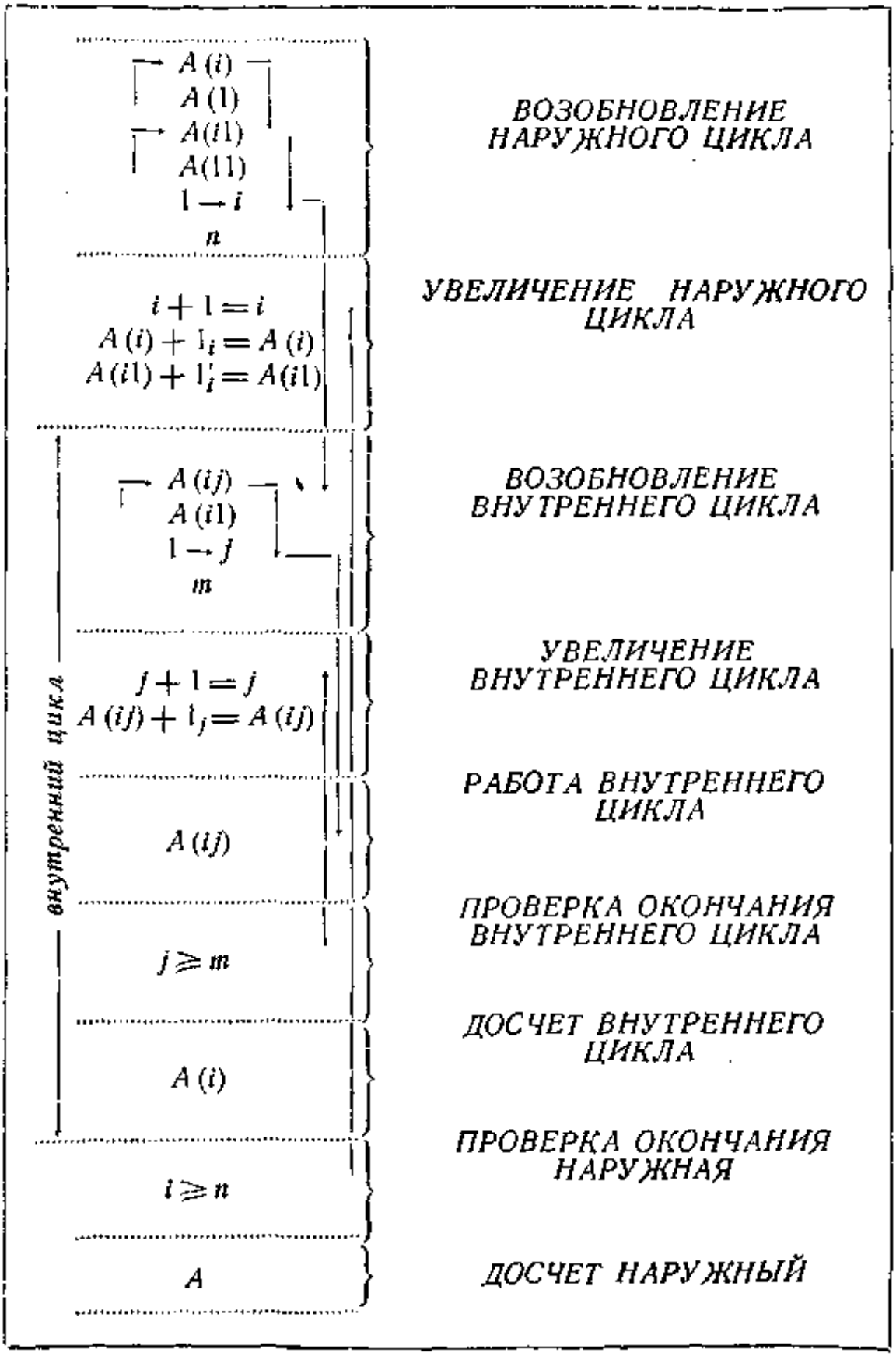

A (i)
A (1)
A(i1)
A(11)
1 → i
n
ВОЗОБНОВЛЕНИЕ НАРУЖНОГО ЦИКЛА
i + 1 = i
A (i) + 1_i = A (i)
A (i1) + 1'_i = A(i1)
УВЕЛИЧЕНИЕ НАРУЖНОГО ЦИКЛА
A (ij)
A (i1)
1 → j
m
ВОЗОБНОВЛЕНИЕ ВНУТРЕННЕГО ЦИКЛА
j + 1 = j
A (ij) + 1_j = A (ij)
УВЕЛИЧЕНИЕ ВНУТРЕННЕГО ЦИКЛА
A (ij)
РАБОТА ВНУТРЕННЕГО ЦИКЛА
j ≥ m
ПРОВЕРКА ОКОНЧАНИЯ ВНУТРЕННЕГО ЦИКЛА
A (i)
ДОСЧЕТ ВНУТРЕННЕГО ЦИКЛА
внутренний цикл
i ≥ n
ПРОВЕРКА ОКОНЧАНИЯ НАРУЖНАЯ
A
ДОСЧЕТ НАРУЖНЫЙ

Рис. 10.3.

содержать внутренний цикл, как подпрограмму. Внутренний цикл важно писать как самостоятельную *возобновляющуюся* программу, т. е. так, чтобы при остановке в середине внутреннего цикла его можно было начать сначала, не обращаясь к наружному циклу. Внутренний цикл пишется тоже по канонической схеме циклической программы. Кроме того, удобно, чтобы наружный цикл передавал внутреннему минимум информации.

Программа двухкратного цикла имеет следующий вид:

$$(1 \to i)\,\overline{\lfloor (i+1)\rfloor\,(1 \to j)\,\overline{\lfloor (j+1)\rfloor\, A(ij)}\,(j \geqslant m)\;\; A(i)\,(i \geqslant n)\;\; A}$$

Здесь $A(i)$ и A — обычные досчеты внутренней и наружной циклической программы.

В содержательных обозначениях каноническая схема изображена на рис. 10.3.

В более общем случае наружная программа может иметь рабочие части $B(i)$ и $C(i)$, расположенные так: первая — между увеличением наружного цикла и возобновлением внутреннего, а вторая — между досчетом внутреннего цикла и проверкой окончания наружного. Кроме того, число внутренних циклов может зависеть от i, так что $m = m(i)$.

Существенно в канонической схеме следующее. *Для команды $A(ik)$ внутреннего цикла, зависящей от двух индексов (т. е. меняющейся и в наружном и во внутреннем цикле), вводится промежуточная команда $A(i1)$, изменяющаяся в наружном цикле и являющаяся восстановителем во внутреннем цикле.* Промежуточные команды можно хранить в рабочих ячейках.

Аналогичным образом пишутся и многократные циклы.

Задача 1. Значения $a_1, a_2, \ldots, a_n$ идут подряд. Вычислить и занести подряд

$$c_{ik} = a_i - a_k$$

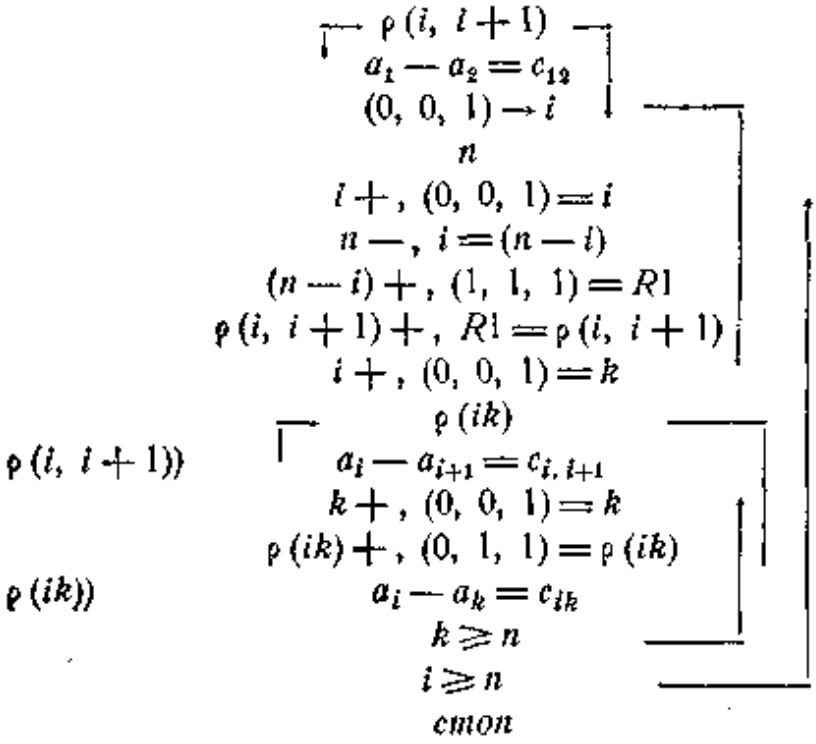
для следующих c_{ik}:

$$c_{12}, c_{13}, \ldots, c_{1n};\;\; c_{23}, c_{24}, \ldots, c_{2n}; \ldots; c_{n-1,n}.$$

Решение дает программа 10.3.

Рассмотрим его. После записи в $c_{i-1,n}$ следует запись в c_{i1}. Ясно, что c_{i1} проще получить из $c_{i-1,n}$ (прибавлением единицы), чем из $c_{i-1,1}$ (прибавлением $n - i + 1$), как это сделано в нашем решении. Но тогда внутренний цикл станет не возобновляемым. От таких упрощений надо удерживаться в практической работе.

Программа 10.3

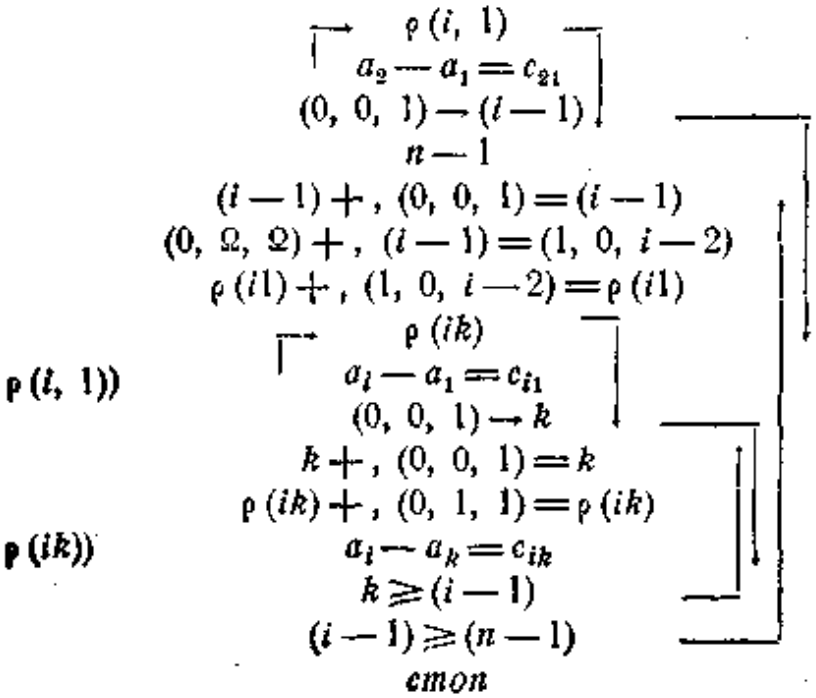


Задача 2. В условиях задачи 1 вычисления выполнить для

$$c_{21};\ c_{31};\ c_{32};\ \ldots;\ c_{n1},\ c_{n2},\ \ldots;\ c_{n,\,n-1}.$$

Решение дает программа 10.4.

Программа 10.4



Задача 3. Умножить матрицу на вектор:

$$y_i = \sum_{k=1}^{n} a_{ik}x_k, \quad i = 1, 2, \ldots, n \qquad (\bar{y} = A\bar{x}).$$

Координаты векторов идут подряд, матрица расположена по строкам, а внутри строки по столбцам, т. е. в порядке $a_{11}, a_{12}, \ldots$ $\ldots, a_{1n}; a_{21}, a_{22}, \ldots$

Решение дается программой 10.5.

Программа 10.5

	$\rho(i)$
$\rho(1))$	$R2+, 0 = y_1$
	$\rho(i1)$
$\rho(11))$	$a_{11} \cdot x_1 = R1$
	$(0, 0, 1) \to i$
	$(0, 0, n)$
$i+1)$	$i+, (0, 0, 1) = i$
	$\rho(i)+, (0, 0, 1) = \rho(i)$
	$\rho(i1)+, (n, 0, 0) = \rho(i1)$
$k=1)$	$0+, 0 = R2$
	$\rho(ik)$
$\rho(i1))$	$a_{i1} \cdot x_1 = R1$
	$(0, 0, 1) \to k$
	$k+, (0, 0, 1) = k$
	$\rho(ik)+, (1, 1, 0) = \rho(ik)$
$\rho(ik))$	$a_{ik} \cdot x_k = R1$
	$R2 + R1 = R2$
	$k \geq n$
$\rho(i))$	$R2+, 0 = y_i$
	$i \geq n; (i+1)$
	стоп

Задача 4. Сделать для предыдущей программы начало, благодаря которому к ней станет возможно следующее обращение (из основной программы):

$s)$	$(я+3) \leftrightarrow \Omega;\ A\bar{x}$
$s+1)$	$(a_{11};\ x_1;\ y_1)$
$s+2)$	$(0, 0, n)$

Решение. Надо заменить *стоп* программы 10.5 командой

$$0 \to 0;\quad \Omega$$

и заполнить ячейки $\rho(1)$, $\rho(11)$, $(0, 0, n)$, $(n, 0, 0)$. Сделать это можно так, как предложено программой 10.6.

Специальным выбором констант для прибавки здесь можно вторую и третью команду заменить одной, а десятую убрать (попробуйте!).

Программа 10.6

```
Ax)  Ω·,  (0, 1, 0) = R1
     R1 —, (0, 2, 0) = R1
     (0 +, 0 = R1) +, R1 = ρ
ρ)   0 +, (s + 1) = R1
     R1 ∧ (Ω, Ω, 0) = R2
     (0 · 0 = R1) +, R2 = ρ(11)
     R1 ∧ (0, 0, Ω) = R1
     (R2 +, 0 = 0) +, R1 = ρ(1)
     ρ +, (0, 1, 0) = (я + 1)
     0 +, (s + 2) = R1
     R1 +, 0 = (0, 0, n)
     (0, 0, n)·, (1, 0, 0) = (n, 0, 0)
```

Переменные ячейки $A(1)$, $\rho(11)$, $(0, 0, n)$, $(n, 0, 0)$ и $\rho(i, 1)$ программы 10.5 удобно выбрать рабочими $R3 \div R7$.

Можно решить эту задачу и иначе: убрать первые четыре команды программы 10.5, а перед пятой поместить программу, возобновляющую $\rho(i)$ и $\rho(i1)$.

10.3. К трехкратному циклу приводит

Задача 5. Перемножить две матрицы $A \cdot B = C$, т. е. вычислить элементы c_{ik} матрицы C:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk} \qquad i = 1, 2, \ldots, p;\ k = 1, 2, \ldots, q.$$

Матрицы A и B записаны по строкам, а внутри строки — по столбцам в последовательных ячейках. Так же нужно записать и матрицу C. Заданы p, q, n и адреса a_{11}, b_{11}, c_{11}.

Задача 6. В ячейках $301 \div 400$ находятся n чисел $x_1, x_2, \ldots, x_n$. Расположить их в порядке возрастания.

Указание. Эта задача имеет различные решения в зависимости от того, хотим мы сократить число занятых ячеек или число действий. Число действий можно сократить от очевидного cn^2 до $cn \log n$, если сперва упорядочить элементы в парах

(x_1, x_2), (x_3, x_4), ..., потом в четверках (x_1, x_2, x_3, x_4), (x_5, x_6, x_7, x_8), ... и т. д. Но при этом потребуется дополнительно n рабочих ячеек.

Задача 7. Внешняя программа в процессе работы вырабатывает в ячейке α очередную величину x_k и обращается к подпрограмме. После очередного обращения $k = 1, 2, \ldots$, подпрограмма должна содержать в порядке убывания в q ячейках от 701 до $700 + q$ максимальные q чисел из $x_1, x_2, \ldots, x_k$. Число q равно

$$q = \begin{cases} k & \text{при} \quad k \leqslant 60_{10}, \\ 60_{10} & \text{при} \quad k \geqslant 60_{10}. \end{cases}$$

Замечание. Подпрограмма должна состоять из двух частей: для первоначальной подготовки и для работы при обращении из внешней программы.

Решение. Обозначим $a_0 \sim 700$, $a_1 \sim 701$, ..., $a_{60} \sim 774$ и занесем при подготовке в a_0 максимально возможное в машине число M, а во все ячейки от a_1 до a_{60} минимально возможное число $-M$. Решение дается программой 10.7.

Программа 10.7

ПОДГОТОВКА

```
      ρ; ρ
   0 — M = a1
эт)  0 — M = a60
      ρ +, (0, 0, 1) = ρ
ρ)    0 — M = ai
      ρ ⩾ эт;
      M +, 0 = a0
      выход
```

РАБОТА

```
      ρ1
   a60 ⩾ α
      ρ2
   a60 +, 0 = a61
ρ2)   ai +, 0 = ai+1
      ρ2 —, (1, 0, 1) = ρ2
      ρ —, (1, 0, 0) = ρ
ρ1)   α ⩾ al
   ρ2 ∧ (0, 0, Ω) = R1
   (0 +, α = 0) +, R1 = (я + 1)
   0 +, α = ai+1
   выход
```

10.4. Мы уже видели, как можно повысить скорость циклических программ, вводя эталон-команду. Этот затруднительный прием еще более усложняется в многократных циклах из-за того, что эталон-команда внутреннего цикла становится переменной (зависит от номера наружного цикла). И мы по-прежнему не рекомендуем пользоваться эталон-командами без особой нужды. Все же рассмотрим три варианта применения эталон-команды в задаче 3 (программы 10.8, 10.9, 10.10).

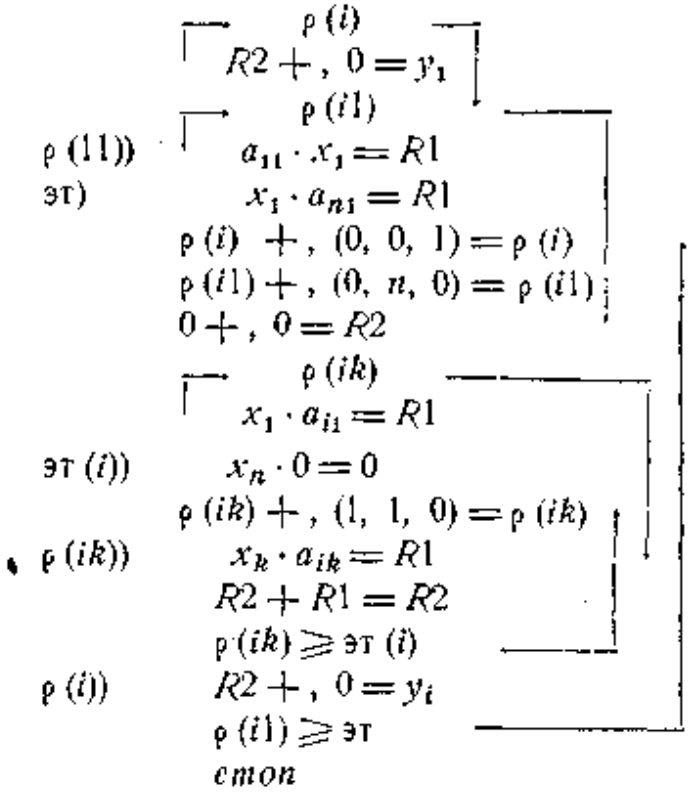
Программа 10.8

```
                ρ (i)
              R2 +, 0 = y_i
                ρ (i1)
ρ (11))     a_11 · x_1 = R1
              эт (i)
            a_1n · x_n = R1
эт)         a_n1 · x_1 = R1
            ρ (i) +, (0, 0, 1) = ρ (i)
            ρ (i1) +, (n, 0, 0) = ρ (i1)
            эт (i) +, (n, 0, 0) = эт (i)
               0 +, 0 = R2
                ρ (ik)
ρ (i1))     a_i1 · x_1 = R1
эт (i))     a_in · x_n = R1
            ρ (ik) +, (1, 1, 0) = ρ (ik)
ρ (ik))       a_ik · x_k = R1
              R2 + R1 = R2
              ρ (ik) ⩾ эт (i)
ρ (i))        R2 +, 0 = y_i
                ρ (i1) ⩾ эт
                 стоп
```

Программа 10.9

```
                ρ (i)
              R2 +, 0 = y_1
                ρ (i1)
ρ (11))     a_11 · x_1 = R1
эт)           a_n1 · x_1 = R1
            ρ (i) +, (0, 0, 1) = ρ (i)
            ρ (i) +, (n, 0, 0) = ρ (i1)
            ρ (i1) +, (n — 1, n — 1, 0) = эт (i)
               0 +, 0 = R2
                ρ (ik)
              a_i1 · x_1 = R1
эт (i))     a_1n · x_n = R1
            ρ (ik) +, (1, 1, 0) = ρ (ik)
ρ (ik).       a_ik · x_k = R1
              R2 + R1 = R2
              ρ (ik) ⩾ эт (i)
ρ (i))        R2 +, 0 = y_i
                ρ (i1) ⩾ эт
                 стоп
```

Программа 10.10



Плох вариант программы 10.8. Введя переменный эталон эт (i), мы приобрели команду изменения этого эталона и, что еще хуже, две ячейки (возобновление и возобновитель эталона) *для возобновления* $i = 1$ *наружного цикла.*

Таким образом, вместо одной ячейки, занятой «эталонной службой», мы получили пять. Если бы цикл был трехкратным — их стало бы 11, для четырехкратного — 19 и т. д. Ясно, что от этого варианта надо отказаться.

В *случае нужды эталон-команды следует организовывать* по типу программы 10.9. Здесь при изменении i эталон эт (i) не изменяется, а образуется из возобновителя ρ ($i1$), на который он наверняка должен быть похож. Благодаря этому возобновление эт (i) при $i = 1$ не потребовалось. При росте кратности циклов преимущество этого варианта перед предыдущим сказывается еще сильнее.

Что касается варианта программы 10.10, то он — плод счастливого случая: переставив сомножители в команде ρ (ik), оказалось возможным сделать эталон эт (i) постоянным.

10.5. Свертывание многократных циклов. Мы познакомились с одно-, двух- и трехкратными циклами. Легко образовать задачи с большей кратностью циклов. Естественно, возникает вопрос, сколь велика может быть кратность циклов. Точнее, может ли случиться коротко сформулированная задача, которая потребует n-кратного цикла *)? Ответ отрицателен: если циклы заданы *регулярным* образом (т. е. задано правило для перехода от наружных циклов к внутренним), *то всегда многократный цикл сворачивается в двухкратный.*

Это полезно знать, потому что ведет к упрощению необозримо сложных программ.

Проиллюстрируем свертывание циклов.

10.5.1. Десятичный счетчик. Пусть задано n последовательных ячеек $a_1, a_2, \ldots, a_n$, изображающих десятичный счетчик, подобный счетчику электроэнергии. Пусть a_n изображает разряд единиц, a_{n-1} — разряд десятков и т. д. Вначале нужно погасить счетчик, т. е. занести нуль в ячейки a_1, $a_2, \ldots, a_n$. После этого программа должна многократно добавлять в счетчик по единице, т. е. повторять следующий процесс:

> найти самый правый разряд, где находится число, меньшее девяти, увеличить это число на единицу и (если этот разряд не последний) занести нуль в разряды, следующие за ним.

Процесс прекратить, когда во всех разрядах будут девятки.

Решение. Если $n = 1, 2, 3$, то решение естественно написать одно-, двух- и трехкратным циклом. Но если n велико, то мы замечаем, что всегда достаточно двухкратного цикла.

Рассмотрим оба случая.

Для малых n решение показано в программах 10.11 ($n = 1$) и 10.12 ($n = 3$).

При большом n решение можно оформить по схеме рис. 10.4 программой 10.13.

---

*) Коротко сформулированная — т. е. сформулированная так, что формулировка не удлиняется при росте n.

Программа 10.11

$0 \to a_1$
$a_1 +, (0, 0, 1) = a_1$
$a_1 \geqslant (0, 0, 11)$
стоп

Программа 10.12

$0 \to a_1$
$a_1 +, (0, 0, 1) = a_1$
$0 \to a_2$
$a_2 +, (0, 0, 1) = a_2$
$0 \to a_3$
$a_3 +, (0, 0, 1) = a_3$
$a_3 \geqslant (0, 0, 11)$
$a_2 \geqslant (0, 0, 11)$
$a_1 \geqslant (0, 0, 11)$
стоп

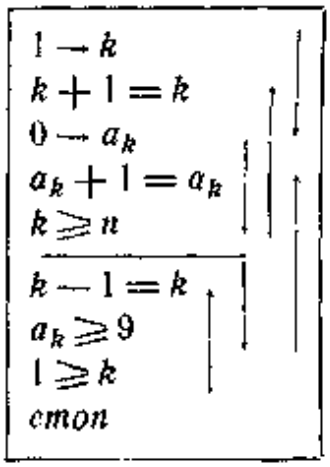


Рис. 10.4.

Программа 10.13

$(1, 1, 1) \to k$
$k +, (1, 1, 1) = k$
$k \wedge (0, \omega, 0) = R1$
$c1 +, R1 = (я + 1)$

$0 \to a_k$
$s1)$ $k \wedge (\omega, 0, \omega) = R2$
$c2 +, R2 = (я + 1)$
$a_k +, (0, 0, 1) = a_k$
$s2)$ $k \geqslant (n, n, n)$

$k -, (1, 1, 1) = k$
$k \wedge (\omega, 0, 0) = R3$
$c3 +, R3 = (я + 1)$
$a_k \geqslant (0, 0, 11)$
$(1, 1, 1) \geqslant k$
стоп

$c1)$ $0 \to a_0$; $s2$
$c2)$ $a_0 +, (0, 0, 1) = a_0$
$c3)$ $a_0 \geqslant (0, 0, 11)$; $s1$

Упражнение. Разберите работу программы 10.13. Составьте самостоятельно программу для десятичного счетчика, например, перестроив нашу схему так, чтобы она не содержала безусловной передачи управления.

10.5.2. Работу десятичного счетчика можно было описать и не так уж точно, как это сделали мы. Можно было не

задавать, в каком порядке гасятся девятки и что делается вначале — гасятся девятки или прибавляется единица переноса. При этом и программа могла бы получиться проще.

Однако наше описание десятичного счетчика привело *к общей* схеме сворачивания многократного цикла, пригодной для задач, связанных с построением и анализом вариантов. Такая схема нужна в задачах планирования, игровых задачах и многих других.

10.5.3. Пусть, например, для достижения определенного результата нужно шаг за шагом выполнять некоторые работы. Требуется

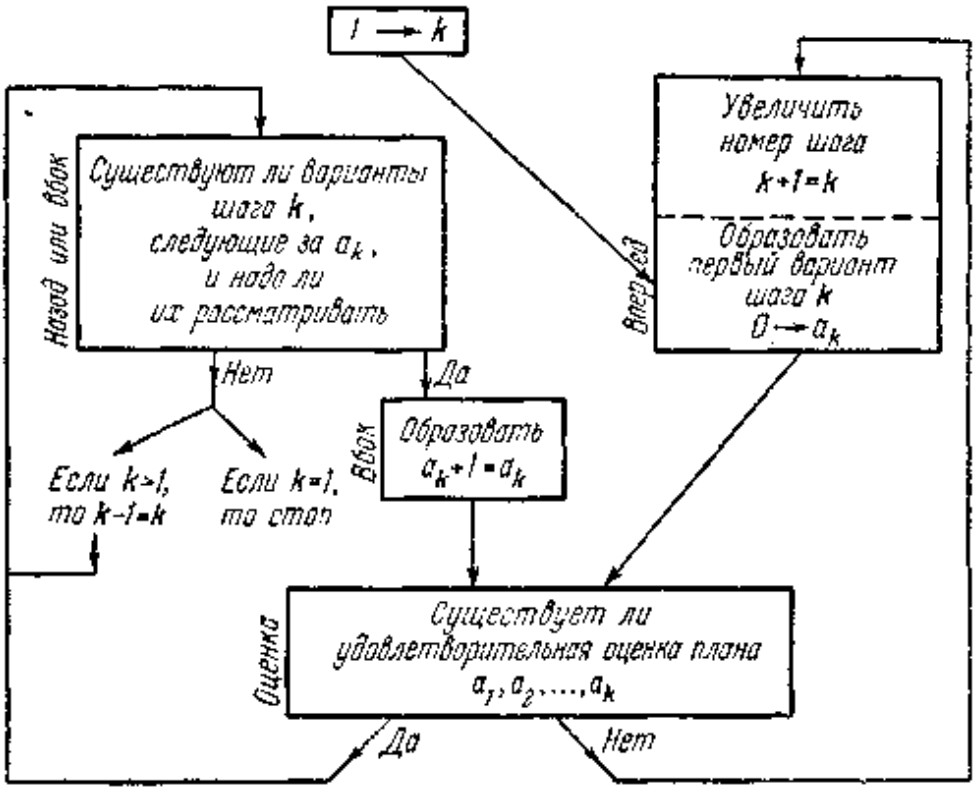


Рис. 10.5.

найти очередность выполнения работ, приводящую к заданному результату с минимальными затратами времени или средств. Эта задача приводит к схеме 10.4, которую мы рассмотрели. Число k становится номером шага, а a_k — номером работы, которая может выполняться на этом шаге при условии, что на предыдущих шагах уже выполнены вполне определенные работы, описанные числами $a_1, a_2, \ldots, a_{k-1}$. Первая работа на шаге k соответствует значению $a_k = 0$, вторая — $a_k = 1$ и т. д. Вся последовательность $a_1, a_2, \ldots$ показывает порядок выполнения работ в рассматриваемом плане.

Правда, в таком более общем случае на k-м шаге рассматривается уже не десять вариантов ($0 \leqslant a_k \leqslant 9$), а некоторое число c_k,

зависящее от номера шага. Впрочем, c_k может и не задаваться заранее, а определяться по ситуации, возникшей к этому моменту. Еще важнее, что и число шагов n (глубина анализа) может не указываться заранее и быть различной для разных вариантов. Достаточно ли ограничиться просмотром вариантов n-го шага или надо перейти к следующему, $(n+1)$-му шагу, может определяться в зависимости от уже проделанных работ, то есть от чисел $a_1, a_2, \ldots, a_n$, которые к этому моменту находятся в ячейках, и от того успеха, которого добилась программа на других, уже рассмотренных вариантах.

Программа, учитывающая эти обобщения, приведена на рис. 10.5.

В каждой конкретной задаче пункты «оценка» и «назад или вбок» приобретают соответствующее содержание. В задаче о десятичном счетчике оно было таким: «оценка» выполнялась сравнением $k \geqq n$, а «назад или вбок» — сравнением $8 \geqq a_k$.

10.5.4. Впрочем, схема 10.4 пригодна и для более общей задачи. Имея план работ $a_1, a_2, \ldots, a_k$, программа может сначала рассматривать *только* такие варианты $a_{k+1} = 0, 1, \ldots$ его продолжения, которые могут привести к значительному выигрышу, по сравнению с уже найденными решениями. Если значительного выигрыша достигнуть не удалось, то программа может снова обследовать варианты a_{k+1}, рассматривая менее заманчивые возможности.

Так, анализируя шахматную позицию, мы сначала ищем только мат. Когда этого не удалось — ищем выигрыш ферзя и т. д. Но когда оказалось, что во всех рассмотренных вариантах у нас теряется ладья, то становится привлекательной и жертва пешки.

11. ПУЛЬТ МАШИНЫ

Вычислитель может вовсе не знать конструкции и принципов работы машины, ее схем и элементов. Но он должен знать возможности машины — систему команд и пульт. За пультом вычислитель окончательно отлаживает программу.

Пульт *ММ* (рис. 11.1) состоит из устройств ввода, вывода и стола с двумя панелями — вертикальной и горизонталь-

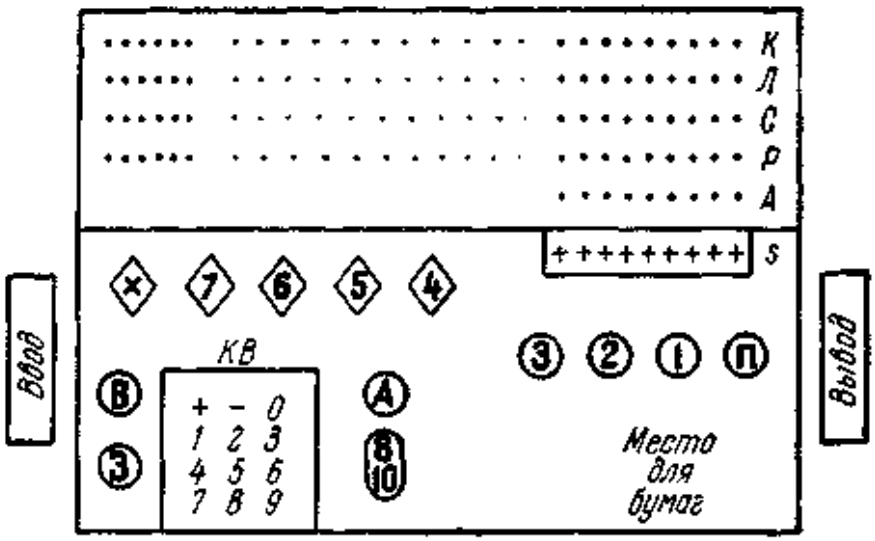


Рис. 11.1.

ной (наклонной). На вертикальной панели находятся пять рядов маленьких лампочек-индикаторов: нижний для адреса выполняемой команды *АВК* (9 лампочек) и ряды *К*, *Л*, *С*, *Р* (по 34 лампочки) — для выполняемой команды, слов, указанных в ее левом и среднем адресах и для результата операции.

На горизонтальной панели находятся различные кнопки и тумблеры управления:

1 — *стоп,*
2 — *шаговая работа,*
3 — *автоматическая работа,*
4 — *стоп команды,*
5 — *стоп чтения,*
6 — *стоп записи,*
7 — *стоп передачи,*
s — *девять тумблеров набора адреса остановки*
╳ — *тумблер, исключающий запись при шаговой работе,*
В — *ввод,*
А — *ввод адреса,*
З — *запись в память,*
П — *печать,*
8, 10 — *тумблер ввода восьмеричных или десятичных чисел,*
КВ — *клавиатура ввода.*

1. Чтобы остановить машину, нужно нажать и отпустить кнопку *стоп*. При этом на лампочках *АВК* загорится адрес очередной команды, а на остальных лампочках — сведения об аргументах и результате этой команды (см. далее — *шаговая работа*).

2. Чтобы ввести программу, надо вставить колоду перфокарт в устройство ввода и нажать (и отпустить) кнопку ввода. Машина остановится, программа введется в память, и ввод прекратится.

3. Чтобы ввести в *АВК* нужный адрес, надо нажать (и отпустить) кнопку **А** (при этом машина остановится) и набрать адрес на клавиатуре по восьмеричной системе. Набранный адрес загорится на лампочках *АВК*, а другие сведения об этой команде — на остальных лампочках.

4. Чтобы пустить машину с адреса, находящегося в *АВК*, нужно нажать кнопку **2** или **3**. В последнем случае (автоматическая работа) режим работы будет зависеть от стоповых тумблеров **4**, **5**, **6**, **7** и адреса, набранного на тумблерах *s*.

5. Для счета по верной (отлаженной) программе, нужно проделать описанные операции, но перед нажатием пусковой

кнопки 3 выключить стоповые тумблеры 4 ÷ 7. Машина будет работать, автоматически выполняя предписания программы.

6. Тумблеры 4 ÷ 7 и *s* употребляются при отладке в режиме автоматической работы (на шаговой работе они не сказываются). Тумблер 4 остановит машину, когда (если) она придет к команде, адрес которой набран на тумблерах *s*. Тумблер 5 остановит машину, если встретится команда, читающая число из ячейки с адресом *s*, а тумблер 6 — при попытке записи в эту ячейку (но до выполнения записи). Наконец, тумблер 7 остановит машину на команде, передающей управление ячейке с адресом *s* (при любом типе передачи: безусловной, условной, стандартной).

7. Употребляется при отладке и шаговая работа, когда команды выполняются по одной за каждое нажатие кнопки 2. При нажатии этой кнопки машина записывает в память результат вызванной команды и вызывает следующую команду, т. е. прочитывает нужные сведения, делает вычисления и показывает результат на лампочках пульта. Если при шаговой работе нужно избежать записи, то перед нажатием кнопки 2 следует включить тумблер ×.

Пусть, например, в *АВ*К находится адрес *я*, по которому записана команда вычитания. Тогда в регистрах *А*, К, *Л*, *С*, *Р* будут гореть соответственно: адрес команды (то есть «*s*»), сама команда (слово из ячейки *s*), уменьшаемое, вычитаемое и вычисленная машиной разность. При нажатии кнопки 2 разность запишется в память, а в *АВ*К загорится *s* + 1.

8. Чтобы посмотреть содержимое какой-либо ячейки, нужно ввести ее адрес в *АВ*К (как указано в 3).

9. Чтобы исправить слово в ячейке, нужно:

а) ввести ее адрес в *АВ*К,

б) установить тумблер 8, 10 в нужное положение,

в) набрать на клавиатуре нужное содержимое ячейки (оно загорится на лампочках верхнего ряда),

г) нажать кнопку записи 3.

Кнопка П служит для печати адреса из *АВ*К и слов из рядов К ÷ *Р* во время отладки.

12. ОРГАНИЗАЦИЯ ПРОГРАММИРОВАНИЯ

12.1. Выбор способа решения задачи производится с учетом следующих обстоятельств:

1. Машина в несколько минут делает вычисления, на которые человеку нужен год.

2. Машина, не путаясь, выполняет сложный логический рисунок вычислений.

3. На логические действия машина тратит почти столько же времени, что и на арифметические.

4. Память машины ограничена.

По этим причинам для машины может оказаться удобным не тот путь решения задачи, который удобен человеку. Машине легче вычислить каждый раз нужное значение синуса, чем хранить в памяти большую таблицу синусов. Интеграл от гладкой функции машине легче сосчитать численно, чем вычислять по длинной формуле или по сходящемуся ряду. Еще более это относится к решению дифференциальных уравнений.

Если человек говорит фразу

«Я думаю о том, что я думаю о том, ... »,

то он уже на третьем-четвертом обороте теряет смысл произносимого (Э. Кант). Машину обороты этого типа не затрудняют.

Но можно привести и обратные примеры, когда задача оказывается неожиданно трудной для машины. Машине легче перемножить все элементы последовательности, чем найти среди них пару равных.

Здесь мы не будем заниматься выбором вычислительных алгоритмов, удобных для машины. Отметим все же, что нельзя начинать программировать, не уяснив в точности, что нужно делать *во всех* возможных случаях. Машина не дает

подправить недодуманную схему во время счета. Она не сообщает о наступлении непредвиденной ситуации. Она тупо продолжает работу, ставшую бессмысленной, и дает в результате дикий ответ, останавливается или вовсе сбивается с программы.

12.2. Блок-программа. Вначале все части программы старались писать подряд, «напрямушку», не выделяя блоков и не передавая им управления специальным образом. Так и мы пишем, например, двойной цикл, помещая внутренний цикл буквально внутри наружного.

С повышением квалификации программистов (и с усложнением задач) программу стали разбивать на блоки—самостоятельные подпрограммы (см. п. 12.3), и появились стандартные приемы обращения к блокам.

Для организации работы блоков пишут *блок-программу*, которая командует подчиненными ей блоками, передавая им управление. Но выходя из блок-программы в блок, нужно знать, к какому (всякий раз единственному!) месту блок-программы вернется управление. Поэтому блок-программа состоит из обращений к блокам (которые обозначаются названиями блоков) и сравнений для условных переходов.

Таким образом, *блок-программа — это программа, в которой целые блоки играют роль команд обычной программы*. Сама блок-программа тоже является блоком программы и обычной программой — она кодируется, перфорируется и вводится в машину. Если блок достаточно сложен, то и он в свою очередь может состоять из блок-программы, которая компонует его подблоки. Эта система продолжается на всех уровнях.

Обращение к любому блоку осуществляется в блок-программе одной стандартной командой

$$n)\ a \multimap \Omega;\ (\textit{начало блока})$$

или ее разновидностью. С блок-программами мы познакомимся в § 14.

12.3. *Каждый блок* программы является самостоятельной подпрограммой и оформляется по правилам, изложенным в п. 8.2. Что касается схем блоков, то это — канонические схемы программы, с которыми мы уже достаточно знакомы.

12.4. Наконец, происходит собственно *программирование* — написание программ в содержательных обозначениях. Оно может вестись независимо разными лицами для разных блоков.

Содержательные обозначения нужно выбирать естественными или стандартными.

Для чисел и букв употребляются их собственные обозначения: ячейку, содержащую величину x^3, обозначают x^2, а эталон наружного цикла — $э_{нар}$ (или $э_i$). Наоборот, обозначение ячейки с x^3 через x^3 хотя и не собьет программу, но увеличит вероятность ошибок, а обозначение $э_{нар}$ через $A1$ или $э_{вн}$ (внешний) затруднит человеку чтение программы или облегчит смешение наружных и внутренних эталонов. Если переменная команда зависит от двух индексов и обозначена $A(ik)$, то легче понять обозначения $A(1k)$ и $A(1,1)$.

Рабочие ячейки для кратковременного хранения неименуемых результатов обозначаются через $R1$, $R2$, ..., а переменные команды программы — через $\rho 1$, $\rho 2$, ...

Всем этим мы, по сути дела, уже занимались. *Программы пишут карандашом (и резинкой!) и педантично отделывают.* Важно отметить, что уже содержательная программа тщательно проверяется.

12.5. Составление *словаря* начинается с распределения памяти, которое отмечают на стандартной форматке (приведена на стр. 116 и 117).

При отсутствии особой тесноты программу не следует особенно ужимать. Блоки программы следует размещать в памяти с небольшими запасными интервалами. Приятно начинать кодировку блока с ячейки, кончающейся на 1. Различные промежуточные результаты следует заносить в различные ячейки — это облегчит отладку. Десятичные константы после перевода в двоичную систему надо записывать на новые места, чтобы программа была возобновляющейся. Само собой разумеется, что все переводимые «$10 \to 2$» константы следует собрать в группы, идущие подряд.

Если программа состоит из небольшого числа блоков, не обращающихся друг к другу, то их рабочие ячейки могут быть общими. Однако чаще одни блоки являются подпрограммами для других. Должно соблюдаться следующее правило: блоками *ранга* 0 называются блоки, не обращающиеся к другим блокам (в частности — библиотечные подпрограммы); блоками *ранга* 1 — те, которые

000	001	002	003	004	005	006	007
020	021	022	023	024	025	026	027
040	041	042	043	044	045	046	047
060	061	062	063	064	065	066	067
100	101	102	103	104	105	106	107
120	121	122	123	124	125	126	127
140	141	142	143	144	145	146	147
160	161	162	163	164	165	166	167
200	201	202	203	204	205	206	207
220	221	222	223	224	225	226	227
240	241	242	243	244	245	246	247
260	261	262	263	264	265	266	267
300	301	302	303	304	305	306	307
320	321	322	323	324	325	326	327
340	341	342	343	344	345	346	347
360	361	362	363	364	365	366	367
400	401	402	403	404	405	406	407
420	421	422	423	424	425	426	427
440	441	442	443	444	445	446	447
460	461	462	463	464	465	466	467
500	501	502	503	504	505	506	507
520	521	522	523	524	525	526	527
540	541	542	543	544	545	546	547
560	561	562	563	564	565	566	567
700	701	702	703	704	705	706	707
720	721	722	723	724	725	726	727
740	741	742	743	744	745	746	747
760	761	762	763	764	765	766	767

010	011	012	013	014	015	016	017
030	031	032	033	034	035	036	037
050	051	052	053	054	055	056	057
070	071	072	073	074	075	076	077
110	111	112	113	114	115	116	117
130	131	132	133	134	135	136	137
150	151	152	153	154	155	156	157
170	171	172	173	174	175	176	177
210	211	212	213	214	215	216	217
230	231	232	233	234	235	236	237
250	251	252	253	254	255	256	257
270	271	272	273	274	275	276	277
310	311	312	313	314	315	316	317
330	331	332	333	334	335	336	337
350	351	352	353	354	355	356	357
370	371	372	373	374	375	376	377
410	411	412	413	414	415	416	417
430	431	432	433	434	435	436	437
450	451	452	453	454	455	456	457
470	471	472	473	474	475	476	477
510	511	512	513	514	515	516	517
530	531	532	533	534	535	536	537
550	551	552	553	554	555	556	557
570	571	572	573	574	575	576	577
710	711	712	713	714	715	716	717
730	731	732	733	734	735	736	737
750	751	752	753	754	755	756	757
770	771	772	773	774	775	776	777

обращаются только к блокам ранга 0; блоками *ранга* 2 называются такие, которые обращаются только к блокам ранга 0 и ранга 1 и т. д. Рабочие ячейки для блоков разных рангов выбираются *различными* и обозначаются

*R*1, *R*2, ...	для блоков ранга 0
*R*11, *R*12, ...	для блоков ранга 1
*R*21, *R*22, ...	для блоков ранга 2 и т. д.

Для часто встречающихся величин следует выбирать удобные, легко запоминающиеся адреса (особенно для рабочих ячеек). Так, например, в среднем столбце адреса подобраны хорошо, а в правом плохо:

*R*11	311	317
*R*12	312	326
*R*21	321	400
*R*22	322	364
$Сч_{нар}$	430	371
$Сч_{вп}$	431	415

То же относится к однородным группам обозначений (x_1, x_2, ..., эт 1, эт 2, ...). Это не только облегчит труд кодировщицы, но и уменьшит вероятность ошибок.

Дополнительные трудности возникают *при решении задач, которые помещаются в памяти с трудом.* Память машины (в отличие от бумаги вычислителя) ограничена и вскоре становится невозможным отводить каждой промежуточной величине собственную ячейку. Когда некоторая величина (промежуточный результат) перестает быть нужной, то отведенную ей ячейку приходится использовать для другой величины. В этом случае в клетку словаря записываются обе (все три и т. д.) величины. Составление словаря становится делом не механическим: приходится следить, какие величины еще понадобятся, т. е. какие ячейки еще заняты, а какие уже освободились.

12.6. Кодирование и перфорация должны выполняться совершенно механически. Они примерно раз в десять менее трудоемки, чем программирование. Кодировку лучше производить *не тому*, кто программировал. Ответственная проверка кодировки вторым человеком *обязательна.*

Проверенная программа *перфорируется*, то есть переносится на специальные карты или ленту с помощью перфоратора. *Перфоратор* имеет клавиатуру для двоичных (+ и —), восьмеричных (от 0 до 7) и десятичных (от 0 до 9) цифр.

Цифры *двоичной* системы (+ и —, т. е. 0 и 1) перфоратор переносит на карту, пробивая отверстие для единицы и оставляя пропуск для нуля; каждую *восьмеричную* цифру перфоратор заменяет *трехзначным* двоичным числом и пробивает его на карте; каждая *десятичная* цифра пробивается *четырехзначным* двоичным числом. Может быть не лишне заметить, что перфоратор переводит в двоичную систему *не числа*, а только *лишь цифры* десятичной системы.

При работе на перфораторе надо закрепить за каждой клавишей один определенный палец и перфорировать только вслепую, не глядя на клавиатуру. Необходимо сначала добиться безошибочного автоматизма перфорации, а потом уже увеличивать скорость.

Готовые карты печатают и считывают с программой. Здесь можно отметить три обстоятельства. Если одновременно с перфорацией происходит и печать, то полученные карты все равно нужно отпечатать. Не исключено, что клавиша нажата верная, отпечатан знак верно, а на карту попал в искаженном виде. Второе обстоятельство заключается в том, что считывание не является эффективным контролем (на счетных станциях такой контроль запрещен и заменен суммированием). Поэтому считывать нужно очень внимательно. Считывают вдвоем. Один читает, а другой (ответственный за карты) — проверяет. И наконец, последнее. Нужно иметь два идентичных экземпляра карт, на случай порчи при вводе в машину.

12.7. Контроль программы является одной из наиболее трудных работ. Организация его требует высокой квалификации работников. Приведем несколько приемов контроля.

Разыгрывание программы. Вычислитель пишет на бумаге адреса ячеек памяти и заносит в них первоначальную информацию кодированной части программы. Затем он выполняет предписываемые программой действия так, как это предписано делать машине. В частности, заносит результаты строго в указанные ячейки. Полной аналогии здесь добиться невозможно по двум причинам. Во-первых, вычислитель пишет числа (но не команды) в десятичной системе. Во-вторых, если некоторый цикл должен быть проделан большое число раз, то вычислитель должен проделать его два-три раза и сообразить,

в каком состоянии будут ячейки и команды после выполнения цикла нужное число раз (или после достижения заданного результата). *Разыгрывание программы является необходимым элементом обучения программированию и вернейшим показателем, умеет ли уже обучающийся программировать.*

Контрольные просчеты. Лучше всего иметь ручной просчет хотя бы для упрощенного варианта задачи. Он выполняется с большой точностью — с 7—8 десятичными знаками. Для контрольного варианта часто берут искусственный случай, в котором ответ известен (может быть найден алгебраически), либо случай, в котором вычисления требуют мало работы — например, интегрирование по уменьшенному числу точек. При выборе входных данных для контрольного просчета следует выбирать все числа различными, избегать чисел вроде нуля и единицы, которые могут не портить результата и при неверном обращении. Все числа следует брать достаточно большими, чтобы не оказались не замеченными малые члены.

Проверка результата. В некоторых случаях (например, при решении системы уравнений), результат трудно получить, но легко проверить — подставить в задание. Такие подстановки можно включать в саму программу задачи в виде отдельных блоков.

12.8. Отладка. Программу можно выверить на руках, но работа эта тяжелая, трудоемкая и может сильно увеличить общие расходы на решение задачи. Поэтому, как правило, программа (карты) содержит ошибки и окончательно отлаживается на машине. Отладка облегчается тем, что программист имеет мало шансов допустить механическую ошибку, которая бы привела к небольшому искажению результата. Как правило, машина останавливается из-за невозможности вести предписанный счет (переполнение разрядов), зацикливается, не выходит на ответ через ожидаемое время или дает резко неправдоподобный результат.

Абсолютно необходимо подготовить память и проверить ввод. Для этого начала кнопкой пульта во все ячейки памяти заносится число «минус нуль». Затем вводится сама программа. Наконец, библиотечная подпрограмма специальным образом суммирует все ячейки памяти и печатает контрольную сумму.

Эта сумма будет полезна в дальнейшем как при плохом вводе, так и (в особенности) при порче карт. Для росписи памяти выбрано слово «минус нуль» потому, что машина останавливается и при команде «минус нуль» и при арифметических действиях с числом «минус нуль».

Отладку предпочтительно вести поблочно, но иногда пускают программу целиком. Если выясняется, что счет идет неверно, то участок, содержащий ошибку, локализуют. Для этого прежде чем пустить машину с нужного адреса, набирают на пульте адрес стоп-ячейки. В качестве такой ячейки надо брать ячейку с командой, непосредственно следующей за проверкой окончания цикла. Можно также предписать машине остановиться при попытке программы записать результат в некоторую ячейку («испортить ячейку») или взять из нее число.

Когда найден неверный цикл, его проходят на шагах — выполняя команды по одной от ручного (не автоматического) управления — и следят по пульту за видом команд, чисел и результатов.

Найденная ошибка исправляется в памяти машины (с помощью пультовой клавиатуры) и в программе. Справа, на полях, пишется дата исправления.

В тяжелых случаях, когда машина уже считает, но дает слегка неверный результат, приходится прибегать к отладочной программе «няня», печатающей заданные промежуточные результаты.

Существенно облегчает отладку наличие в машине барабана (см. далее п. 15.7) или иных частей памяти, которые можно отключать, с тем чтобы информация, записанная на них, не могла быть испорчена машиной. Тогда в отключенной части держат копию отлаживаемой программы. Все исправления вносят в копию и оттуда переносят в оригинал. Копия значительно ускоряет отладку: при порче программы (из-за неверного счета) нет нужды заново вводить карты. Копию можно снимать не только с начального состояния программы, но и с того, в котором она находится после работы ряда блоков. Благодаря этому после порчи программы отладку можно продолжать, не начиная сначала.

По окончании отладки повторяется полностью все сначала, с вводом поправочных карт. Проверяется контрольная сумма и снова производится счет контрольного варианта.

Общих рецептов организации контроля и отладки дать, наверное, нельзя. Следует, однако, помнить, что слова «выхожу на машину» в переводе на старомематический означают: «собираюсь доказать теорему». Запрещается выходить на машину без инструкции (плана отладки). *Инструкция должна быть настолько полной, чтобы ее выполнение было однозначным и могло быть проведено человеком, не знающим ничего о программе.* Продолжительная отладка плоха еще и тем, что не дает возможности отличить тупого от ленивого.

13. БЛОК-ПРОГРАММА

В предыдущей главе было описано назначение и оформление блок-программы. Остается предостеречь читателя от «открытия», что блок-программа не нужна. Действительно, в готовой и отлаженной программе она выглядит излишней. Но блок-программа была нужна, чтобы программа была написана и отлажена человеком, с его способностью ошибаться и неспособностью заранее предусмотреть все, вплоть до распределения памяти. Поэтому всякая программа (если это не пример для упражнения и не специальная подпрограмма) начинается с блок-программы.

13.1 *). Решая задачу, нужно постараться разбить ее на осмысленные части и описать это разбиение блок-программой. Таким образом, программа пишется «сверху» — начиная с блок-программы всей задачи. Этот же принцип употребляется и при написании программ для блоков. Нужно писать программы тех частей, которые пишутся легко и выносить в отдельные подблоки все, что вызывает затруднение. Следует избегать даже двухкратных циклов — легче написать однократный наружный цикл и вынести внутренний цикл в отдельный подблок.

13.2. Рассмотрим примеры составления блок-программ.

Задача 1. Найти вес стального коленчатого вала, заданного чертежом на рис. 13.1 (стр. 125).

Решение. Отнесем к шейке вала цилиндрическую трубу длины l, а к щеке — часть, соединяющую две шейки. Тогда блок-программу 13.1 можно написать так (см. левый столбик):

---

*) Раздел 13.1 с любезного разрешения авторов заимствован из книги Р. С. Гутера, В. Л. Арлазарова и А. В. Ускова «Практика программирования» (справочник, «Наука», 1965).

Блок-программа 13.1

перевод констант	201	01	202	777	211
объем шейки	202	01	203	777	221
объем щеки	203	01	204	777	231
вес вала	204	01	205	777	241
печать веса	205	01	206	777	251
стоп	206	—00	000	000	000

После этого пишутся программы блоков. Предполагается, что l, d, r, ρ, h и $\gamma = 7{,}8$ (удельный вес стали) вводятся в десятичной системе в ячейки l_{10}, d_{10}, ..., γ_{10}, идущие подряд. Тогда

1. Перевод констант

211) Ω +, 0 = *вых*
(*я* + 2) ↔ Ω; *групп*
[(10 → 2); l_{10}, l, 6]

2. Объем шейки

$v_ш = \pi l \rho (2r - \rho)$

221) $r + r = R1$
$R1 - \rho = R1$
$R1 \cdot \rho = R1$
$R1 \cdot \pi = R1$
$R1 \cdot l = v_ш$
$0 \to 0$; Ω

3. Объем щеки

$v_щ = dr(2h - \pi r)$

231) $h + h = R1$
$\pi \cdot r = R2$
$R1 - R2 = R1$
$R1 \cdot r = R1$
$R1 \cdot d = v_щ$
$0 \to 0$; Ω

4. Вес вала

241) $v_ш \cdot 3_n = R1$
$v_щ + v_ш = R2$
$R1 + R2 = R1$
$R1 \cdot \gamma = \alpha$
$0 \to 0$; Ω

5. Печать веса

251) Ω +, 0 = *вых*
(*я* + 1) ↔ Ω; (2 → 10)
выход

Кодировка блок-программы 13.1 (приведенная в правом столбике) получается либо заранее, при ориентировочном распределении памяти, либо после кодирования блоков.

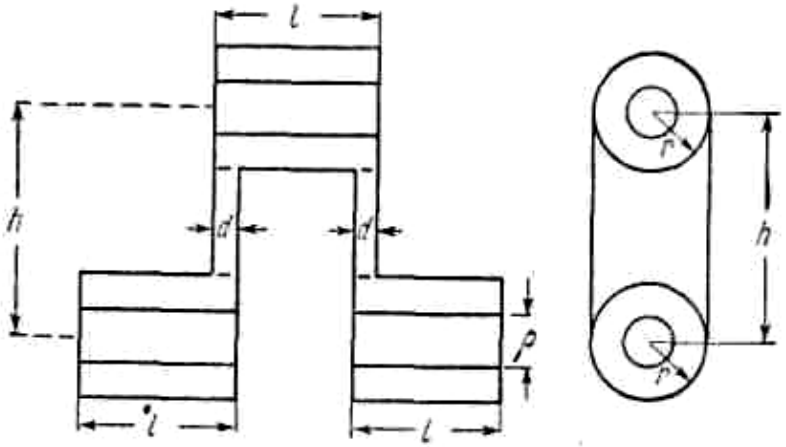


Рис. 13.1.

Задача 2. Найти площадь выпуклого восьмиугольника Q (рис. 13.2), заданного неравенствами

$$a_i x + b_i y + c_i \geqslant 0 \qquad i = 1, \ldots, 8.$$

Решение. Обозначим через li прямую

$$a_i x + b_i y + c_i = 0.$$

Для решения может быть выбран следующий путь. Найти точки r, являющиеся вершинами Q, расположить их в порядке

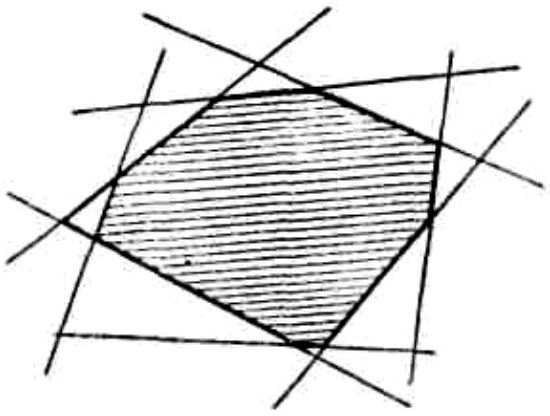


Рис. 13.2.

следования по контуру: $r_1, r_2, \ldots, r_8$. Получить площадь Q, как сумму площадей треугольников $(r_1 r_2 r_3)$, $(r_1 r_3 r_4)$, $(r_1 r_4 r_5)$, ...

$\ldots, (r_1 r_7 r_8)$. Площадь треугольника (r_1, r_i, r_j) равна абсолютной величине числа

$$\frac{(x_i - x_1)(y_j - y_1) - (x_j - x_1)(y_i - x_1)}{2}, \quad \text{где } r_i \equiv (x_i, y_i).$$

Фактически мы поступим так. Рассмотрим точки пересечения прямой $l1$ с остальными. Найдем точку $r_1 = l1 \cap lj$, принадлежащую Q и перенумеруем прямые, обозначив lj через $l2$. Теперь рассмотрим пересечения $l2$ с остальными прямыми. Найдем r_2 и прямую lj, которую следует перенумеровать в $l3$, и т. д. Последняя точка r_8 будет лежать на пересечении $l8$ и $l1$.

При этом для каждой точки пересечения $r_{ij} = li \cap lj$ нам придется определять, принадлежит ли она Q, то есть проверять неравенства

$$l_k(r_{ij}) \equiv a_k x_{ij} + b_k y_{ij} + c_k \geqslant 0$$

для всех k, не равных ни i, ни j.

Блок-программа 13.2 вычисления S_Q приведена ниже.

Блок-программа 13.2

Перейдем к описанию блоков.

Подготовка переводит коэффициенты a_1, b_1, c_1, a_2, b_2, c_2, ... в двоичную систему с $ПЗ$; делает $i=1$ (первая прямая и точка) и заносит 0 в S и R.

Поиск. Входная информация: число i и коэффициенты прямых $L=(l1, l2, \ldots)$, записанные подряд. Поиск находит вершину Q, лежащую на пересечении li и lj, среди $j>i$. Выходная информация: координаты вершины $r_{i+1}\equiv li\cap lj$ в ячейках $r(r_x, r_y)$ и номер j. Если вершины r_{i+1} не найдется, то в j поступает нуль. Поиск имеет свою блок-программу (блок-программа 13.3). Здесь пояснения заслуживают два блока. Блок $r\equiv li\cap lj$ находит r — пересечение прямых с номерами i и j — в последовательности L. Делается это с осторожностью, исключающей переполнение разрядов при делении на нуль (прямые параллельны) или на слишком малое число (точка пересечения далека). В обоих случаях прямые считаются не пересекающимися, и в «пересеч.» заносится 0 (а не (0, 0, 1)).

Блок $r\notin Q$ подставляет координаты r в уравнения lx. Если $lx(r)\geqslant 0$ для всех x, отличных от i и j, то $r\in Q$, и в «верш.» подается (0, 0, 1). В противном случае в «верш.» подается нуль.

Перестройка. *Входная информация: ряд L, числа i и j.* Передвигает коэффициенты lj на место $l(i+1)$, а коэффициенты lk на место $l(k+1)$ для всех $i<k<j$.

Последняя перестройка. Циклически переставляет коэффициенты $l1$, $l2, \ldots, l8$ в порядке $l2$, $l3, \ldots, l8$, $l1$, делает $i=7$ и заносит (0, 0, 1) в R.

Блок S_Δ вычисляет площадь $\Delta(r, r_1, r_2)$.

Печать S работает, разумеется, с помощью перевода $(2\to 10)$.

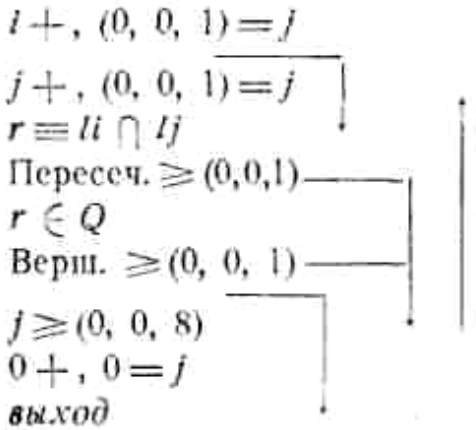
В блок-программе есть еще краткие обозначения. Перенос $r\to r_1$ — это две команды: $r_x\to r_{1x}$ и $r_y\to r_{1y}$; то же относится к переносу $r\to p$. После поиска машина выходит в блок-программе на стоп-ошибку, если нужная вершина не найдена ($j=0$). Проверка конца — это сравнение $R\geqslant(0, 0, 1)$. Напомним, что в подготовке в R заносится нуль, а в последней перестройке в R заносится единица.

Задача 3. Найти площадь выпуклого многоугольника Q (рис. 13.3), *заданного неравенствами*

$$a_ix+b_iy+c_i\geqslant 0, \qquad i=1, 2, \ldots, 8.$$

Указание. Q может оказаться многоугольником с числом сторон, меньшим восьми. В частности, может случиться так, что Q вообще не будет содержать точек.

Блок-программа 13.3



Задача 4 (трудная). Составить блок-схему для вычисления объема восьмигранника $a_i x + b_i y + c_i z + d_i \geqslant 0$, $i = 1, 2, \ldots, 8$.

Прекрасные примеры для составления блок-схем дают игровые задачи:

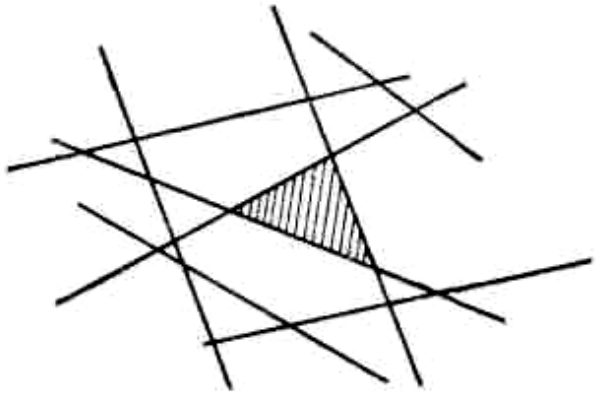


Рис. 13.3.

Задача 5. Король и две ладьи против короля. Сильная сторона начинает и выигрывает. Обратить внимание на трудные случаи начального положения.

Задача 6 (трудная). Король и пешка против короля. Составить блок-схему программы для нахождения оптимального хода.

14. ЛОГИЧЕСКИЕ И БИБЛИОТЕЧНЫЕ ПРОГРАММЫ

Эту главу вычислитель может опустить. Программисту на первых порах достаточно ознакомиться с нею.

Логические и библиотечные программы сложнее счетных. Они требуют изобретательности и порою достигают уровня шахматного этюда. К этому же классу относятся программы, в которых минимизировано число действий или число занятых ячеек. Последнее потеряло практическое значение (с ростом размеров памяти), но сохранило ценность для обучения программированию.

14.1. Логические программы. Программист может использовать слово четырьмя способами: как команду, как число с плавающей запятой, как целое число и, наконец, просто как слово — последовательность нулей и единиц, — когда ячейка рассматривается как множество (последовательность) своих разрядов. Последнее особенно ясно видно при таких операциях, как сдвиг, логическое сложение, логическое умножение. Их трудно описать как действия над числами или командами. В логических командах и программах существенна система счисления (в *ММ* — двоичная) и расположение информации (о числах и командах) в ячейке памяти машины.

Задача 1. Образование дополнения. Пусть задано слово a, у которого кодовые разряды (34—28) равны нулю. Заменить в цифровой части единицы нулями, а нули — единицами.

Решение.

$$(\Omega,\ \Omega,\ \Omega) —,\ a = R1.$$

Задача 2. Размножение последней единицы. В условиях предыдущей задачи образовать слово b, совпадающее со словом a, но имеющее единицы в разрядах, находящихся правее самой правой единицы a.

Решение:

$$a —,\ (0,\ 0,\ 1) = R1$$
$$a \vee R1 = R2$$

Например (в двоичной записи),

$$\begin{array}{r} -, a = +00\,00\ldots01110010011010000 \\ (0,\ 0,\ 1) = +00\,00\ldots00000000000000001 \\ \hline R1 = +00\,00\ldots01110010011001111 \\ a \vee R1 = +00\,00\ldots01110010011011111 \end{array}$$

Задача 3. Гашение хвоста единиц. В условиях задачи 1 образовать число b, у которого все разряды совпадают с разрядами a, но равны нулю последние разряды, занятые у a единицами.

Решение:

$$R1 +, (0,\ 0,\ 1) = R2$$
$$R1 \wedge R2 = R3$$

Задача 4. Выделение младшей единицы. Пусть задано слово a, занимающее 26_{10} правых разрядов (т. е. остальные разряды — нули). Образовать слово b, содержащее единственную единицу — самую правую единицу a. Решение легко получается в три команды, но его можно написать и в две (Ромена Иоффе):

$$2_{\text{ц}}^{26} -, a = R1$$
$$a \wedge R1 = R2$$

Проделать на примерах в двоичной записи.

Задача 5. В условиях предыдущей задачи образовать слово, содержащее единицу против самого правого нуля a и нули в остальных разрядах.

Решение:

$$a +, (0,\ 0,\ 1) = R1$$
$$(\Omega,\ \Omega,\ \Omega) -, a = R2$$
$$R1 \wedge R2 = R3$$

Задача 6. В условиях задачи 4 из каждой группы единиц, стоящих подряд (не разделенных нулями), выделить по одной (правой).

Решение (Г. М. Адельсон-Вельский и В. Л. Арлазаров):

$$(\Omega,\ \Omega,\ \Omega) -, a = R2$$
$$1 - R2 = R3$$
$$R3 +, (0,\ 0,\ 1) = R4$$
$$a \wedge R4 = R1$$

Задача 7. Сочетание k элементов из n (k и n — фиксированные числа, не меньшие, чем 26_{10}) изображается словом a, имеющим k единиц на n правых разрядах (прочие разряды — нули). Сочетания упорядочены в порядке убывания представляющих их слов как *ЦЧ*. По заданному сочетанию получить следующее (без циклов!).

Решение. Программа, пояснения и примеры приведены в табл. 14.1.

Таблица 14.1

Программа		Пример 1	Пример 2	Пример 3
Заданное сочетание a		$X1000$	$X1001$	$X100111$
$a+,(0,0,1)=R1$	гашение хвоста единиц	$X1001$	$X1010$	$X101000$
$a \wedge R1 = b$		$X1000$	$X1000$	$X100000$
$R1-,b=R1$	младший нуль b	00001	00010	0001000
$1^{27}-,b=R2$	младшая единица b	$\overline{X}1000$	$\overline{X}1000$	$\overline{X}100000$
$R2 \wedge b = R2$		01000	01000	0100000
$R2:,(0,0,2)=R2$		00100	00100	0010000
$R2:,R1=R1$		00100	00010	0000010
$b-,R1=c$	результат	$X0100$	$X0110$	$X011110$

В примерах 1, 2, 3 из сочетаний $a = X1000$, $X1001$ и $X100111$ получаются следующие за ними сочетания $c = X0100$, $X0110$ и $X011110$. Здесь X — любая последовательность нулей и единиц, а $\overline{X}$ — обратная ей (и X, и $\overline{X}$ имеют нули в разрядах 34 — 27).

Задача 8. Число a ($0 \leqslant a < 2^{26}$) задано в форме *ЦЧ*. Перевести его в форму *ПЗ*.

Решение:

$$2_n^{26} +, a = R2$$

$$R2 - 2_n^{20} = R3$$

Действительно, 2_n^{26} имеет порядок 27_{10} и мантиссу $^1/_2$ (единицу 27_{10}-го разряда). Поэтому в $R2$ образуется слово, имеющее порядок 27_{10} и мантиссу (целая часть a обозначается $[a]$):

$$^1/_2 + a \cdot 2^{-27} = {}^1/_2 + [a] \cdot 2^{-27}.$$

Это число, как число *ПЗ*, равно

$$2^{27}({}^1/_2 + [a] \cdot 2^{-27}),$$

а число в $R3$ равно $2^{27}({}^1/_2 + [a] \cdot 2^{27}) - 2^{26} = [a]$.

Задача 9. Задано число $0 \leqslant a \leqslant 2^{20}$ в форме *ПЗ*. Разбить его на целую (в $R2$) и дробную (в $R3$) части.

Решение:

$$a + 2^{26} = R4$$

$$R4 - 2^{26} = R2$$

$$a - R2 = R3$$

Задача 10. В условиях предыдущей задачи записать целую часть a в форме *ЦЧ*.

Решение:

$$a + 2^{36} = R2$$

$$(377,\ \Omega,\ \Omega) \wedge R2 = R3$$

Задача 11. Линейная интерполяция. Значения $f_k = f(x_0 + hk)$ находятся в ячейках с адресами $s + k$ $(k = 0, 1, \ldots, n)$. Составить программу, которая по числу x в ячейке α будет доставлять в $R1$ значение $f(x)$ с помощью линейной интерполяции.

Решение. Прежде всего положим

$$f(x) = \begin{cases} f(x_0) & \text{для} \quad x \leqslant x_0, \\ f(x_0 + hn) & \text{для} \quad x \geqslant x_0 + hn. \end{cases}$$

Если $x_0 < x < x_0 + hn$, то x представляется в виде

$$x = x_0 + h(k + \Delta), \text{ где } 0 \leqslant k \leqslant n - 1 \text{ и } 0 \leqslant \Delta < h,$$

причем k и Δ — это целая и дробная части от $(x - x_0) : h$. Тогда

$$f(x) = \Delta f_{k+1} + (1 - \Delta) f_k.$$

Это приводит к программе 14.1. Обратите внимание на получение команды (ρ). Целое число k сдвигается из правого адреса

Программа 14.1

```
        x0 ⩾ x
        f0 → R1; вых
        x ⩾ (x0 + nh)
        fn → R1; вых
    x − x0 = R2
    R2 : h = R2
R2 + 2^26_n = R3
R3 − 2^36_n = R4
    R2 − R4 = Δ
    22 − R3 = (k, 0, 0)
(«s» + 0 = R1) +, (k, 0, 0) = (я + 1)
ρ)  fk + 0 = R1
    1 − Δ = R3
    R2 · R1 = R3
ρ +, (1, 0, 0) = (я + 1)
ρ1) fk+1 + 0 = R1
    Δ · R1 = R1
    R1 + R3 = R1
    выход
```

в левый (на $22_8 = 18_{10}$ разрядов) и к нему прибавляется число s.

Словарь:

$$f_0 \sim s, \quad f_n \sim s + n, \quad \Delta \sim R2.$$

Команды ρ и $\rho 1$ не перфорируются.

Задача 12. В условиях задачи 8 сосчитать число единиц в слове a.

14.2. Библиотечные программы готовятся тщательно и должны удовлетворять ряду условий, которые их сильно усложняют. Программист редко разбирает эти программы, и мы ограничимся тем, что приведем три программы. Программисту достаточно познакомиться с одной из них (по выбору).

Перевод «10 → 2». В ячейке α находится слово

$$uv\bar{p}\bar{a}_1\bar{a}_2\bar{a}_3\bar{a}_4\bar{a}_5\bar{a}_6\bar{a}_7,$$

изображающее десятичное число

$$a = (u) \cdot 10^{(v)p} \cdot 0, \; a_1a_2a_3a_4a_5a_6a_7,$$

где u и v — знаки ($+$ или $-$), а буквы p, a_1, ..., a_7 — десятичные цифры. Буквы с чертой — четырехразрядные двоичные изображения соответствующих десятичных цифр. Таким образом, в ячейке α знаки занимают 2 левых разряда, а цифры $\bar{p}$, $\bar{a}_1$, ..., $\bar{a}_7$ занимают остальные $8 \cdot 4 = 32$ разряда. Требуется перевести число a в двоичную систему в форму *ПЗ* с возможно меньшей ошибкой.

Важное условие: перевести *точно* все, что можно. Это значит, что не слишком большие целые числа и дроби типа 0,5; 0,25; 0,75; ... нужно перевести в двоичную систему *точно*. Ради этого стоит пойти на снижение точности перевода других чисел.

Число a, очевидно, равно

$$a = (u) \cdot 10^{(v)p-i}(a_1 10^{i-1} + a_2 \cdot 10^{i-2} + \ldots + a_i),$$

где a_i — последняя, не равная нулю цифра мантиссы. Мы сначала сосчитаем скобку как Целое Число (что можно сделать точно), а затем уже помножим или поделим результат на десять нужное число раз.

При разборе программы 14.2 рекомендуется проиграть ее на нескольких числах, например

$$10^4 \cdot 0{,}128;\ 10^0 \cdot 0{,}125.$$

Команды $1 \div 6$ переводят порядок p с его знаком в форму *ЦЧ*. Команды 6 и 7 подготавливают ячейку $R1$ (заносят в нее первую цифру) и ячейку $R2$. Цикл $9 \div 15$ вычисляет по Горнеру скобку как *ЦЧ*. Сравнение 15 блокирует случай, когда последняя, отличная от нуля цифра мантиссы равна 8 (что порождает в $R2$ число «минус нуль»). Команды 16 и 17 переводят мантиссу a в форму *ПЗ*. Цикл 21—23 множит и делит нужное число раз на десять. Что именно нужно делать — множить или делить — задается командами 8, 19, 20. Команды 24 и 25 приписывают результату нужный знак.

Программа 14.2

1) $2 \leftarrow \alpha = R4$
2) $36 \rightarrow \alpha = R4$
3) $1 \leftarrow \alpha = R2$
4) $R2 \wedge (-0) = R2$
5) $R4 \vee R2 = R4$
6) $6 \leftarrow \alpha = R2$
7) $36 \rightarrow R2 = R1$
8) $(R1 : 10_n = R1) - \rho;\ 12$
9) $R1 \cdot , (0, 0, 12) = R1$
10) $36 \rightarrow R2 = R3$
11) $R1 + , R3 = R1$
12) $R4 - , (0, 0, 1) = R4$
13) $4 \leftarrow R2 = R2$
14) $|0| \geqslant |R2|;\ 9$
15) $R2 \geqslant 0;\ 9$
16) $2_n^{26} + , R1 = R1$
17) $R1 - 2_n^{28} = R1$
18) $R4 \wedge (+\,00\,000000\,\Omega) = R2$
19) $R4 \geqslant 0;\ 23$
20) $(R1 \cdot \Omega_n = R1) \rightarrow \rho;\ 23$
21, ρ) $R1 \cdot$ (или:) $10_n = R1$
22) $R2 - , (0, 0, 1) = R2$
23) $0 \geqslant R2;\ 21$
24) $0 \wedge \alpha = R2$
25) $R1 \vee R2 = R1$
26) $0 \rightarrow 0;\ \Omega$
константы
27) $R1 \cdot 10_n = R1$
28) $R1 : 10_n = R1$
29) $(0, 0\,012)$
30) $2_n^{26} \approx (+\,33\,400\,000\,000)$

Перевод «$2 \rightarrow 10$» производит операцию, обратную переводу «$10 \rightarrow 2$». Переводимое число a множат или делят на десять столько раз (k), чтобы результат (b) попал в интервал $1 \leqslant |b| < 10$. Тогда $1 - k$ будет десятичным порядком числа a, а число b, будучи переведено в десятичную систему, даст десятичную мантиссу числа a. Для перевода b

в десятичную систему воспользуемся тем, что b_1 — число целых единиц числа b — есть первая цифра искомой мантиссы. Число b_2 целых единиц числа $(b - b_1)\cdot 10$ является второй цифрой этой мантиссы и т. д. Программа 14.3 перевода «2 — 10», в принципе простая, усложняется тем, что машина выполняет действия приближенно, а числа, близкие к круглым, надо перевести круглыми числами. Знаком I_{27-34} обозначено слово, содержащее единицы в разрядах с 27 по 34 и нули в остальных разрядах.

Программа 14.3

1) $|0| \geqslant \alpha$
2) $0 \to R1$; 23
3) $(+77, \Omega, \Omega, \Omega) \wedge \alpha = R2$
4) $1_n \to R1$
5) $R2 \cdot 10_n = R2$
6) $R1 : 2_n = R1$
7) $R2 \geqslant 1_n$
8) $(0 \to R3 = R3) \to 16$
9) $R2 : 10_n = R2$
10) $R1 \cdot 2_n = R1$
11) $(10 - \varepsilon_1)_n \geqslant R2$
12) $(16 + \varepsilon_2)_n + R2 = R2$
13) $1 \leftarrow R2 = R2$
14) $R2 \wedge I_{27-34} = R3$
15) $R2 -,\ R3 = R2$
16) $i \to R3 = R3$
17) $Ri +,\ R3 = R1$
18) $16 +-,\ (4, 0, 0) = 16$
19) $R2 \cdot (0, 0, 12) = R2$
20) $0 \geqslant R2$; 14
21) $(-0) \wedge \alpha = R2$
22) $R1 \vee R2 = R1$
23) $nr\ R1 \cdot \Omega$

константы

24) $0 \to R3 = R3$
25) $(10 - \varepsilon_1)_n = (+\ 04\ 477\ 777\ 774)$
26) $(16 + \varepsilon_2)_n = (+\ 05\ 400\ 000\ 001)$

Быстрая программа $\sqrt{a}$. Пусть

$$a = 2^k \cdot b,$$

где

$$1/2 \leqslant b < 1 \text{ и } |k| < 31.$$

Положим

$$x_0 = 2^\alpha,$$

где α — целая часть $k/2$,

$$x_{n+1} = [(a/x_n) + x_n]/2.$$

Тогда с точностью до единицы последнего разряда мантиссы MM будет $\sqrt{a} \approx x_4$. Для экономии действий удобно

считать двойные итерации:

$$x_{n+2} = \frac{a}{\frac{a}{x_n} + x_n} + \frac{\frac{a}{x_n} + x_n}{4}.$$

Это приводит к программе 14.4 (аргумент — в ячейке α, результат — в $R1$).

Программа 14.4

```
        0 ⩾ α                          α : R2 = R1
        0 → R1; выx                    R2 : 4n = R2
        α ∧ (+36, 0, 0, 0) = R1   x2)  R1 + R2 = R1
        1 → R1 = R1                    α : R1 = R2
        α ∧ (+40, 0, 0, 0) = R2        R2 + R1 = R2
        R1 ∨ R2 = R1                   α : R2 = R1
   x0)  R1 +, (Ω, Ω, Ω) = R1           R2 : 4n = R2
          α : R1 = R2              x4)  R1 + R1 = R1
        R2 + R1 = R2                   выход
```

e^α, $\operatorname{ch}\alpha$ и $\operatorname{sh}\alpha$, $\cos\alpha$ и $\sin\alpha$ — эти пять функций вычисляются с помощью одной бесконечной цепной дроби

$$\lambda = 2 + \cfrac{a^2}{6 + \cfrac{a^2}{10 + \cfrac{a^2}{14 + \ddots}}},$$

где

$$a^2 = \begin{cases} +\alpha^2 & \text{для } e^\alpha,\ \operatorname{ch}\alpha,\ \operatorname{sh}\alpha, \\ -\alpha^2 & \text{для } \cos\alpha,\ \sin\alpha. \end{cases}$$

Далее имеем

$$\begin{matrix}\operatorname{th}\\ \operatorname{tg}\end{matrix}\left(\frac{\alpha}{2}\right) = \frac{\alpha}{\lambda},$$

откуда

$$e^\alpha = 1 + \frac{2\alpha}{\lambda - \alpha},$$

$$\begin{matrix}\operatorname{ch}\\ \cos\end{matrix}(\alpha) = \frac{\lambda^2 + a^2}{\lambda^2 - a^2}, \qquad \begin{matrix}\operatorname{sh}\\ \sin\end{matrix}(\alpha) = \frac{2\alpha\lambda}{\lambda^2 - a^2}.$$

Если $|\alpha|<1$, то вычисления идут по приведенным формулам, а бесконечная дробь λ обрывается на коэффициенте 14, который (для увеличения точности) заменяется на

$$14-\cfrac{(\pi/4)^2}{18-\cfrac{(\pi/4)^2}{22-\ldots}}\approx 13{,}964\ \ldots$$

Если $|\alpha|\geqslant 1$, то $\alpha=2^k\alpha_*$, где $|\alpha_*|<1$. Мантисса α_* легко получается из числа α, все функции вычисляются для аргумента α_*, а окончательный ответ получается k-кратным применением формул удвоения

$$e^{2x}=(e^x)^2,$$
$$\operatorname{ch} 2x=\operatorname{ch}^2 x+\operatorname{sh}^2 x,\qquad \operatorname{sh} 2x=2\operatorname{sh} x\cdot\operatorname{ch} x,$$
$$\cos 2x=\cos^2 x-\sin^2 x,\quad \sin 2x=2\sin x\cdot\cos x.$$

Замена числа 14 на 13,964... выполняется с таким расчетом, чтобы $\sin(\pi/2)$ равнялся нулю.

З а д а ч а. Составьте одну программу с тремя входами для вычисления функций показательной, гиперболических и тригонометрических. Гиперболические и тригонометрические функции надо считать парами (косинус и синус), которые понадобятся в формулах удвоения.

15. О РАЗЛИЧНЫХ МАШИНАХ

Эта глава должна облегчить читателю переход к программированию на тех машинах, на которых ему придется работать.

Каждая машина, разумеется, отличается от машины *ММ*, на которой мы учились программированию. Отметим основные особенности эксплуатируемых машин.

15.1. Число адресов. Кроме трехадресных машин, бывают еще одноадресные, двухадресные и даже полутора- и двухсполовинойадресные.

15.1.1. В команде одноадресной машины указывается код операции и адрес только одной ячейки. Чтобы на машине можно было работать, она должна иметь одну машинно-выделенную ячейку S (называемую сумматором). Сумматор неявно подразумевается в каждой команде, а как он используется, указывает код операции. Из-за этого число команд в одноадресной машине бывает очень велико — до нескольких сотен.

В одноадресной машине результат операции всегда заносится в S, а команда сложения выглядит, например, так:

$$+a \qquad (S+a=S),$$

что означает: добавить к содержимому S содержимое ячейки a и оставить результат в S. Команд вычитания обычно имеется две:

$$-a \qquad (S-a=S),$$
$$a- \qquad (a-S=S).$$

Кроме того, имеются две команды — занесения в сумматор из ячейки a и записи из сумматора в ячейку a:

$$a\downarrow \qquad (a+0=S)$$
$$\downarrow a \qquad (S+0=a)$$

Команда сложения машины *ММ*

$$a + b = c$$

заменяется программой из трех одноадресных команд

1) $a|$
2) $+ b$
3) $|c$

Но если нужно подсчитать «длинную» сумму

$$a_1 + a_2 + \ldots + a_n,$$

то увеличение числа команд не столь заметно.

15.1.2. В командах двухадресной машины тоже подразумевается сумматор S. Два адреса команды могут указывать на адреса двух аргументов, или адреса одного из аргументов (вторым будет S) и результата. Могут быть команды типа $a + b = a$, удобные для переадресации и счета на рабочих ячейках.

15.1.3. В полутора(двухсполовиной)адресной машине адресов два (соответственно три), но один из адресов короткий. В коротком адресе могут фигурировать только первые ячейки памяти (начинающиеся с нескольких нулей). Их используют в качестве рабочих.

Машины с малым числом адресов проще в изготовлении и кажутся быстроходнее (на одну команду приходится меньше обращений к памяти). Их недостатки (большое число команд в программе) обычно снимаются «эквилибристикой» программиста. Но программы при этом пишутся труднее.

15.2. Сравнения такого типа, как у машины *ММ*,

$$a \geqslant b;\ c$$

во многих машинах отсутствуют. Вместо этого при каждом вычитании в специальном устройстве ω запоминается знак $+$ или $-$ разности. Кроме того, имеется двухадресная команда

$$+a;\ b-$$

передающая управление a при $\omega = +$ и b при $\omega = -$. Тогда команда сравнения $a \geqslant b;\ c$ машины *ММ* заменится двумя командами:

$k)$ $a - b = 0$

$k+1)$ $+(k+2);\ c-$

В одноадресных машинах для выполнения сравнения вводится команда

$$+;\ c-$$

передающая управление c при $\omega = -$. Тогда сравнение $a \geqslant b;\ c$, заменяется тремя командами:

$$\begin{array}{rl} a\downarrow & (a+0=S) \\ -b & (S-b=S) \\ +;\ c- & (\omega \geqslant 0;\quad c) \end{array}$$

Особой осторожности требуют машины, у которых знак разности, равной нулю, зависит от знаков уменьшаемого и вычитаемого.

15.3. Регистр переадресации (*РП*) заметно повышает скорость машины и укорачивает программу. *РП* — это короткая (длиной в один адрес) машинно-выделенная ячейка, служащая для переадресации выполняемой команды без изменения содержимого ячейки, в которой команда хранится и без дополнительных затрат времени. В трехадресной машине с *РП* в каждой команде отводится три разряда, показывающих, к каким адресам (левому, среднему, правому) надлежит прибавить *РП* перед выполнением команды. В программе эти адреса отмечают цветными звездочками.

Кроме того, имеется команда типа

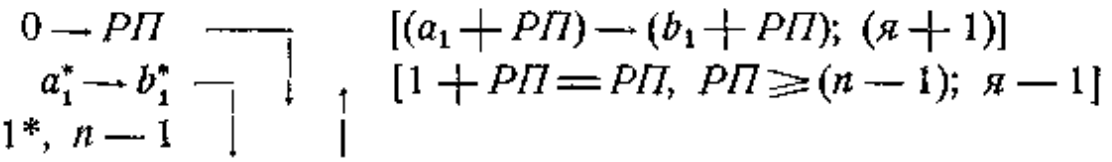
$$k)\ h^*;\ n;\ c\ \text{(то есть } h+PA=PП,\ \ PП \geqslant n;\ c),$$

которая означает: «проверить $PП \geqslant n$, если да — перейти к следующей команде, если нет — добавить к *РП* число h и передать управление команде c». Числа n и h — это короткие числа, написанные в адресах самой команды k.

Цикл переноса $a_k \rightarrow b_k$ от $k=1$ до $k=n$ занимает в машине *ММ* шесть ячеек:

$$\begin{array}{rl} & \rho;\ \rho \\ & a_1 \rightarrow b_1 \\ \text{эт)} & a_n \rightarrow b_n \\ & \rho+,\ (1,\ 1,\ 0)=\rho \\ \rho) & a_k \rightarrow b_k \\ & |\rho| \geqslant |\text{эт}| \end{array}$$

В трехадресной машине с *РП* он пишется в три команды:



и выполняется на 1/3 быстрее. Заметим, что машина может иметь не один индексный регистр, а несколько.

15.4. Знаки команд. Чтобы сделать команды положительными, мы истратили на знак «+» по одному разряду в каждой ячейке, занятой командой. В машинах этого обычно не делают и выбирают один из двух путей.

15.4.1. Разряд знака включают в число разрядов кода команды. От этого команды становятся положительными и отрицательными в зависимости от кода операции. При переадресации нужно учитывать знак команды: чтобы увеличить на единицу правый адрес отрицательной команды, нужно не прибавлять к ней число (0, 0, 1), а вычитать его. Знаки команд стараются сделать запоминающимися: сложение и умножение — положительными, а вычитание и деление — отрицательными и т. д. Тем не менее на первых порах это порождает ошибки в программах.

15.4.2. Другой путь экономии знакового разряда в команде состоит в том, чтобы отказаться от него и в Целых Числах. Тогда все Целые Числа понимаются как неотрицательные числа от 0 до $2^{3n} - 1$, где n — число разрядов в адресе (имеется в виду трехадресная машина). Арифметические действия над этими числами производятся «по модулю 2^{3n}». Это значит, что сначала как бы выполняется обычное действие, а затем к результату прибавляется (или вычитается из него) число 2^{3n} до тех пор, пока результат не попадет в интервал от 0 до $2^{3n} - 1$.

15.5. *Условный порядок.* В машине *ММ* порядок (числа с плавающей запятой) изображается знаком порядка и его величиной. В других машинах порядок p изображается «условным порядком» $P = c + p$. Константу c берут такой, что условный порядок (т. е. P) уже не отрицателен — изменяется от 0 до максимально допустимой величины. Реже встречаются машины, где и мантисса изображается подобным образом.

15.6. Целые Числа в некоторых машинах отсутствуют. Вместо них употребляются числа с запятой, фиксированной *перед* старшим разрядом (а не позади младшего, как в машине *ММ*). При операциях $+$, и $-$, это не заметно, но при $\cdot$, и $:$, результат попадает в другие разряды. Впрочем, умножение и деление индексов — операции редкие. Иногда они вовсе отсутствуют в кодах машины.

15.7. Машины бывают снабжены *внешней памятью* — на барабанах и магнитных лентах. Это дает возможность в процессе работы отправлять туда на хранение полученные результаты, когда их набирается очень много — десятки тысяч, сотни тысяч и миллионы. Из внешней памяти числа могут быть в дальнейшем сравнительно быстро введены во внутреннюю память. Все же обращение к внешней памяти выполняется несравненно медленнее, чем обращение к внутренней памяти машины. При отладке на барабане помещают копию программы, как это описано в п. 12.8.

15.8. В некоторых машинах имеется *долговременное запоминающее устройство* (*ДЗУ*). Это несколько ячеек (от 1 до 10), которые представлены на пульте рядами кнопок. Кнопок в ряде столько, сколько разрядов имеет ячейка машины. Благодаря кнопкам слово в ячейку *ДЗУ* набирается быстрее, чем в ячейку внутренней памяти. Главное же в том, что это слово остается на пульте перед глазами программиста. *ДЗУ* облегчает работу за пультом, особенно при отладке.

15.9. В некоторых машинах *нет действий с плавающей запятой*. Для них приходится пользоваться подпрограммами. Несмотря на наличие специальных сдвигов, подпрограммы оказываются длиннее, чем можно предположить. Так, например, подпрограмма сложения занимает до двадцати ячеек. Другой путь — в регулярной проверке «размеров» чисел и масштабировании. Этот путь нагружает самое узкое место — память программиста.

В наше время некого удивлять тем, что «машина считает». Оцениваются машины по их производительности, зависящей в первую очередь от удобства программирования. Поэтому машин без *ПЗ* (равно как и машин с неудобным вводом и медленным выводом) становится все меньше.

Без *ПЗ* строят еще специализированные машины для управления станками и другими агрегатами. Строят «в радостной надежде», что машина будет работать с числами из заданного диапазона, по программе, которую отладят «раз навсегда». На самом же деле именно производственный процесс дает входную информацию, подверженную колебаниям, зависящим не от программиста. В машине без *ПЗ* эти колебания приходится оценивать все время: и на входе, и в промежуточных выкладках. Легкость переделки программ составляет основное преимущество цифровых электронных управляющих машин перед старыми автоматами. Поэтому на производстве программу постоянно совершенствуют. И решающую роль приобретает неудобство программирования и длительность подбора масштабов на машинах без *ПЗ*.

ПОСЛЕСЛОВИЕ

Физическая работа человека стоит 30 копеек в месяц (по ценам МОГЭСа *)). Зарплата превосходит эту сумму в сотни раз. Значит, деньги человеку платят не за физическую работу, а за нечто иное, и увеличение энергетической мощи человечества слабо разгружает его от повседневной работы. Задача современной науки — это постройка машин, выполняющих вместо человека неэнергетическую часть его труда.

При появлении *ЦЭМ* сразу же было отмечено, что они могут использоваться в этом направлении. Действительно, *ЦЭМ* имеет память для хранения информации, как поступившей извне, так и полученной в процессе работы. Машина может сравнивать различные пути решения задачи, оценивать их эффективность. Случайные числа дают возможность испытывать не предусмотренные пути к решению задачи; эти пути могут, разумеется, быть как вовсе произвольными, так и ограниченными рамками, которые сужаются в результате прошлых неудач. Память современной машины исчисляется десятками тысяч слов, а скорость — сотнями тысяч операций в секунду. Отпали скороспелые утверждения об отсутствии у машин свободы воли, склонностей, ошибок, реакций самозащиты, способности делать то, что не предусмотрено человеком. Сложные игровые и узнающие программы уже обладают этими качествами. Установлена возможность построения самовоспроизводящих машин. В лекциях по кибернетике А. С. Кронрод отметил, что человек, думающий одними голыми мозгами, будет представлять такой же спортивно-эстрадный номер (а не работу на производстве), как и атлет, поднимающий штангу без помощи подъемного крана. Нет

*) Действительно. В месяце 200 рабочих часов. Мощность человека — 1/20 лошадиной силы. 1 лошадиная сила равна 0,76 киловатт. За 1 киловатт-час мы платим 4 копейки.

таких *элементов* творческой деятельности, которых не могли бы выполнять *ЦЭМ*.

Поэтому некоторые ученые нашли возможности *ЦЭМ* безграничными и стали говорить о замене и человеческого труда, и самих людей машинами. Эту антинаучную концепцию очень точно описал (приводим с сокращениями)

*Н. Коржавин**)

НАУКА ИЛИ УТОПИЯ

Сколько было распято, убито!
Как бурлила мысль, томясь во мгле,
Чтоб сегодня просто и открыто
Роботы шагали по земле.
Пусть они пока что неуклюжи,
Как и мы в младенческие дни,
Но уже теперь мы в чем-то хуже
И слабее чем-то, чем они.
Человек ведь многого не может,
Но для беспокойства нет причин.
Превосходство?... Нас ведь не тревожат
Превосходства всех других машин.
Будут крепнуть. Смогут все с годами,
Нам служа, не мучась, не любя,
И воссоздаваться будут сами,
Даже совершенствовать себя.
А на что мы сможем им сгодиться?
Суета, — а пользы никакой...
И они от нас освободиться
Захотят — по логике самой.
И пойдут. И — мертвые — раздавят
С помощью науки нас, живых.
Что мы сможем противопоставить
Точной мысли, воплощенной в них?
Пусть их мир не будет нами признан.
Мы исчезнем в той кромешной мгле...
... И пойдет развитие без жизни
На видавшей всякое земле.
Смолкнут споры... Наша неуклюжесть
Сгинет с нами вмиг в чаду огня:

---

*) «Литературная газета» от 2 июля 1960 г.

Ни таких, как у тебя, замужеств,
Ни таких женитьб, как у меня.
Все быстрей они плодиться станут...
Что ж такого?... Быта колея!
Но настанет день — и не достанет
На земле для этого сырья.
И упрутся роботы натужно,
Не сходя упрямо с колеи.
Будто б им на самом деле нужно
Создавать подобия свои.
Словно есть душа в железном теле...
Словно впрямь доступна неживым
Тяга к счастью... Словно в самом деле
Их существованье нужно им.
... Без приятных чувств и чувств печальных
Вновь в одну из тех густых минут
Ближние сближаться против дальних
Волей древней логики начнут.
Вряд ли целым выйдет кто из боя,
Отодвинув царство вечной тьмы ...
Только нам грустить о том не стоит —
Это будут роботы — не мы!...

Кончается стихотворение бодрым «Быть людьми машинам не дано». Но содержание этой «данности», отличающей, по мнению поэта, машину от человека, уже произнесено: «словно есть душа в железном теле». И дискутировать с поэтом остается лишь на антирелигиозную тему. Еще лаконичнее И. А. Полетаев, отметивший, что по этой концепции только вера в бессмертие души мешает признать возможной сознательную жизнедеятельность машин.

Вслед за такими философскими трактатами должны были бы появиться «думающие» программы. Но тут оказалось, что легко моделируются лишь элементы творческой деятельности, а сколько-нибудь законченные творческие задачи требуют громадного труда и пока что решены только в исключительных случаях.

Основной результат попыток создания думающих программ оказался негативным — мы узнали, как мало можно сделать механически, *как рано* (в сколь простых задачах) ***начинается творчество.***

Рассмотрим, например, чтение. Если буквы стандартного шрифта поочередно располагать в рамке телевизионного устройства, то узнавать их будет легко. Но располагать придется очень точно — буквы «з» и «э» отличаются лишь одной точкой! Потребуется сложнейшая следящая система. Ясно, что человек узнает буквы *как-то проще*. Но сформулировать, что такое «характерные особенности букв» (и как их найти!), мы не умеем. В результате по сей день машины не перепечатали без помощи человека ни одной страницы книги.

Как мало мы знаем о своем собственном механизме узнавания, показывают наши научные определения. Определение льва начинается с того, что у этого животного убираются когти. Кто воспользовался таким определением и остался в живых? Но иного определения в биологии нет. Зато там есть описания, картинки и образцы животных и растений. Но ими-то как раз и нельзя пользоваться механически — второго точно такого же листа, как на образце, во всем лесу не сыскать. Чтобы узнавать — нужно соображать!

Возьмем другой пример. Пока на шахматной доске много фигур, хорошая программа соревнуется с человеком, ибо игра ведется по осмысленным принципам, которые сформулированы и заложены в программу. Но вот наступил эндшпиль. Появилась возможность рассчитать все до конца. И человек делает это, в отличие от всех существующих программ. Порукой тому, что человек рассмотрел все варианты, служит то, что он умеет доказать верность своего решения. Но варианты он рассматривал не индивидуально. В процессе решения одни варианты объединялись, другие отбрасывались, потому что имелись лучшие. Чтобы решать игровые задачи, программа должна уметь думать. Конечность числа вариантов обманчива, их больше, чем песчинок на земле, и механически перебирать все варианты безнадежно.

Рассмотрим более сложный пример. Пусть рабочий берет из ящика беспорядочно набросанные заготовки коленчатых валов и обрабатывает их на станке. Попробуем заменить рабочего автоматом в предположении, что все исполнительные механизмы у нас уже есть и остается только выдавать им руководящие указания. И тут оказывается, что исключительно трудно захватить заготовку и вынуть ее из ящика. Прежде всего нужно идентифицировать различные точки телевизионного изображения, чтобы понять, что они относятся к одной

заготовке. Надо уметь узнавать заготовку, показанную в различных положениях и частично заслоненную другими деталями. Далее нужно решать сложную задачу — какую заготовку взять первой, по какой траектории ее вынимать и какие заготовки и как при этом подвинуть.

Вынимание детали, несомненно, является творческой задачей, хотя мы этого и не подозревали до того, как попытались запрограммировать этот процесс.

В настоящее время программ для такой работы нет. В начале сложнейшей автоматической линии заготовки подает рабочий (если только они не имеют слишком простой конфигурации). Экскурсанты замечают, что «такую простую» операцию забыли автоматизировать, а экскурсоводы обещают, что она будет автоматизирована к следующему разу.

В уже цитированных лекциях А. С. Кронрод заметил, что *ЦЭМ* эффективны там, где у человека нет интуиции и им не приходится соревноваться с поразительным механизмом, созданным самой природой. В арифметических вычислениях наша интуиция не участвует. Она нам не подсказывает, будет ли четной третья цифра произведения 184×275. Уже З. Фрейд отметил, что во сне не перемножают двухзначных чисел, хотя это и проще, чем решать задачу на построение, что доводилось многим.

Поэтому в настоящее время развиваются два направления думающих программ:

1. Исследуются области, где человек не имеет природной интуиции (вычисления, диагностика и т. д.), и в них применяются *ЦЭМ*. Здесь могут сравнительно легко получаться поразительные результаты.

2. Отыскиваются методы решения отдельных задач, решаемых человеком интуитивно. При этом не задаются целью скопировать методы человека (если бы мы только копировали методы живой природы, то не имели бы безногих автомобилей). Но если в трудных случаях удастся понять, «как это делает человек», то успех будет крупным.

Вообще же при составлении узнающих и игровых программ решающую роль приобретает самоусовершенствование программы, в начале достаточно плохой. В сложных случаях оно проходит по путям, которых сам автор программы уже и не знает.