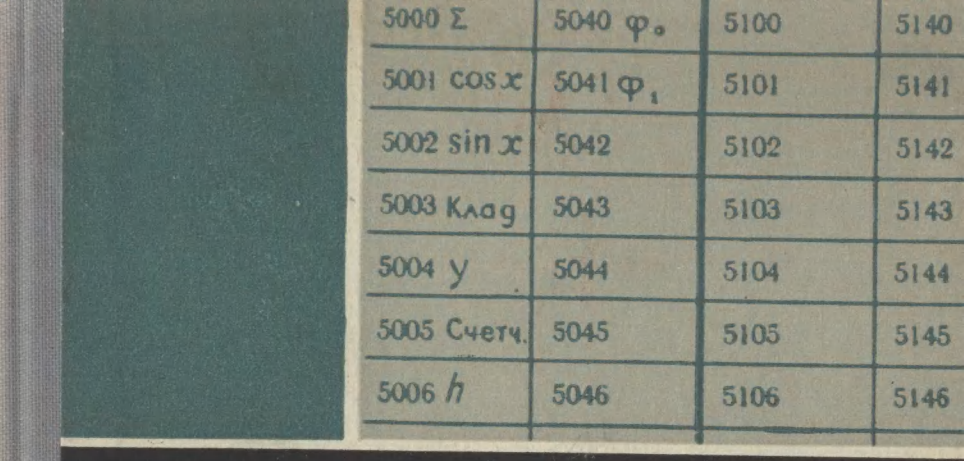
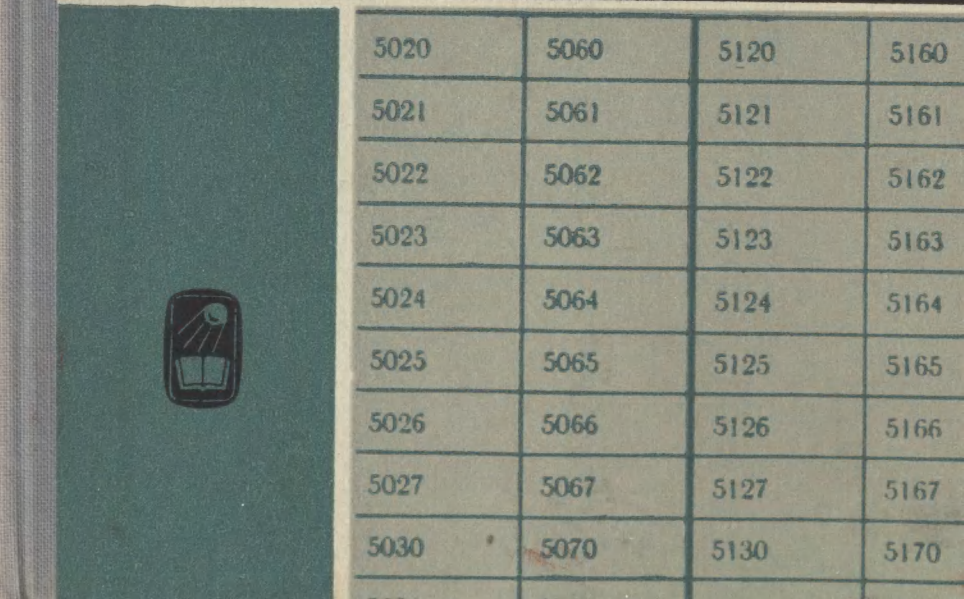


Р. С. ГУТЕР
В. Л. АРЛАЗАРОВ
А. В. УСКОВ

Практика программирования



Р. С. ГУТЕР, В. Л. АРЛАЗАРОВ, А. В. УСКОВ

ПРАКТИКА ПРОГРАММИРОВАНИЯ

СПРАВОЧНИК

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

518(03)
Г 97
УДК 681.142.2 (083.1)

СОДЕРЖАНИЕ

Глава V

Отладка программы

ПРЕДИСЛОВИЕ

Настоящая книга предназначается в качестве справочника для программистов, работающих на трехадресных электронных счетных машинах.

Книг, посвященных программированию для электронных счетных машин, уже немало, и их число все время возрастает. Однако большинство из них посвящено главным образом описанию устройства электронных счетных машин и построению теории программирования как формальной математической теории. С другой стороны, имеется некоторое число книг, в которых, наоборот, рассматривается программирование для одной конкретной машины, точнее, система кодов этой машины и сравнительно небольшое число примеров программ.

В отличие и от тех и от других наш справочник ставит своей целью подробный разбор практических приемов программирования для трехадресных электронных счетных машин. Эта его направленность подчеркивается названием и обусловлена тем, что он написан программистами, в течение ряда лет непосредственно работавшими (и работающими) на машине.

Мы считали необходимым сделать книгу доступной для начинающих программистов и для лиц, впервые знакомящихся с программированием. С этой целью была написана вводная первая глава. По этой же причине изложение материала в различных главах носит различный характер.

Первая глава рассчитана на впервые знакомящихся с программированием. Она написана сжато и конспективно и содержит первоначальные сведения об электронных счетных машинах, элементарных операциях,

программе и командах. Предполагается только, что читатель знаком с различными системами счисления, в частности с двоичной и восьмеричной. Вторая глава содержит основные сведения о программировании. Она будет интересна и имеющим некоторый опыт программирования. Остальные главы рассчитаны уже на более опытного читателя.

Следует иметь в виду, что книга ни в какой степени не является ни учебником, ни справочником по численным методам. Все приведенные в ней программы имеют чисто иллюстративный характер. Метод решения в каждом случае выбирался таким образом, чтобы удобно было иллюстрировать те или иные приемы программирования.

Особое место в книге занимает четвертая глава. Она носит чисто справочный характер и содержит описание конкретной библиотеки стандартных программ и общих принципов построения такой библиотеки. По нашему мнению, применение библиотеки стандартных программ является наиболее приемлемой практически формой автоматизации программирования (по крайней мере, в настоящее время).

Много внимания в книге уделено системе сервисных (обслуживающих) программ в составе библиотеки. Идея организации такой системы и большинство соответствующих программ принадлежат А. С. Кронроду.

Наиболее существенной особенностью нашей книги является систематическое использование способа записи программ, называемого «программированием в содержательных обозначениях». Этот способ был предложен А. Л. Брудно в 1953 г.; его характерными чертами являются естественность и простота изучения и использования.

Чужая программа, написанная в содержательных обозначениях, читается так же просто, как и чужая статья или книга по математике. Запись программы в содержательных обозначениях естественна для всякого математика и вычислителя. А. Л. Брудно довольно долго полагал, что такая запись применяется повсеместно, тем более, что в ряде известных ему организаций именно так и происходило. Только тогда, когда выяснилось, что про-

граммирование в содержательных обозначениях применяется лишь ограниченным кругом программистов, заметка о нем была опубликована А. Л. Брудно в Докладах Академии наук (см. ДАН СССР, 1964, 154, вып. 2, стр. 279—281).

Мы не мыслим себе другой формы записи программы и надеемся, что с помощью настоящего справочника сумеем убедить, по крайней мере, большинство программистов в ее преимуществах. Если же нам это не удастся, то пусть читатель отнесет это не на счет программирования в содержательных обозначениях, а на счет литературной деятельности авторов.

Другой существенной особенностью книги является несколько отличный от обычного аспект применения принципов блочного программирования. Написание блок-программы взамен рисования громоздких блок-схем дает возможность разбивать программу на блоки в самом процессе программирования. Это позволяет обойтись без предварительного составления логической схемы программы и дает возможность по существу объединить процесс программирования с процессом отыскания алгоритма решения задачи. Таким образом, программа превращается в одну из форм записи алгоритма численного решения математической задачи.

Полезно отметить, что само понятие блока является более широким и гибким и содержит меньше ограничений, чем понятие оператора; поэтому операторные схемы программы тоже не являются уже необходимыми.

Описываемая в нашей книге система программирования, а также организация отладки и всей работы на машине сложились и прошли длительную практическую проверку в математической лаборатории Института теоретической и экспериментальной физики.

В работе над книгой мы имели возможность пользоваться советами многих наших товарищей. Г. М. Адельсон-Вельский принимал участие в составлении плана книги и написании второй главы. А. С. Кронрод и Е. М. Ландис участвовали в обсуждении плана книги. А. Л. Брудно, Ф. М. Филлер, П. Т. Резниковский и И. М. Ландис внимательно ознакомились с рукописью и дали большое

число различных советов. Всем им мы выражаем свою глубокую признательность.

При описании библиотеки стандартных программ мы широко пользовались библиотекой стандартных программ ИТЭФ. Многие из этих программ написаны не нами. Их авторами являются Г. М. Адельсон-Вельский, А. С. Кронрод, Г. С. Разина, М. З. Розенфельд, Л. В. Ильков.

Мы благодарим Т. А. Муратову за помощь в кодировке и отладке программ, приводившихся в качестве примеров, Т. Е. Залетову за помощь при работе над рукописью, а также З. Н. Залетову и Р. В. Анопа за оформление рукописи и подготовку ее к печати.

В. Л. Арлазаров,
Р. С. Гутер,
А. В. Усков

Институт теоретической и экспериментальной физики (ИТЭФ) Госкомитета по использованию атомной энергии

ГЛАВА I
ВВЕДЕНИЕ

§ 1. Основные устройства электронной счетной машины. Фиксированная и плавающая запятая

Программист, работающий на электронной счетной машине, должен знать следующие основные устройства:

1) запоминающее устройство (*память*);
2) арифметическое устройство (*арифметика*);
3) устройство управления (*управление*);
4) устройства ввода и вывода (*ввод* и *вывод*).

Память машины предназначена для запоминания программы работы машины, исходных данных, промежуточных и окончательных результатов. Память состоит из отдельных *ячеек*. Каждая из них содержит определенное, постоянное для данной машины число разрядов. Ячейка памяти может хранить набор из нулей и единиц. Каждый такой набор может быть в зависимости от обстоятельств прочитан как число, как команда или какими-либо другими способами.

Обычно память машины делится на две части: *оперативную* и *внешнюю*. Оперативная память характеризуется малым временем обращения, имеет ограниченный объем. Внешняя память, наоборот, характеризуется гораздо большим объемом, но также и бо́льшим временем обращения. Обращение к внешней памяти происходит, как правило, через оперативную. В дальнейшем, употребляя без дополнений слово *память*; мы будем иметь в виду оперативную память. Каждая ячейка памяти имеет определенный номер, который часто называют ее *адресом*. Адрес ячейки есть двоичное число, количество разрядов которого определяется объемом памяти.

Например, если память машины состоит из $4096=2^{12}$ ячеек, то ячейки имеют адреса от 0 до $2^{12}-1$, так что адрес каждой ячейки есть двенадцатиразрядное двоичное число. Его представляют в виде четырех групп из трех разрядов (*триад*). Триада записывается как восьмеричная цифра, так что адрес ячейки можно при таком объеме памяти рассматривать как четырехзначное восьмеричное число. В дальнейшем мы будем считать адрес именно таким.

Содержимое ячейки памяти хранится в ней до тех пор, пока в эту ячейку не будет записано что-либо новое. В последнем случае перед записью предыдущее содержимое автоматически стирается. Если же происходит выборка (чтение) из ячейки, то ее содержимое не изменяется; поэтому одно и то же содержимое можно вызвать из данной ячейки любое число раз.

Арифметика машины служит для выполнения арифметических и иных операций над числами, которые поступают из памяти. *Управление* вызывает из памяти очередную команду, расшифровывает ее, вызывает из памяти нужные числа и засылает их в арифметическое устройство, а полученный результат отсылает затем в память.

Ввод и *вывод* предназначены для общения с машиной: для ввода в машину программы и исходных данных и вывода из нее полученных результатов.

Для ввода в машину программы и исходных данных используются перфоленты или перфокарты. Устройство ввода оборудовано фотоэлементами (фотоввод) или системой читающих щеток (электромеханический ввод). Вывод осуществляется с помощью печати или перфорации, т. е. пробивки отверстий на ленте или картах. Материал, полученный с выходного перфоратора, может быть затем напечатан или же снова введен в машину.

Материал, полученный из печатающего устройства, принято называть *табулограммой*.

Представление о связи перечисленных устройств дает рис. 1, содержащий упрощенную и приблизительную схему соединения частей (*блок-схему*) электронной счетной машины.

В зависимости от формы представления чисел электронные счетные машины делятся на машины с *фиксированной* и с *плавающей* запятой. В машинах с фиксированной запятой число изображается в обычной естественной форме и положение запятой, отделяющей целую часть числа от дробной, фиксируется в определенном месте. Обычно считается, что запятая фиксирована перед первым разрядом, так что в машине записываются только числа, меньшие единицы по абсолютной величине. Поэтому все исходные и выходные данные, так же как и все промежуточные результаты, должны быть меньшими единицы. Это достигается надлежащим выбором единиц измерения и соответствующими масштабными коэффициентами.

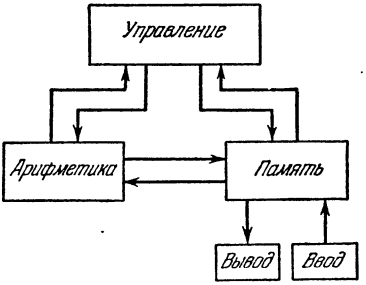


Рис. 1.

В машинах с плавающей запятой число представляется в виде

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

где $|N_0| < 1$. Число N_0 называют *мантиссой*, а p — порядком числа N, 10 означает основание системы счисления. Обычно принимают, что мантисса удовлетворяет неравенству $0{,}1 \leqslant |N_0| < 1$; в этом случае число называют *нормализованным*. При записи числа в ячейке памяти некоторое количество разрядов отводится для изображения порядка, остальные разряды — для мантиссы.

Диапазон чисел, которые могут быть записаны в машине с плавающей запятой, зависит от числа разрядов ячейки памяти. При данном количестве разрядов существует самое большое и самое малое числа, которые могут быть записаны в машине. Числа, по абсолютной величине меньшие самого малого, которое может быть записано, называют *машинным нулем*.

Как правило, электронные счетные машины работают в двоичной системе счисления. Для записи десятичных

чисел применяется так называемая двоично-десятичная система: число представляют в десятичной системе, а каждую его цифру изображают в двоичном виде. Чтобы избежать необходимости в разделительных знаках, для изображения каждой десятичной цифры отводят группу из четырех двоичных разрядов. Такую группу называют *тетрадой.*

§ 2. Программа и команды

Дать точное определение понятий «программа» и «команда» затруднительно. Мы ограничимся поэтому пояснением того, что обычно понимается под этими терминами.

Каждая машина может выполнять некоторое число основных *элементарных операций.* Под *программой* данной задачи, составленной для данной машины, будем понимать сведение процесса решения задачи к выполнению последовательности элементарных операций, имеющихся у этой машины.

Программа представляет собою последовательность *команд,* каждая из которых содержит информацию об одной элементарной операции. Рассмотрим структуру команды, относящейся, например, к арифметической операции.

В команде должно быть указано, какую именно операцию следует выполнять. Для этой цели все элементарные операции машины нумеруются и номер операции называется ее *кодом.*

Кроме кода, показывающего, какую именно операцию нужно выполнить, команда должна содержать также номера ячеек памяти, где хранятся числа, над которыми операция должна быть проделана. Эти номера называют *адресами.* Число адресов в команде у различных машин может быть различно.

Мы ограничимся рассмотрением трехадресных машин, в которых каждая команда содержит, кроме кода операции, три адреса. В команде, соответствующей арифметической операции, эти адреса используются следующим образом: первый и второй адреса означают номера ячеек памяти, из которых должны быть взяты

числа для операции; третий адрес есть номер ячейки памяти, куда следует записать полученный результат. При других операциях использование адресов в команде может быть иным. Подробнее речь об этом будет идти в следующих параграфах (см. §§ 3, 6).

Если в данный момент выполняется команда, лежащая в определенной ячейке памяти машины, то мы будем говорить, что *управление находится в данной ячейке*. Чаще всего такая ячейка обозначается буквой $Я$.

Все команды машины делятся на две большие группы: *команды со стандартным управлением* *) и *команды передачи управления*. Команда со стандартным управлением всегда передает управление следующей ячейке. Иначе говоря, если в ячейке $Я$ находится команда со стандартным управлением, то после ее выполнения будет выполняться команда, взятая из ячейки $Я+1$. Иногда такие команды называют также командами с *принудительным управлением*.

Команды передачи управления предназначены для изменения последовательности выполнения команд. Они делятся, в свою очередь, на команды *безусловной и условной* передачи управления. Команда безусловной передачи управления всегда передает управление в указанную в этой команде ячейку. Команда условной передачи управления может передавать управление в ту или иную ячейку в зависимости от некоторых дополнительных условий.

§ 3. Элементарные операции

Как уже было отмечено в предыдущих параграфах, каждая машина имеет свой набор элементарных операций (*систему команд*). Чтобы познакомить читателя с основными приемами программирования, пригодными для всех трехадресных машин, мы будем рассматривать некоторую условную машину. В качестве элементарных операций этой машины выберем такие, которые имеются у всех или, по крайней мере, у большинства наших трехадресных машин.

*) Более точно—команды со стандартной передачей управления.

Прежде всего следует упомянуть арифметические операции. Сложение, вычитание и умножение являются элементарными операциями на всех машинах. Почти у всех машин имеется также элементарная операция деления *).

Кроме этого на ряде машин имеется также операция вычитания модулей чисел. На некоторых машинах в число элементарных операций включается операция извлечения квадратного корня.

Примем следующий список элементарных арифметических операций условной машины:

сложение,
вычитание,
вычитание модулей чисел,
деление,
умножение.

Все арифметические операции являются операциями со стандартным управлением, т. е. после выполнения арифметической команды из ячейки $Я$ управление переходит в ячейку $Я+1$.

Арифметические операции естественно обозначать следующим образом:

$$\begin{aligned} a + b &= c, \\ a - b &= c, \\ |a| - |b| &= c, \\ a : b &= c, \\ a \cdot b &= c. \end{aligned}$$

При этом запись

$$a+b=c$$

означает команду: число из ячейки, адрес которой обозначен буквой a, сложить с числом из ячейки, адрес которой обозначен буквой b, и сумму записать в

*) Исключение составляет машина «Стрела», в которой вместо операции деления используется специальная программа для нахождения обратной величины, благодаря чему деление можно свести к элементарной операции умножения и нахождению обратной величины на основании тождества $a : b = a \cdot \frac{1}{b}$.

ячейку памяти, адрес которой обозначен буквой c. Аналогичный смысл имеют также и другие приведенные выше записи.

Следующим типом элементарных операций, которые мы рассмотрим, являются *фиксированные действия*. Существуют различные разновидности фиксированных действий. Например, машина М-2 (как и одноадресная машина «Урал-2») может работать как в режиме с плавающей, так и в режиме с фиксированной запятой. Поэтому машина М-2 имеет четыре арифметических действия, рассматривающие полное содержимое ячейки памяти как число с фиксированной запятой; эти действия и называют фиксированными действиями.

С другой разновидностью фиксированных действий мы встречаемся на машинах, работающих лишь с плавающей запятой, например на машинах «Стрела» и БЭСМ-2. Здесь только часть содержимого ячейки рассматривается как число с фиксированной запятой. Так как эти операции применяются главным образом при преобразовании команд (подробнее см. §§ 10, 18), то содержимое ячейки делят на *адресную часть* и *код*.

Мы рассмотрим для условной машины четыре такие операции: сложение и вычитание адресных частей и сложение и вычитание кодов. При сложении и вычитании адресных частей адресные части двух ячеек складываются как числа с фиксированной запятой. Сложение и вычитание происходят по модулю 2^n, где n — число разрядов адресной части. Это означает, что единицы, выходящие за пределы адресной части, не передаются в кодовую часть, а гасятся.

Сложение и вычитание адресных частей мы будем обозначать так:

$$a +, \ b = c,$$
$$a -, \ b = c.$$

Наличие запятой после знаков плюс и минус показывает, что выполняются фиксированные действия. При этом адресные части ячеек с адресами a и b соответственно складываются или вычитаются и записываются на место адресной части в ячейку c. Что касается кодовой части ячейки c, то сюда записывается кодовая

часть ячейки a. Таким образом, фиксированное сложение оказывается некоммутативным: $a+, b \neq b+, a$.

Аналогично сложению и вычитанию адресных частей определяются операции сложения и вычитания кодовых частей ячеек, которые мы будем обозначать

$$a\ ,+b=c,$$
$$a\ ,-b=c.$$

Сложение и вычитание происходят также по модулю 2^m, где m — число разрядов кодовой части ячейки памяти. Адресная часть ячейки в обоих случаях совпадает с адресной частью ячейки a.

При записи в ячейку числа с плавающей запятой обычно адресная часть совпадает с мантиссой, а кодовая — с порядком числа. Поэтому вместо сложения адресных частей часто говорят «*сложение мантисс*», а вместо сложения кодов — «*сложение порядков*». То же относится и к вычитанию.

К фиксированным действиям можно отнести также циклическое сложение. При его выполнении адресные и кодовые части ячеек соответственно складываются как числа с фиксированной запятой, но единицы, выходящие за пределы этих частей, не теряются, а засылаются в соответствующие младшие разряды тех же частей. Циклическое сложение мы будем обозначать символом

$$a \oplus b = c.$$

Кроме арифметических операций каждая электронная счетная машина имеет в числе элементарных также и логические операции. Примем, что рассматриваемая нами условная машина имеет операции логического умножения, логического сложения и сверки (отрицания равнозначности). Эти операции выполняются следующим образом.

При логическом умножении

$$a \wedge b = c$$

происходит поразрядное логическое умножение содержимого ячейки a на содержимое ячейки b по правилам

$$0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0,$$
$$1 \wedge 1 = 1$$

(*конъюнкция*). Результат записывается в ячейку c. Аналогично логическое сложение

$$a \vee b = c$$

выполняется как поразрядное логическое сложение содержимого ячеек a и b, без переноса из разряда в разряд в соответствии с правилами логического сложения (*дизъюнкции*)

$$0 \vee 0 = 0,$$

$$1 \vee 0 = 0 \vee 1 = 1 \vee 1 = 1.$$

Наконец, сверка

$$a \not\vee b = c$$

(отрицание равнозначности) состоит в обычном поразрядном сложении содержимого ячеек a и b без переноса в старший разряд. В результате этого имеем

$$0 \not\vee 0 = 0, \qquad 1 \not\vee 0 = 1,$$

$$0 \not\vee 1 = 1, \qquad 1 \not\vee 1 = 0.$$

Иначе говоря, в тех разрядах, в которых содержимое ячеек a и b одинаково, в ячейке c будет записан нуль, а в тех разрядах, где различно, — единица. При полном совпадении содержимого ячеек a и b в ячейку c будет записан чистый нуль.

Операцию логического сложения часто называют *формированием*, так как с ее помощью можно формировать в ячейке нужное содержимое. Для логического умножения пользуются также терминами «*пересечение*» и «*выделение*». С помощью пересечения можно выделить в ячейке любые нужные разряды, погасив все остальные. Для этой цели достаточно логически умножить содержимое данной ячейки на содержимое другой, в которой стоят единицы в нужных разрядах и нули во всех остальных.

Все рассмотренные выше операции и соответствующие им команды являются командами со стандартным управлением. Это означает, что после выполнения каждой из них управление переходит к следующей ячейке, т. е. выбирается для выполнения команда, лежащая в следующей по номеру ячейке памяти. Изменять порядок выполнения команд можно с помощью специальных

операций *передачи управления.* Команды передачи управления не совершают, вообще говоря, никаких операций над содержимым ячеек памяти, а изменяют лишь порядок выполнения команд. Как указывалось в § 2, существуют команды *безусловной* и *условной* передачи управления.

Команды условной передачи управления обычно строятся на одном из двух различных принципов: передача управления по неравенству или по управляющему сигналу. Первый из этих принципов осуществляется, например, в машинах М-2 и БЭСМ. При этом адреса в команде используются следующим образом. В двух адресах команды (например, в среднем и правом) указываются адреса ячеек, числа из которых сравниваются алгебраически или по абсолютной величине. Если выбранное неравенство справедливо, то управление передается той ячейке, адрес которой указан в оставшемся свободным (например, левом) адресе команды. Если же это неравенство не имеет места, то управление передается следующей ячейке. Такую команду можно условно записать в виде

$$a;\ b < c$$

или

$$a;\ |b| < |c|.$$

Пусть такая команда находится в ячейке $Я$. Тогда, если числа, находящиеся в ячейках памяти b и c, удовлетворяют неравенству, то управление передается в ячейку a; если же это неравенство не имеет места, то управление будет передано в ячейку $Я+1$.

Другой принцип, на котором строятся условные передачи управления, основан на использовании управляющего сигнала. Одновременно с результатом выполняемой операции в машине вырабатывается *управляющий сигнал,* обозначаемый обычно буквой ω, который может принимать значения $\omega=0$ либо $\omega=1$. В зависимости от того, какое значение ω выработалось в результате выполнения предыдущей команды, команда условной передачи управления передает управление в ту или иную ячейку. Условные передачи управления по управляю-

щему сигналу ω применяются, например, в машине «Стрела» и в одноадресных машинах «Урал».

Условная передача управления по неравенству легко может быть заменена передачей по управляющему сигналу ω. Например, при вычитании вырабатывается сигнал $\omega=1$, если результат вычитания отрицателен. Поэтому команду

$$a;\quad b<c$$

можно в машине с передачей управления по сигналу ω заменить двумя командами: командой вычитания

$$b-c=d,$$

после которой следует ставить команду условной передачи управления. Мы примем, что в рассматриваемой нами условной машине применяется принцип условной передачи управления по управляющему сигналу ω.

Для указания адреса ячейки, в которую следует передать управление, достаточно занять в команде один адрес. Два других адреса остаются свободными. Чтобы их использовать, можно, например, совместить с командой передачи управления пересылку содержимого некоторой ячейки в какую-либо другую.

Будем считать, что адрес ячейки, в которую следует передать управление (всегда или в зависимости от выполнения условий), указывается в правом адресе команды и, кроме того, содержимое ячейки, указанной в левом адресе, пересылается в ячейку, адрес которой указан в среднем адресе команды. Таким образом, мы имеем три команды передачи управления, которые можно обозначать следующим образом:

$a\to b;\ c$ — команда безусловной передачи управления,

(0) $a\to b;\ c$ — условная передача управления при $\omega=0$,

(1) $a\to b;\ c$ — условная передача управления при $\omega=1$.

В результате выполнения первой из этих команд управление передается в ячейку c, т. е. следующая команда будет выбрана машиной из ячейки c, а содер-

жимое ячейки a перешлется в ячейку b. Такая же пересылка произойдет при выполнении следующих двух команд, но передача управления в этих командах выполняется иначе.

Именно, команда

$$(0) \quad a \to b; \quad c,$$

находящаяся в ячейке $Я$, передает управление ячейке c в том случае, если в результате выполнения предыдущей операции в машине выработался управляющий сигнал $\omega = 0$. Если же выработался сигнал $\omega = 1$, то управление перейдет не к ячейке c, а к ячейке $Я + 1$. Наоборот, команда

$$(1) \quad a \to b; \quad c$$

передает управление ячейке c при $\omega = 1$ и ячейке $Я + 1$ при $\omega = 0$.

В тех случаях, когда нет нужды в какой-либо пересылке, команды передачи управления можно обозначать короче, опуская левый и средний адрес. Например,

$$\begin{array}{ll} Б & c, \\ (0) & c, \\ (1) & c, \end{array}$$

где буква $Б$ означает безусловную передачу управления. Для команд условной передачи управления пишут также

$$\begin{array}{ll} У_0 & c, \\ У_1 & c; \end{array}$$

буква $У$ означает условную передачу, а индекс — значение сигнала ω, при котором происходит передача ячейке c.

Для пользования командами условной передачи управления необходимо знать, каким образом вырабатывается управляющий сигнал ω.

При сложении, вычитании и вычитании модулей сигнал $\omega = 1$ вырабатывается при отрицательном результате, а сигнал $\omega = 0$ при неотрицательном. При фиксированных действиях сигнал $\omega = 1$ вырабатывается в том случае, если в результате действия появляются или тре-

буются единицы разрядов, выходящих за пределы, соответственно, адресной или кодовой части (например, при вычитании адресных или кодовых частей, если уменьшаемое меньше вычитаемого). При умножении и делении сигнал $\omega=1$ вырабатывается тогда, когда результат превосходит единицу по абсолютной величине. Наконец, во всех логических операциях управляющий сигнал $\omega=1$ вырабатывается в тех случаях, когда результат операции есть чистый нуль. Например, при сверке

$$a \dot{-} b = c$$

будем иметь $\omega = 1$, если a и b совпадают во всех разрядах, и $\omega = 0$, если хотя бы в одном разряде они различны.

Кроме описанных команд управления необходима еще одна команда, называемая *безусловной передачей управления с возвратом*. От уже рассмотренной команды безусловной передачи управления она отличается использованием левого и среднего адресов. Именно, в ячейку b, указанную в среднем адресе, заносится не содержимое ячейки a, а команда передачи управления в ячейку a*). Эту команду мы будем обозначать с двусторонней стрелкой, которая означает, что из ячейки b управление будет передаваться в ячейку a:

$$a \leftrightarrow b; \quad c.$$

К командам управления относится также команда остановки, которую мы будем обозначать словом *стоп*.

Кроме рассмотренных выше элементарных операций у каждой машины имеются еще некоторые специальные операции над командами (описанные выше фиксированные операции также можно отнести к специальным операциям).

Рассмотрим один тип специальных операций: *сдвиги*.

При сдвиге содержимое ячейки памяти может двигаться целиком (*полный сдвиг*), либо может сдвигаться только адресная часть (*сдвиг мантиссы*). Адрес ячейки, содержимое которой надо сдвинуть, указывается во

---

*) На некоторых машинах в ячейку b записывается команда передачи управления в следующую ячейку ($Я + 1$).

втором адресе команды; результат сдвига помещается в ячейку, указанную в третьем адресе. Первый адрес используется для указания направления сдвига (влево или вправо) и числа разрядов, на которые нужно сдвинуть содержимое ячейки. Здесь также есть две возможности.

Число разрядов, на которое следует сдвинуть, может быть прямо написано в первом адресе команды. Такой сдвиг носит название *сдвига по адресу*. В другом случае в первом адресе команды указывается номер ячейки, из которой берется информация о числе разрядов сдвига. Обычно этим числом служит порядок (кодовая часть) содержимого такой ячейки, и поэтому такую операцию называют *сдвигом по порядку*. Более точно эти операции будут описаны в § 6.

Таким образом, получаются четыре различных команды сдвига. Для них мы будем употреблять следующие обозначения. Сдвиг всего содержимого ячейки по адресу мы будем обозначать двойной стрелкой:

$$\overline{a} \Rightarrow b = c.$$

Это означает, что содержимое ячейки b сдвигается на a разрядов и результат заносится в ячейку c. Аналогично фиксированным действиям сдвиг мантиссы будем обозначать

$$\overline{a} \Rightarrow, \; b = c.$$

В этом случае в ячейку c записываются кодовая часть (порядок) содержимого ячейки b без изменения и адресная часть (мантисса), сдвинутая на a разрядов.

Сдвиг по порядку изобразим двойной стрелкой в скобках. Операции сдвигов по порядку будем записывать так:

$$a (\Rightarrow) b = c,$$

$$a (\Rightarrow,) b = c.$$

В первом случае содержимое ячейки b сдвигается на некоторое число разрядов, указанное в кодовой части (в порядке) ячейки a, и результат записывается в ячейку c. Во втором случае сдвигается не все содержимое ячейки b, а только адресная часть (мантисса).

При всех операциях сдвига вырабатывается управляющий сигнал ω, причем $\omega=1$ тогда и только тогда, когда в сдвигаемой части ячейки c оказывается записанным чистый нуль. Во всех остальных случаях будет $\omega=0$.

Остается рассмотреть операции, связанные с обменом информацией между оперативной памятью и внешней памятью.

Ограничимся рассмотрением самого простого случая, когда внешняя память состоит из одного магнитного барабана, имеющего тот же объем, что и оперативная память. Для обращения к внешней памяти достаточно иметь операции, переписывающие содержимое группы ячеек оперативной памяти на барабан, и наоборот. Мы будем называть их *ФБ* и *БФ* *).

Операция

ФБ a b c

переписывает содержимое ячеек оперативной памяти, начиная с a и кончая c, в ячейки барабана, начиная с ячейки b. Аналогично операция

БФ a b c

переносит в оперативную память, в ячейки, начиная с a и кончая c, содержимое ячеек магнитного барабана от b.

Наконец, необходимы операции, позволяющие вывести на печать или перфорацию полученные в машине результаты.

Мы ограничимся командой печати, которую будем условно записывать в виде

Печать a 0 c;

она означает печать содержимого группы ячеек от a до c включительно. Предположим, кроме того, что у нашей машины имеется команда, позволяющая сделать на табулограмме пробел, т. е. просто передвигающая бумажную ленту. Эту операцию мы будем обозначать словом

Интервал.

*) Запись из оперативной (ферритной) памяти на барабан *ФБ* и с барабана в ферритную память *БФ*.

§ 4. Примеры простых программ

Пользуясь обозначениями элементарных операций, приведенными в предыдущем параграфе, легко написать ряд простых программ. Рассмотрим некоторые примеры.

Пример 1.4. Вычислить значение y по формуле

$$y = \frac{x^2 + \frac{a}{x}}{bx + c},$$

предполагая, что значения a, b, c, x известны и введены в машину.

Адреса ячеек, в которых лежат соответствующие числа, естественно обозначить теми же буквами, что и эти числа. Тогда программа вычисления y будет выглядеть так:

Программа 1.4

1) $a : x = R_1$
2) $x \cdot x = R_2$
3) $R_1 + R_2 = R_3$
4) $b \cdot x = R_4$
5) $R_4 + c = R_5$
6) $R_3 : R_5 = y$
7) *стоп.*

Здесь R_1—R_5 означают номера рабочих ячеек, предназначенных для хранения промежуточных результатов вычислений. Пояснений к этой программе не требуется. Приведенную запись программы мы будем называть *записью в содержательных обозначениях.*

Пример 2.4. Составить программу для вычисления квадратного корня $y = \sqrt{x}$.

Значение квадратного корня можно получить способом последовательных приближений из последовательности, каждый следующий член которой получается из предыдущего по формуле

$$y_n = \frac{1}{2}\left(y_{n-1} + \frac{x}{y_{n-1}}\right).$$

Действительно, если эта последовательность сходится, то ее предел y удовлетворяет уравнению

$$y = \frac{1}{2}\left(y + \frac{x}{y}\right),$$

откуда следует $y^2 = x$, т. е. $y = \sqrt{x}$. В качестве начального приближения можно взять, например, значение $y_0 = x/2$.

Как видно из формулы, кроме значения x нужно иметь еще ячейку, в которой записано число $\frac{1}{2}$. Этого достаточно для записи программы, вычисляющей любой член y_n последовательности. Необходимо, однако, еще указать, какой из членов последовательности можно уже принять за значение корня. Можно, например, поступить так: вычислять последовательные значения y_n до тех пор, пока разность между двумя соседними членами не станет по модулю меньше заданного числа ε. В этом случае вычисление корня можно производить по программе:

Программа 2.4

1) $x \cdot \left(\frac{1}{2}\right)_N = y_0$
2) $x : y_0 = R_1$
3) $y_0 + R_1 = R_1$
4) $R_1 \cdot \left(\frac{1}{2}\right)_N = y_1$
5) $y_1 - y_0 = R_1$
6) $|R_1| - |\varepsilon| = 0$
7) (0) $y_1 \to y_0$; 2)
8) *стоп.*

Рассмотрим, как работает эта программа. Команда 1) вычисляет нулевое приближение. При этом $\left(\frac{1}{2}\right)_N$ означает ячейку, содержащую число $\frac{1}{2}$. Команды 2)—4) вычисляют следующее приближение y_1 по предыдущему y_0; R_1 означает рабочую ячейку для записи промежуточных результатов.

Следует обратить внимание на команду 3). Если бы речь шла о числах, то равенство $y_0+R_1=R_1$ означало бы, что $y_0=0$. Однако здесь мы имеем содержательную запись команды: числа из ячеек, адреса которых обозначены буквами y_0 и R_1, сложить и результат снова записать в ячейку R_1. Так как результат предыдущей операции уже использован и больше не потребуется, то его можно стереть и использовать ту же ячейку R_1 для записи следующего промежуточного результата.

Команды 5) и 6) служат для проверки достижения нужной точности. В команде 6) необходимо применить вычитание модулей, поскольку неизвестен знак разности y_1-y_0, полученной в результате выполнения предыдущей команды. При выполнении команды 6) выработается управляющий сигнал ω. Мы получим $\omega=1$, если разность y_1-y_0 по абсолютной величине будет меньше ε, т. е. если уже достигнута нужная точность. В противном случае результат вычитания в команде 6) будет положительным и выработается управляющий сигнал $\omega=0$. Команда 7) передаст тогда управление команде 2) и одновременно перешлет новое приближение значения корня y_1 в ячейку y_0, где содержалось предыдущее. Так будет продолжаться до тех пор, пока не будет выполнено неравенство $|y_1-y_0|<\varepsilon$. Тогда выработается сигнал $\omega=1$ и команда 7) передаст управление не команде 2), а следующей команде 8), которая и остановит машину.

Прежде чем перейти к следующему примеру, сделаем одно замечание. Будем предполагать, что ячейка с нулевым номером всегда содержит число нуль и является незабиваемой, т. е. запись в эту ячейку невозможна. Нулевую ячейку и число нуль будем обозначать одинаково.

Пример 3.4. Написать программу для вычисления тригонометрических функций $\cos x$ и $\sin x$ с помощью рядов

$$\cos x=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots,$$

$$\sin x=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots$$

Обозначим ячейки, предназначенные для значений этих функций, через cos и sin. Вычисления можно производить, например, по следующей программе.

Программа 3.4

1)	$0 \vee 1_N$	$=\cos$
2)	$0 \vee 0$	$=\sin$
3)	$0 \vee 1_N$	$=R_1$
4)	$1_N \to n;$	6)
5)	$n+1_N$	$=n$
6)	$R_1 \cdot x$	$=R_1$
7)	$R_1 : n$	$=R_1$
8)	$\sin+R_1$	$=\sin$
9)	$n+1_N$	$=n$
10)	$0-R_1$	$=R_1$
11)	$R_1 \cdot x$	$=R_1$
12)	$R_1 : n$	$=R_1$
13)	$\cos+R_1$	$=\cos$
14)	$\lvert 0\rvert-\lvert R_1\rvert$	$=0$
15)	(1)	5)
16)	*стоп.*	

Из программы видно (команды 14) и 15)), что вычисления производятся с машинной точностью, т. е. члены ряда считаются до тех пор, пока они остаются отличными от нуля. Нуль в правой части команды 14) означает, что разность должна быть записана в нулевую ячейку. Как было сказано выше, такая запись фактически невозможна, так что разность никуда не запишется. Но это и не требуется, потому что здесь играет роль не величина разности, а значение управляющего сигнала ω, который получается при этом вычитании. В остальном с работой программы читатель разберется самостоятельно без больших затруднений.

Пример 4.4. Приведем программу для нахождения корней квадратного уравнения $ax^2+bx+c=0$, предполагая, что коэффициенты его уже лежат в ячейках, обозначенных теми же буквами.

Так как корни уравнения могут оказаться комплексными, то для каждого из них отведем две ячейки. Пусть, например, действительную и мнимую части первого корня следует записать соответственно в ячейки x_1 и $\tilde{x}_1$, а второго — в ячейки x_2 и $\tilde{x}_2$. В случае действительных корней должно быть $\tilde{x}_1=\tilde{x}_2=0$.

При решении уравнения приходится обращаться к программе извлечения квадратного корня, которая была разобрана в примере 2.4. Кроме того, нужно иметь ячейки, содержащие числа 4 и $\frac{1}{2}$.

Программу решения квадратного уравнения можно записать, например, как программу 4.4.

Команды 1) — 5) этой программы вычисляют знаменатель $2a$, величину $-b$ и дискриминант уравнения $D=b^2-4ac$. В зависимости от знака дискриминанта программа разветвляется. Если $D \geqslant 0$, то в результате выполнения команды 6) выработается управляющий сигнал $\omega = 1$. Поэтому команда 7) передает управление команде 8). Команда 8) передает управление команде 23), которая является начальной командой программы извлечения квадратного корня из числа D и одновременно запишет в ячейку 30) команду передачи управления команде 9). Когда при извлечении корня будет достигнута нужная точность, после выполнения команды 28) выработается сигнал $\omega = 1$ и команда 29) передаст управление команде 30), откуда благодаря записанной там команде управление будет передано команде 9).

Команды 9)—14) завершат вычисление действительных корней уравнения и запишут нули в ячейки, отведенные для мнимых частей. На месте команды 15) можно было бы поставить *стоп*, но тогда программа имела бы два различных конца, что нежелательно. Чтобы этого избежать, здесь помещена команда безусловной передачи управления ячейке 22), которая и остановит машину.

Если $D < 0$, то после команды 6) мы будем иметь $\omega = 1$ и команда 6) передаст управление ячейке 16). Здесь начинается часть программы, соответствующая комплексным корням. Команды 8)—15) не будут в этом случае выполняться.

Программа 4.4

1)		$a + a$	$= R_1$
2)		$0 - b$	$= R_2$
3)		$b \cdot b$	$= R_3$
4)		$a \cdot c$	$= R_4$
5)		$R_4 \cdot 4_N$	$= R_4$
6)		$R_3 - R_4$	$= D$
7)	(1)		16)
8)		9) $\leftrightarrow$ 30);	23)
9)		$R_2 + A$	$= R_4$
10)		$R_4 : R_1$	$= x_1$
11)		$R_2 - A$	$= R_4$
12)		$R_4 : R_1$	$= x_2$
13)		$0 \vee 0$	$= \tilde{x}_1$
14)		$0 \vee 0$	$= \tilde{x}_2$
15)	*Б*		22)
16)		$0 - D$	$= D$
17)		18) $\leftrightarrow$ 30);	23)
18)		$R_2 : R_1$	$= x_1$
19)		$0 \vee x_1$	$= x_2$
20)		$A : R_1$	$= \tilde{x}_1$
21)		$0 - \tilde{x}_1$	$= \tilde{x}_2$
22)		*стоп*	
23)		$D \cdot (1/2)_N$	$= R_5$
24)		$D : R_5$	$= R_6$
25)		$R_5 + R_6$	$= R_6$
26)		$R_6 \cdot \left(\frac{1}{2}\right)_N$	$= A$
27)		$A - R_5$	$= R_6$
28)		$\lvert R_6 \rvert - \lvert \varepsilon \rvert$	$= 0$
29)	(0)	$A \rightarrow R_5$;	24)
30)		*к. п.*	

Прежде всего команда 16) изменит знак подкоренного выражения. Затем команда 17), как в первом случае команда 8), передаст управление программе извлечения корня, но в ячейку 30) будет уже записана команда возврата к 18). После этого команды 18)—21) вычислят действительные и мнимые части корня и команда 22) остановит машину.

Ячейку 30) можно оставлять пустой или записывать в нее все, что угодно. Команда передачи управления в нужное место будет записана в нее самой машиной.

§ 5. Представление чисел и команд в машине

Примем, что рассматриваемая нами условная трехадресная машина имеет ячейку памяти, состоящую из 44 разрядов, и работает с плавающей запятой. Разряды в ячейке нумеруются десятичными номерами от 1 до 44 справа налево.

В ячейке может быть записано как десятичное (естественно, в двоично-десятичном виде) число, что бывает необходимым непосредственно после ввода и перед выводом, так и двоичное. При записи на бланке и при чтении двоичные числа изображаются в двоично-восьмеричном виде — каждая триада, т. е. тройка двоичных разрядов, записывается или читается как восьмеричная цифра.

При записи в ячейке памяти десятичного числа ее разряды используются следующим образом. В 44-м разряде записывается знак числа, в 43-м разряде — знак порядка. При этом в обоих случаях нуль означает знак плюс, а единица — знак минус.

Следующие шесть разрядов (с 42-го по 37-й) предназначены для записи десятичного порядка числа. Они делятся на две группы: диаду (состоящую из двух разрядов 42-го и 41-го), которая отводится для записи десятков порядка, и тетраду (с 40-го по 37-й разряд), которая содержит единицы порядка. Остальные 36 разрядов (с 36-го по 1-й) содержат мантиссу числа (рис. 2). По размеру ячейки мантисса содержит девять тетрад, так что в машине помещается девятиразрядное десятичное число.

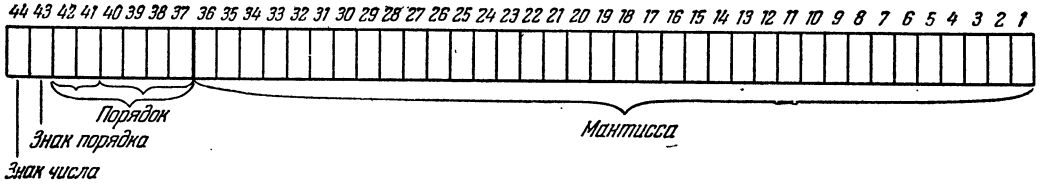


Рис. 2.

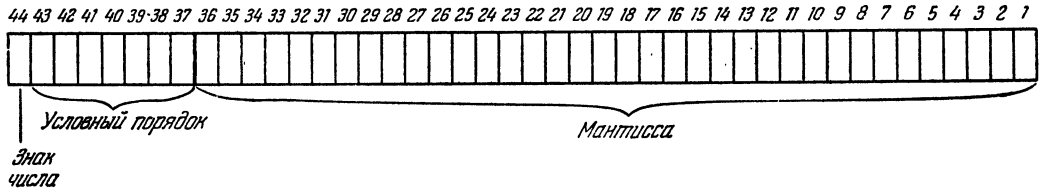


Рис. 3.

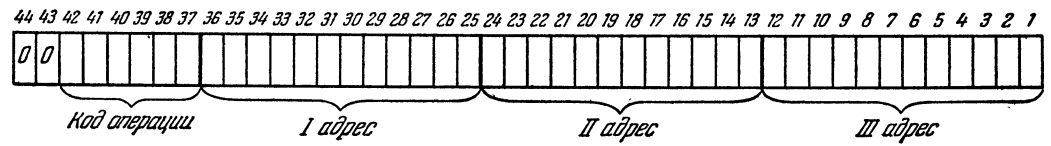


Рис. 4.

Аналогичным образом делятся на части и выходные данные на табулограмме. Например, число 12,5 будет изображено на табулограмме в виде

$$++02 \qquad 125000000,$$

число 0,00037408 в виде

$$+--03 \qquad 374080000,$$

а число $-1{,}22 \cdot 10^{13}$ в виде

$$-+14 \qquad 122000000.$$

При записи в ячейке двоичного числа 44-й разряд по-прежнему содержит знак числа, изображаемый тем же способом, что и для десятичного. В семи разрядах (с 43-го по 37-й) записывается *условный* (или *машинный*) порядок числа

$$p' = p + 100_8,$$

где p означает истинный, а p' — условный порядок числа. Если истинный порядок двоичного числа может быть числом любого знака и заключен в пределах

$$-100_8 \leqslant p \leqslant 77_8,$$

то для условного порядка имеем

$$0 \leqslant p' \leqslant 177_8,$$

так что условный порядок всегда неотрицателен.

Числа первого и второго порядка будут, например, иметь условный порядок соответственно 101 и 102, а отрицательные порядки —1 и —2 приведут к условным порядкам соответственно 77 и 76.

Легко заметить, что при положительном порядке числа условный порядок его больше 100, так что в 43-м разряде будет стоять единица. Наоборот, если порядок числа отрицателен, то его условный порядок меньше 100 и в 43-м разряде должен быть нуль. Поэтому можно считать, что и для двоичного числа порядок записан в 42—37-м разрядах, а 43-й разряд содержит знак порядка, но только здесь 1 означает знак плюс, а 0 — знак минус. Кроме того, отрицательный порядок изображается «в дополнительном коде», т. е. он равен тому

числу единиц, которые нужно прибавить к записанному в 42—37-м разрядах числу, чтобы получить 100_8.

Как и для десятичного числа, разряды 36—1-й отведены для записи мантиссы, которая состоит, следовательно, из 36 двоичных разрядов, или 12 триад, записываемых в виде восьмеричных цифр. Схема использования разрядов при записи двоичного числа показана на рис. 3.

Тридцать шесть двоичных разрядов соответствуют примерно десяти-одиннадцати десятичным. Поэтому промежуточные вычисления ведутся в машине с несколькими запасными знаками по сравнению с входными и выходными данными, что предотвращает большое накопление ошибок округления при арифметических действиях.

Если в ячейку памяти записывается не число, а команда, то разряды 44-й и 43-й не используются. Шесть разрядов с 42-го по 37 й отводятся в ячейке для записи кода операции, который, таким образом, представляет собою шестиразрядное двоичное число или, если его записывают в двоично-восьмеричной форме, двузначное восьмеричное число. Разряды 36—1-й отводятся для трех адресов команды, по двенадцать разрядов на каждый адрес. Левый адрес (36—25-й разряды) является первым, средний (24—13-й разряды) — вторым и правый (12—1-й разряды) — третьим.

Двенадцать разрядов, отведенные на один адрес, позволяют записать $2^{12} = 4096$ различных адресов, поэтому мы будем предполагать, что рассматриваемая нами условная трехадресная машина имеет оперативную память объемом 4096 ячеек.

Таким образом, команда, записанная в ячейке памяти, представляет собою группу из четырех двоичных чисел — одного шестиразрядного *) и трех двенадцатиразрядных (рис. 4). На бланке команда записывается в виде группы из четырех восьмеричных чисел — одного двузначного и трех четырехзначных.

Из сказанного следует, что в ячейке машины адресная часть при записи команды совпадает с мантиссой

*) Не следует забывать, что впереди кода имеется два разряда (43-й и 44-й), не используемые нами при записи команды, в которых предполагаются записанными нули.

при записи числа, а порядок числа (без знаков) — с кодом операции. Поэтому при записи числа на бланке мантиссу тоже пишут в виде трех групп чисел по адресам, двенадцать двоичных разрядов в каждом адресе, т. е. по три тетрады, если число пишется в десятичной форме, и четыре триады — если в восьмеричной. Например, число 276,35147 записывают на бланке в виде

$$++ \quad 03 \quad 276 \quad 351 \quad 470.$$

При записи двоичного числа знаковый разряд записывают как цифру и рассматривают условный порядок вместе со знаком как восьмиразрядное двоичное число или трехразрядное восьмеричное. Так, число

$$285{,}77_{10} = \\ = 100011101{,}11000101000111101011100001010001111 1\ldots_2 = \\ = 435{,}6121727024\ldots_8$$

записывается на бланке в виде

$$111 \quad 4356 \quad 1217 \quad 2703$$

(истинный двоичный порядок 11, условный порядок 111), то же число со знаком минус имело бы вид

$$311 \quad 4356 \quad 1217 \quad 2703,$$

а число $-0{,}109375_{10} = 0{,}000111_2 = 0{,}07_8$ имело бы вид

$$275 \quad 7000 \quad 0000 \quad 0000$$

(истинный порядок — 3, условный порядок $100_8 - 3_8 = 75_8$; мантисса содержит одну значащую цифру 7).

§ 6. Содержательная часть и кодировка

Прежде чем ввести в память машины составленную программу и для того, чтобы иметь возможность это сделать, необходимо заменить содержательную запись каждой команды такой, которая соответствует описанному в предыдущем параграфе способу представления команды в машине. С этой целью нужно указать числовой код каждой элементарной операции и вместо содержательных адресов написать фактические. Переход к программе, записанной в таком виде, от программы, за-

писанной в содержательных обозначениях, называется *кодированием программы.*

Программа в содержательных обозначениях вместе с кодированной пишется на одном бланке. Вид бланка может быть различным и зависит от способа ввода. Если ввод происходит с ленты, то число строк на бланке безразлично. Если же ввод происходит с перфокарт, то удобнее, чтобы число строк на бланке совпадало с числом возможных позиций на одной перфокарте.

Фотография бланка при вводе с перфокарт приведена на рис. 5. Команды в приведенной выше содержательной форме пишутся здесь слева, а в кодированном виде — справа. Поэтому команды, записанные в содержательной форме, мы будем называть «левой частью» программы.

Поскольку кодировка программы может быть легко выполнена по совсем простым формальным правилам, основной задачей программиста является написание левой, содержательной части программы.

Широкая графа в левой половине бланка отводится для записи содержательной части. Слева от нее оставлено место для номеров или условных обозначений команд. Правее идет колонка для записи адреса ячейки, в которой будет лежать данная команда, а затем сама команда — код и три ее адреса.

В первой строке слева от буквы *А* записывается *адресный код* — адрес ячейки, в которую вводится первая из команд. Следующие команды вводятся подряд в следующие ячейки. Если требуется ввести какую-либо команду не подряд, а в другое место, то перед нею необходимо снова поместить соответствующий адресный код.

Для кодирования программы необходимо прежде всего знать коды рассматриваемых операций. Приведем эти коды и некоторые уточнения к отдельным операциям, описанным в § 3.

Коды арифметических операций:

сложение	01
вычитание	02
вычитание модулей	03
деление	04
умножение	05

составил задача блок стр. лист

					A	шифр	№ карты

					A	шифр	№ карты

кодировала перфорировала:

проверила кодировку: проверила карты:

Рис. 5.

Таким образом, команда, записанная в левой части как

$$a + b = c,$$

в кодированном виде будет выглядеть так:

01 адрес a адрес b адрес c.

Коды фиксированных операций:

сложение адресных частей	13
вычитание адресных частей	33
сложение кодовых частей	53
вычитание кодовых частей	73
циклическое сложение	07

Команды

$$a +, b = c,$$
$$a, - b = c,$$
$$x \oplus y = z$$

при кодировке примут соответственно вид:

13	адрес a	адрес b	адрес c,
73	адрес a	адрес b	адрес c,
07	адрес x	адрес y	адрес z.

Коды логических операций:

Логическое сложение	75
Логическое умножение	55
Сверка	15

Таким образом, команда

$$a \dot{+} 0 = b$$

в закодированном виде будет выглядеть так:

15 адрес a 0 адрес b

(напомним, что число 0 лежит в нулевой ячейке памяти, так что адрес нуля есть нуль).

Коды операций управления:

безусловная передача управления с возвратом	16
условная передача управления при $\omega = 1$	36
безусловная передача управления	56
условная передача управления при $\omega = 0$	76

При записи команд передачи управления в левой части возможны некоторые сокращения. Прежде всего, очень часто в командах передачи управления не используются пересылки. В этих случаях можно ограничиться указанием лишь правого адреса, т. е. адреса ячейки, в которую нужно передать управление, и не писать ничего слева и посредине. При кодировании в левом и среднем адресах команды ставятся нули. Например, можно писать

Б 4),

что означает безусловную передачу управления команде 4) и в кодированном виде выглядит так:

56 0, 0; адрес 4)

или

(1) 12),

т. е. в кодированном виде

36 0, 0; адрес 12).

Нумерация всех команд программы очень загромождает ее запись и является излишней. Вполне достаточно нумеровать лишь те команды, которым будет передаваться управление. При этом порядок номеров не играет никакой роли, так что номера могут быть расположены как угодно. Более удобно даже применять не нумерацию команд, а различные условные обозначения команд, которым передается управление.

Наконец, если команда, которой передается управление, написана на той же странице, то удобнее всего провести стрелку из места, откуда передается управление, до строки, соответствующей команде, которой передается управление. Подробнее с этими обозначениями можно познакомиться на примерах, которые будут приведены ниже.

К операциям управления следует также отнести команду *стоп*. Придадим этой команде код 77. Кроме того, в команде можно указать адреса трех ячеек, содержимое которых при остановке машины вызывается на пульт. Следовательно, команда остановки машины с

вызовом на пульт в левой части программы будет записана в виде

стоп *a*, *b*, *c*,

а в кодированной форме

77 адрес *a* адрес *b* адрес *c*.

Коды специальных операций:

Сдвиг мантиссы по адресу	14
Сдвиг мантиссы по порядку	34
Полный сдвиг по адресу	54
Полный сдвиг по порядку	74

При сдвиге по порядку в левом адресе указывается номер ячейки, по кодовой части которой производится сдвиг, а при сдвиге по адресу в левом адресе команды указывается число разрядов, на которые надо произвести сдвиг.

Более точно дело обстоит следующим образом. В семи младших разрядах пишется число $100_8 + p_8$, где p — число разрядов, на которое нужно произвести сдвиг, причем при сдвиге налево считается $p > 0$, а при сдвиге направо принимается $p < 0$. Например, команда

$$\overline{114} \Rightarrow, \quad a = b$$

в кодированном виде выглядит так:

14 0114 адрес *a* адрес *b*

и означает, что мантисса ячейки *a* сдвигается на $14_8 = 12_{10}$ разрядов влево, т. е. на один адрес (второй адрес сдвигается в первый, третий — во второй; на месте третьего адреса будут нули, первый адрес будет утерян; код остается без изменения). Результат запишется в ячейку *b*.

Команда

$$\overline{27} \Rightarrow x = y$$

кодируется в виде

54 0027 адрес *x* адрес *y*

и означает, что содержимое ячейки *x* сдвигается на $100_8 - 27_8 = 51_8 = 41_{10}$ разрядов вправо и результат записывается в ячейку *y*.

При сдвиге по порядку аналогичным образом расшифровывается число, образованное семью младшими разрядами порядка числа, записанного в ячейке, адрес которой указан в первом адресе команды сдвига. Так, команда

$$m\,(\Rightarrow)\,n = p$$

кодируется

74 адрес m адрес n адрес p

и означает сдвиг всего содержимого ячейки n на столько разрядов, каков порядок содержимого ячейки m.

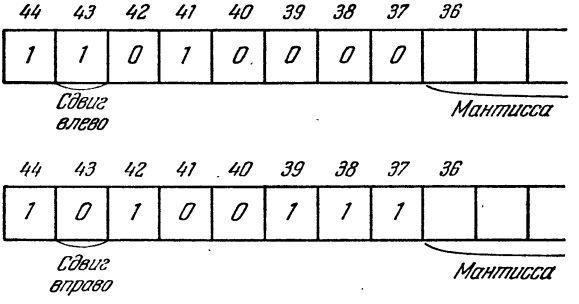


Рис. 6.

Если, например, в ячейке m записано число с условным порядком 320, то сдвиг происходит на $20_8 = 16_{10}$ разрядов влево, а если число с порядком 247, то содержимое ячейки n сдвигается на

$$100_8 - 47_8 = 31_8 = 25_{10}$$

разрядов вправо. Рис. 6 показывает, как выглядит содержимое ячейки m в том и другом случае.

Операции обмена информацией будем кодировать следующим образом:

ФБ	70
Печать	60
БФ	50
Интервал	40

Например, команда

ФБ a b c

в кодированном виде будет выглядеть так:

70 адрес *a* адрес *b* адрес *c*.

При кодировке команды печати в среднем адресе ставится нуль, так что команда

Печать a 0 *b*

в кодированном виде выглядит так:

60 адрес *a* 0 адрес *b*.

В команде *Интервал* все три адреса кодируются нулями.

После того как поставлен код операции, необходимо заменить в команде условные адреса истинными. Для этой цели нужно разместить в памяти программу и числа. Размещение облегчается использованием специальных бланков (шпаргалок), на которых приводится нумерация ячеек памяти. На рис. 7 показана часть шпаргалки с распределением памяти для задачи, разобранной в примере 1.4. На рис. 8 приведена программа этой задачи в закодированном виде, причем ячейки *x*, *y* вызваны на пульт при остановке.

Ясно, что программу можно закодировать на любое место памяти. После того как программа закодирована и набита на перфокартах, нужно только ввести ее вместе с требуемыми входными числовыми данными в память машины и передать управление первой ячейке программы. Это можно сделать с пульта или с помощью специальной операции.

Отметим еще одно часто используемое сокращение. Во многих случаях приходится пересылать некоторое число из одной ячейки в другую без изменений. Мы будем делать это всегда путем логического сложения с нулем. В этих случаях можно в левой части опускать нуль и знак логического сложения. Таким образом, команда пересылки числа из ячейки *a* в ячейку *x* пишется

$$a = x,$$

4400	4440	4500	4540
4401	4441 a	4501 R_1	4541
4402	4442 b	4502 R_2	4542
4403 Программа 1.4	4443 c	4503 R_3	4543
4404	4444 x	4504 R_4	4544
4405	4445 y	4505 R_5	4545
4406	4446	4506	4546
4407	4447	4507	4547
4410	4450	4510	4550
4411	4451	4511	4551

Рис. 7.

	Программа 1.4			4400	A	T шифр	№ 1.4 № карты
1)	$a : x = R_1$	4400	04	4441	4444	4501	
2)	$x \cdot x = R_2$	1	05	4444	4444	4502	
3)	$R_1 + R_2 = R_3$	2	01	4501	4502	4503	
4)	$b \cdot x = R_4$	3	05	4442	4444	4504	
5)	$R_4 + c = R_5$	4	01	4504	4443	4505	
6)	$R_3 : R_5 = y$	5	04	4503	4505	4445	
7)	*стоп* x y 0	6	77	4444	4445	0000	

Рис. 8.

означает

$$0 \vee a = x$$

и кодируется

75 0 адрес a адрес x.

В качестве следующих примеров ниже приведены шпаргалки (рис. 9, 11, 13) для задач, рассмотренных в § 4, и программы (рис. 10, 12, 14) в закодированном виде.

4600 ↑	4640	4700 y_0	4740
4601	4641 R_1	4701 y_1	4741
4602	4642	4702 x	4742
4603 2,4	4643	4703	4743
4604	4644	4704	4744
4605 Программа	4645	4705	4745
4606	4646	4706	4746
4607	4647	4707	4747
4610 $\frac{1}{2}N$	4650	4710	4750
4611 ε ↓	4651	4711	4751
4612	4652	4712	4752
4613	4653	4713	4753
4614	4654	4714	4754
4615	4655	4715	4755

Рис. 9.

В дальнейшем мы будем, в основном, ограничиваться написанием содержательных частей программ, лишь в редких случаях приводя их кодировку.

Сделаем еще несколько замечаний относительно обозначений, применяемых нами при кодировке программ. В содержательной части программы встречаются числа, которые могут иметь различный смысл. Для того чтобы не ошибаться при кодировке, мы будем строго придерживаться следующих условий.

а) Если речь идет о действиях с числами, записанными в ячейке памяти в плавающей форме, то в содержательной части у этого числа будет ставиться знак N. При кодировке на соответствующем месте должен быть

	Программа 2.4			4600	A	T шифр	№ 2.4 № карты
1)	$x \cdot \frac{1}{2}_N = y_0$	4600	05	4702	4610	4700	
2)	$x : y_0 = R_1$	1	04	4702	4700	4641	
3)	$y_0 + R_1 = R_1$	2	01	4700	4641	4641	
4)	$R_1 \cdot \frac{1}{2}_N = y_1$	3	05	4641	4610	4701	
5)	$y_1 - y_0 = R_1$	4	02	4701	4700	4641	
6)	$\lvert R_1 \rvert - \lvert \varepsilon \rvert = 0$	5	03	4641	4611	0	
7)	(0) $y_1 \to y_0$ 2)	6	76	4701	4700	4601	
8)	*стоп* x y 0	7	77	4702	4701	0000	
9)	$\frac{1}{2}_N$	4610	100	4000	0	0	
10)	ε	11	060	4000	0	0	

Рис. 10.

поставлен адрес ячейки, содержащей это число. Например, команда

$$x + 1_N = y$$

кодируется

01 адрес x адрес 1 адресу y.

б) Если число означает номер команды, то в содержательной части мы будем всегда писать этот номер со

скобкой. При кодировке на соответствующем месте нужно ставить адрес ячейки, в которой лежит данная

5000	5040 cos	5100 π	5140
5001	5041 sin	5101 R_1	5141
5002	5042 x	5102	5142
5003	5043	5103	5143
5004	5044	5104	5144
5005	5045	5105	5145
5006	5046	5106	5146
5007	5047	5107	5147
5010	5050	5110	5150
5011	5051	5111	5151
5012	5052	5112	5152
5013	5053	5113	5153
5014	5054	5114	5154
5015	5055	5115	5155
5016	5056	5116	5156
5017 1_N	5057	5117	5157
5020	5060	5120	5160
5021	5061	5121	5161

Зд. 4 Программа

Рис. 11.

команда. Например, команда

$$1_N \to x, \quad 4)$$

кодируется

56 адрес числа 1 адрес x адрес команды 4).

в) В ряде случаев в содержательной части приходится писать число, которое в том же виде должно быть записано в кодированную часть, как, например, в команде сдвига по адресу. Над таким числом в содержательной

	Программа 3.4			5000	A	T шифр	№ 3.4.2 № карты
1)	$1_N = \cos$	5000	75	0000	5020	5040	
2)	$0 = \sin$	1	75	0000	0000	5041	
3)	$1_N = R_1$	2	75	0000	5020	5101	
4)	$1_N \to n$; 6)	3	56	5020	5100	5005	
5)	$n + 1_N = n$	4	01	5100	5020	5100	
6)	$R_1 \cdot x = R_1$	5	05	5101	5042	5101	
7)	$R_1 : n = R_1$	6	04	5101	5100	5101	
8)	$\sin + R_1 = \sin$	7	01	5041	5101	5041	
9)	$n + 1_N = n$	5010	01	5100	5020	5100	
10)	$0 - R_1 = R_1$	1	02	0000	5101	5101	
11)	$R \cdot x = R_1$	2	05	5101	5042	5101	

				5013	A	T шифр	№ 3.4.2 № карты
12)	$R_1 : n = R_1$	5013	04	5101	5100	5101	
13)	$\cos + R_1 = \cos$	4	01	5040	5101	5040	
14)	$\lvert 0 \rvert - \lvert R_1 \rvert = 0$	5	03	0000	5101	0000	
15)	(1) 5)	6	36	0000	0000	5004	
16)	*cmon x* cos sin	7	77	5042	5041	5040	
17)	1_N	5020	101	4000	0000	0000	

Рис. 12.

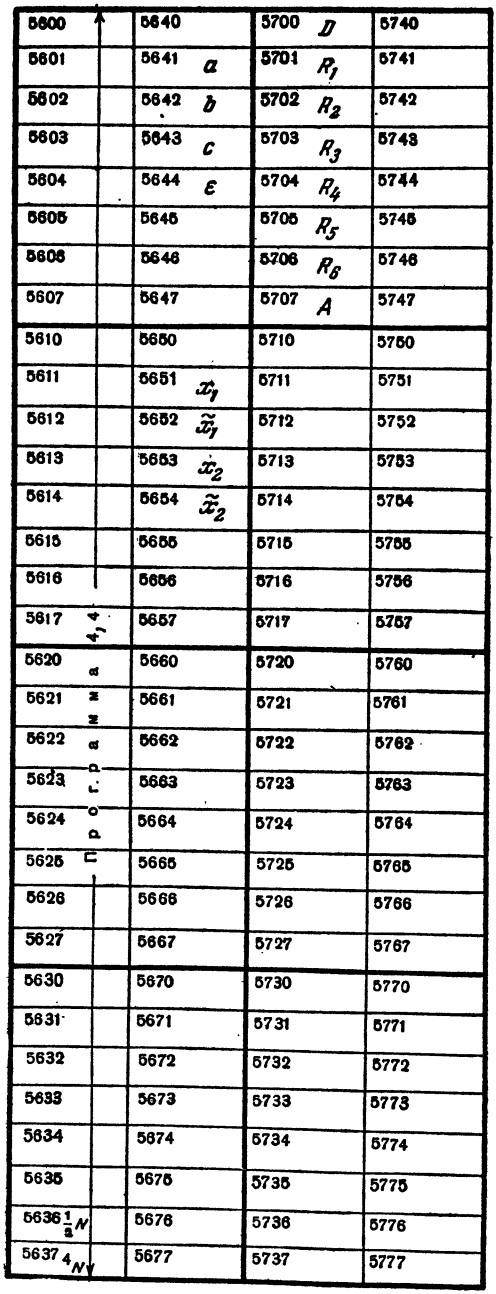

5600
5601
5602
5603
5604
5605
5606
5607
5610
5611
5612
5613
5614
5615
5616
5617
5620
5621
5622
5623
5624
5625
5626
5627
5630
5631
5632
5633
5634
5635
5636
5637
5640
5641 a
5642 b
5643 c
5644 ε
5645
5646
5647
5650
5651 x_1
5652 $\tilde{x}_1$
5653 x_2
5654 $\tilde{x}_2$
5655
5656
5657
5660
5661
5662
5663
5664
5665
5666
5667
5670
5671
5672
5673
5674
5675
5676
5677
5700 D
5701 R_1
5702 R_2
5703 R_3
5704 R_4
5705 R_5
5706 R_6
5707 A
5710
5711
5712
5713
5714
5715
5716
5717
5720
5721
5722
5723
5724
5725
5726
5727
5730
5731
5732
5733
5734
5735
5736
5737
5740
5741
5742
5743
5744
5745
5746
5747
5750
5751
5752
5753
5754
5755
5756
5757
5760
5761
5762
5763
5764
5765
5766
5767
5770
5771
5772
5773
5774
5775
5776
5777
4, 4
Адрес
$\frac{1}{2}N$
4_N

Рис. 13.

	Программа 4.4			5600	A	T шифр	№ 4.4.1 № карты
1)	$a+a=R_1$	5600	01	5641	5641	5701	
2)	$0-b=R_2$	1	02	0000	5642	5702	
3)	$b\cdot b=R_3$	2	05	5642	5642	5703	
4)	$a\cdot c=R_4$	3	05	5641	5643	5704	
5)	$R_4\cdot 4_N=R_4$	4	05	5704	5637	5704	
6)	$R_3-R_4=D$	5	02	5703	5704	5700	
7)	(1) 16)	6	36	0000	0000	5617	
8)	9) ←→ 30); 23)	7	16	5610	5635	5626	
9)	$R_2+A=R_4$	5610	01	5702	5707	5704	
10)	$R_4:R_1=x_1$	11	04	5704	5701	5651	
11)	$R_2-A=R_4$	12	02	5702	5707	5704	
				5613	A	T шифр	№ 4.4.2 № карты
12)	$R_4:R_1=x_2$	5613	04	5704	5701	5653	
13)	$0=\widetilde{x}_1$	4	75	0000	0000	5652	
14)	$0=\widetilde{x}_2$	5	75	0000	0000	5654	
15)	Б 22)	6	56	0000	0000	5625	
16)	$0-D=D$	7	02	0000	5700	5700	
17)	18) ←→ 30); 23)	5620	16	5621	5635	5626	
18)	$R_2:R_1=x_1$	1	04	5702	5701	5651	
19)	$x_1=x_2$	2	75	0000	5651	5653	
20)	$A:R_1=\widetilde{x}_1$	3	04	5707	5701	5652	
21)	$0-\widetilde{x}_1=\widetilde{x}_2$	4	02	0000	5652	5654	
22)	*стоп*	5	77	0000	0000	0000	

Рис. 14*а*.

				5626	A	T шифр	№ 4.4.3 № карты
23)	$D\cdot\frac{1}{2}_N = R_5$	5626	05	5700	5636	5705	
24)	$D : R_5 = R_6$	7	04	5700	5705	5706	
25)	$R_5 + R_6 = R_6$	5630	01	5705	5706	5706	
26)	$R_6\cdot\frac{1}{2}_N = A$	1	05	5706	5636	5707	
27)	$A - R_5 = R_6$	2	02	5707	5705	5706	
28)	$\lvert R_6 \rvert - \lvert \varepsilon \rvert = 0$	3	03	5706	5644	0000	
29)	(0) $A \to R_5$; 24)	4	76	5707	5705	5627	
30)	*н. п.*	5			н. п.		
	$\frac{1}{2}_N$	6	100	4000	0000	0000	
	4_N	7	103	4000	0000	0000	

Рис. 14*б*.

части мы будем ставить черту. Например, команда

$$\overline{27} \Rightarrow,\ a = b$$

кодируется

14 0027 адрес a адрес b.

Все эти обозначения фактически уже использовались выше.

ГЛАВА II
ОСНОВЫ ПРОГРАММИРОВАНИЯ

§ 7. Расписка формулы

Программы однократного вычисления по заданным формулам являются примерами программ, составленных из команд со стандартной передачей управления.

Пример 1.7. Составить программу для вычисления по формуле $y=a_0+a_1x+a_2x^2+a_3x^3+a_4x^4$. Произведем расписку формулы, т. е. определим последовательность арифметических действий для нахождения y. Это можно сделать различными способами. Один из них приводится ниже.

$$
\begin{aligned}
&1)\quad x\cdot x=x^2,\\
&2)\quad x\cdot x^2=x^3,\\
&3)\quad x\cdot x^3=x^4,\\
&4)\quad a_1\cdot x=u_1,\\
&5)\quad a_2\cdot x^2=u_2,\\
&6)\quad a_3\cdot x^3=u_3,\\
&7)\quad a_4\cdot x^4=u_4,\\
&8)\quad a_0+u_1=u_5,\\
&9)\quad u_2+u_5=u_6,\\
&10)\quad u_3+u_6=u_7,\\
&11)\quad u_4+u_7=y.
\end{aligned}
\qquad (1)
$$

Здесь x^2, x^3, x^4, u_1, u_2, u_3, u_4, u_5, u_6, u_7 — промежуточные результаты вычислений. Формулы (1) можно рассматривать как содержательную часть программы вычисления y, однако обозначения в этих формулах и сами они получают иной смысл. Символы x, y, x^2, ..., a_0, ..., u_1, ... обозначают теперь адреса ячеек памяти,

в которых записаны соответствующие числа. При этом в ячейки x, a_0, a_1, a_2 a_3 и a_4 числа должны быть записаны перед началом работы программы, а в остальных ячейках они получаются в результате ее работы. Первые называются *константами* или *аргументами* программы, вторые — *рабочими ячейками.*

Формулы (1) рассматриваются теперь как команды со стандартной передачей управления, лежащие в расположенных подряд ячейках. Таким образом, если управление передано ячейке, в которой лежит команда 1), то после ее выполнения управление перейдет команде 2) и т. д. Когда, наконец, выполнится команда 11), искомая величина будет вычислена и записана в ячейку y. После этого управление перейдет к ячейке, непосредственно следующей за ячейкой с командой 11). В этой ячейке должна быть расположена команда, обеспечивающая правильную работу машины после окончания вычислений. Если считать для простоты, что задача состоит только в вычислении по заданной формуле, то в конце программы надо поместить команду *стоп*. Левая часть программы принимает вид:

Программа 1.7

1) $x \cdot x = x^2$
2) $x \cdot x^2 = x^3$
3) $x \cdot x^3 = x^4$
4) $a_1 \cdot x = u_1$
5) $a_2 \cdot x^2 = u_2$
6) $a_3 \cdot x^3 = u_3$
7) $a_4 \cdot x^4 = u_4$
8) $a_0 + u_1 = u_5$
9) $u_2 + u_5 = u_6$
10) $u_3 + u_6 = u_7$
11) $u_4 + u_7 = y$
12) *стоп.*

3000	Программа 1,7	3040	a_0
3001		3041	a_1
3002		3042	a_2
3003		3043	a_3
3004		3044	a_4
3005		3045	x
3006		3046	x^2
3007		3047	x^3
3010		3050	x^4
3011		3051	u_1
3012		3052	u_2
3013		3053	u_3
3014		3054	u_4
3015		3055	u_5
3016		3056	u_6
3017		3057	u_7
3020		3060	y

Рис. 15.

Кодирование программы 1.7 по составленной шпаргалке производится автоматически. Шпаргалка для программы 1.7 может быть такой, как показано на рис. 15. Закодированная программа с левой частью, написанной на том же бланке, приведена на рис. 16.

Возвратимся снова к программе 1.7. Заметим, что промежуточные результаты после вычисления y не нужны; поэтому их можно записывать в одну и ту же ячейку, если это не мешает дальнейшим вычислениям. Таким образом можно сократить количество рабочих ячеек программы.

	Программа 1.7			3100	A	T шифр	№ 1.7.1 № карты
	$x \cdot x = x^2$	3100	05	3045	3045	3046	
	$x \cdot x^2 = x^3$	1	05	3045	3046	3047	
	$x \cdot x^3 = x^4$	2	05	3045	3047	3050	
	$a_1 \cdot x = u_1$	3	05	3041	3045	3051	
	$a_2 \cdot x^2 = u_2$	4	05	3042	3046	3052	
	$a_3 \cdot x^3 = u_3$	5	05	3043	3047	3053	
	$a_4 \cdot x^4 = u_4$	6	05	3044	3050	3054	
	$a_0 + u_1 = u_5$	7	01	3040	3051	3055	
	$u_2 + u_5 = u_6$	3110	01	3052	3055	3056	
	$u_3 + u_6 = u_7$	1	01	3053	3056	3057	
	$u_4 + u_7 = y$	2	01	3054	3057	3060	

				3113	A	T шифр	№ 1.7.2 № карты
	стоп	3113	77	0000	0000	0000	

Рис. 16.

Легко, например, обойтись ячейками x^2, x^3, x^4, y и еще одной ячейкой. Если эту ячейку обозначить R_1, то наша программа будет выглядеть так:

Программа 2.7

1) $x \cdot x = x^2$
2) $x \cdot x^2 = x^3$
3) $x \cdot x^3 = x^4$
4) $a_1 \cdot x = R_1$
5) $a_0 + R_1 = y$
6) $a_2 \cdot x^2 = R_1$
7) $y + R_1 = y$
8) $a_3 \cdot x^3 = R_1$
9) $y + R_1 = y$
10) $a_4 \cdot x^4 = R_1$
11) $y + R_1 = y$
12) *стоп.*

Расписку формул следует производить непосредственно в виде программы. Рассмотрим, например, вычисление y по схеме Горнера

$$y = \{[(a_4x + a_3)x + a_2]x + a_1\}x + a_0.$$

Мы получим программу:

Программа 3.7

1) $a_4 \cdot x = y$
2) $y + a_3 = y$
3) $x \cdot y = y$
4) $y + a_2 = y$
5) $x \cdot y = y$
6) $y + a_1 = y$
7) $x \cdot y = y$
8) $y + a_0 = y$
9) *стоп.*

Эта программа короче программы 2.7 и требует меньшего числа рабочих ячеек. Кроме того, общее количество действий по ней меньше и потому счет происходит быстрее.

Пример 2.7. Составить программу вычисления по формуле

$$y = x^4 - 2x^3 - 7x^2 + 8x + 1.$$

Здесь и в дальнейшем, если нет специальной оговорки, мы будем полагать, что аргументы, указанные в задании, уже записаны в одноименных ячейках в плавающей форме.

Программа вычислений по схеме Горнера имеет следующий вид:

Программа 4.7

1) $x - 2_N = y$
2) $x \cdot y = y$
3) $y - 7_N = y$
4) $x \cdot y = y$
5) $y + 8_N = y$
6) $x \cdot y = y$
7) $y + 1_N = y$
8) *стоп.*

Заметим, что справедлива формула

$$x^4 - 2x^3 - 7x^2 + 8x + 1 = \\ = (x+2)(x-3)[(x+2)(x-3)+4] - 11.$$

Поэтому можно составить более короткую программу.

Программа 5.7

1) $x + 2_N = R_1$
2) $x - 3_N = y$
3) $R_1 \cdot y = R_1$
4) $R_1 + 4_N = y$
5) $R_1 \cdot y = y$
6) $y - 11_N = y$
7) *стоп.*

Правда, по сравнению с программой 4.7 здесь используется лишняя рабочая ячейка.

Пример 3.7. Составить программу вычисления по формуле

$$y = \frac{x^3 + 3x^2 + 6x - 5}{x^3 - 3x^2 + 6x + 5}.$$

Формулы такого типа часто получаются в результате неудачных алгебраических преобразований исход-

ных формул. В данном случае, очевидно, числитель и знаменатель получаются из величин

$$A = x^3 + 6x,$$
$$B = 3x^2 - 5.$$

Мы имеем следующую программу:

Программа 6.7

1) $x \cdot x = R_1$
2) $3_N \cdot R_1 = y$
3) $y - 5_N = B$
4) $R_1 + 6_N = R_1$
5) $x \cdot R_1 = A$
6) $B + A = R_1$
7) $A - B = y$
8) $R_1 : y = y$
9) *стоп.*

Пример 4.7. Составить программу для вычисления величин p, q, r, заданных формулами:

$$p = \frac{1+x}{1-x}, \quad q = p^2 - 2p + 2, \quad r = \frac{p+2q}{3p+q}. \qquad (2)$$

Программа 7.7

1) $1_N + x = R_1$
2) $1_N - x = p$
3) $R_1 : p = p$
4) $p - 2_N = q$
5) $p \cdot q = q$
6) $q + 2_N = q$
7) $2_N \cdot q = R_1$
8) $p + R_1 = R_1$
9) $3_N \cdot p = r$
10) $q + r = r$
11) $R_1 : r = r$
12) *стоп.*

Пример 5.7. Составить программу для нахождения наибольшего из двух данных чисел

$$y = \max(x_1, x_2). \tag{3}$$

Заметим, что

$$\max(x_1, x_2) = \frac{1}{2}[(x_1 + x_2) + |x_1 - x_2|].$$

По последней формуле составляется

Программа 8.7

1) $x_1 + x_2 = R_1$
2) $x_1 - x_2 = y$
3) $|y| - |0| = y$
4) $R_1 + y = y$
5) $\left(\frac{1}{2}\right)_N \cdot y = y$
6) *стоп*.

§ 8. Разветвляющиеся программы

Простейшим примером разветвляющейся программы является программа вычисления функции, заданной разными формулами на разных участках изменения аргумента.

Пример 1.8. Составить программу вычисления напряженности H электростатического поля, создаваемого зарядом, равномерно распределенным по поверхности шара радиуса r_0.

Как известно, в этом случае напряженность электростатического поля H задается формулами:

$$H = \begin{cases} \dfrac{k}{r^2} & \text{при } r \geqslant r_0, \quad (1) \\ \dfrac{kr}{r_0^3} & \text{при } r < r_0. \quad (2) \end{cases}$$

Здесь k — коэффициент, определяемый величиной заряда и принятыми единицами измерения напряженности, заряда и длины.

Очевидно, в начале программы надо сравнить величины r и r_0, а затем в зависимости от результатов срав-

нения производить вычисления по формуле (1) или (2). Таким образом, получается разветвляющаяся программа.

Программа 1.8

1) $r - r_0 = 0$
2) (1) ——————┐
3) $r \cdot r = H$
4) $k : H = H$
5) *Б* ——————┐
6) $\frac{k}{r_0^3} \cdot r = H$
7) *стоп.*

В этой программе, кроме команд со стандартной передачей управления, имеется команда с условной передачей управления, при помощи которой осуществляется разветвление программы. В левой части программы стрелкой указано, куда должно перейти управление, если значение признака ω совпадает со значением, указанным в команде условной передачи управления. В противном случае управление переходит к следующей команде (так в дальнейшем будет называться команда, расположенная в ячейке со следующим номером).

Если $r \geqslant r_0$, то в результате выполнения команды 1) выработается управляющий сигнал $\omega = 0$ и управление перейдет к команде 3) и затем 4). На место команды 5) можно было бы поставить команду *стоп*. Однако в программе лучше избегать двух различных концов. С этой целью мы помещаем сюда команду передачи управления команде 7), которая и остановит машину.

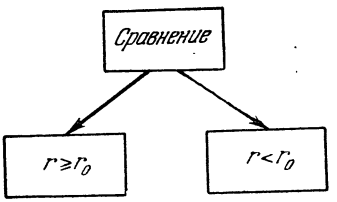


Рис. 17.

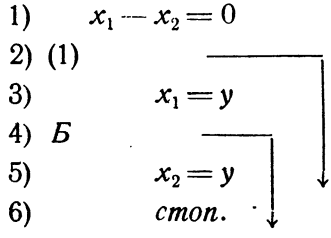
Отметим еще, что ячейку, в которой лежит константа, равная k/r_0^3, мы так и обозначаем в левой части.

Работу программы 1.8 удобно проиллюстрировать схемой программы (рис. 17). В ней прямоугольниками

условно обозначены части программы, осуществляющие алгоритмы, из которых складывается полное решение задачи. Линиями показаны переходы от одной части программы к другой.

Пример 2.8. Составить разветвляющуюся программу для нахождения наибольшего из двух заданных чисел x_1 и x_2 (ср. пример 5.7).

Программа 2.8



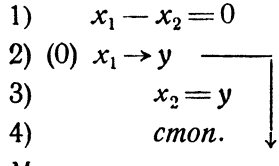
В левой части этой программы использовано сокращенное обозначение

$$x_1 = y$$

вместо $0 \vee x_1 = y$.

Используя совмещение пересылки с командой условной передачи управления, можно сократить программу на две ячейки:

Программа 3.8



Можно составить и неразветвляющуюся программу аналогично программе 5.7, однако она была бы гораздо длиннее программы 3.8.

Пример 3.8. Составить программу упорядочения двух заданных чисел x_1 и x_2, в результате работы которой в ячейки y_1 и y_2 будут положены числа:

$$y_1 = \max(x_1, x_2),$$
$$y_2 = \min(x_1, x_2).$$

Программа 4.8

1) $x_1 - x_2 = 0$
2) (0) $x_1 \to y_1$
3) $x_2 \to y_1$
4) $x_2 \to y_2$
5) $x_1 = y_2$
6) *стоп.*

Схема этой программы по существу совпадает со схемой программы 3.8.

Пример 4.8. Составить программу упорядочения трех заданных чисел x_1, x_2, x_3. В результате работы этой

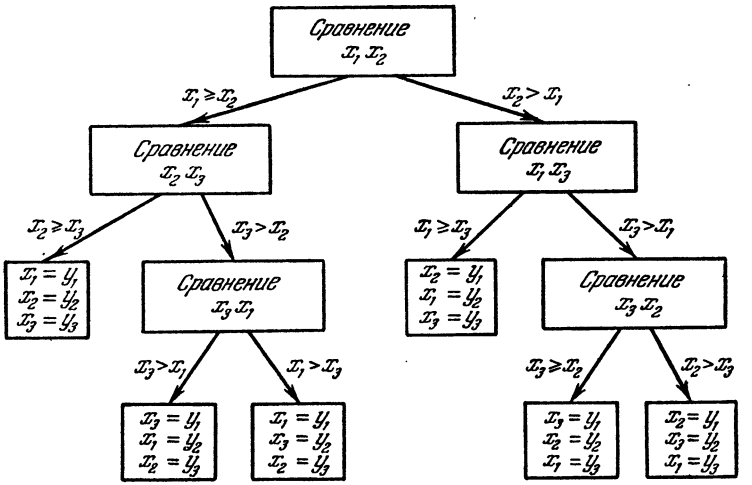


Рис. 18.

программы в ячейки y_1, y_2, y_3 должны быть занесены числа x_1, x_2, x_3 в порядке убывания.

Для составления этой программы удобно предварительно составить ее схему. Будем сначала сравнивать числа x_1 и x_2, а затем меньшее из них с числом x_3.

Схема показана на рис. 18.

По этой схеме легко составить программу:

Программа 5.8

```
    1)      x1 − x2 = 0
    2) (1)          B
A   3)      x2 − x3 = 0
    4) (1)
    5)          x2 = y2
    6)          x3 = y3
    7)          x1 = y1
    8)       стоп
    9)          x2 = y3
   10)      x1 − x3 = 0
   11) (1)
   12)      x3 → y2;  7)
   13)          x1 = y2
   14)      x3 → y1;  8)
B  15)      x1 − x3 = 0
   16) (1)
   17)          x1 = y2
   18)          x3 = y3
   19)      x2 → y1;  8)
   20)          x1 = y3
   21)      x2 − x3 = 0
   22) (1)
   23)      x3 → y2;  19)
   24)      x2 → y2;  14)
```

В этой программе после первой команды

$$x_1 - x_2 = 0$$

происходит разветвление на две большие части.

Части эти названы A и B, и так же названы первые команды этих частей. Таким образом, введены дополнительные обозначения для некоторых команд (кроме номеров).

Необходимо заметить, что нумерация команд в левой части не обязательна. Что же касается этих новых

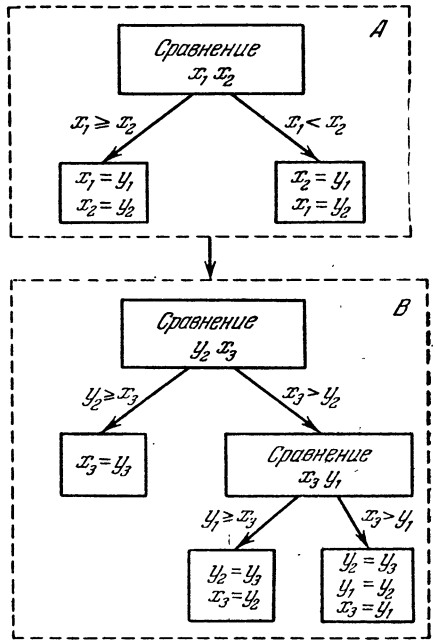
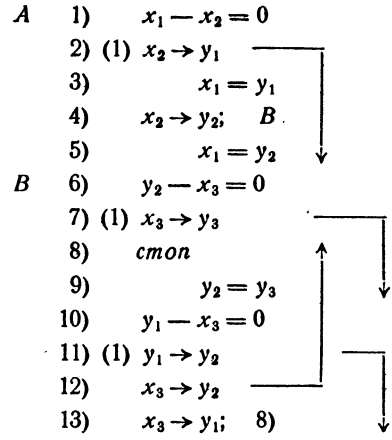


Рис. 19.

обозначений, то, используя их, можно сразу, до того как написана вся программа, написать в левой части команду 2).

Программу 5.8 можно несколько сократить, используя совмещение пересылки с условной передачей управления. Можно составить и более простую схему программы, если несколько изменить алгоритм. Пусть сначала производится упорядочение чисел x_1 и x_2, а затем с упорядоченными числами сравнивается x_3 (рис. 19).

Программа 6.8



Здесь часть A — упорядочение x_1 и x_2, а также первая команда этой части программы; B — сравнение x_3 с упорядоченной парой первых двух чисел и первая команда этой части программы.

Пример 5.8. Составить программу для вычисления величины y, заданной следующим образом:

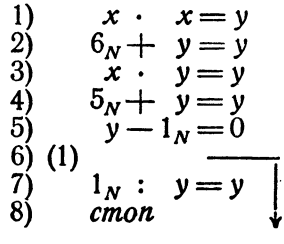
$$y = x^3 + 6x + 5,$$

если $x^3 + 6x + 5 < 1$,

$$y = \frac{1}{x^3 + 6x + 5},$$

если $x^3 + 6x + 5 \geqslant 1$.

Программа 7.8



Во всех приведенных примерах некоторые команды программы выполняются или не выполняются в зависимости от результатов сравнения исходных величин или величин, полученных в процессе работы программы. Это является характерным свойством разветвляющихся программ.

§ 9. Циклы без переменных команд

Ознакомимся теперь с примерами программ, в которых несколько раз производятся вычисления по одной и той же последовательности команд.

Пример 1.9. Составить программу для вычисления по формуле

$$S=1+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{N^2}.$$

Пусть в ячейке S уже лежит сумма первых $n-1$ членов

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

а в ячейке n — соответствующее число; тогда, чтобы добавить следующий член, нужно выполнить команды:

$$1_N : n = R_1,$$
$$R_1 \cdot R_1 = R_1,$$
$$R_1 + S = S.$$

Эти команды составляют *рабочую часть* программы; они должны быть выполнены $N-1$ раз.

Чтобы получить следующее значение n из предыдущего, нужно выполнить команду

$$n+1_N=n,$$

которую естественно назвать командой *изменения*. Для правильной работы программы необходимо в ее начале придать ячейкам n и S первоначальное состояние. В процессе работы программы содержимое этих ячеек будет изменяться, но программа должна быть написана так, чтобы мы могли любое число раз начинать ее

сначала; для этого требуется, чтобы начальное состояние этих ячеек всякий раз заново восстанавливалось. По этой причине команды, приводящие ячейки n и S в их первоначальное состояние, мы будем называть командами *восстановления*.

Далее, после рабочей части необходимо проверить, следует ли повторять цикл еще раз или его выполнение закончено. Это — *проверка окончания* цикла. Таким образом, программу можно записать в следующем виде.

Программа 1.9

1)	$1_N = n$	восстановление
2)	$1_N = S$	
3)	$n + 1_N = n$	изменение
4)	$1_N : n = R$	рабочая часть
5)	$R \cdot R = R$	
6)	$R + S = S$	
7)	$n - N = 0$	проверка окончания
8)	(1)	
9)	*стоп.*	

Легко заметить, что этой программой можно пользоваться при любом N, кроме $N = 1$; в последнем случае она дала бы неверный результат. Это вполне естественно, так как фактически первый член суммы этой программой не вычисляется, а засылается в ячейку S готовым. Если мы хотим вычислять все слагаемые, начиная с первого, то при восстановлении в ячейку S следует засылать не единицу, а нуль.

Восстановление значения n тесно связано с расположением остальных частей программы. Естественно было бы начинать со значения $n = 1$. Но в программе 1.9 этого сделать нельзя (при $S = 0$), так как изменение идет сразу за восстановлением и рабочая часть программы выполняется уже при $n = 2$. Поэтому для правильной работы программы нужно начинать со значения $n = 0$, т. е. предшествующего тому, с которого фактически начинается работа цикла.

Можно было бы начинать с $n = 1$, но переставить команды программы так, чтобы изменение шло после рабочей части. Тогда придется изменить команду проверки окончания. Действительно, проверка окончания будет тогда производиться до того, как вычислен член, соответствующий данному значению n, и последнее слагаемое не сосчитается. В качестве эталона для проверки окончания нужно будет брать не число N, а число $N + 1$.

Обоих этих изменений легко избежать, если оставить изменение впереди рабочей части, но обходить его на первом шагу. Тогда получится такая программа:

Программа 2.9

1)	$0 = S$	}	восстановление
2)	$1_N \to n$	}	
3)	$n + 1_N = n$		изменение
4)	$1_N : n = R$	}	рабочая часть
5)	$R \cdot R = R$	}	
6)	$S + R = S$	}	
7)	$n - N = 0$	}	проверка окончания
8)	(1)	}	
9)	*стоп*	}	

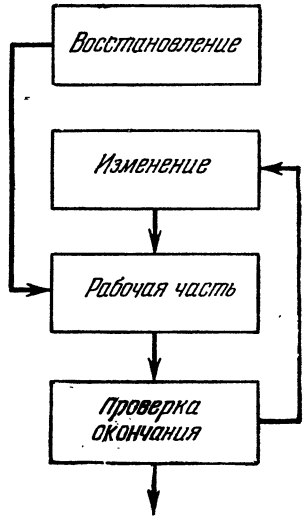


Рис. 20.

Здесь мы имеем обычную схему цикла, которую можно в общем виде изобразить так, как это показано на рис. 20.

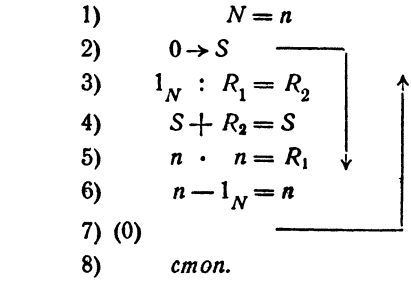
Из сказанного выше ясно, что восстановление всегда должно стоять перед циклом. Расположение остальных его частей может изменяться. Например, можно не менять восстановление в программе 1.9, но она будет давать верный результат и при $N = 1$, если сделать проверку окончания сразу после восстановления.

Программа 3.9

1)	$1_N = n$	
2)	$1_N \to S$;	7)
3)	$n + 1_N = n$	
4)	$1_N : n = R$	
5)	$R \cdot R = R$	
6)	$S + R = S$	
7)	$n - N = 0$	
8)	(1)	3)
9)	*стоп*.	

Рассмотренные программы можно сократить, если начать суммирование не с первого, а с последнего члена.

Программа 4.9



Здесь после восстановления управление сразу передается в середину рабочей части. Вычисление k-го члена и его добавление в S происходят, когда в ячейке n стоит уже $k-1$. Программа начинает работу со значения $n = N$, и затем значение n уменьшается на единицу при каждом повторении рабочей части. При значении $n = 0$ все еще вырабатывается $\omega = 0$ и управление передается в начало рабочей части (команда 3)), которая вычислит и прибавит в S член, соответствующий $n = 1$. Следующее вычитание приведет уже к отрицательной разности, т. е. выработается сигнал $\omega = 1$. В результате этого команда 7) передает управление команде 8), которая и остановит машину.

В программе 4.9 команда 6) служит одновременно и командой изменения и командой проверки окончания. Таким образом, деление цикла на части носит, до известной степени, условный характер.

Заметим, что в программе 4.9 нельзя в качестве рабочей ячейки n использовать константу N или вводить значение N прямо в ячейку n. В процессе работы программы содержимое ячейки n меняется и константа N будет испорчена, так что программу нельзя будет повторить еще раз, не введя ее в машину заново, т. е. она перестанет быть *самовосстанавливающейся*.

Ячейку, к содержимому которой прибавляется (или вычитается) в цикле некоторое число и по которой проверяется окончание цикла, мы будем называть *счетчиком числа циклов* или просто *счетчиком*.

Пример 2.9. Составить программу для вычисления полинома Чебышева $y = T_N(x)$, используя рекуррентную формулу

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x),$$

причем $T_0(x) = 1$, $T_1(x) = x$.

Пусть нами уже вычислено T_n в ячейке y и T_{n-1} в ячейке R_0; тогда T_{n+1} может быть вычислено по программе

$$\begin{aligned} 2x \cdot y &= R_1 \\ R_1 - R_0 &= R_1 \\ y &= R_0 \\ R_1 &= y. \end{aligned}$$

Для того чтобы таким образом сосчитать T_1 при $y = T_0 = 1$, в R_0 должно первоначально стоять x. Программа выглядит так:

Программа 5.9

1)	$2_N \cdot x = R_2$		восстановление
2)	$N = n$		
3)	$x = R_0$		
4)	$1_N = R_1$		
5)	$n - 1_N = n$		изменение и проверка
6)	(1) $R_1 \to y$		окончания
7)	$R_2 \cdot y = R_1$		рабочая часть
8)	$R_1 - R_0 = R_1$		
9)	$y \to R_0$		
10)	*стоп.*		

В этой программе проверка окончания стоит перед рабочей частью уже и формально.

Особенностью этой программы по сравнению с программой 4.9 является и то, что ячейка n здесь не участвует в вычислениях по формуле, а служит только счетчиком числа циклов.

Так как число n записано в ячейке в обычном виде с плавающей запятой, то его можно назвать *плавающим* счетчиком. На практике часто применяются также и *фиксированные счетчики*, в которых число задается как число единиц того или иного адреса, причем единицы адреса рассматриваются как целые числа. В ряде случаев фиксированный счетчик может занимать более чем один адрес.

Задать число N фиксированно в третьем адресе означает задать ячейку, в коде и первом и втором адресах которой нули, а в третьем — число N. Например, при $N=37_8(=31_{10})$ содержимое ячейки будет выглядеть так:

00 0000 0000 0037

Иногда, задавая число N в фиксированной форме, мы будем говорить, что N задано адресно. Ячейку, в которой N задано в третьем адресе, будем обозначать $(0, 0, N)$.

При проверке окончания нужно из счетчика вычитать ячейку, содержащую единицу в самом младшем разряде третьего адреса. Пользуясь тем же способом обозначений, такую ячейку надо обозначать $(0, 0, 1)$. Вычитание производится с помощью фиксированного действия вычитания адресных частей (мантисс).

Программа вычисления полинома Чебышева с использованием фиксированного счетчика имеет следующий вид:

Программа 6.9

1)		$2_N \cdot$	x	$=R_2$
2)			$(0, 0, N)$	$=n$
3)			x	$=R_0$
4)			1_N	$=R_1$
5)		$n -$,	$(0, 0, 1)$	$=n$
6)	(1)	$R_1 \rightarrow$	y	
7)		$R_2 \cdot$	y	$=R_1$
8)		$R_1 \cdot$	R_0	$=R_1$
9)		$y \rightarrow$	R_0	
10)		*стоп.*		

На практике пользуются счетчиками как в фиксированном, так и в плавающем виде.

Пример 3.9. Вычислить

$$S=\sin x=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots$$

Если сумма ряда до члена $a_k = x^n/n!$, лежащего в ячейке R_0, вычислена в ячейке S, то $a_{k+1} = a_k \cdot \frac{(-x^2)}{(n+1)(n+2)}$ и рабочая часть программы вместе с изменением состоит из следующих команд:

$$n + 1_N = R_1$$
$$R_1 + 1_N = n$$
$$n \cdot R_1 = R_1$$
$$(-x^2) : R_1 = R_1$$
$$R_0 \cdot R_1 = R_0$$
$$S + R_0 = S.$$

Поскольку остаток ряда не превосходит величины первого отброшенного члена, то вычисление можно закончить в тот момент, когда прибавление очередного члена ряда к ячейке S не изменит ее содержимого. Такой момент обязательно наступит хотя бы потому, что величина $x^n/n!$ при достаточно большом n станет меньше, чем наименьшее по модулю число, которое можно записать в машине. В таком случае машина будет воспринимать это число как нуль, а следовательно, его прибавление не изменит ячейки S. Однако и до этого число $x^n/n!$ может стать настолько малым по модулю по сравнению с S, что $S + x^n/n!$ будет с машинной точностью совпадать с S. Окончательный вид программы:

Программа 7.9

1)	$x \cdot x = R_2$	
2)	$0 - R_2 = R_2$	
3)	$x = S$	
4)	$x = R_0$	
5)	$1_N = n$	
6)	$n + 1_N = R_1$	↑
7)	$R_1 + 1_N = n$	|
8)	$n \cdot R_1 = R_1$	|
9)	$R_2 : R_1 = R_1$	|
10)	$R_0 \cdot R_1 = R_0$	|
11)	$S + R_0 = R_1$	|
12)	$R_1 + S = 0$	|
13)	(0) $R_1 \to S$	———
14)	*стоп.*	

В этом примере, в отличие от предыдущих, число повторений в цикле не фиксировано заранее, а выясняется в процессе выполнения программы. Здесь мы продолжали вычисления до получения результата с машинной точностью. В других случаях такая точность может оказаться недостижимой или не вызываться необходимостью. Весьма часто результат нужен с точностью намного меньшей, чем позволяют алгоритм и разрядность машины. Продолжение счета после достижения нужной точности является бесцельной тратой машинного времени.

Пример 4.9. Найти с точностью 10^{-4} корень уравнения $x^4 - 10x^3 + 1 = 0$, лежащий на отрезке [0, 1].

Будем искать корень методом Ньютона. В точке $x=1$ имеем $f(1) = -8$, $f''(1) = -48$. Так как $f(1)f''(1) > 0$ и $f'(x) = 4x^3 - 30x^2$, $f''(1) = 12x^2 - 60x$ не обращаются в нуль на интервале (0, 1), то метод Ньютона приложим и за начальное приближение можно взять $x_0 = 1$. Вычисления производим по формуле

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

где

$$f(x_n) = x_n^3(x_n - 10) + 1, \qquad f'(x_n) = 4x_n^3 - 30x_n^2.$$

Счет закончим, когда разность между последующим и предыдущим значениями аргумента станет по модулю меньше $0{,}1 \cdot 10^{-4}$ (мы считаем, что это обеспечит заданную точность в нахождении корня).

Вычисления можно производить по программе 8.9, к которой нужно еще присоединить требуемые константы.

Программа 8.9

	1)	$1_N = x$	
A	2)	$x \cdot x = R_0$	$f(x_n)$
	3)	$x \cdot R_0 = R_1$	
	4)	$x - 10_N = R_2$	
	5)	$R_1 \cdot R_2 = R_2$	
	6)	$R_2 + 1_N = f$	

Продолжение программы 8.9

7) $4_N \cdot R_1 = R_1$
8) $30_N \cdot R_0 = R_0$ } $f'(x_n)$
9) $R_1 - R_0 = f'$
10) $f : f' = R_1$
11) $x - R_1 = R_0$
12) $|R_1| - |0{,}1 \cdot 10^{-4}| = 0$
13) (0) $R_0 \to x$; A
14) *стоп.*

Вообще окончание цикла при достижении желаемой точности характерно для итеративных процессов.

В заключение приведем два примера программ, в которых рабочая часть цикла содержит не только арифметические, но и логические действия.

Пример 5.9. Составить программу вычисления

$$y = x^N,$$

предполагая, что число N задано адресно, т. е. как фиксированное целое число, записанное в ячейке, которую мы тоже будем обозначать N.

Легко составить программу, вычисляющую x^N последовательным умножением на x:

Программа 9.9

1) $N = n$
2) $1_N \to$ y
3) $y \cdot$ $x = y$
4) $n -$, (0, 0, 1) $= n$
5) (0)
6) *стоп.*

Вычисление по этой программе требует выполнения цикла N раз.

Возможна более быстрая, хотя и более длинная программа. Она основана на том, что для получения x^N нет нужды получать все предыдущие степени.

Так как показатель степени может быть представлен в виде суммы степеней двойки, то и степень x^N может быть получена как произведение различных степеней x, каждый из показателей степени которых есть степень двух.

В самом деле, пусть целое число N представлено в виде

$$N = 1 \cdot \delta_1 + 2\delta_2 + 2^2\delta_3 + \ldots + 2^{k-1}\delta_k,$$

где δ_i есть соответствующая двоичная цифра, т. е. нуль или единица. Тогда, очевидно,

$$x^N = x^{\delta_1} \cdot x^{2\delta_2} \cdot x^{2^2\delta_3} \ldots x^{2^{k-1}\delta_k}.$$

Рассмотрим последовательность

$$a_i = x^{2^{i-1}} \quad (i = 1, 2, \ldots).$$

Легко заметить, что

$$a_1 = x,$$
$$a_{i+1} = a_i^2.$$

Теперь для последовательности $\{y_i\}$, члены которой определены формулой

$$y_i = x^{\delta_1} \cdot x^{2\delta_2} \ldots x^{2^{i-1}\delta_i},$$

находим

$$y_{i+1} = \begin{cases} y_i, & \text{если } \delta_{i+1} = 0, \\ y_i a_i, & \text{если } \delta_{i+1} = 1, \end{cases}$$

а искомое значение

$$y = x^N = y_k.$$

Таким образом, можно получить искомое значение x^N, вычисляя элементы последовательностей $\{a_i\}$ и $\{y_i\}$.

Остается вспомнить, что число N было задано в фиксированной форме. Поэтому его двоичная цифра δ_i есть содержимое i-го разряда ячейки N, а k — номер разряда ячейки N, содержащего старшую единицу. Следовательно, для получения очередного значения y_{i+1} нужно выяснять, есть ли единица в данном $(i+1)$-ом разряде ячейки N.

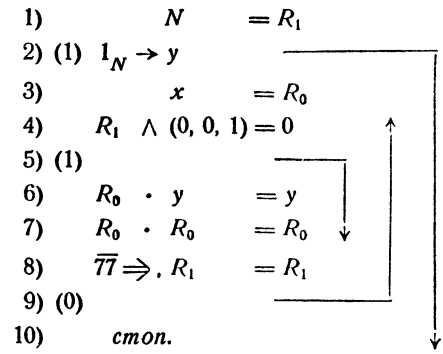
Чтобы проверить значение i-го разряда ячейки N, будем сдвигать содержимое этой ячейки вправо на один разряд. Тогда на i-м шаге на последнем месте окажется как раз i-й разряд и для проверки его значения достаточно пересечь сдвинутую ячейку с константой (0, 0, 1).

В результате логического умножения получится чистый нуль, если в последнем разряде был нуль, и (0, 0, 1), если в последнем разряде была единица. В соответствии с правилом выработки управляющего сигнала при логических действиях во втором случае мы получим $\omega = 0$, а в первом — $\omega = 1$. Следовательно, в этом месте программу можно разветвить, умножая или не умножая y_i на a_i для получения y_{i+1}.

Проверку окончания будем осуществлять по состоянию сдвигаемой ячейки, в которой первоначально было записано число N. После k шагов в этой ячейке останется нуль и выработается $\omega = 1$.

Программа, построенная на описанном алгоритме, имеет вид:

Программа 10.9



Команды 1), 2), 3) — команды восстановления. В ячейке y накапливается результат y_i, в ячейке R_0 лежит a_i, в ячейке R_1 первоначально находится показатель степени N. Если $N = 0$, то по команде 1) выработается сигнал $\omega = 1$ и команда 2) передаст управление концу программы, заслав в ответ единицу ($y = 1$). Команды 4)—8) образуют рабочую часть цикла. По команде 8) одновременно вырабатывается сигнал ω для проверки окончания цикла. После выполнения цикла k раз (k — номер разряда, в котором стоит старшая единица) по команде 8) ячейка R_1 станет нулем и выработается сигнал $\omega = 1$.

Хотя написанная таким способом программа длиннее программы 9.9 и содержит больше команд, повторяющихся в цикле (6 против 3), однако число этих повторений $k = [\log N]$ гораздо меньше и общее время вычисления по программе 10.9 меньше при $N > 4$.

Заметим, что в этом примере рабочая часть цикла представляет собой разветвляющуюся программу. Как будет показано в дальнейшем, рабочая часть может иметь сколь угодно сложную структуру.

Пример 6.9. Найти число единиц в мантиссе ячейки x (т. е. число разрядов мантиссы, в которых стоит единица).

Задачу можно решать аналогично предыдущей: двигать ячейку x вправо на разряд, пока в ней не останется нуль, после каждого сдвига проверять, есть ли единица в младшем разряде, и каждый раз, когда значение младшего разряда будет единица, добавлять единицу в соответствующий счетчик. Рекомендуем читателю самостоятельно написать такую программу. Здесь мы рассмотрим программу, построенную на иной основе *).

Пусть k — номер младшего из отличных от нуля разрядов ячейки x. Посмотрим, что получится после выполнения команды

$$x —, (0, 0, 1) = R_0.$$

*) Излагаемый ниже прием принадлежит Р. Иоффе.

Все разряды R_0 старше k-го совпадут с соответствующими разрядами ячейки x, в k-ом разряде ячейки R_0 будет стоять 0, а во всех разрядах младше k-го — единица. Например,

$$\begin{array}{lll} & \ldots\ 010\ 010\ 101\ 000 & \\ (0,\ 0,\ 1) & \underline{\qquad\qquad\qquad\qquad 1} & (k=4) \\ & \ldots\ 010\ 010\ 100\ 111 & \end{array}$$

(приведены только значения разрядов 3-го адреса).

Если теперь пересечь ячейки x и R_0, т. е. выполнить команду

$$x \wedge R_0 = R_1,$$

то в ячейке R_1 разряды старшие k-го совпадут с соответствующими разрядами ячейки x, так как они у ячеек x и R_0 одинаковы, в k-м разряде R_1 будет нуль, так как нуль стоит в k-м разряде R_0, во всех разрядах, младших k-го, появятся нули, так как нули были в соответствующих разрядах x:

$$\begin{array}{ll} x & \ldots\ 010\ 010\ 101\ 000 \\ R_0 & \ldots\ 010\ 010\ 100\ 111 \\ \hline R_1 & \ldots\ 010\ 010\ 100\ 000 \end{array}$$

Таким образом, мантисса R_0 отличается от мантиссы x только тем, что в ней на месте младшей единицы x стоит нуль. Мы можем повторять этот процесс дальше, добавляя каждый раз в ответ единицу, и закончим его, когда в результате пересечения получится нуль.

Программа 11.9

1) $x = R_0$
2) (1) $0 \rightarrow y$
3) $y + 1_N = y$
4) $R_0 -, (0, 0, 1) = R_1$
5) $R_0 \wedge R_1 = R_0$
6) (0)
7) *стоп.*

Команда 2) осуществляет восстановление ячейки y и, если в мантиссе x — нуль, сразу передает действие в конец. Мы предполагаем, что код x нулевой. Если это не так, программа будет работать неверно, так как команда 5) никогда не выработает сигнал $\omega = 1$. В этом случае надо вместо команды 1) поставить команду пересечения x с ячейкой, в коде которой нуль, а во всех разрядах мантиссы — единицы. Такую ячейку мы будем называть (F, F, F). Первая команда выглядит тогда так:

$$x \wedge (F, F, F) = R_0.$$

§ 10. Переадресация

Пусть в ячейке m лежит команда

$$m)\ a_1 + b_1 = c_1.$$

Посмотрим, что произойдет с этой командой, если к ячейке m прибавить фиксированно ячейку, в младшем разряде среднего адреса которой стоит единица, а во всех остальных разрядах ячейки — нули. Аналогично предыдущему, такую ячейку мы будем изображать (0, 1, 0); точно так же, запись (3, 10, 15) означает, что в младших разрядах первого адреса записано 3_8, в младших разрядах второго адреса 10_8, в младших разрядах третьего 15_8. Итак, мы хотим выяснить, что окажется в ячейке m после выполнения команды

$$m +,\ (0, 1, 0) = m.$$

Код, первый и третий адреса ячейки m не изменятся, а ко второму адресу добавится единица. Если считать, что b_2 — ячейка с адресом на единицу большим, чем b_1, то после сложения команда в m будет выглядеть так:

$$m)\ a_1 + b_2 = c_1.$$

Изменение команды путем прибавления (или вычитания) к ее адресам некоторых чисел мы будем называть *переадресацией* этой команды, а ячейку, в адресах которой записаны прибавляемые числа, — *константой переадресации*. Переадресация всегда имеет вид

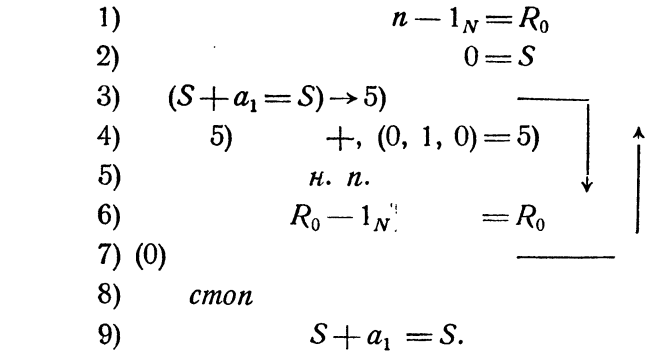
$$m +,\ A = m.$$

Мы сейчас произвели переадресацию ячейки m с константой переадресации (0, 1, 0).

Пример 1.10. Числа $a_1, a_2, \ldots, a_n$ расположены подряд (т. е. номер ячейки a_{k+1} на единицу больше номера ячейки a_k). Поместить их сумму в ячейку S.

Здесь команда 3) есть команда восстановления — она засылает команду $S + a_1 = S$ на место команды 5); в дальнейшем эта команда будет переадресована, как только управление перейдет к команде 4). Разумеется, команда $S + a_1 = S$, называемая *восстановителем*, должна быть записана в какую-либо ячейку заранее. Обычно ее пишут вместе с программой (команда 9).

Программа 1.10



Значок *н. п.* (не перфорировать) у команды 5) означает, что при вводе программы в машину в эту ячейку можно не вводить ничего (или ввести что угодно). Лучше всего ввести в эту ячейку команду *стоп,* чтобы в случае, если пропущено восстановление этой команды, получить на ней стоп. В левой части часто для себя пишут начальное состояние команды.

Иногда восстановитель помещают следом за командой восстановления, поскольку это место все равно обходится. В данной программе команду 9) можно было бы написать на место команды 4), сдвинув все следующие на одну строку вниз. Тогда команда 3) передавала бы управление команде 6), а команда 8) (вместо нынешней 7)) — команде 5).

В счетчик R_0 первоначально засылается число $n-1$, так как проверка окончания стоит после рабочей части, а число повторений цикла равно n. Как и в случае арифметических циклов, счетчик может быть не только плавающим, как в нашем примере, но и фиксированным. После n вычитаний, следовательно, после выполнения цикла n раз, в R_0 окажется отрицательное число, выработается $\omega = 1$ и управление перейдет к команде 8).

Рассмотренный пример является простейшим примером цикла с переадресацией. Схема цикла с переадресацией не отличается от рассмотренной в § 9 схемы цикла без переменных команд; лишь на месте

изменения (или вместе с ним) стоит переадресация (рис. 21).

Восстановление в цикле с переадресацией обязательно содержит засылку команды, которая в цикле будет изменяться. Нельзя вместо этого заранее написать в ячейку, где лежит переадресуемая команда, ее первоначальное состояние. Хотя в этом случае программа будет выполнена верно и мы сэкономим две ячейки (команду и константу восстановления), переменная команда после работы окажется испорченной и выполнить программу второй раз, не вводя ее снова в машину, невозможно. Не помогает здесь и восстановление после цикла, так как при случайном сбое или остановке машины во время выполнения цикла придется вновь вводить программу.

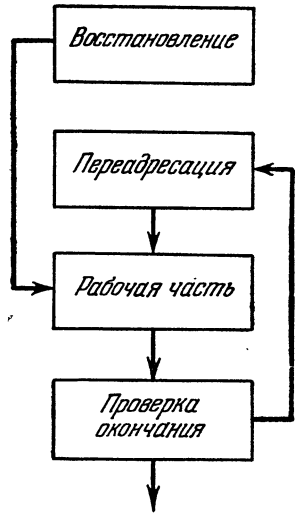


Рис. 21.

Команда 3) в рассмотренном примере 1.10 потому и называется командой восстановления, что ее наличие в программе обеспечивает во всех случаях при обращении к этой программе начальное состояние переменной команды 5).

Приведенный порядок частей в цикле с переадресацией, как и в цикле без переадресации, не является принудительным (кроме восстановления, о б я з а т е л ь н о стоящего в начале). Так, переадресацию часто ставят после рабочей части перед проверкой окончания. Все же программисту следует придерживаться какой-либо одной последовательности частей, если нет специальных причин изменять ее.

П р и м е р 2.10. Вычислить

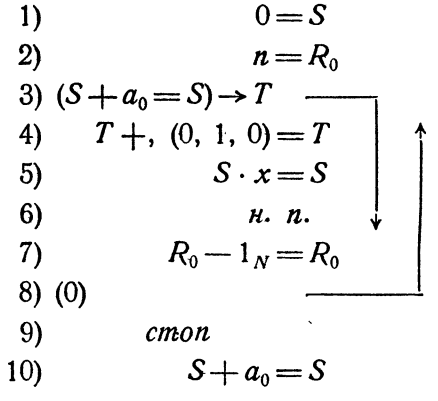
$$S=a_0x^n+a_1x^{n-1}+\ldots a_n,$$

если коэффициенты a_0, a_1, a_2, ..., a_n расположены в ячейках, идущих подряд.

Программа 2.10

Вычисления можно произвести по схеме Горнера. Программа будет выглядеть так:

Программа 3.10

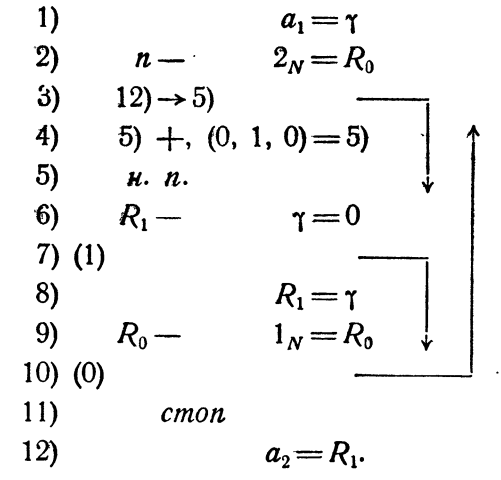


Сравнив написанные программы с программами § 7 (пример 1.7), видим, что при $n \leqslant 4$ бесцикловая про-

грамма не длиннее цикловой и работает существенно быстрее, о чем нельзя забывать. Однако при больших n бесцикловая программа неприемлема. Кроме того, большим достоинством программы с циклом является то, что ею без всяких изменений можно пользоваться при любых n. Этот факт приобретает большое значение, если многочлены в программе приходится считать много раз при различных n (напоминаем, что *стоп*, стоящий в конце программы, чисто условен; вместо него может стоять передача управления в нужное место общей программы).

Пример 3.10. В ячейках $a_1, a_2, \ldots, a_n$, расположенных подряд, записаны числа. Найти среди них наибольшее и положить его в ячейку γ.

Программа 4.10



Пример 4.10. Составить программу скалярного умножения векторов. В ячейках $a_1, a_2, \ldots, a_n$, расположенных подряд, и в ячейках $b_1, b_2, \ldots, b_n$, также расположенных подряд, лежат числа (координаты векторов).

Вычислить сумму

$$S=a_1b_1+a_2b_2+\ldots+a_nb_n.$$

Программа 5.10

	1)	n —	1_N	$=R_0$
	2)		0	$=S$
	3)	$A^0\rightarrow$	A	
	4)	A +,	$(1, 1, 0)$	$=A$
A	5)		*н. п.*	
	6)	S +	R_1	$=S$
	7)	R_0 —	1_N	$=R_0$
	8)	(0)		
	9)		*стоп*	
A^0	10)	$a_1\cdot$	b_1	$=R_1$

Цикл с переадресацией может входить в рабочую часть другого цикла (внешнего по отношению к первому). При этом в тех случаях, когда внешний цикл переадресует те же команды, что и внутренний, следует, вообще говоря, переадресовывать константы восстановления внутреннего цикла.

Пример 5.10. Составить программу умножения вектора на матрицу. Числа $a_1, a_2, \ldots, a_n$ лежат подряд. Элементы матрицы

$$\begin{Vmatrix} b_{11} & b_{12} & \ldots & b_{1m} \\ b_{21} & b_{22} & \ldots & b_{2m} \\ \cdot & \cdot & \cdot & \cdot \\ b_{n1} & b_{n2} & \ldots & b_{nm} \end{Vmatrix}$$

расположены в памяти по строкам, т. е. числа b_{11}, $b_{12}, \ldots, b_{1n}$ расположены подряд, следом идут числа $b_{21}, b_{22}, \ldots, b_{2n}$ и т. д. Нужно найти числа

$$\begin{aligned} c_1 &= a_1b_{11} + a_2b_{21} + \ldots + a_nb_{n1}, \\ c_2 &= a_1b_{12} + a_2b_{22} + \ldots + a_nb_{n2}, \\ &\cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ c_m &= a_1b_{1m} + a_2b_{2m} + \ldots + a_nb_{nm} \end{aligned}$$

и положить их также последовательно в ячейки.

Это можно сделать с помощью программы 6.10.

Программа 6.10

```
 1)      m  —      1_N = R_0
 2)                B^0 = B
 3)      A^0 →     R_1
 4)      B  +, (0, 0, 1) = B
 5)      R_1 +, (0, 1, 0) = R_1
 6)      n  —      1_N = R_2
 7)                0   = S
 8)      R_1 →     A
 9)      A  +, (1, n, 0) = A
A 10)              н. п.
11)      S  +      R_3 = S
12)      R_2 —     1_N = R_2
13) (0)
B 14)              н. п.
15)      R_0 —     1_N = R_0
16) (0)
17)      стоп
A^0 18)  a_1 ·     b_11 = R_3
B^0 19)            S = c_1.
```

внутренний цикл

Внутренний цикл этой программы почти в точности совпадает с циклом предыдущего примера, но в качестве восстановителя команды A используется не заранее заготовленная константа, а рабочая ячейка, которая готовится и переадресуется во внешнем цикле.

Пример 6.10. В ячейках $a_1, a_2, \ldots, a_n$ лежат числа. Выбрать из них те, которые больше заданной константы C, и расположить в ячейках начиная с b_1 подряд. Определить, сколько таких чисел.

Программа 7.10

Команды 1), 2), 3) — восстановление. После него действие сразу передается команде 9), вычитающей 1 из счетчика. По команде 10) действие передается началу рабочей части — команде 4) и одновременно в рабочую ячейку R_0 засылается первое (а впоследствии очередное) число a_i. Эта команда переадресуется (командой 8)). Команды 4)—6) образуют рабочую часть программы. При помощи команд 4), 5) проверяется выполнение условия $a_i > c$. Если оно не выполнено, то действие передается на переадресацию (команде 8)), а затем снова на проверку окончания цикла. Если же условие $a_i > c$ выполнено, то число a_i из ячейки R_0 переносится в ячейку b_s командой 6). Эта команда также переадресуется командой 7), но не каждый раз в цикле, а только в тех случаях, когда выполняется команда 6). В конце работы программы команда 11) по состоянию команды B определяет количество чисел, удовлетворяющих условию. Оно определяется как число единиц правого адреса ячейки M.

В левой части на месте засылаемых и изменяемых команд 6) и 10) мы написали их первоначальное состояние для удобства чтения программы. Однако значки *н. п.* справа показывают, что в правой части эти команды надо закодировать как *стоп*.

Из данного примера видно, что рабочая часть цикла с переадресацией может быть разветвляющейся программой и некоторые из переадресаций могут выполняться не на каждом повторении цикла, а в зависимости от условий.

В заключение сделаем одно предостережение. В ряде случаев, когда число повторений цикла с переадресацией известно заранее, имеется возможность уменьшить число команд следующим образом. Запишем команду, которая переадресуется, в том виде, какой она имеет после окончания работы программы, в виде константы. Вместо засылки счетчика и изменения его будем осуществлять проверку окончания, сравнивая эту константу с изменяющейся командой. В тот момент, когда они совпали, нужно окончить цикл.

Несмотря на экономию ячейки (исчезает команда восстановления счетчика), писать цикл таким образом ни в коем случае не следует. Вычисление окончательного состояния изменяемой команды утомляет, превращает организацию цикла из простого почти формального дела в умственную работу и является источником ошибок.

Проверку окончания цикла с фиксированным числом повторений, все равно — известным заранее или вычисленным к моменту начала работы цикла, нужно вести только по счетчику.

ГЛАВА III

ОРГАНИЗАЦИЯ ПРОГРАММЫ

§ 11. Подпрограммы

Подпрограммой мы будем называть часть программы, имеющую самостоятельное значение. В этом смысле все примеры, рассмотренные в предыдущих главах, можно считать подпрограммами, которые могут быть использованы при решении тех или иных задач. Любая программа состоит из некоторого числа подпрограмм и команд, связывающих их. Подпрограммы, входящие в состав данной программы, мы будем называть ее *блоками*.

Как уже говорилось, команда *стоп*, стоящая в конце каждой программы в предыдущих главах, являлась условной. Вместо нее блок должен заканчиваться свободной ячейкой, предназначенной для записи команды передачи управления нужной части программы, что будет сделано самой машиной в процессе работы.

Пример 1.11. Вычислить $z = P(x) + P(y)$, где $P(\alpha) = \alpha^3 - 3\alpha^2 + 2\alpha + 3$, а значения x и y предполагаются записанными в соответствующих ячейках.

Напишем сначала программу вычисления функции $\gamma = P(\alpha) = [(\alpha - 3)\alpha + 2]\alpha + 3$.

Программа 1.11 (счет $P(\alpha)$)

P	$\alpha - 3_N = \gamma$
	$\alpha \cdot \quad \gamma = \gamma$
	$2_N + \gamma = \gamma$
	$\alpha \cdot \quad \gamma = \gamma$
	$3_N + \gamma = \gamma$
KP	*н. п.*

Последняя ячейка, обозначенная KP, оставлена свободной для записи в нее команды передачи управления. Написанная программа будет служить подпрограммой общей программы вычисления z

Программа 2.11 (счет z)

1) $x = \alpha$
2) 3) $\leftrightarrow$ KP ; P
3) $\gamma = R_1$
4) $y = \alpha$
5) 6) $\leftrightarrow$ KP ; P
6) $\gamma + R_1 = z$
7) *стоп.*

Команда 1) пересылает значение x в ячейку α, являющуюся аргументом программы счета $P(\alpha)$.

Особое внимание следует обратить на команду 2). Она передает управление ячейке P, являющейся началом программы вычисления функции. Одновременно команда 2) засылает в ячейку KP, которая служит концом той же программы, команду передачи управления — возврат в программу счета z. Эту команду 2) мы будем называть командой обращения к подпрограмме.

После выполнения команды 2) начнется вычисление по программе 1.11, а в ячейке KP будет лежать команда

$$KP:\ 0 \leftrightarrow 0;\ 3).$$

Когда управление перейдет к ячейке KP, то эта команда передаст управление команде 3) программы 2.11. К этому моменту в ячейке γ будет находиться значение $P(x)$. Команда 3) перешлет это значение в ячейку z для запоминания.

Команды 4) и 5) аналогичны командам 1) и 2). При этом команда 5), как и 2), передает управление ячейке P, а в ячейку KP засылает команду передачи управления к команде 6) основной программы. К моменту выполнения команды (6) в ячейке γ будет находиться значение $P(y)$. Заметим, что команда 3) была необходима, так как без нее вычисленное ранее значение $P(x)$ было

бы безвозвратно потеряно при повторном обращении к программе 1.11.

Наличие у каждой программы своей ячейки, в которую нужно засылать команду передачи управления, не вызывается необходимостью. Значительно удобнее использовать для этой цели одну определенную ячейку, общую для всех подпрограмм. В этом случае каждая подпрограмма должна оканчиваться командой передачи управления этой выделенной ячейке. В дальнейшем такую ячейку мы будем всегда обозначать Ω.

Если воспользоваться этим приемом, то программы примера 1.11 будут выглядеть так, как это показано на рис. 22, где они приведены в закодированном виде. Места для расположения этой программы и ее рабочих ячеек могут быть выбраны совершенно произвольно и независимо друг от друга. В начале работы программы управление должно быть передано первой ячейке программы счета z.

Пример 2.11. Вычислить и напечатать таблицу значений функции

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

на участке от $x=a$ до $x=b$ с шагом Δx.

Подпрограмму счета y в отдельной точке и печати можно записать в виде

Программа 3.11 (*Счет y*)

F	$x \cdot x$	$=x^2$
	x^2+1_N	$=R_1$
	x^2-1_N	$=R_2$
	$R_2 : R_1$	$=y$
Печать	$x \quad 0$	y
Б		Ω.

Теперь можно написать общую программу. Это будет обычный цикл, в качестве рабочей части которого служит обращение к подпрограмме *F*.

Счет z

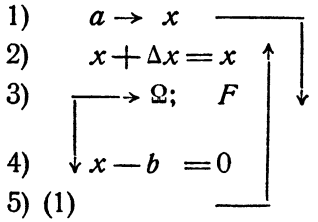
Составил	Задача	Блок		Стр.		Лист	
	Программа 2.11			3500	A	T шифр	№ 2.11.1 № карты
1)	$x=\alpha$	3500	75	0	3600	0140	
2)	3) $\leftrightarrow$ 2; p	1	16	3502	0007	3520	
3)	$\gamma=R_1$	2	75	0	0160	3541	
4)	$y=\alpha$	3	75	0	3601	0140	
5)	6) $\leftrightarrow$ 2; p	4	16	3505	0007	3520	
6)	$\gamma+R_1=z$	5	01	0160	3541	3542	
7)	*стоп*	6	77	0000	0000	0000	

Счет P

				3520	A	T шифр	№ 2.11.2 № карты
	$\alpha-3_N=\gamma$	3520	02	0140	0103	0160	
	$\alpha\cdot\gamma=\gamma$	1	05	0140	0160	0160	
	$2_N+\gamma=\gamma$	2	01	0102	0160	0160	
	$\alpha\cdot\gamma=\gamma$	3	05	0140	0160	0160	
	$3_N+\gamma=\gamma$	4	01	0103	0160	0160	
	Б 2	5	56	0000	0000	0007	

Рис. 22.

Программа 4.11



Стрелка в команде 3) слева заменяет написание здесь команды 4). Команда печати может быть помещена не в подпрограмме счета y, а в основной программе, но она должна выполняться на каждом шагу цикла, так как все вычисления ведутся в тех же ячейках и при каждом следующем обращении к программе F предыдущие результаты теряются. Поэтому они должны быть сначала напечатаны.

Расположение частей программы и рабочих ячеек может быть совершенно произвольным. Однако здесь уже есть некоторое ограничение, состоящее в том, что ячейки x и y должны быть расположены рядом, так как команда печати печатает содержимое группы ячеек подряд.

Нужно иметь в виду, что содержимым ячеек x и y являются числа, записанные в двоичной системе. Поэтому фактически необходимо вместо команды печати иметь обращение к блоку печати, который предварительно переведет числа в десятичную систему. Подробнее этот вопрос будет освещен в следующей главе.

§ 12. Блочное программирование

При написании любой программы ее следует разбивать на отдельные блоки. Каждая ее часть, имеющая сколько-нибудь самостоятельное значение, должна быть выделена как отдельный блок, хотя бы она и состояла всего из нескольких команд. Более того, при вычислении по сравнительно длинной формуле ее полезно разбить на отдельные части, выделив вычисление по каждой из частей в самостоятельный блок.

Рассмотрим пример разбиения программы на блоки.

Пример 1.12. Найти скалярное произведение $A = \boldsymbol{p}_1\boldsymbol{p}_2$ векторов $\boldsymbol{p}_1$ и $\boldsymbol{p}_2$, заданных своими модулями p_1, p_2 и направляющими косинусами $\cos\alpha_1$, $\cos\beta_1$, $\cos\gamma_1$ и $\cos\alpha_2$, $\cos\beta_2$, $\cos\gamma_2$ в некоторой координатной системе.

В этой программе естественно выделить следующие блоки: вычисление декартовых координат первого вектора, вычисление декартовых координат второго вектора и вычисление скалярного произведения. Эти программы выглядят так:

Программа 1.12 (*координаты первого вектора*)

K1 $p_1 \cdot \cos\alpha_1 = x_1$
$p_1 \cdot \cos\beta_1 = y_1$
$p_1 \cdot \cos\gamma_1 = z_1$
Б Ω.

Программа 2.12 (*координаты второго вектора*)

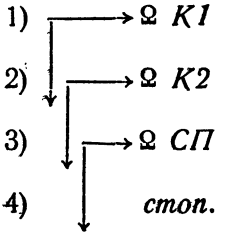
K2 $p_2 \cdot \cos\alpha_2 = x_2$
$p_2 \cdot \cos\beta_2 = y_2$
$p_2 \cdot \cos\gamma_2 = z_2$
Б Ω.

Программа 3.12 (*скалярное произведение*)

СП $x_1 \cdot x_2 = R_1$
$y_1 \cdot y_2 = R_2$
$R_1 + R_2 = A$
$z_1 \cdot z_2 = R_1$
$A + R_1 = A$
Б Ω.

Для связи этих трех блоков в единую программу нужно написать несколько команд, образующих собирающую программу.

Программа 4.12 (*собирающая*)



Если блок имеет короткое название и обращение к нему требует передачи управления в первую его ячейку, то в собирающей вместо условного обозначения первой ячейки блока можно прямо указывать его название. Собственно говоря, обозначения *К1*, *К2* и *СП* можно считать названиями блоков.

Программа вычисления величины *А* состоит из всех четырех написанных программ. Для ее работы необходимо все эти программы ввести в память и передать управление первой ячейке собирающей программы.

Легко видеть, что собирающая программа, как и последние команды каждого из блоков, передающих управление ячейке Ω, представляют собою накладные расходы. Их можно было бы исключить, в результате чего программа сократилась бы на 6 ячеек, т. е. более чем в полтора раза. Такое сокращение, ликвидирующее накладные расходы, соответствовало бы доставке продуктов непосредственно от производителя к потребителю. Однако такая непосредственная доставка, целесообразная, например, для молочницы, разносящей молоко от одной коровы, уже очень невыгодна, если речь идет о молочной ферме колхоза или совхоза. Блочная структура программы является аналогом отделения производства от потребления, причем роль посредника между ними осуществляется собирающей программой.

Блочная структура программы обладает следующими достоинствами:

1) При написании программы каждый блок программируется отдельно. При этом можно не заботиться ни о программе в целом, ни об остальных блоках,

Более того, различные блоки могут писаться разными людьми.

2) Разбиение программы на блоки позволяет при небольших изменениях задания ограничиваться переделками одного или нескольких блоков, не затрагивая остальных частей программы.

Например, если бы в примере 1.12 изменилась форма задания одного из векторов, то при блочной структуре программы достаточно было бы переделать один блок, тогда как без нее пришлось бы писать программу заново.

3) Блочная структура программы облегчает ее отладку. Во-первых, она позволяет отлаживать отдельные блоки независимо друг от друга и даже блоки программы, не написанной еще до конца. Более того, можно программировать и отлаживать отдельные блоки до того, как составлен полный алгоритм решения задачи. Во-вторых, вся программа оказывается легко обозримой, так как в каждый момент отладки мы имеем дело либо с отдельным блоком, либо с собирающей.

4) Программа, составленная из блоков, обладает большой гибкостью. Одни и те же блоки могут быть использованы в различных частях программы.

§ 13. Блок-схема и блок-программа

Не следует думать, что возможность обращения к подпрограмме есть привилегия только собирающей программы. Наоборот, очень часто в различных блоках приходится в свою очередь обращаться к другим блокам.

Обращение одного блока к другому мешает использованию ячейки Ω в качестве единого конца всех блоков. Действительно, обращаясь к одному из них, мы засылаем в ячейку Ω команду возврата из этого блока в нужную часть программы. При обращении этого блока ко второму в ту же ячейку Ω необходимо заслать команду возврата в первый блок, в результате чего ранее засланная туда команда потеряется.

Чтобы не нарушать правильной работы программы, рекомендуется поступать следующим образом. Последнюю ячейку блока (*конец*) будем оставлять не

перфорируемой, а начинать блок командой

$$\Omega = конец.$$

При этом команда передачи управления в нужное место программы, записанная в ячейке Ω при обращении к данному блоку, будет перенесена в конец блока и ячейка Ω может быть снова использована для той же цели.

Пример 1.13. Многоугольник задан следующим образом. Внутри него выбрана фиксированная точка и заданы расстояния от этой точки до вершин многоугольника и длины сторон. Вычислить площадь этого многоугольника, если длины всех отрезков в порядке, указанном на рис. 23, записаны в ячейках a_1, a_2, ..., a_{2n}, лежащих подряд.

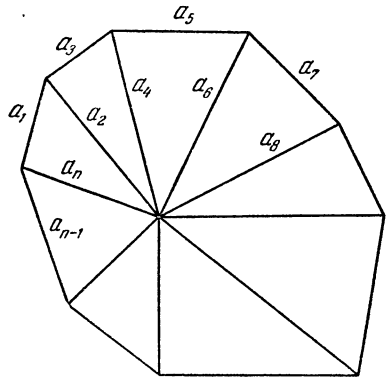


Рис. 23.

Площадь многоугольника мы будем считать как сумму площадей треугольников, на которые он разбит, выделив вычисление площади треугольника в отдельный блок. Аргументами для этого блока будут ячейки α_1, α_2, α_3, которые должны содержать длины сторон очередного треугольника, а выходной ячейкой, содержащей площадь треугольника, будет служить ячейка γ.

В основной программе нужно сложить полученные площади всех треугольников. Кроме того, перед обращением к блоку, вычисляющему площадь треугольника, нужно позаботиться, чтобы в ячейки α_1, α_2, α_3, являющиеся аргументами этого блока, были засланы длины сторон соответствующего треугольника. Для первого из них это будут a_n, a_1, a_2, для второго a_2, a_3, a_4 и т. д. Основную программу вычисления площади многоугольника можно написать следующим образом.

Программа 1.13

	1)	0	$=S$
	2)	$n \quad -1_N$	$=$ *Счетчик*
	3)	a_n	$=\alpha_1$
	4)	T^0	$=T$
	5)	$U^0 \rightarrow U$	
	6)	$T +, (0, 2, 0) = T$	
	7)	$U +, (0, 2, 0) = U$	
T	8)	$(a_1 = \alpha_2)$	*н. п.*
U	9)	$(a_2 = \alpha_3)$	*н. п.*
	10)	11) $\leftrightarrow \Omega$;	*Площадь*
	11)	$S + \gamma = S$	
	12)	*Счетчик* $- 1_N =$ *Счетчик*	
	13) (0)	$\alpha_3 \rightarrow \alpha_1$ 6)	
	14)	*стоп*	
T^0	15)	$a_1 = \alpha_2$	
U^0	16)	$a_2 = \alpha_3$	
	17)	$(0, 2, 0)$.	

Здесь мы имеем обычный цикл с переадресацией. Команды 1)—5) являются здесь командами восстановления. При этом команда 1) приводит в начальное состояние ячейку S. Команда 2) образует счетчик цикла. Команда 3) засылает первоначальное значение в аргумент α_1 блока вычисления площади, т. е. выполняет задачу, возложенную на рабочую часть цикла. Однако мы относим эту команду к командам восстановления, поскольку здесь речь идет лишь о начальном состоянии, а в дальнейшем в ячейку α_1 число будет засылаться иначе. Команды 4) и 5) восстанавливают первоначальное состояние команд рабочей части, засылающих числа в ячейки α_2 и α_3.

Далее идут команды 6) и 7), которые переадресуют команды T и U рабочей части на две единицы среднего адреса. Действительно, если для первого треугольника

в ячейке α_2 должна лежать a_1, то для второго это будет a_3, для третьего a_5 и т. д. Аналогично в ячейке α_3 должны лежать, соответственно, сначала a_2, затем a_4, потом a_6 и т. д.

Команды 12) и 13) являются командами проверки окончания. Кроме того, команда 12) является одновременно и командой изменения счетчика, а команда 13) благодаря возможности совмещения пересылки с передачей управления выполняет также и задачу рабочей части, пересылая в ячейку α_1 новое содержимое. Ячейки 15)—17) содержат восстановители переадресуемых команд и константу переадресации.

Укажем еще одно важное обстоятельство, которое следует иметь в виду при блочном программировании. До сих пор счетчик цикла мы обозначали как любую рабочую ячейку, например через R_0. При блочном программировании необходимо следить, чтобы рабочие ячейки, используемые различными блоками, были различны. В противном случае результаты, полученные в одном блоке, могут быть испорчены при обращении к другому. Чтобы избежать этого, следует, по крайней мере, для ячеек, содержимое которых требует длительного хранения, выбирать не обезличенные обозначения вроде R_0, а характерные. С этой целью мы и применили название *Счетчик*.

Напишем теперь блок *Площадь*. Поскольку требуется нахождение площади треугольника по трем сторонам, то можно воспользоваться формулой Герона, которая требует извлечения квадратного корня. Программа для извлечения квадратного корня, которую мы рассматривали в § 4, будет еще одним блоком нашей программы. Аргумент этого блока назовем α, а ответ γ_1. Блок *Площадь* можно написать так, как это сделано в программе 2.13.

Для полноты приведем здесь же программу 3.13 извлечения квадратного корня, оформленную как блок нашей программы.

При обращении основной программы к блоку *Площадь* в ячейку Ω засылается команда возврата в команду 11) основной программы. Первая же команда блока *Площадь* (Ω=конец) переносит команду возврата в конец этого блока. После этого о содержимом

ячейки Ω можно не заботиться, поскольку оно уже использовано и возвращение в команду 11) основной программы после окончания блока *Площадь* обеспечено.

Программа 2.13

$\Omega = $ *конец*
$\alpha_1 + \alpha_2 = R_1$
$R_1 + \alpha_3 = R_1$
$R_1 \cdot \left(\frac{1}{2}\right)_N = p$
$p - \alpha_1 = R_1$
$p \cdot R_1 = R_2$
$p - \alpha_2 = R_1$
$R_1 \cdot R_2 = R_1$
$p - \alpha_3 = R_2$
$R_2 \cdot R_1 = \alpha$
$\longrightarrow \Omega$ *корень*
$\downarrow \quad \gamma_1 = \gamma$
конец.

Программа 3.13

$\Omega = $ *конец*
$\alpha \cdot \left(\frac{1}{2}\right)_N = R_3$
$V \quad \alpha : R_3 = \gamma_1$
$R_3 + \gamma_1 = \gamma_1$
$\gamma_1 \cdot \left(\frac{1}{2}\right)_N = \gamma_1$
$R_3 - \gamma_1 = R_4$
$|R_4| - |\varepsilon| = 0$
(0) $R_4 \rightarrow R_3$; V
конец.

Теперь при обращении к блоку *Корень* внутри блока *Площадь* мы можем спокойно засылать снова в ячейку Ω команду возврата в блок *Площадь*, которая таким же способом будет помещена в конец блока *Корень*.

Так как блок *Корень* является самым внутренним и ни к каким другим блокам не обращается, то в нем команда Ω=*конец* является излишней. Тем не менее мы считаем безусловно необходимым начинать этой командой каждый блок. Прежде всего это делается с целью стандартизации написания блоков, что очень важно. Нужно помнить также, что, начиная писать блок, нельзя заранее быть уверенным в том, что какую-либо его часть не захочется выделить как отдельный блок.

Читателю может показаться, что, оставляя в конце блока свободную ячейку, мы возвращаемся к тому, что каждый блок программы должен иметь свой конец. Однако это не так, потому что при обращении к блоку не требуется знать адрес его конца; команда возврата всегда засылается в стандартную ячейку Ω.

Следующим шагом в том же направлении является выделение группы ячеек памяти в качестве заголовков (начал) очередных блоков. В этом случае мы будем иметь возможность писать обращение к блокам, которые еще не запрограммированы, и кодировать написанные блоки до того, как следующим блокам отведены места в памяти. Потребуется только начинать каждый блок командой передачи управления из заголовка в то место памяти, где этот блок фактически расположен. Между прочим, это не вызовет удлинения программы, поскольку передачу управления можно совместить с пересылкой Ω в конец.

Такая система заголовков пока еще редко используется в вычислительных программах, но широко используется в библиотеках стандартных программ. Подробнее система заголовков будет рассматриваться в следующей главе.

Из рассмотренных примеров видно, что блоки могут быть весьма различными. Это может быть расписка формулы (бесцикловая), программа, содержащая циклы, обращения к подпрограммам и т. д. Вообще блок может быть сколь угодно сложным образованием. Един-

ственным свойством блока, которое можно считать его точным определением, является следующее: *при обращении к блоку A с помощью команды*

$$S \leftrightarrow \Omega;\ A$$

можно быть уверенным в том, что после окончания его работы управление перейдет к ячейке S.

Будем обозначать блоки программы прямоугольниками и нарисуем схему связи между ними, указывая

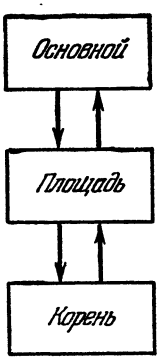


Рис. 24.

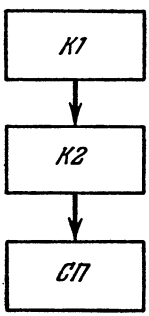


Рис. 25.

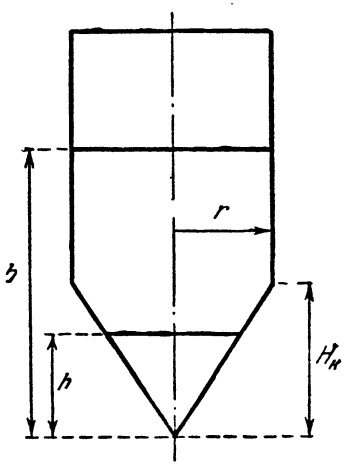


Рис. 26.

стрелками последовательность работы блоков. Мы получим *блок-схему* программы. Например, для программы примера 1.13 блок-схема имеет вид, изображенный на рис. 24, а для программы примера 1.12 — на рис. 25.

Рассматривая каждый из блоков программы как некоторый оператор, мы превратим блок-схему в логическую схему программы. Заменяя стрелки в блок-схеме командами обращения к блокам, мы получим *блок-программу,* которую мы уже фактически ввели в предыдущем параграфе под названием собирающей.

Таким образом, блок-программа содержит команды обращения к блокам программы. Кроме того, если

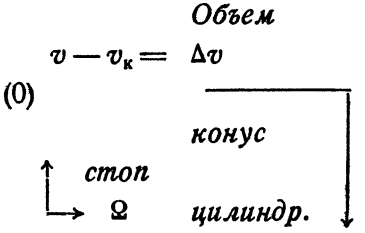
обращение к некоторым блокам является условным, то блок-программа должна содержать проверки соответствующих условий.

Если передача управления некоторому блоку зависит от результатов работы предыдущего, хотя бы и промежуточных, то предыдущий блок должен быть устроен так, чтобы он вырабатывал какие-либо признаки, проверка которых должна производиться в собирающей, но не в блоке. Это требование непосредственно вытекает из данного нами выше определения блока. Можно добавить, что признаки следует делать такими, чтобы проверка была возможно более простой и короткой. Если признаки сложны, то упрощение их может быть выделено в качестве отдельного блока.

Пример 2.13. Резервуар с горючим имеет форму опрокинутого конуса, завершенного цилиндром. Форма и размеры поперечного сечения показаны на рис. 26. В резервуар влито v литров горючего. Требуется определить высоту h уровня жидкости, считая от нижней точки.

Высоту h следует вычислять по различным формулам в зависимости от того, больше или меньше объема конуса объем v. Поэтому прежде всего нужно вычислить объем конуса. Назовем соответствующий блок *Объем*. После вычисления $v_к$, если $v<v_к$, высоту h следует искать как высоту конуса (блок *Конус*), а при $v>v_к$ как высоту цилиндра (блок *Цилиндр*). Таким образом, блок-программа задачи будет иметь вид:

Программа 4.13 (*Собирающая*)



Здесь мы воспользовались еще одним удобным сокращением. Так как при уходе на блок в среднем ад-

ресе всегда пишется Ω, то в левой части ее можно и не писать — она подразумевается. Если в левом адресе нужно ставить $Я + 1$ ($Я$ — адрес команды обращения к блоку), то его тоже можно не писать в левой части; при кодировке он будет поставлен автоматически. Если же возврат идет не к $Я + 1$, а в другое место программы, то это следует указывать символом или стрелкой, как это сделано при обращении к блоку *Цилиндр*.

Блок *Объем* состоит всего из нескольких команд. При этом, кроме объема конуса, попутно вычисляется площадь его основания, которая равна площади основания цилиндра (ячейка Q).

Программа 5.13 (*Объем*)

$$\Omega = \textit{конец}$$
$$r \cdot r = R_1$$
$$\pi \cdot R_1 = Q$$
$$Q \cdot H_к = v_к$$
$$v_к \cdot \left(\frac{1}{3}\right)_N = v_к$$
конец.

Блок *Конус* будет работать в том случае, если $v < v_к$. Тогда жидкость будет иметь форму конуса с высотой h и радиусом основания

$$r_1 = r\frac{h}{H_к},$$

что легко следует из подобия треугольников. Мы имеем

$$v = \frac{1}{3}\frac{\pi r^2}{H_к^2}h^3,$$

откуда

$$h = \sqrt[3]{\frac{3vH_к^2}{Q}},$$

где $Q = \pi r^2$ уже вычислено в предыдущем блоке. Таким образом, можно написать программу 6.13.

Программа 6.13 (*Конус*)

$\Omega =$ *конец*

$H_к \cdot H_к = R_2$

$v \cdot R_2 = R_2$

$R_2 \cdot 3_N = R_2$

$R_2 : Q = \alpha$

корень

$\gamma = h$

конец.

Здесь выделено в качестве блока извлечение кубического корня. Прежде чем заниматься этим блоком, напишем программу блока *Цилиндр*. Так как основная программа дает значение Δv, а площадь Q поперечного сечения уже известна, то остается очень мало работы.

Программа 7.13 (*Цилиндр*)

$\Omega =$ *конец*

$\Delta v : Q = h$

$h + H_к = h$

конец.

Теперь напишем программу для извлечения кубического корня. Для того чтобы не обращаться ни к каким другим программам, воспользуемся алгоритмом, основанным на итерации по формуле *)

$$x_{n+1} = \frac{1}{3}\frac{a}{x_n^2} + \frac{2}{3}x_n.$$

Легко проверить, что последовательность $\{x_n\}$ сходится к значению $\sqrt[3]{a}$. В качестве нулевого приближения можно взять $x_0 = \frac{a}{3}$. Получим программу 8.13.

*) Напомним еще раз, что мы не имеем в виду рекомендовать этот алгоритм для извлечения кубического корня. Он используется нами лишь для того, чтобы довести задачу до конца, не обращаясь к программам, ранее не рассматривавшимся.

Программа 8.13 (*Корень*)

```
      Ω    = конец
      α  · (1/3)_N = x
∇     x  ·  x    = x²
      α  :  x²   = R3
      R3 · (1/3)_N = R3
      x  +  x    = R4
      R4 · (1/3)_N = R4
      R3 +  R4   = γ
      γ  —  x    = R5
     |R5| — |ε|  = 0
(0)   γ  →  x;    ∇
      конец.
```

§ 14. Как писать программу

Настоящий параграф содержит некоторые практические советы, полезные в процессе программирования. При этом мы считаем, что программа является одним из наиболее естественных способов записи алгоритма решения задачи. Поэтому будем предполагать, что разработка алгоритма и написание программы не есть два различных этапа решения задачи и что математик, решая задачу, записывает алгоритм ее решения непосредственно в виде программы.

Приступая к решению задачи, следует прежде всего разбить ее на основные части, а затем написать внешнюю*) блок-программу. *При составлении программы какой-либо части задачи ее нужно писать «сверху», делая то, что уже ясно, и вынося в качестве отдельных*

*) Так как некоторые блоки, в свою очередь, могут являться собирающими, то блок-программу всей задачи естественно назвать внешней.

блоков те места, которые пока неясны или вызывают затруднения. Покажем на примере, что мы имеем в виду.

Пример 1.14. Функция $y=f(x)$ удовлетворяет уравнению $a(x^4+y^4)-2a^3(x^2+y^2)=a^4(x+y)$ с неизвестным значением параметра a. Значения этой функции измерены в $n+1$ точках $x_0, x_1, x_2, \ldots, x_n$ и равны, соответственно, $y_0, y_1, y_2, \ldots, y_n$. Требуется подобрать параметр a таким образом, чтобы среднее квадратичное отклонение вычисленных значений функции от измеренных было наименьшим (способ наименьших квадратов), предполагая, что возможные его значения заключены между a_0 и a_1. Значения $x_0, x_1, \ldots, x_n$ и $y_0, y_1, \ldots, y_n$ будем предполагать лежащими в ячейках памяти подряд.

Будем поступать следующим образом. Пройдем по всем значениям a в заданном интервале с некоторым шагом Δa. Для каждого значения a будем вычислять значения функции в точках $x_0, x_1, \ldots, x_n$, затем сумму квадратов отклонений этих значений от измеренных и сравнивать между собою отклонения, полученные для различных a, выбирая наименьшее. Наименьшую сумму квадратов мы будем помещать в ячейку $\sigma_{\text{мин}}$, в которую сначала запишем наибольшее возможное в машине число. Его мы обозначим буквой *Щ*.

Таким образом, блок-программу нашей задачи можно записать так:

Программа 1.14

Назначение каждого блока ясно из его названия. Заметим, что мы написали собирающую, не решив, как будут написаны отдельные блоки.

Обратимся теперь к блокам программы. Блок *Функция* должен при уже заданном a и любом $x\,(=x_0, x_1, \ldots)$ вычислять соответствующие значения $f(x)$ и размещать их подряд в ячейках $y_0^{\text{выч}}, y_1^{\text{выч}}, \ldots, y_n^{\text{выч}}$. Эту задачу можно решить такой программой:

Программа 2.14 (*Функция*)

```
                    Ω = конец
                (n, 0, 0) = Сч. 1
                    T0 = T
        S0 →  S;        T
              T+, (0, 1, 0) = T          ↑
              S+, (0, 0, 1) = S          |
T                   (x0 = α)             | н. п.
                 Уравнение               |
S                   (γ      = y0выч)     | н. п.
        Сч. 1 —, (1, 0, 0) = Сч. 1        |
  (0)                                     |
      конец
T0              x0      = α
S0              γ       = y0выч.
```

Это — обычный цикл с переадресацией, причем основная рабочая часть вынесена в отдельный блок *Уравнение*, который должен при заданных a и x найти соответствующее значение y, т. е. решить уравнение $a(x^4+y^4)-2a^3(x^2+y^2)-a^4(x+y)=0$.

Заметим, что блок *Функция* можно сократить на одну команду, совместив пересылку γ в ячейку $y^{\text{выч}}$ с условной передачей управления (см. пример 5.8).

Отложив написание блока *Уравнение*, напишем следующие основные блоки программы. Блок *Отклонение*,

который считает сумму квадратов отклонений

$$\sigma_{\text{срав}} = \sum_{k=0}^{n} (y_k^{\text{выч}} - y_k)^2,$$

можно написать в виде:

Программа 3.14 (*Отклонение*)

```
                    Ω = конец
                    0 = σ срав
                (n, 0, 0) = Сч. 2
          U0→     U
          U +,  (1, 1, 0) = U
U         (y0выч —      y0 = R1)↓      н. п.
          R1 ·          R1 = R1
          σ срав +      R1 = σ срав
          Сч. 2 —,  (1, 0, 0) = Сч. 2
    (0)
                конец
U0                y0выч — y0 = R1.
```

Блок *Сравнение* сравнивает полученное значение суммы квадратов отклонений с наименьшим, полученным ранее для других a и хранящимся в ячейке $\sigma_{\text{мин}}$. Если это новое значение меньше предшествующих, то оно переносится в ячейку $\sigma_{\text{мин}}$. Одновременно значение a, соответствующее этому наименьшему отклонению, записывается в ячейку $a_{\text{отв}}$ (см. программу 4.14).

Обратимся теперь к блоку *Уравнение*. Выберем сначала способ решения уравнения. Для простоты ограничимся таким способом. Будем идти от некоторого значения $y_{\text{нач}}$ с шагом Δy до тех пор, пока не обнаружим участка, на котором функция, равная левой части уравнения, меняет знак. После нахождения такого участка будем делить его последовательно пополам, пока не получим участка достаточно малой длины. Тогда блок *Уравнение* можно написать в виде программы 5.14.

Программа 4.14 (*Сравнение*)

$\Omega = $ *конец*

$\sigma_{срав} - \sigma_{мин} = 0$

(0) ——————

$\sigma_{срав} = \sigma_{мин}$

$a = a_{отв}$

конец.

Программа 5.14 (*Уравнение*)

$\Omega = $ *конец*

$\Delta y_0 = \Delta y$

$y_{нач} = y$

Левая часть

$\gamma = f$

$y + \quad \Delta y = y$

Левая часть

$f \cdot \quad \gamma = R_2$

$R_2 + \quad 0 = R_2$

(0) $\gamma \to f;$ ——————

$\Delta y \cdot \left(\frac{1}{2}\right)_N = \Delta y$

$0 - \quad \Delta y = \Delta y$

$|\Delta y| - \quad |\varepsilon| = 0$

(0) ——————

конец

Остается написать блок *Левая часть*, предназначенный для вычисления функции

$$\gamma = a(x^4 + y^4) - 2a^3(x^2 + y^2) - a^4(x + y),$$

причем значение x лежит в ячейке α. Здесь мы имеем простую расписку формулы.

Программа 6.14 (*Левая часть*)

$\Omega = $ *конец*
$\alpha \cdot \alpha = x^2$
$x^2 \cdot x^2 = x^4$
$y \cdot y = y^2$
$y^2 \cdot y^2 = y^4$
$a \cdot a = a^2$
$a^2 \cdot a = a^3$
$a^3 + a^3 = 2a^3$
$a^2 \cdot a^2 = a^4$
$x^4 + y^4 = R_3$
$a \cdot R_3 = R_3$
$x^2 + y^2 = R_4$
$2a^3 \cdot R_4 = R_4$
$x + y = R_5$
$a^4 \cdot R_5 = R_5$
$R_4 + R_5 = R_4$
$R_3 - R_4 = \gamma$
конец.

Заметим, что блок *Уравнение* годится для решения любого уравнения выбранным способом. Нужно только написать соответствующий блок *Левая часть.*

Подобно тому, что мы писали блоки *Отклонение* и *Сравнение* до того, как выбрали способ решения уравнения, можно и отладку этих блоков вести независимо от остальных блоков. Подробнее речь об этом будет идти в гл. V.

Пользуясь данной программой, мы достигаем точности нахождения значения a, равной Δa. Выбор очень маленького значения Δa для получения большой точности может вызвать сильное увеличение времени работы программы. Целесообразнее поступать иначе: сначала найти $a_{\text{отв}}$ при некотором сравнительно большом

Δa, а затем пройти более мелким шагом по участку $a_{отв} - \Delta a$, $a_{отв} + \Delta a$ с помощью той же программы. Для этого нужно нашу собирающую превратить в блок, работающий на участке ($a_{лев}$, $a_{прав}$) с шагом Δa, причем значения $a_{лев}$, $a_{прав}$, Δa должны засылаться некоторой новой более внешней собирающей.

Назвав блок, полученный из прежней собирающей, *Поиском*, получим две новые программы:

Программа 7.14 (*Внешняя собирающая*)

$a_0 = a_{лев}$

$a_1 = a_{прав}$

$\Delta a_0 \to \Delta a$

$a_{отв} - \Delta a = a_{лев}$

$a_{отв} + \Delta a = a_{прав}$

$\Delta a \cdot \left(\frac{1}{8}\right)_N = \Delta a$

поиск

$\Delta a - \varepsilon_a = 0$

(0)

стоп.

Программа 8.14 (*Поиск*)

$\Omega = конец$

$Щ = \sigma_{мин}$

$a_{лев} \to a$

$a + \Delta a = a$

Функция

Отклонение

Сравнение

$a - a_{прав} = 0$

(1)

конец.

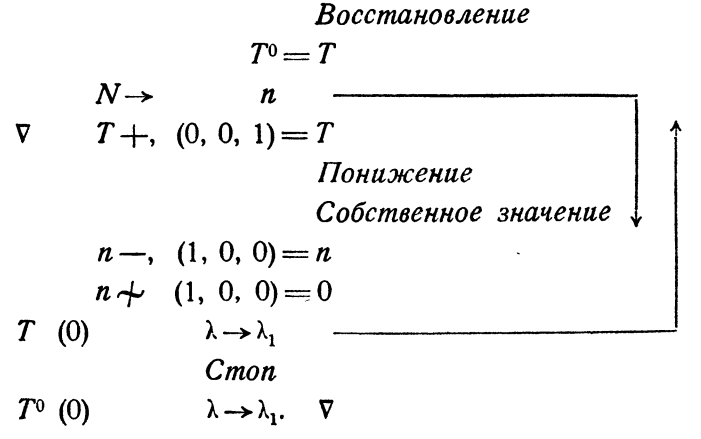
Пример 2.14. Найти все собственные числа матрицы порядка N.

Предположим, что элементы матрицы записаны подряд в ячейках памяти, начиная с ячейки a_{11}. Порядок матрицы будем считать заданным адресно, т. е. в фиксированной форме, в виде числа единиц первого адреса.

Наибольшее по модулю собственное число легко находится методом итераций. Следующие собственные числа будем искать методом понижения *): после того как одно собственное число и соответствующий ему собственный вектор найдены, можно преобразовать матрицу порядка n в такую новую матрицу порядка $n-1$, все собственные числа которой совпадают с собственными числами предыдущей матрицы, кроме уже найденного.

Блок-программу задачи можно написать поэтому следующим образом:

Программа 9.14 (*Собирающая*)



Блок *Восстановление* осуществляет пересылку элементов заданной матрицы на рабочее поле. На нем

*) См. Д. К. Фаддеев и В. Н. Фаддеева, Вычислительные методы линейной алгебры, Физматгиз, 1960, стр. 375.

мы сейчас останавливаться не будем. Понижение порядка матрицы состоит из двух частей: преобразования матрицы и ее пересылки на рабочее место. Поэтому блок *Понижение* тоже можно написать в виде собирающей.

Программа 10.14 (*Понижение*)

$\Omega =$ *конец*

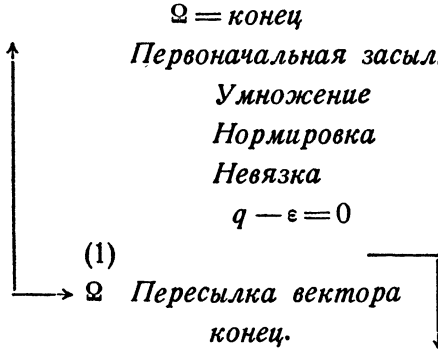
Преобразование матрицы

Пересылка матрицы

конец.

Блок *Собственное значение* должен находить наибольшее по модулю собственное значение очередной матрицы. Как мы условились, это будет сделано с помощью итераций. Поэтому блок можно написать так:

Программа 11.14 (*Собственное значение*)



Ячейка q содержит невязку, подсчитанную как сумма модулей разностей координат векторов, получающихся двумя последовательными итерациями. Нормировка вектора, как это диктуется методом понижения, производится таким образом, чтобы первая координата вектора равнялась единице. Что касается блока *Умножение*, то здесь речь идет об умножении матрицы на вектор. Такая программа уже рассматривалась нами в §10 (см. пример 5.10).

1000	1001	1002	1003 Собирающая	1004	1005	1006	1007
1020	1021	1022	1023	1024 Восстановление	1025	1026	1027
1040	1041 Понижение	1042	1043	1044	1045	1046	1047
1060	1061	1062	1063 Преобразование		1065	1066 матрицы	1067
1100	1101	1102	1103	1104	1105	1106	1107
1120	1121	1122	1123 Пересылка	1124	1125	1126 матрицы	1127
1140	1141	1142	1143 Сосста.	1144 значения	1145	1146	1147
1160	1161 Первоначальная	1162	1163	1164 засылка	1165	1166	1167
1200	1201	1202	1203	1204	1205	1206	1207
1220	1221	1222	1223	1224	1225	1226	1227
1240	1241	1242	1243 Нормировка	1244	1245	1246	1247
1260	1261	1262	1263	1264	1265 Невязка	1265	1267
1300	1301	1302	1303	1304	1305	1306	1307
1320 n	1321 $n_{зап}$	1322	1323	1324	1325	1326	1327
1340 n_{10}	1341 n_{11}	1342 n_{12}	1343 n_{13}	1344	1345	1346	1347
1360 λ	1361 λ_1	1362	1363	1364	1365	1366	1367
1400 a_{11}	1401 a_{12}	1402	1403	1404	1405	1406	1407
1420	1421	1422	1423	1424	1425	1426	1427
1440	1441	1442	1443	1444	1445	1446	1447
1460	1461	1462	1463	1464	1465	1466	1467
1500 $a_{11}^{запас}$	1501	1502	1503	1504	1505	1506	1507
1520	1521	1522	1523	1524	1525	1526	1527
1540	1541	1542	1543	1544	1545	1546	1547
1560	1561	1562	1563	1564	1565	1566	1567
1600 $a_{11}^{нач}$	1601	1602	1603	1604	1605	1606	1607
1620	1621	1622	1623	1624	1625	1626	1627
1640	1641	1642	1643	1644	1645	1646	1647
1660	1661	1662	1663	1664	1665	1666	1667
1700 u_1	1701	1702	1703	1704	1705	1706	1707
1720 $U_{зап}$	1721 $A^{зап}$	1722	1723	1724	1725	1726	1727
1740 q	1741 S	1742	1743	1744	1745	1746	1747
1760 ε	1761	1762	1763	1764	1765	1766	1767

Рис. 27 (левый).

1010	1011	1012	1013	1014	1015	1016	1017
1030	1031	1032	1033	1034	1035	1036	1037
1050	1051	1052	1053	1054	1055	1056	1057
1070	1071	1072	1073	1074	1075	1076	1077
1110	1111	1112	1113	1114	1115	1116	1117
1130	1131	1132	1133	1134	1135	1136	1137
1150	1151	1152	1153	1154	1155	1156	1157
1170	1171	1172	1173	1174	1175	1176	1177
1210 Умножение	1211	1212	1213	1214	1215	1216	1217
1230	1231	1232	1233	1234	1235	1236	1237
1250	1251	1252	1253	1254	1255	1256	1257
1270	1271	1272	1273	1274	1275	1276	1277
1310	1311	1312	1313	1314	1315	1316	1317
1330 n_0	1331 n_1	1332 n_2	1333 n_3	1334 n_4	1335 n_5	1336 n_6	1337 n_7
1350 R_0	1351 R_1	1352 R_2	1353	1354	1355	1356	1357
1370	1371	1372	1373	1374	1375	1376	1377
1410	1411	1412	1413	1414	1415	1416	1417
1430	1431	1432	1433	1434	1435	1436	1437
1450	1451	1452	1453	1454	1455	1456	1457
1470	1471	1472	1473	1474	1475	1476	1477
1510	1511	1512	1513	1514	1515	1516	1517
1530	1531	1532	1533	1534	1535	1536	1537
1550	1551	1552	1553	1554	1555	1556	1557
1570	1571	1572	1573	1574	1575	1576	1577
1610	1611	1612	1613	1614	1615	1616	1617
1630	1631	1632	1633	1634	1635	1636	1637
1650	1651	1652	1653	1654	1655	1656	1657
1670	1671	1672	1673	1674	1675	1676	1677
1710 v_1	1711	1712	1713	1714	1715	1716	1717
1730	1731	1732	1733	1734	1735	1736	1737
1750	1751	1752	1753	1754	1755	1756	1757
1770	1771	1772	1773	1774	1775	1776	1777

Рис. 27 (правый).

Нахождение собственных чисел матрицы

Составил | Задача | Блок | Стр. 1 | Лист 1

	Собирающая			1000	A	T шифр	№ 2.14.1 № карты
	Восстановление	1000	16	1001	0007	1020	
	$T_0 = T$	1	75	0	1012	1010	
	$N \to n$	2	56	2000	1320	1006	
∇	$T+, (0, 0, 1) = T$	3	13	1010	0121	1010	
	Понижение	4	16	1005	0007	1040	
	$n-, (1, 0, 0) = n$	5	33	1320	0124	1320	
	Собств. значение	6	16	1007	0007	1140	
	$n \not\vdash (1, 0, 0) = 0$	7	15	1320	0124	0	
T	(0) $\lambda \to \lambda_1$;	1010			*н. п.*		
	стоп	1	77	0	0	0	
T^0	(0) $\lambda \to \lambda_1$; ∇	2	76	1360	1361	1003	

Составил | Задача | Блок | Стр. 1 | Лист 2

	Восстановление			1020	A	T шифр	№ 2.14.2 № карты
	$\Omega = конец$	1020	75	0	0007	1033	
	$A^0 = A$	1	75	0	1034	1026	
	$N = n_0$	2	75	0	2000	1330	
	$N-, (1, 0, 0) = n_{зап}$	3	33	2000	0124	1321	
	$n_0-, (1, 0, 0) = n_0$	4	33	1330	0124	1330	
	(1) $n_{зап} \to n$	5	36	1321	1320	1033	
A	$a_{11}^{нач} = a_{11}$	6			*н. п.*		
	$A+, (0, 1, 1) = A$	7	13	1026	0123	1026	
	$n-, (1, 0, 0) = n$	1030	33	1320	0124	1320	
	(0)	1	36	0	0	1026	
	Б	2	56	0	0	1024	

				1033	A	T шифр	№ 2.14.3 № карты
	конец	1033			*н. п.*		
A^c	$a_{11}^{нач} = a_{11}$	4	75	0	1600	1400	

Рис. $28_{1,2}$.

Нахождение собственных чисел матрицы

Составил Задача Блок Стр. 1 Лист 3

	Понижение			1040	A	T шифр	№ 2.14.4 № карты
	Ω = *конец*	1040	75	0	0007	1043	
	Преобразование матрицы	1	16	1042	0007	1060	
	Пересылка матрицы	2	16	1043	0007	1120	
	Конец	3			*н. п.*		

Составил Задача Блок Стр. 1 Лист 4

	Преобразование матрицы			1060	A	T шифр	№ 2.14.5 № карты
	Ω = *конец*	1060	75	0	0007	1101	
	$V^0 +,\ n = V$	1	13	1103	1320	1075	
	$U^0 = U_{зап}$	2	75	0	1102	1720	
	$n -,\ (1, 0, 0) = n_{зап}$	3	33	1320	0124	1321	
	$n_{зап} \to n_1$	4	56	1321	1331	1067	
	$U_{зап} +,\ (0, 1, 0) = U_{зап}$	5	13	1720	0122	1720	
	$V +,\ (1, 0, 0) = V$	6	13	1075	0124	1075	
	$n_1 -,\ (1, 0, 0) = n_1$	7	33	1331	0124	1331	
	(1) $n_{зап} \to n_2$	1070	36	1321	1332	1101	
	$U_{зап} = U$	1	75	0	1720	1074	
	$n_2 -,\ (1, 0, 0) = n_2$	2	33	1332	0124	1332	

				1073	A	T шифр	№ 2.14.6 № карты
	(1)	1073	36	0	0	1065	
U	$a_{12} \cdot u_2 = R_0$	4			*н. п.*		
V	$a_{22} - R_0 = a_{11}^{зап}$	5			*н. п.*		
	$U +,\ (1, 0, 0) = U$	6	13	1074	0124	1074	
	$V +,\ (1, 0, 1) = V$	7	13	1075	0125	1075	
	Б	1100	56	0	0	1072	
	конец	1			*н. п.*		
U^0	$a_{12} \cdot u_2 = R_0$	2	05	1401	1701	1350	
V^0	$a_{12} - R_0 = a_{11}^{зап}$	3	02	1401	1350	1500	

Рис. $28_{3,4}$.

Нахождение собственных чисел матрицы

Составил | Задача | Блок | Стр. 1 | Лист 5

	Пересылка матрицы			1120	A	T шифр	№ 2.14.7 № карты
	Ω = *конец*	1120	75	0	0007	1133	
	$A^0 = A$	1	75	0	1134	1126	
	n —, (1, 0, 0) = n_0	2	33	1320	0124	1330	
	n_0 —, (1, 0, 0) = $n_{зап}$	3	33	1330	0124	1321	
	n_0 —, (1, 0, 0) = n_0	4	33	1330	0124	1330	
	(1) $n_{зап} \to n_{13}$; *конец*	5	36	1321	1343	1333	
A	$a_{11}^{зап} = a_{11}$	6			*н. п.*		
	A +, (0, 1, 1) = A	7	13	1126	0123	1126	
	n_{13} —, (1, 0, 0) = n_{13}	1130	33	1343	0124	1343	
	(0)	1	76	0	0	1126	
	Б	2	56	0	0	1124	

				1133	A	T шифр	№ 2.14.8 № карты
	конец	1133			*н. п.*		
A^0	$a_{11}^{зап} = a_{11}$	4	75	0	1500	1400	

Составил | Задача | Блок | Стр. 1 | Лист 6

	Собств. значение			1140	A	T шифр	№ 2.14.9 № карты
	Ω = *конец*	1140	75	0	0007	1150	
	Первонач. засылка	1	16	1142	0007	1160	
	Умножение	2	16	1143	0007	1200	
	Нормировка	3	16	1144	0007	1240	
	Невязка	4	16	1145	0007	1260	
	$q - \varepsilon = 0$	5	02	1740	1760	0	
(1)		6	36	0	0	1150	
	Ω; *Пересылка вектора*	7	16	1142	0007	1300	
	конец	1150			*н. п.*		

Рис. $28_{5,6}$.

Нахождение собственных чисел матрицы

Составил Задача Блок Стр. 1 Лист 7

	Первоначальная засылка			1160	A	Т шифр	№ 2.14.10 № карты
	$\Omega = конец$	1160	75	0	0007	1166	
	$n-, (1, 0, 0) = n_5$	1	33	1320	0124	1335	
	$M^0 \to M$	2	56	1167	1165	1164	
	$M+, (0, 1, 0) = M$	3	13	1165	0122	1165	
	$n_5-, (1, 0, 0) = n_5$	4	33	1335	0124	1335	
M	(0) $1_N \to v_1$	5			н. п.		
	конец	6			н. п.		
M^0	(0) $1_N \to v_1$	7	76	0101	1710	1163	

Составил Задача Блок Стр. 1 Лист 8

	Умножение			1200	A	Т шифр	№ 2.14.11 № карты
	$\Omega = конец$	1200	75	0	0007	1221	
	$n-, (1, 0, 0) = n_6$	1	33	1320	0124	1336	
	$B^0 = B$	2	75	0	1223	1216	
	$A^0 \to A^{зап}$	3	56	1222	1721	1206	
	$B+, (0, 0, 1) = B$	4	13	1216	0121	1216	
	$A^{зап}+, n = A^{зап}$	5	13	1721	1320	1721	
	$n-, (1, 0, 0) = n_7$	6	33	1320	0124	1337	
	$0 = S$	7	75	0	0	1741	
	$A^{зап} \to A$	1210	56	1721	1212	1212	
	$A+, (1, 1, 0) = A$	1	13	1212	0126	1212	
A	$a_{11} \cdot v_1 = R_1$	2			н. п.		

				1213	A	Т шифр	№ 2.14.12 № карты
	$S + R_1 = S$	1213	01	1741	1351	1741	
	$n_7-, (1, 0, 0) = n_7$	4	33	1337	0124	1337	
	(0)	5	76	0	0	1211	
B	$S = u_1$	6			н. п.		
	$n_6-, (1, 0, 0) = n_6$	7	33	1336	0124	1336	
	(0)	1220	76	0	0	1204	
	конец	1			н. п.		
A^0	$a_{11} \cdot v_1 = R_1$	2	05	1400	1710	1351	
B^0	$S = u_1$	3	75	0	1741	1700	

Рис. $28_{7, 8}$.

Нахождение собственных чисел матрицы

Составил | Задача | Блок | Стр. 1 | Лист 9

	Нормировка			1240	A	T шифр	№ 2.14.13 № карты
	$\Omega = конец$	1240	75	0	0007	1250	
	$u_1 = \lambda$	1	75	0	1700	1360	
	$n-, (1, 0, 0) = n_{10}$	2	33	1320	0124	1340	
	$Q^0 \to Q$	3	56	1251	1245	1245	
	$Q+, (1, 0, 1) = Q$	4	13	1245	0125	1245	
Q	$u_1 : \lambda = u_1$	5			н. п.		
	$n_{10}-, (1, 0, 0) = n_{10}$	6	33	1340	0124	1340	
	(0)	7	76	0	0	1244	
	конец	1250			н. п.		
Q^0	$u_1 : \lambda = u_1$	1	04	1700	1360	1700	

Составил | Задача | Блок | Стр. 1 | Лист 10

	Невязка			1260	A	T шифр	№ 2.14.14 № карты
	$\Omega = конец$	1260	75	0	0007	1272	
	$n-, (1, 0, 0) = n_{11}$	1	33	1320	0124	1341	
	$P^0 = P$	2	75	0	1273	1265	
	$0 \to q$	3	56	0	1740	1265	
	$P+, (1, 1, 0) = P$	4	13	1265	0126	1265	
P	$u_1 - v_1 = R_2$	5			н. п.		
	$\lvert R_2 \rvert - \lvert 0 \rvert = R_2$	6	03	1352	0	1352	
	$q + R_2 = q$	7	01	1740	1352	1740	
	$n_{11}-, (1, 0, 0) = n_{11}$	1270	33	1341	0124	1341	
	(0)	1	76	0	0	1264	
	конец	2			н. п.		

				1373	A	T шифр	№ 2.14.15 № карты
P^0	$u_1 - v_1 = R_2$	1373	02	1700	1710	1352	

Рис. 28$_{9, 10}$.

Нахождение собственных чисел матрицы

Составил | Задача | Блок | | Стр. 1 | A | T шифр | № 2 14.16 № карты

	Пересылка вектора			1300	A	T шифр	№ 2 14.16 № карты
	$\Omega = конец$	1300	75	0	0007	1306	
	$n-, (1, 0, 0) = n_{12}$	1	33	1320	0124	1342	
	$L^0 \to L$	2	56	1307	1305	1304	
	$L+, (1, 0, 1) = L$	3	13	1305	0125	1305	
	$n_{12}-, (1, 0, 0) = n_{12}$	4	33	1342	0124	1342	
L	(0) $u_1 \to v_1$	5			н. п.		
	конец	6			н. п.		
L^0	(0) $u_1 \to v_1$	7	76	1700	1710	1303	

Составил | Задача | Блок | | Стр. 1 | A | T шифр | № 2.14.17 № карты

	Константы			0121	A	T шифр	№ 2.14.17 № карты
	(0, 0, 1)	0121	0	0	0	0001	
	(0, 1, 0)	2	0	0	0001	0	
	(0, 1, 1)	3	0	0	0001	0001	
	(1, 0, 0)		0	0001	0	0	
	(1, 0, 1)	5	0	0001	0	0001	
	(1, 1, 0)	6	0	0001	0001	0	
				0101	A		
	1_N	0101	101	4000	0	0	

Составил | Задача | Блок | | Стр. 2 | A | T шифр | № 2.14.18 № карты

	Константы			1600	A	T шифр	№ 2.14.18 № карты
	$a_{11} = 4{,}541$	1600	103	4424	7737	1666	
	$a_{12} = 0{,}238$	1	076	7473	3105	5035	
	$a_{13} = -1{,}914$	2	301	7517	6763	5544	
	$a_{21} = 12{,}565$	3	104	6220	5075	3412	
	$a_{22} = 2{,}118$	4	102	4170	6517	6764	
	$a_{23} = 0{,}538 \cdot 10^{-1}$	5	074	6705	6543	4206	
	$a_{31} = 1{,}327$	6	101	5235	5442	6416	
	$a_{32} = -0{,}544$	7	300	4264	1625	4020	
	$a_{33} = 0{,}873$	1610	100	6767	6355	4427	
				1760	A		
	$\varepsilon = 1$	1760	101	4000	0	0	

				2000	A	T шифр	№ 2.14.19 № карты
	N (3, 0, 0)	2000	0	0003	0	0	

Рис. 28$_{11, 12, 13}$.

Мы не станем выписывать отдельно содержательные части программ этих и остальных блоков программы, а отсылаем читателя к рис. 27 и 28, на которых приведены шпаргалка и вся программа задачи в закодированном виде. При этом каждый отдельный блок рекомендуется писать на отдельном бланке, оставляя, если придется, нижние строки пустыми *).

Заметим, что ячейки $n_0 - n_{13}$ являются фиксированными счетчиками. Нет никакой нужды иметь эти счетчики различными во всех блоках, поскольку в данном случае блоки работают независимо друг от друга. Тем не менее следует поступать именно так всегда, чтобы гарантировать себя от ошибок, если блоки будут работать в другой зависимости друг от друга.

Обратим внимание читателя на то, что при кодировке блоков между ними необходимо оставлять несколько свободных ячеек для того, чтобы иметь возможность внести, если это потребуется, необходимые изменения или поправки.

*) На рис. 28 пустые части бланков отрезаны в целях экономии места.

ГЛАВА IV
БИБЛИОТЕКА СТАНДАРТНЫХ ПРОГРАММ

§ 15. Стандартные программы и обращение к ним

При решении самых различных по содержанию задач часто приходится сталкиваться с необходимостью написания одних и тех же блоков. Например, в разнообразных задачах требуется по ходу решения вычислять значения тех или иных элементарных функций, решать системы линейных уравнений с различным числом неизвестных, находить корни функций и т. д. Неразумно, конечно, для каждой задачи писать такие блоки заново. Наиболее употребительные блоки (программы) объединяются в *библиотеку стандартных программ*. Таким образом, библиотека представляет собой набор программ, а каждый программист, решая задачу, может использовать те из них, которые ему нужны для решения этой задачи, в качестве блоков своей программы. Стандартные программы хранятся либо на внешнем запоминающем устройстве (барабан, лента), либо на перфокартах (перфоленте) и вводятся в оперативную память вместе с основной программой. Каждая библиотечная программа имеет известный «начальный адрес», и к ней можно обращаться так же, как к остальным блокам.

Библиотека стандартных программ существенно облегчает работу программиста. Вообще, с точки зрения программиста наличие стандартной программы равносильно тому, что в машине имеется команда, выполняющая ту же самую задачу, только эта команда может записываться не в одну, а в несколько ячеек. Использование стандартных программ как бы увеличи-

вает число команд, имеющихся в машине. Но нельзя забывать, что стандартные программы занимают наравне с блоками основной программы место в оперативной памяти и на работу их уходит значительно больше машинного времени, чем если бы те же операции выполнялись при помощи специальных устройств.

Ввиду того, что стандартные программы используются во многих задачах, к написанию их предъявляются особые требования. Прежде всего программы должны быть написаны так, чтобы обращение к ним было простым и удобным.

Рассмотрим пример. Программа умножения векторов, приведенная в § 10 (см. пример 5.10), является блоком, только в конце вместо команды *«стоп»* следует поставить команду «Б Ω». Однако использование этого блока в качестве стандартной программы было бы неудобно по нескольким причинам. Во-первых, программист кроме адреса начала программы должен знать адреса ячеек a_1, b_1, c, n, что само по себе достаточно неприятно. Кроме того, он должен перед обращением к блоку *Умножение векторов* перенести один из векторов в ячейки $a_1, \ldots, a_n$, другой — в ячейки $b_1, \ldots, b_n$ и поместить в ячейку n размерность вектора. Наконец, совершенно не ясно, на каком расстоянии друг от друга надо разместить ячейки a_1 и b_1. С одной стороны, их хотелось бы разместить как можно ближе друг к другу, чтобы не разбивать память (т. е. чтобы программа была расположена компактно), с другой стороны, расстояние между адресами b_1 и a_1 не должно быть меньше n (размерность векторов), которое заранее неизвестно.

Вместо того чтобы заставлять программиста помещать аргументы в заданные ячейки и указывать ему, откуда брать ответ, при составлении стандартных программ идут по другому пути. Требуют от программиста, чтобы он при обращении к стандартной программе указал адреса аргументов и адреса ячеек, в которых он желал бы получить ответ. Например, для рассматриваемой программы ему достаточно вслед за обращением к стандартной программе *Умножение векторов* как к любому блоку написать две ячейки информации

для стандартной программы *):

$Я$)		$Я+3) \leftrightarrow \Omega$		*Умножение векторов*
$Я+1)$	$\sim$	a_1	b_1	c
$Я+2)$	$\sim$	n	$\sim$	$\sim$,

где a_1 — адрес ячейки, в которой лежит 1-я координата 1-го вектора;
b_1 — адрес ячейки, в которой лежит 1-я координата 2-го вектора;
c — адрес ячейки, в которую следует поместить ответ;
n — размерность вектора, заданная адресно **)

Стандартная программа должна быть составлена так, чтобы расшифровывать данную при обращении к ней информацию. Это принципиально возможно, так как по содержимому правого адреса ячейки Ω легко установить номера ячеек, в которых эта информация записана.

Уже на этом простом примере видны требования, предъявляемые к стандартным программам. Во-первых, вся информация о том, где находятся агрументы, куда положить ответ, рабочее поле и т. д., должна указываться при обращении к ней. Тогда программист перед обращением к стандартной программе не должен будет проделывать никаких предварительных операций.

Во-вторых, от стандартной программы требуется, чтобы она занимала известное число ячеек памяти (желательно расположенных подряд), не зависящее от размерности аргументов. Если для работы программы необходимо рабочее поле переменной длины, то место для него должен указать программист.

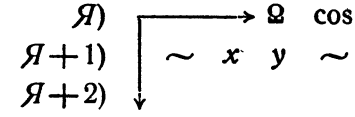
Приведем примеры обращения к стандартным программам.

---

*) $\sim$ (что угодно) означает, что содержимое соответствующего адреса или кода безразлично.

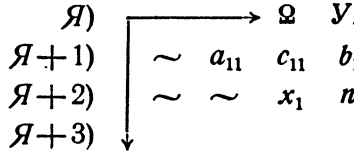
**) В другом варианте n — адрес ячейки, в котором записана размерность векторов; при этом должно быть заранее оговорено, в какой форме записана размерность.

1. Программа вычисления косинуса:



Программа вычислит $y=\cos x$ и передаст управление к $Я+2$.

2. Программа решения линейных уравнений методом Гаусса:



Решается система уравнений

$$a_{11}x_1+\ \ldots\ +a_{1n}x_n=b_1,$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$a_{n1}x_1+\ \ldots\ +a_{nn}x_n=b_n.$$

Элементы матрицы

$$\left\|\begin{matrix} a_{11} & \ldots & a_{1n} \\ \ldots & \ldots & \ldots \\ a_{n1} & \ldots & a_{nn} \end{matrix}\right\|$$

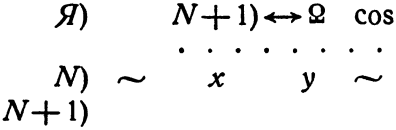
считаются расположенными подряд по строкам, начиная с ячейки a_{11}, свободные члены также считаются расположенными подряд, начиная с ячейки b_1. Программа положит ответ в ячейки $x_1, \ldots, x_n$, лежащие подряд; n — число уравнений, заданное адресно. Для решения системы требуется рабочее поле в n^2 ячеек, c_{11} — адрес первой ячейки этого рабочего поля *).

Примечание. В отличие от (обычных) блоков, после которых можно передать управление любой ячейке, при обращении к стандартной программе в ячейку Ω необходимо заслать передачу управления к ячейке, следующей за информационными, так как это

*) Разумеется, стандартная программа не должна портить ячеек, в которых лежат ее аргументы, иначе можно было бы в качестве рабочего поля использовать саму матрицу $a_{11}, \ldots, a_{nn}$.

единственное указание стандартной программе, где лежит информация. Такие программы называются программами с *принудительным концом* в отличие от первых, называемых программами со *свободным концом*. Однако следование этих информационных ячеек за обращением формально не обязательно.

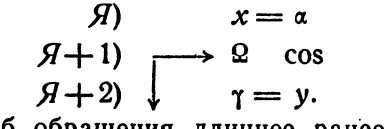
Например, обращение к косинусу может выглядеть и так:



где N — произвольная ячейка.

От требования указания всей информации при обращении к программам библиотеки можно отказаться в так называемых программах без информации, которые точнее было бы назвать программами с малой информацией. Сюда относятся программы вычисления функций одного переменного, выдачи случайной величины и некоторые другие.

Выделим в библиотеке ячейку для аргумента элементарных функций, назовем ее α, и ячейку для ответа, назовем ее γ. Все программы функций одного аргумента устроим как программы, вычисляющие $\gamma = f(\alpha)$. Стандартность этих программ заключается лишь в том, что ячейки α и γ — одни и те же для всех программ этого типа. Если программисту необходимо вычислить $y = \cos x$, то ему нужно написать следующую программу:



Этот способ обращения длиннее ранее рассмотренного, но имеет и свои преимущества. Если аргумент программы получается в результате вычислений и в дальнейшем не нужен, то его можно сразу при вычислении помещать в ячейку α. Точно так же ответ часто можно использовать, не пересылая его из γ в другую ячейку.

Пример 1.15. Вычислить

$$\cos x + \cos(x+h) + \ldots + \cos(x+nh);$$

имеется стандартная программа $\gamma = \cos\alpha$.

Программа выглядит так*):

Программа 1.15

1) $0 = S$
2) $x = \alpha$
3) $n \to Cu$
4) $\alpha + h = \alpha$
5) cos
6) $S + \gamma = S$
7) $Cu - 1_N = Cu$
8) (0)
9) *стоп.*

Легко видеть, что в этом примере мы даже проиграли бы, имея cos как стандартную программу с информацией. Заметим, что сама стандартная программа с информацией всегда длиннее, чем соответствующая программа без информации, и работает дольше. Существуют библиотеки, в которых программы рассмотренного типа строятся как по первому, так и по второму принципу.

§ 16. Описание библиотеки стандартных программ

В этом параграфе мы перечислим некоторые стандартные программы, входящие в библиотеку, используемую в ряде вычислительных центров. Мы рассмотрим программы, составляющие общую часть таких библиотек и используемые всюду. Другие программы, которые определяются спецификой задач, решаемых на данной машине, нами не рассматриваются. Мы считаем, что приведенные нами стандартные программы составляют тот минимум, который необходим для библиотеки стандартных программ любой машины.

*) Напоминаем еще раз, что подобные примеры приводятся лишь в качестве иллюстрации; фактически такую сумму выгоднее считать иначе.

Описание носит справочный характер: оно в большинстве случаев ограничивается перечислением программ и указанием информации, необходимой при обращении к этим программам. Алгоритмов работы этих программ мы не касаемся за недостатком места, хотя нам вполне ясна неразрывная связь алгоритмов и их осуществления в виде программ. При выборе формы задания информации мы руководствовались только одним принципом — удобством использования программистом стандартной программы. В некоторых случаях это делало стандартные программы более сложными.

Некоторые программы библиотеки устроены так, что имеется возможность влиять на их работу с пульта машины. Этой цели служат специальные регистры пульта, устроенные так же, как ячейки памяти. Мы будем предполагать, что на пульте имеется три таких регистра, по 44 разряда каждый. Их мы будем называть ДЗУ1, ДЗУ2, ДЗУ3; разряды в каждом регистре нумеруются справа налево, и для удобства обращения мы будем считать номера разрядов восьмеричными. Информация в эти регистры заносится программистом путем нажатия соответствующих клавиш (нажатая клавиша означает единицу). Машина может брать информацию из этих регистров, как из ячеек памяти с адресами соответственно 7775, 7776, 7777 (см. ниже, § 20).

Библиотека разбита на части, каждая из которых обозначается буквой Б с соответствующим номером и имеет свое наименование.

Б1. Константы и рабочие ячейки. В библиотеке естественно иметь наиболее употребительные константы. К ним в первую очередь относятся фиксированные константы, а именно, константы переадресации и высекатели адресов. Их удобно располагать в следующем порядке *):

Константы переадресации	Высекатели адресов
0, 0, 1	0, 0, *F*
0, 1, 0	0, *F*, 0
0, 1, 1	0, *F*, *F*
1, 0, 0	*F*, 0, 0
1, 0, 1	*F*, 0, *F*
1, 1, 0	*F*, *F*, 0
1, 1, 1	*F*, *F*, *F*

Ячейкам, в которых расположены эти константы, естественно придавать такие адреса, чтобы по адресу можно было угадать содержимое ячейки, и наоборот. Например, последняя цифра адреса ячейки, содержащей константу переадресации, может совпадать с цифрой, которая получается, если рассматривать эту константу как триаду. Так, константа 1, 0, 1, должна лежать в ячейке, адрес которой оканчивается цифрой 5.

*) Буквой *F*, записанной в адресе, мы обозначаем число 7777. Таким образом, 0, 0, *F* означает число 0, 0, 7777.

Кроме фиксированных констант в библиотеке необходимы плавающие константы, т. е. числа, записанные в плавающей форме. Сюда входят целые числа от 1 до 10, дроби $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{8}$, 0,1, 0,01, 0,001, $\frac{\pi}{2}$, $\frac{\pi}{180}$, e, $\ln 2$, $\sqrt{2}$, C (постоянная Эйлера).

Для нужд стандартных программ в библиотеке отведено достаточное количество рабочих ячеек. Этими ячейками можно воспользоваться при программировании как рабочими ячейками программы, однако этого не стоит делать, так как при обращении к стандартным программам библиотеки любая из этих ячеек может быть испорчена, т. е. ее содержимое изменено.

Несколько рабочих ячеек библиотеки имеют специальное назначение. Прежде всего это ячейка α для аргумента и ячейки γ_0, γ_1 для результатов при работе со стандартными программами без информации. Далее, к выделенным ячейкам относится ячейка Ω. О других выделенных ячейках речь будет идти позже.

Стандартная программа не портит своего аргумента, которым может служить ячейка α, либо любая ячейка, указанная в информации. Однако стандартная программа, аргументом которой не является ячейка α, может использовать ее в качестве рабочей и, следовательно, изменить ее содержимое.

Б2. Переводы. Печать. Основой программ этой группы являются программы перевода из одной системы счисления в другую, которые мы будем называть программами «2 → 10» и «10 → 2». Первая из них является необходимой при выводе числового материала из машины. Вторая употребляется несколько реже, так как при наличии небольшого числа входных констант их нетрудно перевести в двоичную систему вручную.

Обе программы являются программами без информации и со свободным концом, т. е. обращение к ним имеет вид

$$Я) \quad S \leftrightarrow \Omega; \quad 10 \to 2$$

или

$$Я) \quad S \leftrightarrow \Omega; \quad 2 \to 10,$$

где S — произвольная ячейка. Для программы «10 → 2», как и для всех стандартных программ без информации, аргументом служит выделенная ячейка α, а ответом — ячейка γ_0. Программа «2 → 10» является в этом смысле единственным исключением. Так как часто приходится печатать ячейки α и γ_0, то для программы «2 → 10» выбраны другие ячейки аргумента и ответа. По аналогии со всеми остальными стандартными программами без информации, ячейки аргумента и ответа для программы «2 → 10» мы будем обозначать $\alpha_{\text{печ}}$ и $\gamma_{\text{печ}}$.

Работа каждой из этих программ состоит в том, что число, лежащее в ячейке, служащей аргументом данной программы, переводится из одной системы счисления в другую и результат помещается в соответствующую ответную ячейку.

Непосредственное обращение к этим программам встречается редко. Чаще всего приходится обращаться к программам печати

или перевода, обрабатывающим целый массив входных или выходных данных. Прежде чем описывать эти программы, напомним, что среди элементарных операций машины мы предполагаем наличие команды десятичной печати. При этом содержимое ячейки рассматривается как двоично-десятичное число в таком виде, как это было написано в § 5 первой главы. Числа печатаются так, как они записываются на бланке (см. там же). В тех же случаях, когда тетрада не соответствует никакой десятичной цифре, на этом месте никакая цифра не печатается, а остается пробел.

Для вывода числового массива из машины обычно применяется одна из двух следующих программ.

Печать по адресу (печать от ... до, *Печать 1*). Здесь в информации к стандартной программе указываются адреса начальной и конечной ячеек печатаемого массива. Обращение к программе имеет вид

Я)		*Я*+2	↔	Ω	*Печать* 1
Я+1)	kl_1l_2	a_1		$\overline{q}$	a_n.

Числа из ячеек от a_1 до a_n включительно переводятся в десятичную систему и печатаются группами по q штук в каждой. Число q задается адресно. Перед каждой группой печатается адрес ячейки, из которой берется первое число группы, после группы дается интервал.

Печать адресов может быть блокирована нажатием определенного разряда какого-либо ДЗУ. Мы будем считать, что это делается нажатием 43-го разряда ДЗУ2. Число kl_1l_2 в коде ячейки информации используется для блокировки печати. Здесь k означает одну из восьмеричных цифр 1, 2 или 3, которую мы будем считать номером ДЗУ, а l_1l_2 — двузначное восьмеричное число от 01 до 54, означающее номер разряда в этом ДЗУ. Нажатие соответствующей клавиши блокирует работу программы печати и эта программа передает управление сразу ячейке $Я + 2$. При $q = 0$ массив от a_1 до a_n выпечатывается без интервалов и без печати адреса.

Пример. При обращении к программе печати с информацией

Я)		*Я*+2	↔	Ω	*Печать* 1
Я+1)	$\overline{341}$	$\overline{4000}$		$\overline{q}$	$\overline{4012}$

произойдет печать, как показано на рис. 29, а затем управление перейдет к $Я + 2$. Если будет нажат 41-й разряд ДЗУ3, то управление сразу перейдет к $Я + 2$, а печати не будет.

В ряде случаев размеры печатаемого массива меняются от варианта к варианту и указать заранее адрес ячейки a_n бывает затруднительно. Тогда можно воспользоваться другой программой печати с иной информацией.

Печать по числу (печать от ... сколько, *Печать 2*). Здесь в информации к стандартной программе указываются начальная ячейка и число ячеек печатаемого массива. Обращение к ней имеет вид

Я)		*Я*+2	↔	Ω	*Печать* 2
Я+1)	kl_1l_2	a_1		$\overline{q}$	n

Программа работает так же, как и печать по адресу, с той лишь разницей, что n означает адрес ячейки, в которой в фиксированной форме в виде числа единиц третьего адреса задано количество чисел печатаемого массива.

$q = 4$

+ +0	4000
+ +00	100 002 000
+ —01	300 405 607
— —02	800 900 010
— +03	200 300 040
+ +0	4 004
— —10	987 654 321
— +08	823 400 000
+ +03	711 700 000
+ +00	000 000 000
+ +0	4 010
— +04	312 541 303
+ —02	156 270 651
— —07	781 353 255

$q = 777$

+ +0	4000
+ +00	100 002 000
+ —01	300 405 607
— —02	800 900 010
— +03	200 300 040
— —10	987 654 321
— +08	823 400 000
+ +03	711 700 000
+ +00	000 000 000
— +04	312 541 303
+ —02	156 270 651
— —07	781 353 255

$q = 0$

+ +00	100 002 000
+ —01	300 405 607
— —02	800 900 010
— +03	200 300 040
— —10	987 654 321
— +08	823 400 000
+ +03	711 700 000
+ +00	000 000 000
— +04	312 541 303
+ —02	156 270 651
— —07	781 353 255

$q = 4$ при нажатом 43-м разряде ДЗУ 2

+ +00	100 002 000
+ —01	300 405 607
— —02	800 900 010
— +03	200 300 040
+ +0	
— —10	987 654 321
— +08	823 400 000
+ +03	711 700 000
+ +00	000 000 000
+ +0	
— +04	312 541 303
+ —02	156 270 651
— —07	781 353 255

Рис. 29.

Аналогичные программы имеются и для перевода числового массива из десятичной системы в двоичную.

В зависимости от задаваемой информации можно пользоваться такими программами.

Перевод по адресу (перевод от ... до, *Перевод 1*). Обращение к этой программе:

$$\begin{array}{llll} Я) & & Я+2 \leftrightarrow Ю & \textit{Перевод 1} \\ Я+1) & kl_1l_2 & a_1 \quad b_1 & a_n. \end{array}$$

Все числа из ячеек от a_1 до a_n включительно переводятся из десятичной системы в двоичную и размещаются в ячейках, начиная с b_1. В случае надобности можно полагать $b_1 = a_1$, и тогда числа, переведенные в двоичную систему, размещаются в тех же ячейках, в которых были десятичные. Однако пользоваться этой возможностью имеет смысл только при недостатке ячеек памяти, так как при этом программа делается несамовосстанавливающейся.

После перевода в двоичную систему следует обратный перевод чисел из ячеек от b_1 в десятичную систему и их печать. Полученную табулограмму можно использовать для проверки перевода и перфорации. Число kl_1l_2 имеет тот же смысл, что и в программах печати: нажатие разряда с номером l_1l_2 на ДЗУ с номером k блокирует обратный перевод и печать.

Перевод по числу (перевод от ... сколько, *Перевод 2*) работает так же, как и *Перевод 1*. Обращение

$$\begin{array}{llll} Я) & & Я+2 \leftrightarrow Ю; & \textit{Перевод 2} \\ Я+1) & kl_1l_2 & a_1 \quad b_1 & n \end{array}$$

отличается от обращения к *Переводу 1* заданием информации: вместо адреса a_n последней ячейки задается адрес n ячейки, в третьем адресе которой задан фиксированным числом размер переводимого массива.

Печать программы. Часто нужно печатать массив ячеек, рассматривая их содержимое не как плавающие числа, а как команды.

В таком случае естественно хотелось бы напечатать эти ячейки в том же виде, в каком они кодируются на бланке, т. е. в восьмеричной форме. Однако поскольку ячейка печатается тетрадами, то выпечатываемую команду нужно предварительно подвергнуть преобразованию. При печатании ячейки на печать выходит лишь десять цифр, принимающих любое значение от 0 до 7 (9 цифр в мантиссе и 1 в порядке, другая цифра порядка принимает лишь значение не больше 3); поэтому печать команды можно произвести лишь в две строки. Для преобразования команд к необходимому виду и выпечатывания их служат следующие две программы:

Печать программы по адресу (от ... до, *ППП*). Обращение:

$$\begin{array}{llll} Я) & & Я+2 \leftrightarrow Ю; & \textit{ППП} \\ Я+1) & kl_1l_2 & a_1 \quad \sim & a_n. \end{array}$$

Программа берет содержимое ячеек a_1, ..., a_n, переводит его в печатную форму и печатает. Перед первой печатаемой ячейкой и перед печатью ячейки с круглым адресом (т. е. оканчивающимся на нуль) печатается адрес очередной ячейки.

В коде число kl_1l_2 имеет то же значение, что и в программе Печать 1 (блокирует печать).

Печать программы по числу (от „сколько, *ПП2*). Обращение:

Я)		*Я*+2 ↔ Ω;		*ПП2*
Я+1)	kl_1l_2	a_1	~	n.

Работает так же, как *ПП1*, но печатается массив не от a_1 до a_n, а от a_1 всего n_3 штук, где n_3 означает число единиц третьего адреса ячейки n_4.

```
++0  2712
++02   05  2230
++0  2224  2230
++03   05  2225
++0  2226  2231
++04   04  2230
++0  2231  2231
++05   02  0101
++0  2231  2072
++06   02  0101
++0  2051  2230
++07   05  2230
++0  2224  2230
++0  2720
++00   05  2225
++0  2227  2231
++01   04  2230
++0  2231  2231
++02   02  0101
++0  2231  2073
```

Рис. 30.

Массив, напечатанный при помощи программы *ПП1* или *ПП2*, имеет вид, показанный на рис. 30 (напечатаны ячейки от 2712 до 2722, содержимое их взято произвольно).

Знаки + и цифра 0, стоящие слева в каждой печати, ничего не обозначают.

Вторая цифра есть последняя цифра адреса печатаемой ячейки. Для сравнения те же ячейки приведены записанными на бланке (рис. 31).

Составил	Задача	Блок		Стр.		Лист	
				2712	А	Шифр	№ карты
		2712	05	2230	2224	2230	
		3	05	2225	2226	2231	
		4	04	2230	2231	2231	
		5	02	0101	2231	2072	
		6	02	0101	2051	2230	
		7	05	2230	2224	2230	
		2720	05	2225	2227	2231	
		1	04	2230	2231	2231	
		2	02	0101	2231	2073	

Рис. 31.

Б3. Функции одного переменного. Все программы вычисления функций одного переменного работают как программы без информации со свободным концом. Обращение к ним

$$S \leftrightarrow \Omega; \quad f$$

где f — начало программы нужной функции, а S — произвольная ячейка. При работе стандартной программы аргумент берется из ячейки α; ответы помещаются: в ячейку γ_0, если ответ один, или в ячейки γ_0, γ_1, если ответа два. После работы программы управление переходит в S.

Имеются следующие программы:

Элементарные функции

e^{α},	$\gamma_0 = e^{\alpha}$,	
$e^{-\alpha}$,	$\gamma_0 = e^{-\alpha}$,	
cos, sin,	$\gamma_0 = \cos\alpha$,	$\gamma_1 = \sin\alpha$,
ch, sh,	$\gamma_0 = \operatorname{ch}\alpha$,	$\gamma_1 = \operatorname{sh}\alpha$,
ln,	$\gamma_0 = \ln\alpha$,	
arcctg, arctg,	$\gamma_0 = \operatorname{arcctg}\alpha$,	$\gamma_1 = \operatorname{arctg}\alpha$,
arccos, arcsin,	$\gamma_0 = \arccos\alpha$,	$\gamma_1 = \arcsin\alpha$.

Таким образом, если ответа два, то из пары функций в ячейку γ_1 всегда помещается нечетная.

Некоторые специальные функции

erf $\quad \gamma_0 = \operatorname{erf} \alpha = \dfrac{2}{\sqrt{\pi}} \int\limits_0^{\alpha} e^{-t^2}\, dt,$

$$\gamma_1 = e^{\alpha^2}(1 - \operatorname{erf} \alpha) = \frac{2}{\sqrt{\pi}} e^{\alpha^2} \int\limits_{\alpha}^{\infty} e^{-t^2}\, dt;$$

Γ $\quad \gamma_0 = \Gamma(\alpha) = \int\limits_0^{\infty} x^{\alpha-1} e^{-x}\, dx.$

Б 4. Обслуживание. В эту группу входит ряд программ, облегчающих работу программиста, стандартизуя некоторые часто встречающиеся детали программ.

Перенос по адресу (перенос от ... до, *Перенос 1*). Программа с информацией. Обращение имеет вид:

$$Я) \quad Я+2 \leftrightarrow Ω \quad \textit{Перенос 1}$$
$$Я+1) \quad a_1 \quad b_1 \quad a_n.$$

Программа переносит содержимое ячеек от a_1 до a_n включительно в ячейки, начиная с b_1.

Чтобы понять необходимость такого рода переносов, достаточно, например, вспомнить блоки *Перенос матрицы* и *Перенос вектора* в примере 2.14.

Перенос по числу (перенос от ... сколько, *Перенос 2*). Обращение к программе:

$$Я) \quad Я+2 \leftrightarrow Ω \quad \textit{Перенос 2}$$
$$Я+1) \quad a_1 \quad b_1 \quad n.$$

Переносится содержимое ячеек начиная от a_1 числом n_3 (n_3 означает содержимое третьего адреса ячейки n) в ячейки, начиная с b_1.

Переводы из фиксированной формы в плавающую. Три программы без информации со свободным концом, обозначаемые $ФП_1$, $ФП_2$, $ФП_3$. Программа $ФП_k$ ($k = 1, 2, 3$) переводит содержимое k-го адреса ячейки α, рассматриваемое как целое число, в плавающую форму и заносит его в ячейку γ_0. Обращение

$$S \leftrightarrow Ω; \quad ФП_k,$$

где S — произвольная ячейка программы.

Например, если содержимое ячейки α: 0053, 0012, 0037, то программа $ФП_3$ занесет в ячейку γ_0 число $37_8 = 31_{10}$ в плавающей форме, т. е. число

105 7600 0000 0000,

а программа $ФП_1$ — число $53_8 = 43_{10}$, т. е. число

106 5300 0000 00000.

Переводы из плавающей формы в фиксированную. Три программы, аналогичные рассмотренным выше, с обращением

$$S \leftrightarrow Ω; \quad ПФ_k.$$

Содержимое ячейки α рассматривается как число в плавающей форме, его целая часть переводится в фиксированную форму и записывается в виде числа единиц k-го адреса ячейки γ_0. Например, если в ячейке α записано число $10{,}3_{10}$, то программа $ПФ_2$ запишет в ячейку γ_0 содержимое (0, 0012, 0).

Фиксированное умножение (*ФИУ*). Программа с принудительным концом. Обращение:

Я)	*Я*+2 ↔ Ω;	*ФИУ*		
Я+1)	*a*	*b*	*c*	

Содержимое адресных частей (мантисс) ячеек a и b рассматривается как два целых числа. Эти числа перемножаются и произведение (либо его правые 36 разрядов, если оно получается бо́льшим) записывается в адресную часть ячейки c. Например, если a: (0, 0, 0025) и b: (0, 0, 0010), то в ячейке c будет (0, 0, 0250).

К группе обслуживания относятся также информационная программа (*ИНФО*) и программа группового формирования команд (*ГФК*), которые будут подробно описаны в параграфе, посвященном формированию команд (см. § 18).

Б5. Линейная алгебра. В эту группу библиотечных программ входят три программы, предназначенные для работы с матрицами.

Решение системы линейных уравнений (*ЛиС*). Обращение:

Я)	*Я*+3 ↔ Ω;	*ЛиС*	
Я+1)	a_{11}	c_{11}	b_1
Я+2)	~	x_1	n

Здесь a_{11} означает адрес начала массива коэффициентов, которые предполагаются записанными подряд; c_{11} — начало рабочего поля, размеры которого совпадают с размерами матрицы коэффициентов. Это рабочее поле необходимо, например, для решения системы методом Гаусса. Можно принять $c_{11} = a_{11}$; в этом случае программа будет работать верно, но матрица коэффициентов будет испорчена. Далее, b_1 — адрес начала столбца правых частей, расположенных подряд, x_1 — начальная ячейка для ответов. Порядок системы n задан адресно.

Обратная матрица (*ОМ*). Обращение к этой программе:

Я)	*Я*+2 ↔ Ω;	*ОМ*	
Я+1)	a_{11}	n	c_{11}.

По квадратной матрице порядка n, расположенной по строкам подряд в ячейках, начиная с a_{11}, строится обратная матрица,

элементы которой располагаются в том же порядке в ячейках, начиная с c_{11}. Порядок матрицы n задан адресно. Разрешается задавать $c_{11} = a_{11}$.

Умножение матриц ($M \times M$). Обращение:

Я)	Я+3 ↔ Ω;		$M \times M$
Я+1)	a_{11}	b_{11}	c_{11}
Я+2)	k	l	m.

Перемножаются матрицы порядков $k \times l$ и $l \times m$, расположенные подряд; a_{11} и b_{11} — адреса их первых элементов. Матрица-произведение записывается подряд в ячейки, начиная с c_{11}.

Б6. Интеграл. В библиотеке имеются две программы интегрирования. Одна из них считает интеграл, разбивая отрезок интегрирования на задаваемые информацией n частей, и на каждой вычисляет его методом Гаусса по пяти точкам.

Обращение к этой программе:

Я)	Я+3 ↔ Ω		$\int$
Я+1)	$x_{\min}$	$x_{\max}$	x
Я+2)	S_1	f	n.

Здесь $x_{\min}$, $x_{\max}$ — нижний и верхний пределы интегрирования, x — переменная интегрирования, f — начало блока счета подынтегральной функции, вычисляющего $\gamma_0 = f(x)$; S_1 — адрес первой из трех идущих подряд рабочих ячеек программы, которые должны быть выделены программистом, n — число участков разбиения интервала интегрирования, заданное адресно. Нуль в коде ячейки $Я + 2$ обязателен. Блок вычисления функции $f(x)$ должен быть написан программистом отдельно и оформлен как обычный блок. Значение интеграла помещается в ячейку γ_0.

Другая программа интегрирования носит название Интеграл с уточнением. Она имеет то же начало, что и предыдущая, но несколько отличное обращение: в коде ячейки $Я + 2$ ставится 001 при вычислении с заданной относительной точностью и 002 при вычислении с заданной абсолютной точностью. В третьем адресе ячейки $Я + 2$ вместо адресно заданного n указывается номер ячейки ε, в котором приведена требуемая точность.

Таким образом, обращение к интегралу с уточнением при вычислении, например, с относительной точностью имеет вид

Я)		Я+3 ↔ Ω		$\int$
Я+1)	~	$x_{\min}$	$x_{\max}$	x
Я+2)	001	S_1	f	ε.

Программа вычисляет интеграл по квадратурным формулам Гаусса с уточнением по формуле уточняющих квадратур А. С. Кронрода *),

*) А. С. Кронрод, Узлы и веса квадратурных формул, «Наука», 1964.

изменяя число участков разбиения до достижения требуемой абсолютной или относительной точности. Значение интеграла, как и в предыдущем случае, помещается в ячейку γ_0.

Обратим внимание читателя на отличие программ интегрирования от всех стандартных программ, встречавшихся ранее. При обращении к программе интеграла указываются не только адреса ячеек аргументов и результатов, но также адрес начала некоторого блока. К этому блоку стандартная программа интеграла обращается в процессе своей работы.

Блок вычисления $f(x)$ может быть как угодно сложной программой. В частности, он может включать вычисление интеграла (например, в случае повторного интегрирования). Программа интегрирования устроена таким образом, что она допускает обращение программы f снова к программе интегрирования. Следовательно, оказывается, что *программа интегрирования допускает обращение к себе самой, как к подпрограмме.* Такие программы мы будем называть *совершенно стандартными программами.*

Необходимо только проследить, чтобы при повторном обращении к этой программе отведенные для ее работы три рабочие ячейки $S_1 - S_3$ были другими. Впервые такая программа была составлена М. З. Розенфельд *).

Б7. Корень функции. Программа вычисляет самый левый на заданном участке корень функции $y = f(x)$. Программа проходит этот участок слева направо с некоторым заранее заданным шагом Δx до первой перемены знака функции. Выделенный корень находится с нужной точностью и помещается в ячейку γ_0. Тем самым предполагается, что начальный шаг Δx выбран достаточно малым. Если функция $f(x)$ не изменяет знака, то в ячейку γ_0 записывается *Щ*.

Обращение к этой программе имеет вид:

Я)	*Я*+4 ↔ Ω		*Корень*
Я+1)	$x_{\min}$	$x_{\max}$	x
Я+2)	S_1	f	y
Я+3)	0	Δx	ε.

Здесь $x_{\min}$ и $x_{\max}$ — левый и правый концы участка, на котором ищется корень; x и y — аргумент и значение функции $y = f(x)$; f — начало блока счета функции; S_1 — первая из трех рабочих ячеек, отведенных для работы программы; Δx — адрес ячейки, содержащей начальный шаг, и, наконец, ε — адрес ячейки, в которой задана требуемая точность вычисления корня.

Программа *Корень функции*, как и программа интегрирования, является совершенно стандартной, т. е. допускает обращение программы f к себе обычным образом. Благодаря этому она может быть применена для решения системы уравнений с несколькими неизвестными.

---

*) См. М. З. Розенфельд, О совершенно стандартных программах, ДАН СССР, 1964, т. 154, вып. 5, стр. 1050—1051.

Б10 *). Табличная функция (ТФ). Программа предназначена для вычисления значений функции $y = f(x)$, заданной таблицей, т. е. является программой интерполяции. По заданному значению аргумента x программа вычисляет значение функции с помощью интерполяционного многочлена заданной степени n. Обращение:

$Я$)	$Я+3 \leftrightarrow$ Ω;			$ТФ$
$Я+1$)	n	$x_{\min}$	$x_{\max}$	x
$Я+2$)	δ	y_0	y_m	y.

Здесь n — степень интерполяционного многочлена, заданная адресно (т. е. в виде фиксированного числа) в коде ячейки $Я + 1$; x и y — адреса ячеек, содержащих значение аргумента и ответ. Значения функции содержатся в идущих подряд ячейках, начиная с y_0 и кончая y_m. Признак δ в коде ячейки $Я + 2$ может принимать значения 0 и 1; $\delta = 0$ есть признак таблицы с постоянным шагом. В этом случае $x_{\min}$ и $x_{\max}$ означают адреса наименьшего и наибольшего значений x, которые могут быть расположены любым образом. Наоборот, $\delta = 1$ есть признак таблицы с произвольным шагом. Тогда $x_{\min}$ и $x_{\max}$ означают адреса начальной и конечной ячеек, в которых подряд расположены соответствующие значения аргумента.

Б11. Бесселевы функции **). Программа вычисляет функции Бесселя первого и второго рода $J_\nu(x)$ и $Y_\nu(x)$, а также функции чисто мнимого аргумента $I_\nu(x)$ при любых действительных ν и x. Обращение к программе:

$Я$)	$Я+2 \leftrightarrow$ Ω			*Бессель*
$Я+1$)	δ	ν	x	y.

Признак δ может принимать значения $\delta = 0$, 1, 2 и указывает, какую из функций требуется вычислять. При этом $\delta = 0$ означает функцию J, $\delta = 1$ — функцию I, а $\delta = 2$ — функцию Y. Далее, x и y означают адреса ячеек аргумента и ответа, ν — адрес ячейки, в которой записано значение индекса вычисляемой функции Бесселя.

Б12. Полиномы Лежандра (PP'). Обращение:

$Я$)	$Я+2 \leftrightarrow$ Ω;			PP'
$Я+1$)	n	x	P_0	P'_0.

Степень вычисляемого полинома задается адресно в коде ячейки информации; x — адрес аргумента. Программа вычисляет значения всех полиномов Лежандра и их производных до степени n включительно. Значения полиномов записываются подряд в ячейки, начиная с P_0, а производных — в ячейки, начиная с P'_0. Кроме того, значения $P_n(x)$ и $P'_n(x)$ записываются в ячейках γ_0 и γ_1.

Если во втором адресе ячейки $Я + 1$ стоит нуль, то записи значений полиномов степеней, меньших n, не будет. Аналогично, если

---

*) По причинам, которые будут разъяснены в § 19, мы применяем восьмеричную нумерацию частей библиотеки.

**) Алгоритм программ вычисления бесселевых функций принадлежит М. Г. Белкиной.

нуль стоит в третьем адресе $Я + 1$, то не будут записываться значения производных. Таким образом, если требуются только значения $P_n(x)$ и $P_n'(x)$, то их можно взять из ячеек γ_0 и γ_1, не занимая других ячеек памяти.

Б13. Интегрирование системы дифференциальных уравнений. В библиотеке имеется программа интегрирования системы дифференциальных уравнений методом Рунге — Кутта. Она состоит из двух программ.

Программы работают с системой дифференциальных уравнений вида

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \ldots, y_n), \qquad i = 1, 2, \ldots, n.$$

Перед началом работы задаются три группы по $n + 1$ ячеек и группа из n ячеек, расположенных подряд:

1) $\tilde{x}$, $\tilde{y}_1$, $\tilde{y}_2$, ..., $\tilde{y}_n$; в $\tilde{x}$ лежит текущее значение аргумента, в $\tilde{y}_1$, $\tilde{y}_2$, ..., $\tilde{y}_n$ — соответствующие текущие значения функции;

2) $x_{раб}$, $y_{1раб}$, $y_{2раб}$, ..., $y_{n\,раб}$ — соответствующие рабочие ячейки;

3) x^0, y_1^0, y_2^0, ..., y_n^0 — перед шагом решения уравнения в этих ячейках помещаются начальные значения.

4) f_1, f_2, ..., f_n, в этих ячейках будут размещаться значения правых частей уравнения. Кроме этого задается адрес ячейки, содержащей шаг Δx по независимой переменной.

Программа счета правых частей должна брать аргументы из ячеек $x_{раб}$, $y_{1раб}$, $y_{2раб}$, ..., $y_{n\,раб}$ и выдавать ответы в ячейки f_1, f_2, ..., f_n, после чего управление должно передаваться ячейке Ω.

Первая из этих программ — подготовка и первый шаг (*ПШРК*). Обращение:

$Я$)		$Я+3 \leftrightarrow \Omega$		*ПШРК*
$Я+1$)		x^0	$x_{раб}$	$\tilde{x}$
$Я+2$)	n	Δx	*СПЧ*	f_1.

Здесь n — порядок системы, заданный адресно в коде ячейки; *СПЧ* — начало блока счета правых частей.

Программа выполняет всю нужную подготовку и вычисляет значения искомых функций после первого шага, помещаемые в ячейки $\tilde{x}$, $\tilde{y}_1$, $\tilde{y}_2$, ..., $\tilde{y}_n$.

Вторая программа — очередной шаг Рунге — Кутта (*ШРК*) — в начале своей работы переносит текущие значения $\tilde{x}$, $\tilde{y}_1$, $\tilde{y}_2$, ..., $\tilde{y}_n$ в ячейки начальных значений x^0, y_1^0, y_2^0, ..., y_n^0, а затем вычисляет текущие значения $\tilde{x}$, $\tilde{y}_1$, $\tilde{y}_2$, ..., $\tilde{y}_n$. Значения x^0, y_1^0, y_2^0, ..., y_n^0 сохраняются. Поскольку вся информация уже указана в обращении к предыдущей программе, программа очередного шага не требует информации. Обращение к ней:

$$Я)\ S \leftrightarrow \Omega \quad ШРК,$$

где S — любая ячейка программы.

Б14. Комплексная библиотека. Для упрощения работы с комплексными числами служат программы комплексной библиотеки. Комплексное число $a_0 + ia_1$ мы полагаем записанным в двух последовательных ячейках: в первой a_0 — действительная часть, во второй a_1 — мнимая.

В том же виде получается ответ. Код информационной ячейки во всех программах произволен.

Умножение $\otimes$. Обращение:

$$\begin{array}{cccc} Я) & Я+2 \leftrightarrow Ω & \otimes & \\ Я+1) & a_0 & b_0 & c_0. \end{array}$$

Программа производит умножение чисел $a_0 + ia_1$ и $b_0 + ib_1$ и кладет действительную часть результата в c_0, а мнимую в c_1.

Деление $\odot$. Обращение:

$$\begin{array}{cccc} Я) & Я+2 \leftrightarrow Ω & \odot & \\ Я+1) & a_0 & b_0 & c_0. \end{array}$$

Вычисляется $(a_0 + ia_1) : (b_0 + ib_1)$; результат заносится в ячейки c_0, c_1.

Перевод числа из алгебраической формы в тригонометрическую *(AT)*. Обращение:

$$\begin{array}{cccc} Я) & Я+2 \leftrightarrow Ω & AT & \\ Я+1) & a_0 & \sim & c_0 \end{array}$$

Число в алгебраической форме $a_0 + ia_1$ переводится в тригонометрическую форму $\rho e^{i\varphi}$. В c_0 заносится $\rho \geqslant 0$, в c_1 заносится φ $(-\pi < \varphi \leqslant \pi)$.

Перевод числа из тригонометрической формы в алгебраическую *(TA)*. Обращение:

$$\begin{array}{cccc} Я) & Я+2 \leftrightarrow Ω & TA & \\ Я+1) & a_0 & \sim & c_0. \end{array}$$

Число $a_0 e^{ia_1}$ переводится в форму $c_0 + ic_1$.

Модуль (mod z). Обращение:

$$\begin{array}{cccc} Я) & Я+2 \leftrightarrow Ω & \mathrm{mod}\, z & \\ Я+1) & a_0 & \sim & c_0. \end{array}$$

В ячейку c_0 заносится число $|a_0 + ia_1|$.

Комплексные функции. Обращение:

$$\begin{array}{cccc} Я) & Я+2 \leftrightarrow Ω & f & \\ Я+1) & a_0 & \sim & c_0, \end{array}$$

где f — начало программы.

Вычисляется $c_0 + ic_1 = f(a_0 + ia_1)$. Имеются программы

$$\ln z, \quad e^z, \quad \sqrt{z}.$$

Несколько по-другому задается информация к программе $\cos z$, $\sin z$. Обращение:

$$Я) \quad Я+2 \leftrightarrow Ω \quad \cos,\ \sin$$
$$Я+1) \quad a_0 \qquad c_0 \qquad d_0.$$

Вычисляется $c_0 + ic_1 = \cos(a_0 + ia_1)$ и $d_0 + id_1 = \sin(a_0 + ia_1)$. В этой программе, если $c_0 = 0$, занесения в c_0 и c_1 не будет; то же для d_0. Это сделано, чтобы можно было вычислить одну из функций $\cos z$ или $\sin z$, не отводя места под другую.

Б15. Случайная величина. Поскольку в машине не предусмотрен специальный датчик случайных чисел, случайная величина вырабатывается алгоритмически специальной программой — программой со свободным концом. Обращение:

$$Я) \quad S \leftrightarrow Ω; \quad СВ.$$

Программа кладет в ячейку γ_0 очередное случайное число, равномерно распределенное на интервале (0,1). Аргументами этой программы служат специальные ячейки (*образующие*), перерабатывающиеся при каждом обращении к программе.

Для того чтобы счет по программе с участием случайной величины можно было повторить или продолжить, применяются две программы настройки образующих.

Абсолютное начало. Обращение: *Я*) $S \leftrightarrow Ω$ *Абсолютное начало*
Продолжение. Обращение: *Я*) $S \leftrightarrow Ω$ *Продолжение.*

Программа *Абсолютное начало* заносит в образующие некоторые константы. Это — восстановление первоначального состояния образующих. Программа *Продолжение* печатает образующие при помощи программы *ПП1*. Ни *Абсолютное начало* ни *Продолжение* не вырабатывают в γ_0 случайной величины.

При необходимости повторить работу программы с момента последнего обращения к *Продолжению* достаточно ввести напечатанные образующие в машину (разумеется, перед этим надо набить их на перфокарту) и начать работу программы, обойдя *Абсолютное начало*. По этой причине обращение к *Абсолютному началу* следует делать первой командой программы.

Б16. Библиотека дальнего плавания. Библиотека дальнего плавания предназначена для работы с повышенной точностью и более широким диапазоном чисел. Для записи каждого числа в форме дальнего плавания отводится $k+1$ ячеек памяти, расположенных подряд. В первой из них (заглавной) в коде записывается адресно число k, а в мантиссе — порядок числа в фиксированной форме. Следующие k ячеек содержат мантиссу числа, которая записана подряд в мантиссе этих ячеек. В коде этих ячеек для удобства действий записывается условный порядок 100_8.

Все программы этой библиотеки имеют общий заголовок ДП. Имеются следующие программы.

Арифметические действия. Пять программ, соответствующих имеющимся в машине арифметическим командам: сложение, вычитание, вычитание модулей, деление и умножение.

Обращение к каждой из этих программ имеет вид:

$$Я) \quad Я+2 \leftrightarrow Ω; \quad ДП$$
$$Я+1) \quad a_0 \quad * \quad b_0 = c_0,$$

где * означает символ требуемой операции. При кодировании машинный код соответствующей операции ставится в коде $Я + 1$, т. е. ячейка $Я + 1$ кодируется по обычным правилам; a_0, b_0, c_0 означают, соответственно, заглавные ячейки чисел, участвующих в операции и результата.

Например, обращение:

$$1244) \quad \overline{1246} \leftrightarrow Ω; \quad ДП$$
$$1245) \quad a_0 \; + \; b_0 = c_0$$

кодируется следующим образом:

16 1246 адрес Ω адрес *ДП*

01 адрес a_0 адрес b_0 адрес c_0.

По окончании работы программы арифметических операций вырабатывается управляющий сигнал ω, так же как и при обычных операциях с плавающими числами.

Логические действия. Имеются программы логического сложения и сверки. Обращение к ним аналогично обращениям к арифметическим операциям. Например, для логического сложения:

$$Я) \quad Я+2 \leftrightarrow Ω; \quad ДП$$
$$Я+1) \quad a_0 \quad \vee \quad b_0 = c_0.$$

Программы вырабатывают сигнал ω, как и при обычных логических операциях.

Во всех арифметических и логических операциях количество ячеек, в которых записаны участвующие в них числа, может быть различным.

Элементарные функции. Обращение к любой программе вычисления элементарных функций имеет вид:

$$Я) \quad Я+2 \leftrightarrow Ω; \quad ДП$$
$$Я+1) \quad f \quad (x_1) = y_1.$$

Здесь x_1 и y_1 — адреса заглавных ячеек аргумента и ответа, которые пишутся, соответственно, во втором и третьем адресах; f — номер программы требуемой элементарной функции, который помещается в первом адресе. Имеются следующие программы:

$$\sin x, \quad \cos x, \quad \ln x, \quad e^x, \quad \arcsin x, \quad \operatorname{arctg} x.$$

Б17. Сервисные программы. Под сервисными программами мы понимаем группу программ, предназначенных для обеспечения удобной работы программиста за пультом, главным образом в процессе отладки. Обращение ко всем этим программам происходит с пульта, и все они после работы выходят на *стоп*. Те из них, которые требуют информации, берут ее также с пульта по состоянию ДЗУ.

В библиотеке имеются следующие программы.

Туда. Программа производит контрольное суммирование всей памяти, за исключением рабочих ячеек библиотеки, и сравнивает полученную сумму с содержимым выделенной ячейки *КΣ*. При несовпадении этих сумм машина останавливается, зажигая обе эти суммы на регистрах пульта. При повторном нажиме на пуск, как и в случае совпадения сумм, происходит запись всей суммируемой части памяти на барабан и печать контрольной суммы. Если машина имеет несколько барабанов *), то перемотка происходит на барабан, номер которого указан в двух старших разрядах ДЗУ2, и этот номер печатается вместе с контрольной суммой.

Обратно. Программа *Обратно* переписывает содержимое барабана в оперативную память. Если барабанов несколько, то перемотка идет с барабана, номер которого указан, как и в предыдущей программе. В конце работы программы печатаются контрольная сумма и номер барабана.

Поправка с пульта. Программа предназначена для контролируемого внесения изменений в какую-либо ячейку памяти. Адрес изменяемой ячейки набирается в среднем адресе ДЗУ1, вводимое вновь содержимое — в ДЗУ3. Осуществляется перемотка программы с барабана в оперативную память, как в программе *Обратно*, затем следует остановка. В регистрах пульта зажигается содержимое ДЗУ1 и ДЗУ3. Теперь программист имеет возможность прочесть и еще раз проверить адрес исправляемой ячейки и новое ее содержимое. При нажиме на пуск новое содержимое вносится в ячейку и производятся снова контрольное суммирование памяти и запись всей программы на барабан. При этом печатаются адрес исправленной ячейки и ее новое содержимое с помощью печати программы, а также новая контрольная сумма.

Надслой. Программа позволяет изменить некоторую числовую константу без предварительного перевода ее в двоичную систему. Адрес изменяемой константы набирается в среднем адресе ДЗУ1, а сама константа в десятичной форме — в ДЗУ3. Программа работает аналогично *Поправке с пульта.*

Подслой. Программа выпечатывает содержимое массива ячеек памяти, начало и конец которого указаны во втором и третьем адресах ДЗУ1. Печать происходит, как и в программе *Печать 1*, группами по 8_{10} ячеек в каждой.

Печать программы с пульта. Программа работает подобно описанной в Б2 программе *Печать программы.* Информация к ней задается, как и в предыдущей программе *Подслой.*

Для управления работой библиотечных программ удобно выделить какое-либо одно ДЗУ. Будем считать, что им является ДЗУ2, и укажем возможное распределение разрядов этого ДЗУ при работе с библиотекой (рис. 32).

Как уже было указано, два старших разряда отводятся для указания номера барабана (если их несколько, но не больше четырех), с которым работают программы *Туда* или *Обратно.* Следующие

*) Разумеется, в этом случае команды перемотки должны иметь другой вид, чем это указано в § 3.

два разряда (52 и 51) можно отвести для работы с магнитными лентами, если они есть и их число не превосходит четырех. 50-й разряд включает работу выходного перфоратора и вывод происходит не только на печать, но и на перфорацию. Если печать не работает или табулограмма не нужна, а достаточен вывод на перфорацию (или на огни), то можно выключить печать нажатием 47-го разряда. Выключение печати удобно также в том случае, когда нужно пройти уже отлаженную часть программы.

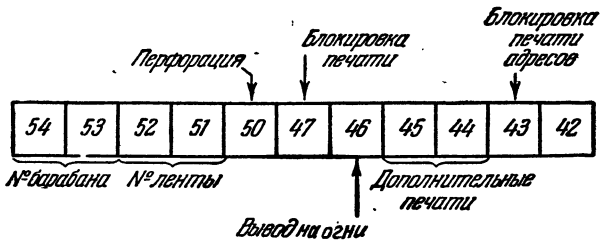


Рис. 32.

Если нажат 46-й разряд ДЗУ2, то при работе всех программ печати перед выдачей на печать или на перфорацию очередного числа машина останавливается, вызывая это число на один из регистров пульта. На другом регистре зажигается адрес ячейки, в котором это число находится.

Если работает программа десятичной печати, то число загорается в десятичной форме; когда же работает печать программы, то на пульт вызывается команда в своем обычном виде. При нажиме на кнопку *Пуск* машина переходит к обработке следующей ячейки, которая должна печататься.

Разряды 45-й и 44-й используются для получения повторных печатей.

При каждом обращении к программе печати печатаемая группа будет напечатана дополнительно столько раз, каково четверичное число, записанное в этих разрядах.

Как уже было упомянуто, 43-й разряд используется для блокировки печати адресов ячеек.

§ 17. Примеры программ с использованием библиотеки

Пример 1.17. Пусть заданы n значений $x_1, x_2, \ldots, x_n$ аргумента и соответствующие им n значений функции $y_1, y_2, \ldots, y_n$, полученные как результаты эксперимента. Требуется подобрать по этим экспериментальным данным параметры a_1 и a_2 функции вида $y = e^{a_1+a_2x}$ по способу наименьших квадратов, вычислить соответствующие заданным значениям аргумента сглаженные значения функции и напечатать найденные значения параметра и отклонения вычисленных значений от экспериментальных.

Как известно, параметры a_1 и a_2 могут быть найдены из системы линейных уравнений *)

$$a_1 n + a_2 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} \ln y_i,$$

$$a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i \ln y_i.$$

Дело сводится, таким образом, к составлению и решению системы линейных уравнений.

Блок-программу задачи можно написать так:

Программа 1.17

Ввод данных
Составление матрицы
Решение системы
Счет отклонений
Печать
Стоп

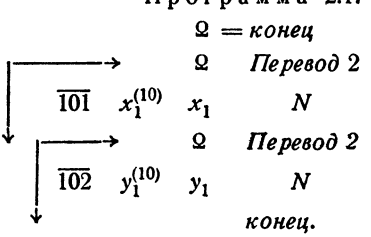
Блок *Ввод данных* состоит из обращений к программе перевода. Так как для работы программы нужна ячейка

$$N = (0, 0, n),$$

в которой число значений аргумента задано в фиксированной форме в третьем адресе, то мы воспользуемся программой перевода по числу (*Перевод 2*).

Программа блока *Ввод данных* может быть тогда записана так:

Программа 2.17 (*Ввод данных*)



Блок *Составление матрицы* требует перевода n в плавающую форму и вычисления нескольких сумм.

*) См., например, Р. С. Гутер и Б. В. Овчинский, Элементы численного анализа и математической обработки результатов опыта, Физматгиз, 1962, стр. 347.

Программа 3.17 (*Составление матрицы*)

```
                Ω      = конец
           (0, 0, n)   = α
              ФП3
                γ0     = a11
                0      = a12
                0      = a22
                0      = b1
                0      = b2
        N —, (0, 0, 1) = nраб
                S0     = S
          T0  → T;     S
□       S +, (0, 1, 0) = S
        T +, (0, 1, 0) = T
S                (x1 = xраб)        н. п.
      a12 +  xраб    = a12
      xраб ·  xраб   = R0
       a22 +   R0    = a22
T                (y1   = α)         н. п.
                 In
      b1 +      γ0     = b1
       xраб ·   γ0     = R1
       b2 +     R1     = b2
    nраб —, (0, 0, 1) = nраб
 (0)  a12 →  a21;     □
          конец
S0               x1   = xраб
T0               y1   = α.
```

Для решения уравнений достаточно просто обратиться к соответствующей стандартной программе.

Программа 3.17 (*Решение системы*)

```
        Ω  = конец
   ┌──→Ω     ЛиС
   │  a11  c11   b11
   │   0   a1    2
   ↓
   конец.
```

Блок *Счет отклонений*, который включает вычисление функции по формуле

$$y = e^{a_1 + a_2 x}$$

с полученными параметрами также не вызывает сомнений.

Программа 4.17 (*Счет отклонений*)

```
                Ω    = конец
          N —, (0, 0, 1) = счетчик
                U⁰   =  U
          V⁰ →,  V
          U +, (1, 0, 0) =  U
          V +, (0, 1, 1) =  V
U         (x₁ ·   a₂   =  R₂)  ↓   н. п.
          a₁ +    R₂   =  α
                  e^α
V         (γ₀ —   y₁   =  ε₁)      н. п.
     счетчик —, (0, 0, 1) = счетчик
(0)
     конец
U⁰        x₁ ·   a₂   =  R₂
V⁰        γ₀ —   y₁   =  ε₁
```

Остается написать блок печати. Он будет иметь вид:

Программа 5.17 (*Печать*)

```
          Ω   =   конец
              Печать 1
a₁        2̄          a₂
              Печать 2
ε₁        10̄          N
     конец.
```

Параметры a_1, a_2 естественно напечатать отдельной парой. Что касается отклонений, то их удобнее печатать группами по $10_8 = 8_{10}$ штук, с тем чтобы затем сравнивать с напечатанными программой ввода измеренными значениями функций.

Программа в закодированном виде приведена на рис. 33.

Подбор параметров

Составил | Задача | Блок | Стр. 1 | Лист 1

	Собирающая			5000	A	T шифр	№ 1.17.1 № карты
	Ввод данных	5000	16	5001	0007	5020	
	Составление матрицы	1	16	5002	0007	5040	
	Решение уравнения	2	16	5003	0007	5100	
	Счет отклонений	3	16	5004	0007	5120	
	Печать	4	16	5005	0007	5150	
	стоп	5	77	0	0	0	

					A	шифр	№ карты

Рис. 33_1

Подбор параметров

Составил Задача Блок Стр. 1 Лист 2

	Ввод данных			5020	A	T шифр	№ 1.17.2 № карты
	2 = конец	5020	75	0	0007	5025	
	Перевод 2	1	16	5023	0007	7742	
101	$x_1^{(10)} x_1 N$	2	101	1000	1500	3000	
	Перевод 2	3	16	5025	0007	7742	
102	$y_1^{(10)} y_1 N$	4	102	2000	2500	3000	
	конец	5			*к.п.*		

					A	шифр	№ карты

Рис. 33_2.

Подбор параметров

Составил Задача Блок Стр. 1 Лист 3

	Составление матрицы			5040	A	T шифр	№ 1.17.3 № карты
	$\Omega = конец$	5040	75	0	0007	5070	
	$(0, 0, n) = \alpha$	1	75	0	3001	0140	
	$ФП_3$	2	16	5043	0007	7733	
	$\gamma_0 = a_{11}$	3	75	0	0160	3002	
	$0 = a_{12}$	4	75	0	0	3003	
	$0 = a_{22}$	5	75	0	0	3005	
	$0 = b_1$	6	75	0	0	3006	
	$0 = b_2$	7	75	0	0	3007	
	$N-, (0, 0, 1) = n_{раб}$	5050	33	3000	0121	3010	
	$S^0 = S$	1	75	0	5071	5055	
	$T^0 \succ T$	2	56	5072	5061	5055	

				5053	A	T шифр	№ 1.17.4 № карты
V	$S+, (0, 1, 0) = S$	5053	13	5055	0122	5055	
	$T+, (0, 1, 0) = T$	4	13	5061	0122	5061	
S	$x_1 = x_{раб}$ ↓	5			н. п.		
	$a_{12} + x_{раб} = a_{12}$	6	01	3003	3011	3003	
	$x_{раб} \cdot x_{раб} = R_0$	7	05	3011	3011	3020	
	$a_{22} + R_0 = a_{22}$	5060	01	3005	3020	3005	
T	$y_1 = \alpha$	1			н. п.		
	ln	2	16	5063	0007	7714	
	$b_1 + \gamma_0 = b_1$	3	01	3006	0160	3006	
	$x_{раб} \cdot \gamma_0 = R_1$	4	05	3011	0160	3021	
	$b_2 + R_1 = b_2$	5	01	3007	3021	3007	

Рис. 33$_3$.

Подбор параметров

Составил Задача Блок Стр. 2 Лист 4

	Составление матрицы			5066	A	T шифр	№ 1.17.5 № карты
	$n_{раб}$ —, (0, 0, 1)=$n_{раб}$	5066	33	3010	0121	3010	
	(0) a_{12} → a_{21}; V	7	76	3003	3004	5053	
	конец	5070			*н. п.*		
S^0	$x_1=x_{раб}$	1	75	0	1500	3011	
T^0	$y_1=\alpha$	2	75	0	2500	0140	

					A	шифр	№ карты

Рис. 33$_4$.

Подбор параметров

Составил Задача Блок Стр. 1 Лист 5

	Решение уравнения			5100	A	T шифр	№ 1.17.6 № карты
	Ω=*конец*	5100	75	0	0007	5104	
	ЛиС	1	16	5104	0007	7701	
	a_{11} c_{11} b_1	2	0	3002	3032	3006	
	0 a_1 N	3	0	0	3030	3000	
	конец	4			*н. п.*		

					A	шифр	№ карты

Рис. 33_5.

Подбор параметров

Составил Задача Блок Стр. 1 Лист 6

	Счет отклонений			5120	A	T шифр	№ 1.17.7 № карты
	$\Omega = конец$	5120	75	0	0007	5134	
	$N-$, (0, 0, 1)$=счетчик$	1	33	3000	0121	3027	
	$U^0 = U$	2	75	0	5135	5126	
	$V^0 \to V$	3	56	5136	5131	5126	
	$U+$, (1, 0, 0)$=U$	4	13	5126	0124	5126	
	$V+$, (0, 1, 1)$=V$	5	13	5131	0123	5131	
U	$(x_1 \cdot a_2 = R_2)$	6			*н. п.*		
	$a_1 + R_2 = \alpha$	7	01	3030	3022	0140	
	e^{α}	5130	16	5131	0007	7710	
V	$(\gamma_0 - y_1 = \varepsilon_1)$	1			*н. п.*		
	счетчик$-$, (0, 0, 1)$=счетчик$	2	33	3027	0121	3027	

				5133	A	T шифр	№ 1.17.8 № карты
	(0)	5133	76	0	0	5124	
	конец	4			*н. п.*		
U^0	$x_1 \cdot a_2 = R_2$	5	05	1500	3031	3022	
V^0	$\gamma_0 - y_1 = \varepsilon_1$	6	02	0160	2500	3500	

Рис. 33$_6$.

Подбор параметров

Составил Задача Блок Стр. 1 Лист 7

	Печать			5150	A	T шифр	№ 1.17.9 № карты
	2 = *конец*	5150	75	0	0007	5155	
	Печать 1	1	16	5153	0007	7751	
	a_1 $\overline{2}$ a_2	2	0	3030	0002	3031	
	Печать 2	3	16	5155	0007	7752	
	ε_1 $\overline{10}$ N	4	0	3500	0010	3000	
	конец	5			*н. п.*		

					A	шифр	№ карты

Рис. 33$_7$.

Пример 2.17. Вычислить двойной интеграл

$$I = \int_G \int f(x) \exp \sin (x + y)\, dx\, dy,$$

где $f(x)$ задана таблицей, область G ограничена кривой

$$(x^2 + y^2)^3 - x^3 (x^2 + y^2) - \frac{3}{4} x^4 = 0.$$

Напишем программу в виде:

Программа 6.17

Перевод 1

$z^{10}(0)$ z_0 $z^{10}\left(\frac{3}{2}\right)$

Интеграл

Печать 1

I 0 *I*

стоп.

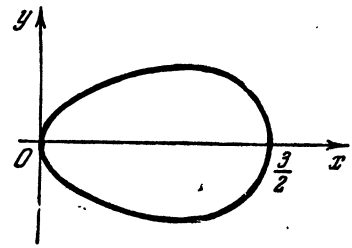


Рис. 34.

Из вида области (рис. 34) следует, что x меняется на интервале $(0, {}^3/_2)$.

При этом мы предполагаем, что таблица функции задана на участке $(0, {}^3/_2)$ и ее значения вводятся подряд в десятичной системе в ячейки от $z^{10}(0)$ до $z^{10}({}^3/_2)$, адреса которых нам известны.

Блок *Интеграл* содержит обращение к стандартной программе вычисления интеграла с уточнением. Для этого представим наш интеграл в виде

$$I = \int_0^{3/2} F_1(x)\, dx,$$

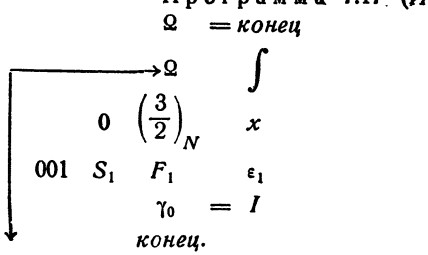
где

$$F_1(x) = f(x) \int_{y_1(x)}^{y_2(x)} \exp \sin (x + y)\, dy,$$

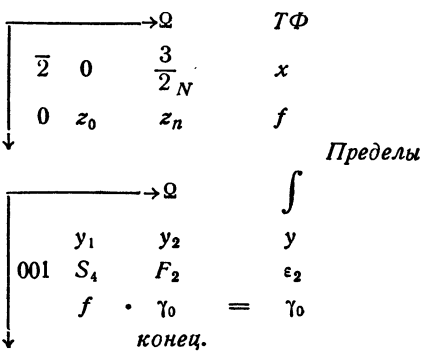
причем пределы интегрирования $y_1(x)$ и $y_2(x)$ нужно находить, решая уравнение, задающее границу области, относительно y при данном x. Мы получим программу 7.17.

В свою очередь блок F_1, считающий подынтегральную функцию, тоже содержит обращение к интегралу (см. программу 8.17).

Программа 7.17 (*Интеграл*)

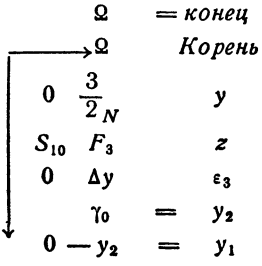


Программа 8.17 (*Функция* F_1)



Блок *Пределы* служит для нахождения пределов интегрирования; для этого нужно обращаться к программе «Корень». Из того, что уравнение содержит y лишь в четных степенях, следует, что уравнение при любом x должно иметь два корня, симметричные относительно оси Ox. Поэтому достаточно искать один корень y_2, лежащий между нулем и $^3/_2$.

Программа 9.17 (*Пределы*)



Остается написать программы F_2 для подынтегральной функции внутреннего интеграла и F_3 для левой части уравнения.

Программа 10.17 (*Функция* F_2)

$$\begin{aligned} \Omega &= \textit{конец} \\ x + y &= \alpha \\ &\quad \cos,\ \sin \\ \gamma_1 &= \alpha \\ &\quad e^{\alpha} \end{aligned}$$

конец.

Программа 11.17 (*Функция* F_3)

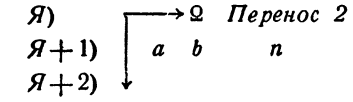
$$\begin{aligned} \Omega &= \textit{конец} \\ x \cdot x &= x^2 \\ x \cdot x^2 &= x^3 \\ x^2 \cdot x^2 &= x^4 \\ y \cdot y &= y^2 \\ x^2 + y^2 &= R_1 \\ R_1 \cdot R_1 &= R_2 \\ R_1 \cdot R_2 &= R_2 \\ R_1 \cdot x^3 &= R_1 \\ R_2 - R_1 &= R_1 \\ x^4 \cdot 3_N &= R_2 \\ \left(\frac{1}{4}\right)_N \cdot R_2 &= R_2 \\ R_1 - R_2 &= z \end{aligned}$$

конец.

§ 18. Формирование команд. Примеры стандартных библиотечных программ

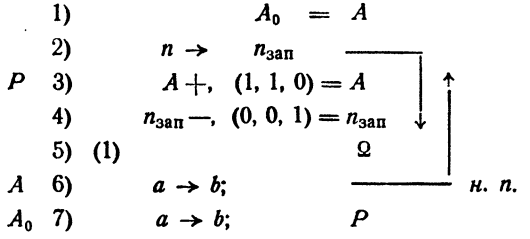
Перейдем теперь к вопросу о том, как писать стандартную программу, т. е. как извлекать и обрабатывать полученную от программиста информацию. Мы рассмотрим несколько примеров, стандартных программ. В некоторых случаях информация к ним не совпадает с той, которая была предложена в предыдущем параграфе.

Пример 1.18. Написать стандартную программу *Перенос по числу (Перенос 2)*, которая по обращению



переносила бы содержимое ячеек a, $a+1$ и т. д. соответственно в ячейки b, $b+1$ и т. д., всего n чисел, где n задано адресно.

Если бы все числа a, b, n были известны, то программа выглядела бы так:



При написании стандартной программы мы, очевидно, не имеем восстановителя A_0 (так как не знаем заранее адресов a и b) и ячейки n, поэтому для первоначального создания команды и счетчика $n_{зап}$ мы можем использовать только ячейку, где лежит информация. Прежде всего нужно ее извлечь.

В правом адресе ячейки Ω стоит число $Я+2$, на единицу большее адреса ячейки с информацией, а в двух других адресах — нули. Чтобы перенести информацию, скажем, в ячейку $R0$, нужно выполнить команду

$$0 \vee Я+1 = R0. \qquad (1)$$

Чтобы получить такую команду, надо сдвинуть адресную часть ячейки Ω на 12 разрядов, т. е. на один адрес влево, затем к третьему адресу полученной ячейки $R1$ прибавить $R0$, из второго адреса вычесть единицу и заменить код в ячейке $R1$ на код логического сложения. Заготовим ячейку B_0 с содержимым

$$B_0: \quad F \vee F = R0;$$

чтобы получить команду (1), достаточно сложить B_0 с $R1$:

1) $\overline{114} \Rightarrow$ Ω $= R1$
2) $B_0 +$, $R1$ $= Q$
Q 3) (0 $\vee$ $(Я+1) = R0$) н. п.

Это типичный пример *формирования команд*, частным случаем которого является переадресация, рассмотренная в гл. II, § 10. Формированием команды мы будем называть одно или несколько действий, цель выполнения которых — создание команды.

В результате выполнения написанных трех команд мы перенесли информацию в известную нам ячейку $R0$.

Следующие действия имеют целью формирование команды A и счетчика $n_{зап}$:

4) $(F, F, 0) \wedge R0 = R1$

5) $(0 \to 0, P) \vee R1 = A$

6) $R0 \wedge (0, 0, F) = n_{зап}$.

Этим заканчивается восстановление. Дальше стандартная программа ничем не отличается от обычной.

Ее окончательный вид таков:

	1)	$\overline{114} \Rightarrow \Omega$	$= R1$
	2)	$(F \vee F = R0) +, R1$	$= Q$
Q	3)	$(0 \vee (Я + 1)$	$= R0)$ *н. п.*
	4)	$(F, F, 0) \wedge R0$	$= R1$
	5)	$(0, 0, P) +, R1$	$= A$
	6)	$R0 \wedge (0, 0, F)$	$= n_{зап}$
	7) Б		9) *)
P	8)	$A +, (1, 1, 0)$	$= A$
	9)	$n_{зап} -, (0, 0, 1)$	$= n_{зап}$
	10) (1)		Ω
A	11)	$(a \to b$	) *н. п.*
	12)	$F \vee F$	$= R0$
	13)	$0 \to 0;$	P

Команды 12) и 13) — константы, которые используются при формировании. Константы $(0, 1, 0)$, $(1, 0, 1)$, $(0, 0, F)$ и $(F, F, 0)$ мы здесь не выписываем, так как они входят в состав констант библиотеки.

Пример 2.18. Написать стандартную программу

$$z = x^y$$

с обращением:

*) Здесь можно было бы сэкономить ячейку следующим образом. Если опустить команду 7), то к команде A прибавится $(1, 1, 0)$. Если бы мы заранее сделали команду A на $(1, 1, 0)$ меньше, чем нужно в момент ее первого исполнения, то команду 7) можно было бы опустить; для этого достаточно взять константу $(0 \to 0; P)$ на $(1, 1, 0)$ меньше, т. е. положить ее равной $(0 \to 0; P) -, (1, 1, 0) = (F - 1 \to F; P)$, что означает $(\overline{7776} \to \overline{7777}; P)$.

считая, что в библиотеке уже имеются стандартные программы

$$\gamma = e^{\alpha}$$

и

$$\gamma = \ln \alpha,$$

работающие как программы без информации.

Если бы адреса x, y, z были известны, то программа выглядела бы так:

1) $\Omega = конец$
A 2) $x = \alpha$
3) $\longrightarrow \Omega \quad \ln \alpha$
B 4) $\downarrow \gamma \cdot y = \alpha$
5) $\longrightarrow \Omega \quad e^{\alpha}$
C 6) $\downarrow \gamma = z$
7) *конец.*

В этой программе нам предстоит сформировать команды 2), 4), 6). Прежде всего перенесем информацию в рабочую ячейку.

1) $\Omega = конец$
2) $\overline{114} \Rightarrow \quad \Omega = R1$
3) $(F \vee F = R0) +, \quad R1 = Q$
Q 4) $(0 \quad \vee (Я + 1) = R0) \quad н.\ п.$

Из ячейки $R0$, в которой теперь лежит информация (x, y, z), нам нужно сформировать команды

$$x = \alpha$$
$$\gamma \cdot y = \alpha$$
$$\gamma = z.$$

Первую команду удобнее формировать в виде $x \vee 0 = \alpha$:

5) $R0 \wedge (F, 0, 0) = R1$
6) $(0 \vee 0 = \alpha) +, \quad R1 = A$
7) $R0 \wedge (0, F, 0) = R1$
8) $(\gamma \cdot 0 = \alpha) +, \quad R1 = B$
9) $R0 \wedge (0, 0, F) = R1$
10) $(0 \vee \gamma = 0) +, \quad R1 = C$

При формировании команд A, B, C мы каждый раз из $R0$ извлекаем нужный адрес и добавляем к соответствующей константе. Далее следует сама программа:

A 11) $(x \vee 0 = \alpha)$ *н. п.*
12) $\longrightarrow \Omega \ln \alpha$
B 13) $(\downarrow \gamma \cdot y = \alpha)$ *н. п.*
14) $\longrightarrow \Omega \ e^{\alpha}$
C 15) $(\downarrow 0 \vee \gamma = z)$ *н. п.*
16) *конец*

константы
17) $0 \vee \gamma = 0$
18) $0 \vee 0 = \alpha$
19) $\gamma \cdot 0 = \alpha$.

Вообще превращение произвольной программы в стандартную достигается формированием тех команд, которые зависят от информации, т. е. добавлением в начале программы некоторого числа команд. При этом становится ненужным восстановление переменных команд, если последние формируются.

Число команд программы, которые нужно сформировать, может быть весьма велико; это могут быть не только арифметические или логические команды, но и команды передачи управления, информационные ячейки и т. д. Иногда формируются почти все команды основной программы.

Рассмотрим еще один пример.

Пример 3.18. Составить стандартную программу для вычисления корня уравнения

$$f(x) = 0.$$

Информация к программе:

$Я$)	$Я+3 \leftrightarrow \Omega$		*Корень*
$Я+1$)	a	b	x
$Я+2$)	f	Δx	ε.

Коды ячеек информации $Я+1$ и $Я+2$ могут быть любыми.

Здесь a и b — ячейки, в которых указаны начало и конец отрезка, на котором ищется корень; f — адрес первой ячейки программы, вычисляющей функцию

$$\gamma = f(x);$$

x — аргумент этой программы (ответ ее γ); Δx — первоначальный шаг, с которым проходится отрезок $[a, b]$; ε — точность, с которой должно быть вычислено значение корня. Найденное значение корня должно быть положено в ячейку γ.

Программа, только не стандартная, была приведена в § 14, стр. 105. Мы приведем ее здесь лишь с небольшими изменениями:

1) $\Omega = конец$
2) $\Delta x = \delta$
3) $a = x$
4) $\longrightarrow \Omega\ f$
5) $\gamma = \xi$
6) $x + \quad \delta \quad = x$
7) $b - \quad x \quad = 0$
8) (1) $Щ \rightarrow\ \gamma;\ конец$
9) $\longrightarrow\ \Omega;\ f$
10) $\xi \cdot \quad \gamma \quad = R_0$
11) $R_0 + \quad 0 \quad = 0$
12) (0) $\gamma \rightarrow\ \xi$
13) $\delta \cdot \left(-\frac{1}{2}\right)_N = \delta$
14) $|\varepsilon| - |\delta| \quad = 0$
15) (1) $x \rightarrow\ \gamma; \quad 6)$
16) $конец.$

Алгоритм нахождения корня здесь следующий: определяем значение $f(x)$ при $x = a$ и кладем его в ячейку ξ (3), 4), 5)), затем вычисляем значение $f(x)$, изменяя x на $\delta = \Delta x$ (6)), пока мы не выйдем за пределы отрезка (7), 8)) или пока на отрезке $(x, x + \delta)$ функция не будет принимать значения противоположных знаков (9), 10), 11), 12)).

В первом случае программа заканчивается и в ответ выдается $Щ$ — самое большое возможное в машине число, в знак того, что корень не найден (8)).

Во втором случае отрезок $(x, x + \delta)$ проходится в обратном направлении с вдвое меньшим шагом (13)) и т. д., пока δ не окажется меньше заданного эталона (14), 15)). Тогда очередное значение x объявляется корнем.

Посмотрим, какие команды этой программы нужно формировать, чтобы сделать программу стандартной. Для этого, очевидно, достаточно проверить, в каких командах упоминаются ячейки, адреса которых задаются в информации.

Нужно сформировать команды: 2), 3), 4), 6), 7), 9), 14), 15). Восемь из четырнадцати рабочих команд программы, оказывается, нужно формировать! Как мы уже видели, для формирования каждой команды нужно потратить много ячеек: кроме самих формирующих команд нужно держать еще константы. Поэтому в стандартной программе желательно иметь как можно меньше формируемых команд.

Приведенную выше программу нетрудно изменить, значительно уменьшив число формируемых команд.

```
    1)        Ω          = конец
A   2)        Δx         = δ
B   3)        a          = z
    4)   ┌──→ Ω,          17)
    5)   ↓     γ          = ξ
    6)     z + δ         = z
C   7)     b — z         = 0
    8) (1) Щ → γ; конец
    9)   ┌──→Ω                    ┐
   10)   ↓ξ · γ          = R0     │
   11)     R0 + 0        = 0      │
   12) (0)   γ → ξ;        6)     │
   13)     δ · (−1/2)    = δ      │
D  14)     |ε| — |δ|     = 0      │
   15) (1)  z → γ;         6)     │
   16)                  конец     │
E  17)     z → x; f                ↓
```

Здесь вместо ячейки x для внутреннего пользования заведена ячейка z. Пересылка же из z в x производится одновременно с обращением к функции. Кроме того, обращение к программе происходит только из одного места. Команда 17 служит как бы заголовком к программе f.

Хотя программа удлинилась на одну ячейку, но вместо восьми команд, которые нужно сформировать, в ней осталось лишь пять (2), 3), 7), 14), 17)). Напишем формирование этих команд. Начнем, как и раньше, с перенесения информации в рабочие ячейки. В отличие от предыдущих примеров здесь нужно перенести две ячейки: $Я+1$ и $Я+2$.

```
    1)                          Ω = конец
    2)              114⇒        Ω = R1
    3) (F ∨ F = R1)+,           R1 = Q
Q   4)              (0 ∨ (Я+2) = R1)  н. п.
    5)              Q —, (0, 1, 1) = P
P   6)              (0 ∨ (Я+1) = R0)  н. п.
```

Теперь содержимое ячейки $Я+1$, непосредственно следующей за обращением к программе *Корень*, перенесено в $R0$, а содержимое

$Я+2$ — в $R1$, т. е. теперь в

$$R0\colon \sim a \quad b \quad x$$
$$R1\colon \sim f \quad \Delta x \quad \varepsilon.$$

Затем следует само формирование:

7)	$R1 \wedge (0,\ F,\ 0) = R2$
8)	$(0 \vee 0 = \delta) \vee R2 = A$
9)	$R0 \wedge (F,\ 0,\ 0) = R2$
10)	$(0 \vee 0 = z) \vee R2 = B$
11)	$R0 \wedge (0,\ F,\ 0) = R2$
12)	$\overline{114} \Rightarrow, R2 = R2$
13)	$(0 - z = 0) \vee R2 = C$
14)	$\overline{130} \Rightarrow, R1 = R2$
15)	$(\lvert 0 \rvert - \lvert \delta \rvert = 0) \vee R2 = D$
16)	$R0 \wedge (0,\ 0,\ F) = R2$
17)	$\overline{114} \Rightarrow, R2 = R2$
18)	$\overline{50} \Rightarrow, R1 = R3$
19)	$R2+, R3 = R2$
20)	$(z \to 0;\ 0) \vee R2 = E.$

Далее без всяких изменений выписывается сама программа, начиная со 2-й ее команды (1-я команда $\Omega =$ *конец* уже написана).

Мы не будем переписывать программу. К ней, разумеется, следует приписать константы, которые были использованы при формировании:

$$F \vee F = R1$$
$$0 \vee 0 = \delta$$
$$0 \vee 0 = z$$
$$0 - z = 0$$
$$\lvert 0 \rvert - \lvert \delta \rvert = 0$$
$$z \to 0;\ 0.$$

Итак, на формирование пяти команд пришлось потратить 19 команд и держать шесть констант, т. е. добавить к основной программе 25 ячеек. Для формирования каждой команды требуется дополнительно минимум три ячейки (две команды и константа), а на формирование команды E даже шесть. Время работы стандартной программы мало отличается от времени работы нестандартной, так как команды формирования выполняются лишь единожды, в то время как программа включает в себя цикл, повторяющийся много раз, и в цикле обращается к программе счета функции, которая сама может быть очень громоздкой. Однако стандартная программа занимает в 2,5 раза больше места, чем нестандартная.

Из рассмотренных примеров видно, что обработка информации, которая задается стандартной программе, состоит из двух частей: первая часть — это перенесение информации, лежащей в ячейках $Я+1$, $Я+2$ и т. д., в рабочие ячейки ($Я$ — ячейка обращения); вторая часть — формирование команд, зависящих от информации. При этом само формирование команд осуществляется обычно применением весьма небольшого числа действий. Наиболее частые из них:

1. Высечение адреса из информационной ячейки.
2. Сдвиг ячейки на один или два адреса, т. е. так, чтобы один адрес сдвинулся на место другого.
3. Сложение (или вычитание) подготовленных таким образом ячеек с заранее заготовленными константами.

В связи с этим можно создать стандартные программы для переноса информации и формирования команд. В предыдущем параграфе мы уже говорили, что предполагаем наличие в библиотеке двух таких обслуживающих программ (стр. 133). Здесь мы дадим их полное описание.

ИНФО. Программа служит для переноса информации к стандартной программе в рабочие ячейки библиотеки. Программа со свободным концом. Обращение:

$$S \longleftrightarrow \textit{Конец инфо};\ \textit{ИНФО}.$$

Из правого адреса ячейки Ω программа берет номер m. Содержимое четырех ячеек

$$m-4,\ m-3,\ m-2,\ m-1$$

переносится в выделенные идущие подряд ячейки библиотеки *Инф1, Инф2, Инф3, Инф4.* Поскольку аргументом этой программы служит ячейка Ω, эта программа имеет свой конец.

Обычно обращение к *ИНФО* бывает первой командой стандартной программы. Возьмем для примера только что разобранную программу *Корень*. После обращения к программе

$Я)$	$Я+3 \longleftrightarrow$	Ω	*корень*	
$Я+1)$	a		b	x
$Я+2)$	f		Δx	ε

в ячейке Ω будет стоять $0 \longleftrightarrow 0$, $Я+3$. Если обратиться в начале программы *Корень* к *ИНФО*, то ячейка $Я+2$ будет перенесена в *Инф4*, $Я+1$ в *Инф3*; $Я$ — в *Инф2*, $Я-1$ — *в Инф1*. Фактически мы используем только ячейки *Инф4* и *Инф3*, в которых лежит вся информация. Перенос же четырех ячеек сделан в расчете на то, что могут встретиться программы, информация к которым занимает четыре ячейки. Итак, после обращения к *ИНФО*

Инф3:	a	b	x
Инф4:	f	Δx	ε

и тем самым первая часть — перенос информации в рабочие ячейки — осуществлена командой обращения к *ИНФО*. Переходим ко второй части.

Групповое формирование команд (*ГФК*). Программа с принудительным концом. Обращение:

Я)		*Я*+1 ↔ Ω		*ГФК*
Я+1)	0	Ψ	P^0	P
Я+2)	код	A_1	A_2	A_3
Я+3)	. . . . .			
Я+4)	. . . . .			
. . . . .				
Я+(2*n*−1))	. . . . .			
Я+2*n*)	. . . . .			

Информацией к программе служат n пар ячеек. Каждая пара предназначена для формирования одной команды. После работы программы управление передается ячейке *Я* + (2*n* + 1). Окончание информации опознается программой по ненулевому коду. Все информационные пары устроены одинаково, поэтому достаточно рассмотреть первую.

Я+1)	0	$\alpha_1\alpha_2\alpha_3\beta_1\gamma_1\beta_2\gamma_2\beta_3\gamma_3$	P^0	P
Я+2)	код	A_1	A_2	A_3.

Программа формирует команду и записывает ее в ячейку P. Код этой команды берется из кода *Я* + 2. Адреса формируемой команды P получаются из адресов ячейки P^0 и адресов ячеек A_1, A_2, A_3 или самих чисел $\overline{A}_1$, $\overline{A}_2$, $\overline{A}_3$. Признаки α_1, α_2, α_3 принимают значения 0 или 1 (один разряд). Если $\alpha_1 = 0$, то для формирования первого адреса команды берется первый адрес P^0; если $\alpha_1 = 1$, то 0. γ_1, γ_2, γ_3 принимают значения от 0 до 3 (два разряда); γ_1 указывает, какой адрес A_1 нужно взять для формирования первого адреса команды. Если $\gamma_1 = 0$, то берется само число $\overline{A}_1$ β_1, β_2, β_3 принимают значения 0 или 1 (один разряд). Если $\beta_1 = 0$, то первый адрес получается прибавлением к первому адресу P^0 (или нулю) нужного адреса ячейки A_1 (или самого числа $\overline{A}_1$). Если $\beta_1 = 1$, то 1-й адрес P получается вычитанием из 1-го адреса P^0 (или нуля). Второй и третий адреса команды P формируются аналогично. Обозначим через $[A]_k$ — k-й адрес ячейки $\overline{A}$, если $k = 1, 2, 3$, и $\overline{A}$, если $k = 0$. Тогда закон образования адресов ячейки P можно записать так:

$$[P]_i = (1-\alpha_i)[P^0]_i + (-1)^{\beta_i}[A_i]_{\gamma_i} \qquad (i = 1, 2, 3).$$

Чаще всего ячейку P^0 можно брать нулем. Роль A_1, A_2, A_3 в стандартных программах играют обычно ячейки *Инф4* — *Инф1*.

В качестве примера покажем, какую информацию нужно написать для *ГФК*, чтобы сформировать ячейку *Е* в рассмотренной программе *Корень*. Команда *Е* должна выглядеть так: $z \to x$; *f*. Информация к программе лежит в ячейках *Инф4* и *Инф3*, как указано выше. В команде, которую нужно сформировать, первый адрес z известен, а второй и третий могут быть взяты из ячеек *Инф3* и *Инф4*. Поэтому $P^0 = 0$. Соответственно α_1, α_2, α_3 могут быть взяты

произвольно (так как все адреса P^0 все равно нули). Мы положим их равными нулю. Поскольку все адреса z, f, x нужно написать или взять из *Инф3* и *Инф4* и прибавить к адресам P^0 (нулям), то β_1, β_2, β_3 равны нулю.

Адрес z известен, поэтому в качестве A_1 может быть взято z, а γ_1 положено равным нулю. Адрес f записан в первом адресе *Инф4*, следовательно, $A_2 =$ *Инф4*, а $\gamma_2 = 1$. Наконец, адрес x находится в третьем адресе *Инф3*, следовательно, $A_3 =$ *Инф3*, $\gamma_3 = 3$. Код второй информационной ячейки соответствует коду формируемой команды. Итак,

0, 000 000 011 001 0 E

56, $z \longrightarrow$ *Инф3*, *Инф4*.

Обычно первый адрес первой ячейки, в котором записано условное число Ψ, записывают не двоичными, а сразу восьмеричными цифрами. Код второй ячейки не пишется, а указывается так, как если бы это была команда. Переписанные ячейки информации к *ГФК* выглядят так:

0 $\overline{0031}$, 0, E

$z \longrightarrow$ *Инф3* *Инф4*

Перепишем теперь, используя программы *ИНФО* и *ГФК* всю формирующую часть программы *Корень*:

1) Ω $= конец$

2) $\longrightarrow$ *конец ИНФО*; *ИНФО*

3) $\downarrow$ $\longrightarrow$ Ω *ГФК*

4) $\downarrow$ 0 $\overline{0020}$, 0, *A*

5) 0 $\vee$ *Инф4* $= \delta$

6) 0 $\overline{0010}$, 0, *B*

7) 0 $\vee$ *Инф3* $= z$

8) 0 $\overline{0200}$, 0, *C*

9) *Инф3* $- z = 0$

10) 0 $\overline{0300}$, 0, *D*

11) $|$*Инф4*$| - |\delta| = 0$

12) 0 $\overline{0031}$, 0, *E*

13) $z \longrightarrow$ *Инф3*; *Инф4*.

Таким образом, длина формирующей части программы сильно уменьшилась: она сократилась с 25 ячеек до 12 (мы нигде не считаем все равно необходимую команду $\Omega =$ *конец*). Вообще при помощи *ГФК* на формирование одной команды обычно тратится всего две ячейки (информация для *ГФК*). Лишь в редких случаях требуется константа P^0, или команду приходится формировать в два этапа.

Относительно использования программы *ГФК* нужно сказать следующее. Программа *ГФК* очень удобна, так как при пользо-

вании ею формирование команд ведется по стандартным правилам. Однако на формирование каждой команды программа *ГФК* тратит очень много действий (а значит, и времени). В таких программах как корень функции, собственное время работы которых очень велико, такая добавка несущественна. Однако писать формирования при помощи *ГФК* для очень быстрых программ, таких как элементарные функции (если оформлять их как программы с информацией), невыгодно, ибо это может существенно замедлить скорость их работы.

Идея программы *ГФК* была высказана А. С. Кронродом в 1960 г. В том же году им была создана первая программа такого рода. В 1962 г. группой инженеров под руководством Н. И. Бессонова была разработана и введена в действие команда *ФК*, делающая то же, что и программа *ГФК* для одной пары ячеек информации. Команда записывается в две ячейки:

$$\begin{array}{llllll} Я) & ФК & \alpha_1\alpha_2\alpha_3\beta_1\gamma_1\beta_2\gamma_2\beta_3\gamma_3 & P^0 & P \\ Я+1) & код & A & B & C. \end{array}$$

Команда формирует ячейку P, как и программа *ГФК*. Если $P \neq 0$, то управление переходит ячейке $Я + 2$. Если $P = 0$, то, прежде чем передать управление ячейке $Я + 2$, машина выполнит сформированную команду (не записывая ее, так как в ячейку 0 ничего нельзя записать). Замена стандартной программы командой, идущей примерно в 40 раз быстрее, резко увеличивает возможности применения формирования команд. Теперь формирование можно ставить в цикле, формируя и сразу выполняя нужную команду. Кроме того, команду *ФК* можно использовать даже в тех программах, где на обработку информации можно потратить лишь очень небольшое время. Удобства, предоставляемые ею, столь велики, что аналогичную команду, по-видимому, следовало бы иметь на всех машинах.

Несмотря на то, что формирование команд стандартной программы, которые зависят от задаваемой информации, выполняется по несложным, почти формальным правилам и при наличии обслуживающих программ *ИНФО* и *ГФК* занимают немного места, написание стандартных программ является делом трудным. Чтобы уметь писать стандартную программу, нужно ясно представлять ячейку памяти машины и уметь преобразовывать ее логическими действиями, без чего, вообще говоря, можно и обойтись при написании не очень трудных счетных программ. Но дело не только в этом. Дело в том, что стандартная программа, входящая в состав библиотеки, должна быть возможно более короткой и быстрой.

Если программа, составляемая программистом для решения какой-либо задачи, свободно размещается в памяти, то не имеет значения, будет ли она несколько длиннее или короче. В любой программе можно выделить несколько блоков, обращение к которым ведется наиболее часто и на которые падает львиная доля общего времени работы программы. При написании программы нужно заботиться о времени работы только таких блоков. Для

остальных блоков вовсе несущественен выигрыш или проигрыш во времени даже и в несколько раз. Это правило обычно формулируется так: *«Экономь время только во внутренних циклах»*.

При написании библиотечной программы составитель должен быть готов как к тому, что эта программа будет использована в программе с загруженной памятью, так и к тому, что она окажется во внутреннем цикле. Поэтому выбор алгоритма и написание наиболее употребительных библиотечных программ являются одной из самых квалифицированных работ и проводятся с большой тщательностью.

Часто бывает выгодно какой-либо из блоков программы, составляемой для решения конкретной задачи, оформлять как стандартную программу. Такая программа не входит в состав библиотеки, но может быть использована для других задач. Вообще, если это не составляет большого труда, выгодно оформлять как стандартные все программы, которые впоследствии могут быть использованы. Это создает фонд стандартных программ и позволяет избегнуть нередко встречающейся ситуации, когда многие программисты (или даже один программист в разное время) пишут совершенно одинаковые блоки (часто трудные и длинные), нужные им для решения различных задач.

§ 19. Способы размещения библиотеки в памяти

Самый простой способ устройства библиотеки — расположение программ на постоянных адресах. За каждой стандартной программой закрепляется определенное, никаким другим программам не принадлежащее место в оперативной памяти. Перед началом работы программист вводит нужные части библиотеки в закрепленные за ними места, а затем свою программу. При всей естественности такого способа устройства библиотеки он имеет много недостатков.

Во-первых, нужные программисту части библиотеки, вообще говоря, могут и не находиться в памяти подряд. Поэтому часть памяти, оставшаяся для основной программы, разбита. Это вызывает большие неудобства в тех случаях, когда программисту нужны большие сплошные массивы рабочих ячеек, как, например, при решении систем линейных уравнений с большим числом неизвестных.

Во-вторых, при такой системе невозможно иметь библиотеку стандартных программ, занимающую место большее, чем объем оперативной памяти машины, хотя это не противоречит идее библиотеки, даже и не пользующейся внешними запоминающими устройствами, — в принципе важно лишь, чтобы те части библиотеки, которые нужны для данной программы, вместе с самой программой умещались в памяти машины.

В-третьих, каждое изменение в программах библиотеки и связанное с ним изменение начальных адресов этих программ требует соответствующего исправления обращения в основной программе, часто уже отлаженной и работающей, тогда как изменений, не

вызывающихся особой необходимостью, в работающей программе всегда лучше не делать *).

Последнее затруднение можно обойти, введя заголовки. Каждой стандартной программе придается одна ячейка — ее заголовок, в которой стоит команда передачи управления началу программы. Программист, вместо того чтобы обращаться к началу стандартной программы, обращается к ее заголовку. Заголовки всех стандартных программ написаны подряд отдельно от самих программ.

Например, если стандартные программы $\sin\alpha$ и e^{α} начинаются соответственно с адресов 0517 и 0542, а заголовки написаны начиная с 7700, то их содержание таково:

заголовок $\sin\alpha$ 7700: 56 0 0 0517

заголовок e^{α} 7701: 56 0 0 0542.

Программист же вообще не знает адресов 0517 и 0542, а только 7700 и 7701.

При всех изменениях, происходящих в программах библиотеки, адреса заголовков остаются прежними, а меняется только их содержимое, так что программисту даже не сообщается об этих изменениях. Во многих случаях заголовок — потерянная ячейка, но часто удается совместить передачу управления с какой-либо нужной в программе засылкой.

Все же и с заголовками остаются затруднения с изменением библиотеки, если одна из программ библиотеки удлиняется. Тогда нужно либо перекодировать все идущие за ней библиотечные программы, что практически невозможно, либо разбивать измененную программу и, оставив часть ее на старом месте, остальное дописать в конце библиотеки.

Что же касается неудобств с разбиением памяти и невозможностью иметь большую библиотеку, то эти недостатки вообще неустранимы в библиотеке на постоянных адресах.

Для того чтобы устранить эти недостатки, необходимо научиться *автоматически перекодировать,* или, как мы будем иногда говорить, *сдвигать* программы.

Рассмотрим простейший пример.

В ячейках $x_1, x_2, \ldots, x_n$ расположены числа, найти

$$S = \sum_{k=1}^{n} \frac{1}{x_k}.$$

Это обычный цикл с переадресацией. Написание его не вызывает затруднений.

---

*) Исправления такого рода часто являются источником ошибок. Программист может и не помнить всех мест своей программы, где он обращался к измененной программе библиотеки. Эти места он находит обычно, просто просматривая свою программу, и некоторые из них может пропустить.

Программа 1.19

```
      1)                0 = S
      2)     n − 1_N    = R_0
      3)     A_0 → A;
      4)     A +, (0, 1, 0) = A
A     5)     1_N : x_k  = R_1      н. п.
      6)     S + R_1    = S
      7)     R_0 − 1_N  = R_0
     10)  (0)
     11)  Б             Ω
A_0  12)     1_N : x_1  = R_1.
```

Программа, закодированная начиная с адреса 1100, приведена на рис. 35. Посмотрим, какие изменения нужно внести в правую часть, чтобы закодировать (сдвинуть) программу на другое место. При этом будем считать, что адреса ячеек (0, 1, 0); x_1, x_2, ..., x_n; S, n, R_0, R_1 зафиксированы и не зависят от положения программы

Составил Задача Блок Стр. Лист

				1100	A	шифр	№ карты
	$0=S$	1100	75	0	0	4002	
	$n-1_N=R_0$	1	02	4003	0101	4000	
	$A^0\to A$	2	56	1111	1104	1104	
	$A+,(0,\ 1,\ 0)=A$	3	13	1104	0122	1104	
A	$1_N:x_k=R_1$	4			н. п.		
	$S+R_1=S$	5	01	4002	4001	4002	
	$R_0-1_N=R_0$	6	02	4000	0101	4000	
	(0)	7	76	0	0	1103	
	$Б$ Ω	1110	56	0	0	0007	
A^0	$1_N:x_1=R_1$	1	04	0101	5000	4001	

Рис. 35.

в памяти. Та же программа, закодированная начиная с адреса 1200, приведена на рис. 36. Сравнивая эти программы, мы видим, что коды всех команд не изменились, а в мантиссах отличаются адреса

I, II, III команды 3), адреса I и III команды 4), адрес II команды 10), короче те адреса, где упоминается одна из ячеек программы. Эти адреса легко найти, даже не заглядывая в левую часть и шпаргалку, так как они заключены в интервале, где расположена программа. Такие адреса мы будем называть *внутренними*. Для данной программы внутренними адресами являются адреса 1100—1111.

Составил Задача Блок Стр. Лист

				1200	A	шифр	№ карты
	$0=S$	1200	75	0000	0000	4002	
	$n-1_N=R_0$	1	02	4003	0101	4000	
	$A^0\to A$	2	56	1211	1204	1204	
	$A+,(0,\ 1,\ 0)=A$	3	13	1204	0122	1294	
	$1_N:x_k=R_1$	4			*н. п.*		
	$S+R_1=S$	5	01	4002	4001	4002	
	$R_0-1_N=R_0$	6	02	4000	0101	4000	
	(0)	7	76	0000	0000	1203	
	Б 2	1210	56	0000	0000	0007	
A^0	$1_N:x_1=R_1$	1	04	0101	5000	4001	

Рис. 36.

Все эти адреса в сдвинутой программе отличаются от соответствующих адресов первоначальной программы на величину сдвига (1200—1100). Легко написать программу, которая по введенной в ячейки 1100—1111 программе 1.19 положила бы в ячейки 1200—1211 программу 2.19.

Информацией для такой программы являются начальный и конечный адреса программы, которую нужно сдвинуть, и величина сдвига. Программу, осуществляющую такую перекодировку, назовем *сдвигающей*. Такая программа обследует адрес за адресом данной ей для сдвига программы и переносит их на новое место, внося, где нужно, соответствующие изменения. Однако, для того чтобы определить, нужно ли менять данный адрес, недостаточно знать, является ли он внутренним для сдвигаемой программы.

Действительно, программе могут принадлежать числовые константы, адреса которых вовсе не означают номеров ячеек. Но эти адреса могут случайно оказаться внутренними. Разумеется, их не надо менять при сдвиге. Кроме того, есть команды (например,

сдвиг), один из адресов которых никогда не означает номера ячейки. Их тоже не надо менять.

Наконец, при помощи некоторых команд можно создавать константы. Так, например, по команде *Я*: $\overline{0020} \leftrightarrow R_0 \sim$ в ячейке R_0 запишется: 16 0 0 0020, и если мы потом хотим использовать ячейку R_0 как константу (например, счетчик циклов), то при сдвиге менять первый адрес в ячейке *Я* не нужно, даже если он внутренний.

Бывает и обратное: адрес, который не является внутренним для данной программы, нужно при сдвиге изменить. Это те случаи, когда адрес некоторой команды формируется из условной команды добавлением к ней константы, зависящей от расположения программы в памяти. При сдвиге такую константу надо изменять, даже если ее адреса не являются внутренними.

Перечисленные нами случаи, в которых трудно произвести перекодировку по формальным правилам (а мы далеко не исчерпали их), отнюдь не равноценны.

Исследуя не только адреса, но и коды команд сдвигаемой программы, можно установить, что некоторые адреса не надо изменять. Так, в команде $\overline{a \Rightarrow}$, $b = c$ (т. е. в команде с кодом «сдвиг мантиссы») заведомо не нужно менять первый адрес, так как он означает величину сдвига и не есть адрес ячейки.

Другие трудности носят более принципиальный характер. Без дополнительной информации, по одному лишь содержимому, нельзя догадаться, лежит ли в данной ячейке команда или числовая константа. Если при помощи команды создается константа, то одни ее адреса могут подвергаться изменениям при сдвиге программы, а другие не могут. Ясно, что искусственные ограничения на написание программы вроде запрета использовать внутренние адреса, не подвергающиеся сдвигу, не являются выходом из положения.

Тем не менее необходимость сдвигающих программ очевидна, и такие программы создаются. Все они основаны на том, что о сдвигаемой программе сообщается, кроме ее местоположения и места, в которое ее следует сдвинуть, дополнительная информация. Можно использовать два способа. Первый заключается в том, что сдвигающая программа обследует и перекодирует программу, которую нужно сдвинуть, используя при этом еще таблицу исключений, где указаны тем или иным способом команды и адреса, которые нужно (или не нужно) перекодировать нестандартным образом. Место, где лежит таблица исключений, сообщается сдвигающей программе.

Второй, более простой и, быть может, более удобный способ состоит в следующем. Каждой команде сдвигаемой программы соответствуют три разряда в информационной ячейке. Эти разряды относятся к трем адресам команды. Если соответствующий адрес перекодируется при сдвиге, то в разряде ставится единица, если нет, то нуль. На 14 команд программы нужно $14 \times 3 = 42$ разряда, т. е. одна информационная ячейка. Информационные ячейки составляются вместе с программой; сдвигающей программе сообщается их место в памяти машины. Теперь сдвигающей программе точно указано, какие адреса каких команд нужно перекодировать при сдвиге. Эту информацию называют *таблицей характеристик*.

Значение сдвигающих программ не ограничивается их использованием в библиотеке. Задача сдвига есть часть более общей задачи составления *компилирующей* программы. Ставится она так. Дана программа, расположенная на известных местах, не обязательно подряд, и известны «куски» памяти машины, в которые программист хотел бы поместить эту программу. Компилирующая программа должна разместить и перекодировать данную ей программу на требуемые места. Задача сложна как с точки зрения чисто математической — как скомпоновать части данной программы, чтобы, по возможности меньше разбивая их, можно было разместить их на требуемых «кусках», так и с программистской — организация сдвига, решение вопроса о том, можно ли разбивать тот или иной блок, перекодировка команд обращения одних частей к другим.

Вернемся, однако, к библиотеке стандартных программ. Мы будем считать, что имеется сдвигающая программа и для каждой из программ библиотеки написана информация, необходимая для ее обработки сдвигающей. В таком случае программы библиотеки могут быть закодированы на совершенно произвольных местах. Для каждой части библиотеки отведем разряд одного из ДЗУ, например ДЗУ1. Так, части Б1 соответствует первый разряд ДЗУ1, Б2 — второй и т. д. (Поэтому, кстати, мы и установили восьмеричную нумерацию частей библиотеки.)

Перед вводом основной программы вводятся библиотека и сдвигающая программа. Мы полагаем, что сдвигающая программа занимает в памяти место свободное от библиотеки. Программист набирает на ДЗУ1 разряды, соответствующие нужным частям библиотеки, и передает управление сдвигающей программе. Сдвигающая определяет по положению ДЗУ1, какие программы библиотеки нужны, и перекодирует их таким образом, чтобы они расположились подряд в конце памяти. Далее программист вводит основную программу.

Такой способ размещения библиотеки имеет ряд преимуществ по сравнению с размещением на постоянных адресах. Библиотечные программы здесь расположены компактно в конце памяти, и если программист кодирует свою программу начиная с первых ячеек, то единственное, о чем он должен подумать, — это о том, начиная с какого места будут расположены библиотечные программы, нужные ему, чтобы не использовать для своей программы места, которые впоследствии будут заняты библиотекой.

При использовании сдвигающей место памяти, в котором будет работать та или иная библиотечная программа, вовсе не зависит от того, где она была закодирована первоначально. Благодаря этому все изменения одной из стандартных программ никак не затрагивает остальные: даже если заменить стандартную программу другой, написанной на другом месте, после сдвига снова образуется компактный массив.

Практически работа с библиотекой производится так. Библиотека вместе со сдвигающей программой вводится в память, а затем записывается на барабан (или ленту). На барабане эти программы будут находиться в течение всей работы машины, пока по причине какой-либо неисправности содержимое барабана не испортится.

Программист перед началом работы набирает на ДЗУ1 номера нужных частей библиотек и вводит в машину одну перфокарту. На этой перфокарте набита программа перемотки содержимого барабана в оперативную память и передачи управления сдвигающей программе. Передав действие введенной программе перемотки, программист получает в оперативной памяти библиотеку и сдвигающую. После этого работа происходит как описано выше. Таким образом, вместо того чтобы вводить каждый раз заново библиотеку и сдвигающую программу, что занимает много (2—3 *мин*) времени, вводится всего одна перфокарта. Перемотка же осуществляется очень быстро. Точно так же следует поступать и в тех случаях, когда используется библиотека на постоянных адресах.

В дальнейшем мы всюду будем предполагать, что имеется библиотека со сдвигающей программой и что она вызывается в память указанным способом.

Если программа должна подвергнуться сдвигу, т. е. работать не на том месте, куда введена, то вообще нет необходимости вводить программу на то место, где она могла бы работать. Так, программа, приведенная на рис. 35, закодированная с адреса 1100, может быть введена на адреса начиная с 1000 (рис. 37). Сдвигающая

Составил	Задача	Блок		Стр.			Лист
				1000	A	шифр	№ карты
		1000	75	0000	0000	4002	
		1	02	4003	0101	4000	
		2	56	1111	1104	1104	
		3	13	1104	0122	1104	
		4			*н. п.*		
		5	01	4002	4001	4002	
		6	02	4000	0101	4000	
		7	76	0000	0000	1103	
		1010	56	0000	0000	0007	
		1	04	0101	5000	4001	

Рис. 37.

программа, которой, разумеется, теперь должны быть указаны не только место, где написана программа (1000—1011), и место, куда ее нужно перенести (с 1200), но и место, где она может работать

в начальном состоянии (1100), перенесет программу с адресов 1000—1011 на адреса 1200—1211, перекодировав ее так, чтобы она верно работала на этих адресах.

Итак, при составлении библиотечных программ нет никакой необходимости заботиться о том, чтобы все программы были закодированы на разные места памяти. Можно, например, как это и сделано в некоторых библиотеках, все библиотечные программы кодировать так, чтобы они работали на одном и том же месте памяти. Вводить их в машину, разумеется, нужно на разные места. После работы программы сдвига они займут свое истинное место и приобретут соответствующую этому месту кодировку. До сдвига их можно хранить во внешней памяти, используя барабан или ленту. Если у машины их несколько, то число заготовленных стандартных программ может быть увеличено.

Существует еще один принцип построения библиотеки стандартных программ, также основанный на использовании сдвигающей программы. В библиотеках, построенных по этому принципу, для работы стандартных программ отводится определенное рабочее поле. Все библиотечные программы записаны на барабане или на ленте (быть может, на нескольких). Во время работы с библиотекой в оперативной памяти постоянно находятся заголовки стандартных программ и так называемая *интерпретирующая программа.* Все библиотечные программы записываются и работают на одном и том же рабочем поле библиотеки. Если какая-либо из программ находится в данный момент в оперативной памяти машины, то в ее заголовке лежит команда передачи управления этой программе. Если же данной программы в оперативной памяти нет, то в заголовке лежит команда передачи управления интерпретирующей программе. Интерпретирующая программа проверяет наличие на рабочем поле места, не занятого другими стандартными программами. В том случае, когда свободного места достаточно, нужная программа вызывается с барабана и сдвигается на это место. При отсутствии места интерпретирующая программа предварительно стирает все рабочее поле библиотеки (для чего достаточно изменить все заголовки) и вызывает требуемую программу в начало рабочего поля.

При таком принципе построения библиотеки пользование ею полностью автоматизировано, что в некотором смысле очень удобно; программисту даже не нужно выписывать те части библиотеки, которые ему потребуются.

При всех преимуществах, которые дает использование библиотекой сдвигающей или интерпретирующей программы, нельзя забывать о тех огромных преимуществах, которые имеет перед ними библиотека, работающая на постоянных адресах. Библиотечные программы, написанные в истинных адресах, лежат перед программистом на бланке в том самом виде, в каком она фактически работает.

При остановке машины в какой-либо из библиотечных программ мы всегда знаем, где именно это произошло, и легко можем установить ошибки, допущенные при обращении к этой стандартной программе.

Наоборот, после сдвига программа, работающая в машине, настолько отличается от записанной на бланке, что разобраться в ней и выяснить причины ошибок несравненно сложнее. Более того, при использовании сдвига или интерпретирующей программы подчас трудно установить, в какой из библиотечных программ произошла ошибка и какой ошибкой нашей программы она вызвана. Часто почти невозможно определить, из какого места своей программы мы попали в библиотечную.

Поэтому мы считаем целесообразным не пользоваться сдвигом библиотечных программ во всех тех случаях, когда это допускается объемом оперативной памяти. Наиболее удобным является такое построение библиотеки, при котором можно воспользоваться любым из рассмотренных принципов. Однако во всех случаях сервисные программы и константы всегда должны быть в оперативной памяти. Тем самым естественно всегда иметь для них постоянные адреса.

Обратим внимание читателя на одно немаловажное обстоятельство. Мы нигде не предполагаем, и это в самом деле не так, что программы библиотеки не используют других библиотечных программ из той же или из других частей библиотеки. Нет никакой нужды избегать такого рода зависимостей. Например, для решения систем линейных уравнений естественно воспользоваться программами фиксированного умножения и переноса по числу, а для программы функции $\operatorname{erf} \alpha = \frac{2}{\sqrt{\pi}} \int_0^{\alpha} e^{-t^2}\, dt$ обращаться к программе функции e^{α}. Столь же естественно, что две программы печати (*Печать 1* и *Печать 2*, см. § 16) представляют собою, собственно, единую программу с двумя входами, так же как программы cos, sin и ch, sh. Таких примеров можно привести множество. Небольшой проигрыш места, возникающий в случае, когда нужна только одна из таких программ, с лихвой окупается удобствами, возникающими благодаря большому числу возможностей различного обращения.

ГЛАВА V

ОТЛАДКА ПРОГРАММЫ

§ 20. Пульт машины. Работа программиста у пульта

Для того чтобы отладить программу, т. е. убедиться в ее правильности, и устранить имеющиеся ошибки, необходимо проверить программу на машине. Приемы этой проверки существенно зависят от возможностей, которые предоставляются пультом машины. Поэтому знакомство с приемами отладки необходимо начать со знакомства с пультом машины.

Как и в случае набора элементарных операций, для пульта мы примем в рассмотрение те возможности, которые имеются у всех больших трехадресных машин или, по крайней мере, у многих из них, причем лишь те, которые нужны для программиста, а не для инженера.

Пульт состоит из двух панелей: горизонтальной и вертикальной. На первой расположены тумблеры, кнопки и переключатели, управляющие работой машины, а на второй — ряды лампочек, позволяющие прочесть содержимое тех или иных ячеек.

На горизонтальной панели находятся (рис. 38):

1. Кнопка *Ввод*. Ее нажатие включает работу устройства ввода, в которое предварительно должна быть заложена программа и все исходные данные, набитые на перфокартах (или перфоленте). Весь этот материал вводится в соответствующие ячейки памяти машины.

2. Кнопка *Пуск*, нажатие которой включает работу машины.

3. Кнопка *Стоп*, которая останавливает работу машины или устройства ввода.

4. *Переключатель режима работы машины.* Он может находиться в двух состояниях: *автоматический режим* и *шаговый режим*. При работе в автоматическом режиме (*на автомате*) машина после нажатия кнопки *Пуск* выполняет программу — команда за командой, автоматически. При работе в шаговом режиме (*на шагах*) после нажатия кнопки *Пуск* машина выполняет

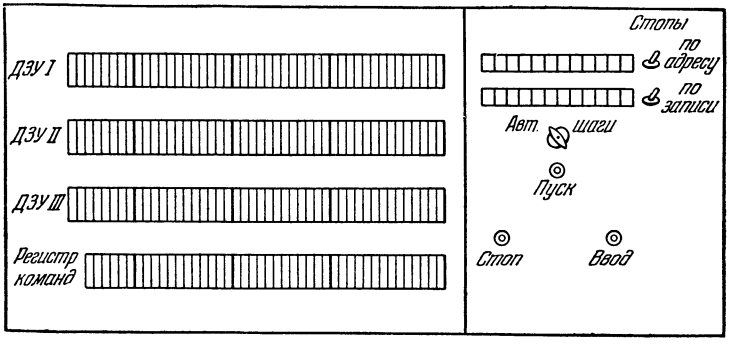


Рис. 38.

очередную команду, после чего останавливается. Для того чтобы выполнить следующую команду, нужно снова нажать кнопку *Пуск*.

5. Тумблеры *отладочных стопов*, позволяющие при работе машины в автоматическом режиме остановить ее в нужном месте. У каждого тумблера имеется 12-разрядная клавиатура, на которой можно набрать адрес соответствующей ячейки.

Мы будем предполагать наличие на пульте двух отладочных стопов.

Стоп по адресу. При наборе на клавиатуре у этого тумблера адреса некоторой ячейки и включении этого тумблера машина останавливается перед выполнением команды, лежащей в ячейке с указанным адресом.

Стоп по записи. При наборе на клавиатуре у этого тумблера адреса некоторой ячейки и включении тумблера машина останавливается перед записью какого-либо содержимого в ячейку с указанным адресом.

6. *Регистр команд.* Он представляет собою 42-разрядную*) клавиатуру, на которой можно набрать любую команду; она будет выполнена при нажиме на кнопку *Пуск.* В частности, регистром команд приходится пользоваться в начале работы для передачи управления первой ячейке программы (или любому другому месту).

7. *Дополнительные запоминающие устройства.* Их может быть несколько. Каждый из них представляет собою 44-разрядную клавиатуру, соответствующую полной ячейке памяти машины, на которой может быть набрано любое содержимое. При этом нажатая клавиша означает единицу в соответствующем разряде, не нажатая — нуль. Машина может брать информацию из этих дополнительных запоминающих устройств либо с помощью специальной команды, либо как из обычной ячейки памяти.

Для простоты и определенности мы будем предполагать, что на пульте имеется три дополнительных запоминающих устройства, которые мы будем называть ДЗУ1, ДЗУ2, ДЗУ3, и что машина может брать из них информацию, как из обычных ячеек памяти с номерами, соответственно, 7775—7777. Запись в эти ячейки из машины, разумеется, невозможна.

На вертикальной панели пульта находятся несколько рядов неоновых лампочек, составляющих следующие регистры (рис. 39).

1. *Командный регистр адреса* — 12-разрядный регистр, указывающий адрес ячейки, в которой находится управление, т. е. команды, которая будет выполнена, если нажать *Пуск.*

2. *Регистр команд* — 42-разрядный регистр, в котором находится содержимое ячейки, где находится управление, т. е., иначе говоря, команда, которая должна выполняться при нажиме кнопки *Пуск.* При наборе команды на клавиатуре регистра команд горизонтальной панели пульта здесь будет гореть набранная команда.

---

*) Как было указано в § 5, 44- и 43-й разряды при записи команды в ячейке не используются, поэтому для команды достаточно 42-разрядного регистра.

3. *Регистры содержимого ячеек* — три 44-разрядных регистра Р1, Р2, Р3; в них можно прочесть содержимое ячеек, адреса которых указаны, соответственно, в первом, втором и третьем адресах предыдущей, только что выполненной команды.

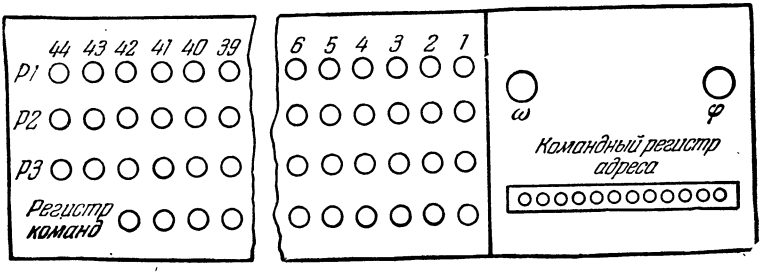


Рис. 39.

Во всех регистрах число изображается в том виде, в каком оно лежит в ячейке, т. е. в двоично-восьмеричном или двоично-десятичном; горящая лампочка означает единицу, не горящая — нуль.

Кроме перечисленных регистров на вертикальной панели пульта находятся две особые лампочки: управляющий сигнал ω и сигнал аварийного стопа при переполнении разрядной сетки, который мы будем называть сигналом φ. Будем считать, что блокировка сигнала φ отсутствует и во всех случаях, когда в результате плавающих действий происходит переполнение разрядной сетки машины, машина останавливается. При этом на пульте загорается лампочка сигнала φ. Это же происходит и при делении на нуль.

Перед выходом на машину для отладки программы необходимо тщательно продумать порядок действий за пультом, возможности, которые могут встретиться, и реакцию на каждую из них. План работы за пультом должен быть записан в специальную инструкцию, которая должна составляться программистом для себя на каждую отладку. Особенно важно иметь подробную и тщательно продуманную инструкцию для работы у пульта в случае *безавторной отладки*, когда програм-

мист не выходит за пульт сам, а работа по инструкции поручается оператору.

Организация отладки существенно зависит от организации всей работы на машине, весьма различной в разных местах. Чаще всего для отладки выделяются небольшие промежутки времени, порядка 5—10 минут. В других случаях, наоборот, программисту предоставляется для работы у пульта час-полтора. Естественно, что отладку в том или другом случае следует вести по-разному.

И тот, и другой способы распределения времени имеют свои преимущества и свои недостатки. Небольшие промежутки времени требуют более тщательной подготовки к отладке и безусловно экономят машинное время. С другой стороны, при выходе на машину на 5—10 минут трудно за такое время обнаружить больше одной ошибки, тогда как в программе их бывает обычно несколько. Поэтому отладка большими порциями, хотя и требует больше машинного времени, очень сильно экономит календарное: для того чтобы устранить, например, шесть ошибок, может потребоваться около часу машинного времени, но шесть выходов на машину, т. е. наверняка несколько дней. При работе большими порциями на устранение этих ошибок может понадобиться и полтора-два часа, но всю отладку можно провести за один день. При этом надо иметь в виду, что отладка существенно облегчается специальными сервисными программами, упомянутыми в § 16. Об их использовании будет идти речь в § 23.

По-видимому, наиболее целесообразной организацией работы на машине является совмещение этих двух методов: один-два часа в день должны быть разбиты на маленькие промежутки, а остальное время — на большие, размер которых определяется в зависимости от трудности и срочности задачи. Следует знать, что для большинства задач машинное время, затраченное на отладку, как правило, заметно превосходит время, необходимое для счета. Это не относится лишь к тем программам, по которым счет приходится вести очень часто, т. е. где считается большое число одинаковых вариантов.

При проверке программы (или отдельных ее блоков) на машине в наличии ошибок можно убедиться по одному из следующих признаков:

1) машина не доходит до нужного места (например, до конца программы или блока), останавливаясь в результате переполнения разрядной сетки или невозможности выполнения указанных в программе действий;

2) машина не доходит до нужного места, продолжая все время двигаться по некоторому замкнутому циклу, из которого нет выхода, — з а ц и к л и в а е т с я;

3) машина доходит до нужного места, но выдает неверный результат.

Тому, как по этим признакам определить место и характер сделанных ошибок и затем устранить их, будут посвящены следующие параграфы. Здесь мы ограничимся только одной рекомендацией, выполнение которой даст возможность использовать еще один признак неверной работы программы.

Перед вводом программы в память машины и началом работы полезно производить не очистку всей оперативной памяти нулями, а «роспись *Щ*». Буквой *Щ* мы обозначаем, как уже было замечено в § 14, наибольшее положительное число, которое может поместиться в машине. В нашем случае оно имеет вид

$$Щ = 177\ \ F\ F\ F = 177\ 7777\ 7777\ 7777.$$

Программа *Роспись Щ* расписывает всю память этим числом, т. е. заносит его во все ячейки памяти, кроме, разумеется, тех нескольких ячеек, в которых находится сама эта программа. После того как произведена роспись *Щ*, вводится нужная программа. Таким образом, в ячейках памяти, содержащих программу задачи и все нужные константы, будет находиться требуемое содержимое, а во всех остальных ячейках — число *Щ*.

Если в результате ошибки в программе или адресного сбоя машины в командный регистр адреса будет занесен адрес, не предусмотренный программой, т. е. если машина будет брать команду из той ячейки, где никакая команда не предусмотрена, то она остановится, так как число *Щ* воспринимается как команда с кодом 77, т. е. *стоп*. Если по тем же причинам машина возьмет

содержимое непредусмотренной ячейки в качестве числа, с которым нужно совершать плавающие действия, то в большинстве случаев машина остановится в результате переполнения разрядной сетки, так как *Щ* есть наибольшее возможное число и плавающие действия с ним чаще всего (к сожалению, не всегда) невозможны. При работе на шаговом режиме, даже при возможности действий с числом *Щ*, оно будет сразу замечено.

Итак, в результате предварительной росписи *Щ* мы получаем еще один признак неправильной работы машины — выход на *Щ*. Под этим мы будем подразумевать и попытку выполнить *Щ* как команду, и попытку воспринимать его как аргумент при арифметических действиях. Наша практика работы показывает, что не менее 50% ошибок обнаруживаются именно благодаря выходу машины на *Щ*.

§ 21. Проверка работы блоков. Организация ручного просчета

При блочном программировании отладку естественно начинать с проверки правильности работы отдельных блоков. Как уже говорилось в главе III, работу каждого блока можно проверять отдельно. Только в тех случаях, когда программа невелика и связи между блоками совсем просты, можно начинать процесс отладки всей задачи в целом. Но и тогда приходится проверять работу каждого отдельного блока.

Неправильная работа программы при условии нормально работающей машины может быть вызвана ошибками программирования, кодировки и перфорации. Если с возможностью ошибок программирования необходимо считаться всегда и полностью исключить их не удается, то ошибки кодировки и перфорации могут быть практически исключены правильной организацией работы.

Для этой цели необходимо прежде всего создание специальных постоянных групп для кодировки и контроля и для перфорации. Программа или ее отдельные блоки пишутся программистом в содержательных обо-

значениях и затем передаются в группу кодировки и контроля вместе с составленной программистом шпаргалкой задачи. В отдельных случаях и составление шпаргалки может быть поручено кодировщице; важно лишь, чтобы шпаргалка существовала в одном экземпляре и чтобы изменения в нее вносились только ее составителем. Кодировка программы должна быть проверена другим человеком.

Работу группы перфорации целесообразно организовать следующим образом:

1) программа набивается на картах перфораторщицей;

2) набитая колода дублируется на репродукторе;

3) полученная из репродуктора колода пропечатывается на специальном устройстве;

4) пропечатка колоды сверяется с программой другой перфораторщицей.

Бланки программы должны быть подписаны обеими кодировщицами и обеими перфораторщицами. Ответственность за обнаруженные ошибки кодировки несет кодировщица, проверявшая ее; за ошибки перфорации отвечает перфораторщица, проверявшая карты. При такой организации кодировки и перфорации возможность ошибок устраняется почти полностью. Остается находить и устранять ошибки программирования.

Перед вводом в память отлаживаемой программы нужно ввести требуемые для ее работы части библиотеки. Мы будем предполагать, что нужные части библиотеки вводятся с помощью программы *Вызов* и в случае необходимости сдвигаются. Программа *Вызов* расписывает *Щ* во все ячейки, не занятые введенными частями библиотеки. Для отладки отдельного блока полезно написать специальную отладочную программу. Она должна содержать засылку данных контрольного варианта в аргументы блока, обращение к отлаживаемому блоку и пропечатку его выходных ячеек. Полученные результаты должны быть сверены с результатами контрольного ручного просчета.

Вообще перед отладкой любого блока необходимо провести контрольный ручной просчет. Без такого просчета и сверки программист не может быть уверенным в том, что программа работает правильно.

Организация ручного просчета является самой ответственной и самой квалифицированной частью отладки программы. Так как мы не можем стандартизовать организацию ручного просчета хотя бы в той степени, в какой это было сделано в предыдущих главах для организации программы, то ограничимся некоторыми практическими рекомендациями.

Как правило, ручной просчет для блока должен быть проведен так, чтобы, начиная от исходных данных контрольного варианта, служащих аргументами блока, получить все значения, которые являются его выходными результатами. В качестве контрольного можно, например, взять один из фактических вариантов задачи. При этом входные данные контрольного варианта не должны быть нулями или выделенными для данной задачи числами. Ручной просчет проводится на клавишных вычислительных машинах в две руки и с возможно бо́льшим числом знаков. Расписка для ручного просчета должна составляться по формулам задания независимо от программы.

Если при сверке выходные результаты блока совпадают с результатами контрольного просчета, то блок можно считать отлаженным. При несовпадении результатов нужно сверять результаты промежуточных вычислений. Чтобы облегчить такую сверку, полезно при написании программы отводить для результата каждого промежуточного действия отдельную рабочую ячейку. После работы блока можно пропечатать все рабочие ячейки, получив, таким образом, все результаты промежуточных вычислений. Количество рабочих ячеек при этом резко возрастает, и здесь необходимо считаться с возможностями, предоставляемыми памятью машины. Но при этом резко возрастает также и скорость отладок; поэтому если память машины не позволяет иметь различные рабочие ячейки для всех промежуточных результатов, желательно иметь их различными хотя бы внутри блока, используя одни и те же группы рабочих ячеек в различных блоках, не обращающихся друг к другу.

Сверяя результаты промежуточных вычислений в ручном просчете и на машине, легко обнаружить, в ка-

ком месте сделана ошибка. При этом нужно считаться также с возможностью ошибки и в ручном просчете.

Если блок содержит цикл, повторяющийся большое число раз, то его нельзя пройти ручным просчетом до конца. В таком случае можно поступить одним из следующих способов.

1) Провести ручной просчет для двух-трех повторений цикла, сверяя затем каждый из результатов с результатами того же числа повторений на машине. Для этого необходимо останавливать машину с помощью тумблера *Стоп по адресу* в команде проверки окончания цикла, либо проходить цикл на шагах.

2) Если проверка окончания цикла происходит по счетчику, то можно с помощью отладочной программы изменить начальное состояние счетчика таким образом, чтобы выход из цикла происходил после двух-трех его повторений. В этом случае можно проводить полный ручной просчет как описано выше.

Отладочные программы для каждого блока можно заменить отладочными печатями, вставляемыми в конце каждого блока. В ней предусматривается печать всех нужных результатов и должна быть предусмотрена возможность блокировки печати разрядом ДЗУ. После отладки блока печать его результатов можно блокировать.

Этот способ предполагает отладку программы в целом. Его преимущество в том, что счет происходит затем по той самой программе, которая отлаживалась (следует только блокировать все ненужные печати). Некоторым недостатком его является необходимость отлаживать блоки в определенном порядке, так как отладка следующего блока невозможна до получения верных результатов предыдущего. Это относится и к ручному просчету.

До сих пор мы предполагали, что машина проходит отлаживаемый блок (или программу) до конца. Рассмотрим случаи, когда это не так.

Как уже было указано, мы можем встретиться с несколькими возможностями. Чаще всего машина не доходит до нужного места программы, останавливаясь вследствие переполнения разрядной сетки. По существу,

этот случай должен рассматриваться аналогично получению неверного результата. При остановке машины по переполнению необходимо запомнить (лучше записать) адрес команды, в результате выполнения которой это произошло. Далее следует пропечатать (с помощью программы *Подслой*, см. § 16, Б17) рабочие ячейки данного блока, что позволит выяснить место и причину ошибки. Очень часто переполнение вызывается попыткой действия с *Щ*. В таких случаях обнаружить ошибку легко даже и без ручного просчета, проследив, какие именно действия приводят к невозможному результату.

Другим типом возможной ошибки является зацикливание машины, которое можно заметить, например, по времени работы машины. При зацикливании внутри блока нужно проходить блок на шагах; при наличии циклов нужно обращать особое внимание на команды проверки окончания цикла и команды изменения счетчиков, если цикл идет по счетчику. При зацикливании в собирающей ее нужно проходить следующим образом. Все команды собирающей, кроме команд обращения к блокам, проходят на шагах. Команды обращения к блоку проходят на автомате, поставив «стоп» по следующей команде.

Так же поступают и в тех случаях, когда один блок обращается к другому.

Таким образом собирающую можно проходить как отдельный блок, не вникая в содержание каждой части программы. В результате находится либо ошибка в собирающей, либо блок, в котором происходит зацикливание.

При выходе на *Щ* для нахождения ошибки достаточно найти команду, передающую сюда управление. Прежде всего нужно обратить внимание на командный регистр адреса и выяснить, находится ли управление в ячейке программы или вне ее. Если выход на *Щ* произошел в ячейке программы, то это значит, что в нее заслано неверное содержимое. Необходимо проверить формирование этой команды (если она формируемая). В том случае, когда эта команда не формируется, следует пройти программу снова до этого места, поставив

Стоп по записи в данную ячейку. Так же следует поступать и в том случае, если выясняется, что содержимое какой-либо ячейки изменяется непредвиденным образом.

Если же выход на *Щ* произошел вне пределов отлаживаемой программы, то легче всего обнаружить ошибку с помощью *Стопа по передаче управления.* Однако такой отладочный стоп на машине бывает в редких случаях и его наличие у рассматриваемой условной машины мы не предполагали. Поэтому ошибка при выходе на *Щ* вне пределов отлаживаемой программы ищется так же, как и при зацикливании.

Итак, мы рассмотрели все возможные признаки неправильной работы программы. Хотя в каждом из этих случаев мы дали какие-либо рекомендации, читатель не должен обольщаться — отладка программы является трудной и высококвалифицированной работой, при выполнении которой возникает множество непредвиденных ситуаций.

Единственный способ преодоления всех трудностей отладки состоит в том, чтобы не допускать ошибок при программировании.

Обнаруженные в процессе отладки ошибки должны быть отмечены и исправлены на бланке. После отладки необходимо перебить соответствующие карты, с тем чтобы к следующему выходу на машину иметь колоду, в которой все замеченные ошибки исправлены. Ни в коем случае не следует допускать поправок, набитых на отдельные карты и вводящихся после основной колоды, так как при этом в некоторые ячейки дважды вводится различное содержимое и окончательный результат зависит от порядка, в котором подложены карты. Это может в дальнейшем служить источником разнообразных неприятностей.

Заметим, что мы все время говорили о контроле правильности программы, считая, что машина работает правильно. Контроль правильности работы машины, если в нем возникает необходимость, осуществляется обычно при помощи повторного счета. На этом вопросе мы останавливаться не будем.

§ 22. Логические программы и логические блоки

До сих пор основное внимание в книге уделялось программированию вычислительных задач, т. е. реализации в виде программы того или иного вычислительного метода. С точки зрения программирования характерной чертой этих задач является использование содержимого ячеек памяти одним из следующих двух способов. Части ячеек в процессе работы программы будет передаваться управление, т. е. их содержимое воспринимается машиной как команда. По существу, командами являются также и восстановители в циклах с переадресацией, хотя этим ячейкам управление не передается. Другие ячейки (точнее, их содержимое) используются как числа, плавающие или фиксированные, введенные в машину вместе с программой или получившиеся в результате вычислений.

В описании библиотеки стандартных программ мы познакомились также и с другим использованием содержимого ячеек памяти. При обращении к библиотечной программе с информацией одна или несколько ячеек, следующих после команды обращения, должны содержать информацию для соответствующей программы. Здесь ячейка содержит уже не число и не команду, а части команд — отдельные адреса, которые будут использованы при формировании команд стандартной программы перед ее работой. Кроме того, такая ячейка может содержать также и некоторые признаки, например число печатей в группе или номер ДЗУ и разряда, блокирующего печать при обращении к программе печати.

Если содержимое такой ячейки можно, с известной натяжкой, рассматривать как команду, то в логических программах мы часто встречаемся с таким случаем, когда содержимое некоторой ячейки памяти представляет собою набор нулей и единиц, несущий ту или иную смысловую нагрузку и играющий роль информации.

Укажем несколько примеров задач, в которых существенную роль играет использование ячейки памяти для записи информации, отличной от команд и чисел.

Игровые программы. Рассмотрим, например, программу для игры в шахматы. Независимо от того, составляется ли универсальная программа, играющая в любой позиции, или пишется программа, приспособленная для выбора хода в определенном узком классе позиций, первым вопросом, который встает перед программистом, является вопрос о записи в машине положения фигур на доске.

Возможен, например, следующий способ записи. Каждой горизонтали шахматной доски отводится одна ячейка памяти. В этой ячейке каждому полю, расположенному на данной горизонтали, отводится четыре разряда, в которых записывается информация о фигуре, находящейся на этом поле. В левом из четырех разрядов ставится единица в том случае, если на этом поле находится фигура черных, и нуль, если поле свободно или на нем стоит фигура белых. В остальных трех разрядах записан условный знак стоящей на поле фигуры, например:

000 — поле свободно,
001 — пешка,

010 — конь,
011 — слон,
100 — ладья,
101 — ферзь,
110 — король.

В ячейке, отведенной для одной горизонтали, оказываются занятыми $8 \times 4 = 32$ правых разряда. В остальных разрядах можно постоянно держать нули. Восемь идущих подряд ячеек содержат, таким образом, всю информацию о расположении фигур на доске.

Узнающие и (или) диагностирующие программы. Под этим названием обычно объединяют большое число весьма различных задач. Сформулируем одну из них. Пусть дано некоторое множество предметов, разбитых на несколько классов, задан набор признаков, которыми может обладать или не обладать каждый предмет, и о каждом из предметов известно, какими из данных признаков он обладает и к какому классу относится. Пусть теперь поступил новый предмет с известными признаками и требуется определить, к какому классу он относится.

Информацию о каждом предмете можно записать в машине следующим образом. Предположим, что число признаков меньше числа разрядов в ячейке. Отведем каждому предмету одну ячейку памяти, а каждому признаку — определенный разряд. Тогда в ячейке, соответствующей данному предмету, будут стоять единицы в разрядах, соответствующих признакам, которыми он обладает, и нули в остальных. Если известно, к какому классу этот предмет относится, то удобно в отдельных разрядах той же ячейки записать номер класса.

Переводящие программы. Прежде чем пытаться с помощью машины переводить текст с одного языка на другой или, что значительно сложнее, научить машину отвечать хотя бы на простейшие вопросы «по прочитанному тексту», нужно уметь записать текст в машину.

Можно, например, поступить так. Занумеруем все буквы алфавита; придадим также номера промежутку между словами, знакам препинания и цифрам. По-видимому, общее число требуемых знаков не превосходит 64 и их можно пронумеровать шестизначными двоичными числами от 000000 до 111111. Занимая в ячейке 42 разряда (естественно — подряд и справа), можно записать в ней 7 знаков. Таким образом, в массиве из n ячеек можно записать текст, состоящий из $7n$ знаков.

Каждый из предложенных выше способов записи *(кодировки)* информации далеко не единственный и, быть может, не лучший. Для нас здесь важно то обстоятельство, что мы имеем дело с существенно иным использованием содержимого ячейки памяти: оно воспринимается не как команда и не как число, а как информация другого типа.

В программах указанного вида обычно очень мало арифметических (плавающих) действий. Основное место в таких программах занимают логические операции, проверки выполнения различного рода условий и передачи управления, которых в логических программах бывает гораздо больше, чем в вычислительных.

Необходимо иметь в виду, что и в вычислительных программах могут встречаться блоки логического характера, к которым также относится все, сказанное выше. В качестве одного из примеров вычислительных программ, требующих задания информации описанного вида, можно, например, указать задачу расчета магнитного поля, создаваемого магнитом сложной конфигурации. Границы магнита можно задавать рисунком: группу из n ячеек, отведенных для этой цели, можно рассматривать как прямоугольник с $44 \times n$ клетками. На этом прямоугольнике можно изобразить границу магнита зачерненными клетками и ставить единицы в тех разрядах ячеек, которые этим клеткам соответствуют. Этот способ аналогичен получению изображения на телевизионном экране и называется заданием информации с помощью *растра*.

Писать логические программы трудно. Помимо трудностей «принципиального» характера, связанных с выбором алгоритма *), все время приходится решать «чисто технические» задачи: в каком виде хранить информацию, чтобы она занимала мало места и чтобы было удобно перерабатывать ее и пользоваться ею. Нередко эти «технические задачи» перерастают в проблемы и не дают возможности осуществить, казалось бы, вполне «жизнеспособный» алгоритм.

Несмотря на это, а, быть может, наоборот, вследствие этого логические задачи занимают важное место среди задач, решаемых на вычислительных машинах, и их значение и роль неуклонно растут. По существу, как раз такие программы позволяют, с основанием или без него, пользоваться термином «думающие машины». Именно вокруг вопроса о том, может или не может машина решать сложные логические задачи лучше, чем человек, велась еще недавно столь жаркая, хотя и малосодержательная дискуссия.

При всех своих специфических качествах логические задачи программируются по тем же правилам, что и вычислительные: те же разветвления, такая же организация цикла, те же приемы блочного программирования и та же отладка. В логических программах требуется еще более тщательно следить за разбиением на блоки и еще более внимательно готовить отладку.

Программисту при написании сколько-нибудь сложной логической программы приходится предусматривать большое число различных случаев, которые могут встретиться в зависимости от различных начальных условий, и для каждого из возможных случаев писать свои участки программы. Многие из таких возможностей осуществляются на самом деле чрезвычайно редко или даже вообще не осуществляются.

Проверив программу лишь на одном-двух отладочных вариантах и отладив, таким образом, основные блоки и коммуникации программы, программист рискует оставить непроверенным большое число участков программы и, следовательно, оставить незамеченным и неисправленным большое число ошибок. Найти эти ошибки впо-

*) Достаточно сказать, что ни одна из описанных выше задач не решена еще в том смысле, что ни для одной из них не существует пока программы, решающей эту задачу хотя бы не хуже, чем это делает человек.

следствии будет очень сложно, так как они влияют на результат лишь в редких случаях. Так возникают программы, ошибки в которых «вылавливаются» месяцами или даже годами.

Из сказанного выше вытекают два практических вывода. Прежде всего при отладке логических программ ни в коем случае не следует полагаться на отладку программы в целом, что часто можно делать в вычислительных программах. Блоки логической программы, как и логические блоки вычислительной программы, необходимо отлаживать только отдельно. Кроме того, при отладке каждого блока нужно тщательно следить за тем, чтобы набор исходных данных обеспечивал прохождение всех участков блока. Отладочные варианты должны проверять все сравнения в обе стороны. Это означает, что для каждого ветвления программы должен быть и такой вариант, при котором выполнение программы пойдет по одной ветви, и такой, при котором осуществится другая ветвь. Составление таких отладочных вариантов не требует слишком большого труда, потому что ответ в каждом из них обычно легко определяется без сложных просчетов.

Логические блоки полезно разыграть вручную, выполняя для некоторых исходных данных команду за командой вручную и проверяя при этом соответствие получаемых результатов задуманной цели и правильность передач управления различным частям блока.

Времени на подготовку отладки и на саму отладку логических блоков жалеть не следует. Эта работа окупается. Если человеку, пишущему логическую программу, трудно пообещать спокойный сон, то выполнение приведенных рекомендаций сможет, по крайней мере, уберечь его от кошмаров.

§ 23. Использование сервисных программ. Управление программой с пульта

Тысячам людей, работающих на машинах, грезится такая картина. Программист, написавший и подготовивший программу, отдает ее оператору. Оператор в отсутствие программиста вводит программу в машину и передает действие началу программы. Машина работает необходимое время и печатает нужные результаты. Задача снимается и вводится следующая.

Для того чтобы эта картина стала явью, нужно всего лишь одно: чтобы все программы были написаны без ошибок.

Ошибки появляются оттого, что человек, пишущий программу, должен следить одновременно за очень большим числом деталей выполнения намеченного им плана решения задачи.

Если задание часто может быть записано в несколько строк, а алгоритм решения, понятный для человека, может быть изображен несколькими формулами, то чтобы сделать этот алгоритм понятным машине, нужно в твердо определенном порядке расставить огромное число цифр и значков.

Даже самый слабый вычислитель, встретившись с положением, про которое ему не было дано указаний, может найти правильное решение на основании предыдущего опыта. В крайнем случае, он

обратится с вопросом к составителю алгоритма. Ему можно рассказать алгоритм, опуская детали. Машина не имеет опыта. Вопросы она задает только путем остановки.

Чтобы избежать ошибок, разрабатываются специальные способы написания программ. Операторный метод предусматривает представление программы в виде последовательности формальных операторов. Создаются программирующие программы, цель которых составить программу по операторной схеме и описанию операторов. *Но пока при написании информации для машины (будь то программа или информация для программирующей программы) нужно указывать то, что человек считает абсолютно очевидным, столь естественная, казалось бы, картина работы на машине остается мечтой.* В действительности значительная часть машинного времени затрачивается на отладку программ. Сервисные программы предназначены для облегчения отладки.

Формальное описание сервисных программ, которые мы имеем в виду, дано в § 16. Здесь мы подробнее рассмотрим работу с этими программами.

Программа *Туда.* Отладка любой программы начинается (после того как программа введена в память) с обращения к программе *Туда.* Для этого достаточно передать управление с пульта заголовку этой программы. Программа *Туда* производит контрольное суммирование всей памяти, кроме рабочих ячеек библиотеки, и выходит на *стоп*, зажигая в одном из регистров пульта, например P1, полученную контрольную сумму. При нажиме на пуск эта контрольная сумма переносится в выделенную ячейку библиотеки, обозначаемую $K\Sigma$, и все содержимое оперативной памяти переносится на барабан. При этом происходит печать контрольной суммы $K\Sigma$ программой *Печать программы.* Таким образом, в процессе отладки на барабане хранится программа в ее первоначальном состоянии.

При повторной работе с той же программой полученную раньше контрольную сумму можно вместе с программой ввести в ячейку $K\Sigma$. В этом случае программа *Туда* остановится после контрольного суммирования только тогда, когда полученная контрольная сумма памяти не совпадает с содержимым ячейки $K\Sigma$. В регистрах пульта загораются тогда обе суммы, что является сигналом неправильности ввода. Поскольку перед вводом программы вся оперативная память расписана стандартным содержимым, то при правильном вводе программы контрольная сумма памяти должна быть всякий раз одна и та же (если, конечно, в программу не внесено никаких изменений). Таким образом, программа *Туда* является одновременно контролем правильности ввода. Если же контрольные суммы совпадают, то запись на барабан и печать контрольной суммы происходят без остановки машины и вызова суммы на пульт *).

*) На самом деле программа *Туда* всегда сверяет полученную сумму с содержимым ячейки $K\Sigma$. Поскольку сначала в ячейке $K\Sigma$ находится $Щ$, либо контрольная сумма вызванных частей библиотеки, то в первый раз остановка машины произойдет обязательно.

На некоторых машинах предусмотрена возможность проверки правильности ввода путем контрольного суммирования, осуществляемого самим устройством ввода. При этом, однако, проверяется лишь тот факт, что вводимый материал верно «прочитан» устройством ввода, тогда как суммирование по программе *Туда* проверяет еще и верность его записи в память машины.

Программа *Обратно.* В результате ошибок программы или случайных сбоев машины (при систематических сбоях следует прекратить работу на машине и заняться ее наладкой; но это уже дело инженеров, обслуживающих машину) отдельные команды или даже целые куски программы могут быть испорчены. Это затрудняет поиски ошибок, так как программу нельзя повторить. Но даже после нахождения ошибки программу необходимо вводить заново.

С помощью программы *Обратно* можно переписать с барабана в оперативную память отлаживаемую программу в ее первоначальном состоянии, записанную на барабан программой *Туда* в начале отладки. Если испорчена и сама программа *Обратно*, то она вводится отдельной картой, называемой *Аварийной* или *Картой сбоя.*

Поправка с пульта. При обнаружении легко исправляемых ошибок (типа описки) может оказаться полезным исправить ее сразу же, для того чтобы продолжать отладку. Это особенно важно при длительных отладках. Внесение исправлений осуществляется программой *Поправка с пульта.*

Для обращения к ней в среднем адресе ДЗУ1 набираем адрес исправляемой команды, а в ДЗУ3 — содержимое, которое мы хотим ввести. После этого управление с пульта передается началу программы *Поправка.* Программа переписывает в оперативную память первоначальное состояние программы, как и программа *Обратно,* а затем выходит на *стоп* и зажигает в регистрах пульта адрес исправляемой ячейки и ее новое содержимое. При нажатии пуска поправка вносится в память и производятся контрольное суммирование памяти и запись на барабан, как в программе *Туда.* Одновременно происходят печать поправки программой *Печать программы* и печать контрольной суммы.

Внесение поправки можно было бы осуществить непосредственно, набрав и выполнив команду занесения из ДЗУ в ячейку. Описанная программа имеет следующие преимущества.

1) Перед внесением поправки она восстанавливает программу с барабана и затем записывает ее на барабан в исправленном виде.

2) Вынуждает дважды контролировать адрес исправляемой ячейки и вносимое содержимое, благодаря чему резко уменьшается число самых частых и опасных ошибок — ошибок при поправках.

3) Благодаря тому что каждая внесенная поправка печатается, у программиста остается протокол проделанной на машине работы.

Надслой. Программа *Надслой* позволяет внести поправку числовой константы без предварительного ручного перевода ее в двоичную систему. Программа работает так же, как и *Поправка с пульта.* В ДЗУ3 набирается константа в двоично-десятичной форме. Перед занесением в ячейку она переводится в двоичную систему.

Подслой. Программа *Подслой* является одной из наиболее употребительных сервисных программ. С помощью этой программы

можно выпечатать в десятичном виде содержимое группы ячеек. Для этого в среднем адресе ДЗУ1 набирается номер начальной, а в правом адресе — номер конечной ячейки печатаемой группы, после чего управление передается с пульта началу программы.

Эта программа позволяет проверять промежуточные результаты в процессе работы отлаживаемой программы в любом ее месте.

Разумеется, для той же цели можно воспользоваться и отладочными печатями. Однако они должны быть предусмотрены заранее и быть записаны в программе, тогда как при написании программы трудно предусмотреть, в какой момент нужно узнать содержимое той или иной рабочей ячейки.

Печать программы с пульта. Эта программа работает так же, как и *Подслой*, выпечатывая содержимое ячеек в командной форме как печать программы. Она применяется при наличии в отлаживаемой программе большого числа переменных, или формируемых команд, или логической части. Фиксированные счетчики, информационные ячейки, рабочие ячейки логических программ необходимо печатать в командном виде.

Вообще каждую отладку полезно заканчивать печатанием нужной части памяти с помощью *Подслоя* и *Печати программы с пульта*.

Обращение к описанным нами сервисным программам осуществляется программистом с пульта при остановленной машине. Работа этих программ происходит независимо от работы отлаживаемой программы.

Существуют сервисные программы, построенные на других принципах, — это программы, работающие совместно с отлаживаемой и дающие о ней ту или иную информацию. Такие программы обычно называют *прокруточными*. Например, программа *Луч* по ходу работы отлаживаемой программы печатает адреса команд безусловной передачи управления и команд условной передачи управления, если эта передача фактически происходит.

Мы не будем останавливаться на описании такого рода программ. При их использовании резко уменьшается скорость работы отлаживаемой программы *), а ценность информации, получаемой от них для отладки, сомнительна. Нахождение грубых ошибок в передачах управления, на которое нацелены прокруточные программы, является одной из самых простых частей отладки.

Наличие дополнительных запоминающих устройств на пульте не только облегчает отладку, но и позволяет управлять работой программы в процессе счета. Наблюдая за получающимися по ходу решения задачи результатами, программист может обнаружить, что требуются некоторые дополнительные сведения, печать которых хотя и предусмотрена, но в настоящее время выключена. Как было сказано, с помощью разрядов ДЗУ можно, не останавли-

*) Замедление объясняется тем, что прокруточная программа переносит команды отлаживаемой программы на свое рабочее поле, анализирует их и затем только выполняет. Для коротких программ такое замедление не играет существенной роли. Однако для больших программ может потребоваться чересчур много времени, чтобы добраться до блока, содержащего ошибки.

вая машины, включать или выключать соответствующие печати. Такой возможностью не следует пренебрегать, так как во многих задачах время печати составляет львиную долю общего времени счета.

При решении многих задач, например при решении систем дифференциальных уравнений, табулировании функций и т. п., вычисления ведутся с аргументом, меняющимся с некоторым заранее заданным шагом. От выбора шага зависит скорость работы и точность получающихся результатов. Выбор наиболее удобного шага часто является задачей трудно алгоритмизируемой, хотя и сравнительно легко решаемой человеком, наблюдающим за ходом вычислений. В этих случаях удобно выбрать в качестве ячейки, в которой записан шаг, одно из ДЗУ пульта машины. Это позволяет менять шаг расчетов в процессе работы программы. Точно так же в ДЗУ удобно выносить различные эталоны.

Аналогично обстоит дело в некоторых типах задач, где требуется, например, подобрать параметры таким образом, чтобы некоторая функция принимала нужные значения. Если эта функция сложна, то обычные методы подбора параметров могут оказаться неприменимыми. Вместе с тем при разумном взаимодействии человека и машины такие задачи часто могут быть решены.

Нужно иметь в виду, что управление работой программы с пульта требует активной, напряженной и трудной работы программиста непосредственно за пультом машины, что очень утомительно. Поэтому такого рода деятельностью не следует злоупотреблять.

§ 24. Примеры. Некоторые практические советы

В качестве примера организации отладки и ручного просчета рассмотрим отладку программы нахождения собственных чисел матрицы, приведенной в § 14 (см. пример 2.14). Ручной просчет требуется здесь для четырех блоков: умножение, нормировка, невязка и преобразование матрицы. Поскольку программа может работать при любом N, естественно для контрольного варианта выбрать матрицу третьего порядка.

Прежде всего выберем матрицу третьего порядка, элементы которой являются числами различных порядков и знаков. Пусть, например, это будет матрица

$$\begin{Vmatrix} 4{,}541 & 0{,}238 & -1{,}914 \\ 12{,}565 & 2{,}118 & 0{,}538 \cdot 10^{-1} \\ 1{,}327 & -0{,}544 & 0{,}873 \end{Vmatrix}$$

Рассмотрим отладку одного из блоков программы, например блока *Умножение*. Для ручного просчета здесь достаточно выбрать произвольный вектор, скажем, вектор (2,3; 1,27; —0,46), и найти произведение матрицы на вектор. Мы получим вектор (11,627; 31,564612; 1,95964). Проверку работы блока проще всего провести с помощью отладочной программы. Элементы матрицы мы будем предполагать введенными в ячейки, начиная с $a_{11}^{\text{нач}}$ в двоичной форме. Так как при умножении берется матрица, расположенная на

своем рабочем поле, то в отладочной программе нужно прежде всего переслать эти значения в ячейки, начиная с a_{11}. Здесь можно воспользоваться библиотечной программой пересылки по адресу, так как речь идет о матрице третьего порядка, расположение всех элементов которой хорошо известно.

Элементы v_1, v_2, v_3 вектора могут быть сразу введены в соответствующие ячейки, поскольку отлаживаемый блок не портит этих ячеек. После переноса матрицы на рабочее поле отладочная программа должна обратиться к отлаживаемому блоку *Умножение*, после чего нужно напечатать элементы матрицы, вектора-множителя и вектора, полученного в результате умножения. Отладочную программу можно, например, написать так:

Программа 1.23

$\longrightarrow \Omega$ *Перенос 1*

$a_{11}^{\text{нач}}$ a_{11} $a_{33}^{\text{нач}}$

$\downarrow$ *Умножение*

Печать 1

a_{11} $\bar{3}$ a_{33}

Печать 1

v_1 $\bar{3}$ v_3

Печать 1

u_1 $\bar{3}$ u_3

стоп.

Можно не пользоваться стандартной программой переноса, а использовать для этой цели блок программы *Восстановление*, который решает ту же задачу. В этом случае мы одновременно отлаживаем оба блока. Программу удобно написать так, чтобы сначала проверить блок *Восстановление*, а затем уже *Умножение*. Это можно сделать, например, так:

Программа 2.23

Печать 1

$a_{11}^{\text{нач}}$ $\bar{3}$ $a_{33}^{\text{нач}}$

Восстановление

Печать 1

a_{11} $\bar{3}$ a_{33}

Умножение

Печать 1

v_1 $\bar{3}$ v_3

Печать 1

u_1 $\bar{3}$ u_3

стоп.

Программа 2.23 в закодированном виде помещена на рис. 40.

Составил Задача Блок Стр. Лист 0

	Отладочная			3000	A	T шифра	№ 0.1 № карты
	Печать 1	3000	16	3002	0007	7751	
	$a_{11}^{\text{нач}} \overline{3} a_{33}^{\text{нач}}$	1	00	1600	0003	1610	
	Восстановление	2	16	3003	0007	1020	
	Печать 1	3	16	3005	0007	7751	
	$a_{11} \overline{3} a_{33}$	4	00	1400	0003	1410	
	Умножение	5	16	3006	0007	1200	
	Печать 1	6	16	3010	0007	7751	
	$v_1 \overline{3} v_3$	7	00	1710	0003	1712	
	Печать 1	3010	16	3012	0007	7751	
	$u_1 \overline{3} u_3$	1	00	1700	0003	1702	
	стоп	2	77	0000	0000	0000	

Рис. 40.

Для работы с этой программой за пультом необходима инструкция, которую можно написать, например, так:

Инструкция к отладке блоков

Восстановление и Умножение

1. Ввести библиотеки Б1, Б2 и Б17 *).
2. Ввести программные карты 2—3, 11—12 и отладочную карту 0.
3. Пуск на автомате с программы *Туда*. После *стопа* нажать *пуск* еще раз.
4. Поставить *стоп* по 3005. Пуск на автомате с 3000. Сверить две напечатанные матрицы.
5. При совпадении (т. е. при верной работе блока *Восстановление*) снова нажать *пуск*. Получить и сверить печати двух векторов.

Последний пункт существенно зависит от времени, отведенного на отладку. Если есть возможность проверить работу неверного

*) Они могут быть введены с карт, либо с барабана с помощью программы *Вызов*. Тогда нужно указывать, какие разряды ДЗУ1 набрать.

блока, то следует предусмотреть и записать в инструкции, что именно нужно проверять в первую очередь в случае неверного переноса матрицы или неверного умножения. Аналогично отлаживаются и остальные блоки.

Так как рассматриваемая программа невелика, то нет никакой надобности отлаживать ее блоки отдельно, тем более, что блоки связаны между собой очень просто. При отладке всей программы в целом можно не писать отладочной программы, а для проверки результатов блоков пользоваться *Подслоем*.

Для организации ручного просчета можно выбрать один из двух следующих путей. Если идти по программе, то после переноса матрицы на рабочее место мы переходим к отгонке старшего собственного числа матрицы. Можно сделать одну-две итерации вручную и подобрать границу невязки ε таким образом, чтобы эта точность была достигнута уже после второй итерации. Тогда дальнейший ручной просчет можно вести с вектором и собственным числом, которые получены в результате второй итерации.

Возможен и другой путь. Можно сразу поставить требуемую точность ε отгонки старшего собственного числа. Проверив работу блоков отгонки собственного вектора (*Умножение*, *Нормировка*, *Невязка*) на первых двух итерациях, можно пустить программу до получения собственного числа и собственного вектора с нужной точностью, а затем полученные таким образом данные использовать для ручного просчета при проверке блоков, работающих после (*Преобразование матрицы* и *Пересылка*).

В рассматриваемой задаче применение этого второго пути нецелесообразно, однако в некоторых случаях такого рода использование полученных из машины результатов работы блоков для ручного просчета при отладке следующих может оказаться очень удобным.

После первого умножения и нормировки мы получаем невязку $q = 4{,}56572423$, а после второй $q = 0{,}89813742$. Поэтому можно положить $\varepsilon = 1$ и после двух итераций программа выйдет из блока *Собственное значение*. При ручном просчете для блока *Преобразование матрицы* можно исходить из собственного вектора (1; 5,0420793; —0,20747155). Преобразованная матрица второго порядка имеет вид

$$\begin{Vmatrix} 0{,}91798513 & 0{,}45090055 \\ -0{,}49462177 & 0{,}47589946 \end{Vmatrix}.$$

Приведем примеры инструкций для отладки задачи, исходя из того, что отладка проводится дважды (инструкция № 2 может быть написана только после того, как проведена первая отладка).

Инструкция № 1

1. Ввести Б1, Б2 и Б17.
2. Ввести программу, карты 1—17 *).

*) Мы хотим получить контрольную сумму программы, не зависящую от констант, которые в других случаях могут быть иными.

3. Пуск на автомате с программы *Туда*. После *стопа* нажать пуск снова.
4. Ввести отладочные константы, карты 18—19.
5. Повторить п. 3.
6. Пропечатать *Подслоем* введенную матрицу (1600—1610).
7. *Стоп* по 1006. Пуск на автомате с 1000.
8. Пропечатать матрицу на рабочем поле (1400—1410) *).
9. Вызвать на пульт команду Т (1010). Должно быть: 76 1360 1003 1361.
10. Пройти на шагах от 1006 до 1164. Проверить команду М (1165).
11. *Стоп* по 1166. Пуск на автомате.
12. Пропечатать $v_1 - v_3$ (1710—1712). Должны быть единицы.
13. Пройти на шагах от 1166 до 1200.
14. *Стоп* по 1143. Пуск на автомате.
15. Пропечатать $u_1 - u_3$ (1700—1702). Должен быть вектор 2,865; 14,7368; 1,656.
16. *Стоп* по 1146. Пуск на автомате с 1143.
17. Напечатать вектор $u_1 - u_3$ (1700—1702) q (1740) и ε (1760). Должно быть: (1; 5,1437347; 0,570801047) $q = 4{,}5657242$ и $\varepsilon = 1$.
18. *Стоп* по 1142. Пуск на автомате с 1146.
19. Пропечатать $v_1 - v_3$ (1710—1712). Должен быть перенос того же вектора.
20. *Стоп* по 1150. Пуск на автомате.
21. Напечатать q (1740). Должно быть 0,89813742.
22. Пройти на шагах от 1150 до 1040.

Инструкция № 2

1. Ввести Б17.
2. Ввести программу, карты 1—17 и карту $K\Sigma$ **).
3. Пуск на автомате с программы *Туда*. Должна напечататься контрольная сумма.
4. Ввести отладочные константы, карты 18—19.
5. *Стоп* по 1042. Пуск на автомате с 1000.
6. Напечатать преобразованную матрицу (1500—1503). Должна быть матрица

$$\begin{Vmatrix} 0{,}91798513 & 0{,}45090055 \\ -0{,}49462177 & 0{,}47589946 \end{Vmatrix}.$$

---

*) Пропечатку результатов работы каждого блока нельзя считать обязательной. Часто можно попытаться пройти сразу несколько блоков или даже всю программу и выпечатать результаты последнего в том случае, если их правильность гарантирует правильность работы не только этого последнего блока, но и всех предыдущих.

**) Контрольная сумма набивается по результатам предыдущей отладки. Если были поправки в программе, то нужно перебить карты, но новая контрольная сумма также должна быть известна, если мы пользовались сервисной программой *Поправка с пульта*.

7. *Стоп* по 1006. Пуск на автомате с 1042.

8. Напечатать матрицу на рабочем поле (1400—1403). Должна быть та же матрица.

В первой части работы по инструкции № 1 отлаживается отгонка старшего собственного значения матрицы с помощью итераций и все блоки, относящиеся к этой части. При работе по инструкции № 2 отлаживаются преобразование матрицы (понижение порядка) и пересылка вновь полученной матрицы на рабочее поле. Если при выполнении какого-либо пункта инструкции мы получим результаты, не совпадающие с тем, что должно быть, то следует пройти соответствующий блок на шагах, выпечатывая в случае необходимости рабочие ячейки или формируемые команды программы.

В заключение мы выскажем несколько замечаний, которые следует иметь в виду программисту. Многие из них относятся не столько к процессу отладки, сколько к процессу программирования. Речь идет о том, как следует писать программу, чтобы облегчить отладку. Некоторые из этих замечаний были высказаны в той или иной форме раньше. Мы считаем полезным собрать их воедино и снабдить общим заголовком.

Заповеди программиста.

1. Программа должна быть самовосстанавливающейся. Это означает: программа должна быть написана так, чтобы, прервав ее выполнение в любом месте, можно было бы начать выполнять ее с начала (без нового ввода). В частности, поэтому

восстановление переменных команд должно идти до их первого исполнения;

перевод «10→2» нужно вести в другие ячейки, т. е. не стирая десятичных чисел. Отступление от этой последней рекомендации можно разрешить только при остром дефиците места в памяти.

2. Избегай длинных блоков. Блоки должны быть короткими и легко обозримыми. Длина обычного блока не должна превышать полутора-двух десятков команд. Всякие сколько-нибудь самостоятельные образования следует выделять в качестве отдельных блоков.

3. Избегай двойных циклов. Внутренние циклы лучше всего выделять в качестве отдельных блоков. Если рабочая часть цикла сложна, то и ее нужно оформлять как отдельный блок.

4. Пиши циклы только по счетчику. В случае многократных циклов счетчики внешних циклов являются удобными аргументами для рабочей части внутреннего цикла. Не следует проверять окончание цикла по состоянию переменных команд.

5. Сокращая программу, удлиняешь отладку. Нередко представляется возможность с помощью того или иного трюка сделать программу на одну-две ячейки короче. От этого трудно удержаться, но не следует стремиться к этому.

Программа должна быть не столько короткой, сколько понятной. Красивая и точная формулировка теоремы не всегда самая понятная.

Усложнение программы трюками не только увеличивает возможность ошибки, но и затрудняет отладку. Искусство программирования состоит в том, чтобы уметь сократить любую программу и не пользоваться этим умением, если речь идет не о библиотечных программах.

6. Свободные ячейки прибыли не приносят. Если программа легко умещается в оперативной памяти, то нет никакой нужды экономить рабочие ячейки. Наоборот, выгоднее для каждого промежуточного результата иметь свою рабочую ячейку. Их пропечатка *Подслоем* облегчает отладку арифметических блоков.

7. Не пользуйся рабочими ячейками библиотеки. Для этого есть две причины: во-первых, обращение к любой стандартной программе может испортить их, т. е. изменить их содержимое; во-вторых, содержимое рабочей ячейки библиотеки трудно проверить, так как его нельзя напечатать *Подслоем* (так же как и *Печатью программы*).

8. Блок начинается командой Ω=конец. Без этого нельзя не только обратиться внутри блока к другому блоку или стандартной программе, но и продолжить выполнение программы, если внутри блока мы останавливались и выпечатывали какие-либо ячейки (*Подслоем* или *Печатью программы*).

9. Не разбрасывай рабочие ячейки. Их лучше всего располагать в памяти подряд, единым массивом. Это удобнее и при кодировке программы и при отладке для пропечатки массива.

10. Оставляй зазоры между блоками. При кодировке программы между блоками нужно оставлять несколько свободных ячеек, чтобы иметь возможность внести исправления или изменения.

11. Время решения короткой задачи есть время отладки. Поэтому программы, время работы которых исчисляется минутами (или десятками минут), если только они не будут использоваться много раз, нужно писать наиболее простым способом. Здесь нет смысла думать об экономии времени счета, если это связано с усложнением программы.

12. Экономь календарное время. В комплексе «программирование — отладка — счет» оптимальным является такой вариант, при котором календарное время от начала работы над задачей до ее завершения будет минимальным.

13. Лучше (лишний) *стоп*, чем (лишний) «ляп». Нужно стараться организовывать программу так, чтобы в случае неверного хода решения у нее было как можно больше возможностей выйти на *стоп*. Ни в коем случае не следует, например, вводить блокировки переполнения при вычислении элементарных функций за пределами их области определения. В частности, для получения *стопа* при ошибке совершенно необходима.

14. Роспись Щ. Перед вводом программы все ячейки памяти нужно расписать числом

$$Щ = 177\ F\ F\ F,$$

которое воспринимается как команда *стоп* и является самым большим числом, помещающимся в машине. Машина остановится как в случае передачи управления такой ячейке, так и в большинстве случаев плавающих действий с таким числом. Далее, для той же цели очень полезно

15. Набивать *стоп*, в неперфорируемые ячейки. При ошибках восстановления или формирования машина остановится.

16. Программа есть карты, так что карты всегда должны точно соответствовать написанной на бланках программе. При внесении поправок необходимо немедленно перебивать карты. Никакая ячейка не должна

быть набита на двух картах, так как иначе ее содержимое зависит от порядка ввода карт.

Все карты в колоде должны быть перенумерованы и их не следует перекладывать. Тем не менее на каждой карте должен быть набит адресный код, т. е. колода должна быть такой, чтобы ее можно было бы перетасовать.

17. При *стопе* в библиотечной программе проверь обращение и аргумент. При нормальной работающей машине остановка в библиотечной программе возможна лишь по одной из трех причин: а) неверно написано обращение к ней; б) задан невозможный аргумент или в) стандартная программа испорчена основной. Других ошибок в библиотечных программах не должно быть.

18. Не ставь *стоп* по адресу первой команды блока. Первая команда блока пересылает Ω в конец. Если остановиться до выполнения этой команды, то выход из блока не будет обеспечен. Если нужно остановиться перед блоком, то *стоп* следует ставить по команде обращения к нему в собирающей.

19. Думай перед отладкой. Выход на машину должен быть тщательно подготовлен и продуман. Программист должен точно знать, какую цель он перед собой ставит. Необходимо наметить точный план работы и изложить этот план в форме подробной инструкции.

20. Думай во время отладки. Машинное время надо экономить; однако отсюда не следует, что нужно торопиться, суетиться и дергаться. Это обходится очень дорого. Работать за пультом следует спокойно и вдумчиво.

21. Если программа доходит до конца, это еще не значит, что она работает верно. Этот пункт в комментариях не нуждается.

ЗАКЛЮЧЕНИЕ

Образованному читателю может показаться странным отсутствие в нашей книге главы, посвященной автоматизации программирования, которая, как кажется многим, определяет лицо современного программирования. На самом же деле использование современных языков автоматического программирования имеет такое же значение для самого программирования, как изобретение нового печатного станка для поэзии.

Создатель системы программирования в содержательных обозначениях А. Л. Брудно писал: «Умение программировать становится элементом культуры. Кстати сказать, оно также не является наукой, как правила умножения столбиком или пользование арифмометром и логарифмической линейкой. Оно должно быть таким же простым орудием в руках образованного человека, как эти правила и приборы» *).

Для человека, умеющего программировать, не составит труда овладеть любым языком автоматического программирования и решить вопрос о целесообразности его использования. Наши соображения по этому вопросу мы хотим изложить в настоящем заключении. Мы адресуем его читателям, имеющим представление об автоматизации программирования, например о языке АЛГОЛ-60 **).

Своим происхождением системы автоматизации программирования обязаны следующим обстоятельствам.

---

*) А. Л. Брудно, Введение в программирование, «Наука», 1965. Разрядка автора.

**) С языком АЛГОЛ-60 можно познакомиться по книге: Г. Боттенбрух, Структура АЛГОЛ-60 и его использование, ИЛ, 1963.

В течение длительного времени программы для электронных вычислительных машин писались в действительных адресах и кодах машины, т. е. сразу в кодированном виде без предварительно написанной содержательной части. Такие программы очень трудно писать и совсем невозможно читать. Предлагаем читателю еще раз убедиться в этом, попытавшись прочесть следующую программу:

```
1037  01  1052  1054  0140
  40  56  1055  1045  1042
  41  13  1045  1056  1045
  42  05  0140  0140  0010
  43  02  0010  1052  0011
  44  02  0010  1053  0010
  45  77  0000  0000  0000
  46  01  0140  1054  0140
  47  02  0140  1053  0000
1050  36  0000  1000  1041
  51  77  0000  0000  0000
  52  73  4564  2564  2564
  53  77  7312  6252  6253
  54  76  4061  1156  4571
  55  04  0010  0011  1511
  56  00  0000  0000  0001
```

Легко убедиться, что единственный способ прочесть эту программу, это написать предварительно содержательную часть.

Как средство борьбы с этими трудностями возникло естественное желание создать способ записи алгоритма решения задачи (*язык*), по возможности близкий к привычному языку математики. Хотелось, чтобы этот язык позволял математику оставаться на уровне формул и избавлял от необходимости механически выписывать длинные колонки цифр, не допуская ни малейшей погрешности или описки.

Другое пожелание к создаваемому языку — сделать его независимым от того, на какой машине будет реа-

лизован данный алгоритм *). В то же время он должен быть таким, чтобы для каждой машины можно было написать программу (*транслятор*), переводящую запись алгоритма с этого языка на машинный.

Одной из таких попыток, получившей наибольшее распространение, является язык АЛГОЛ-60. Знакомый с ним читатель без труда заметит некоторое его сходство с программированием в содержательных обозначениях. Это сходство состоит в том, что все рассматриваемые величины обозначаются в программе так же, как и в формулах, и сами формулы имеют привычный для математика вид.

Как способ записи программы язык АЛГОЛ-60 имеет ряд достоинств. В программе, написанной на АЛГОЛе, формулы выглядят даже более привычно, чем в содержательных обозначениях. Программисту не приходится заботиться о распределении памяти и кодировке — это сделает транслятор. Наконец, нет нужды заниматься организацией циклов; циклы, особенно многократные, пишутся на АЛГОЛе значительно проще. К сказанному следует добавить, что язык АЛГОЛ-60 реализует, таким образом, оба пожелания, которые были высказаны двумя абзацами раньше.

Остановимся теперь на недостатках АЛГОЛа. Прежде всего необходимо отметить невозможность использования АЛГОЛа для программирования логических задач.

Запись информации для игровых, узнающих или переводящих программ, описанных в § 22, или программ аналогичного типа или назначения немыслима без учета структуры ячейки памяти. Дело в том, что в терминах АЛГОЛа нельзя записывать или обрабатывать информацию, занимающую группу из нескольких разрядов ячейки. АЛГОЛ допускает рассмотрение величин лишь двух видов: числа (фиксированные или плавающие) и булевские (логические) переменные, принимаю-

*) По всей вероятности, такое пожелание было особенно сильным, например, в США, где существует большое число разнообразных типов машин, выпускаемых различными фирмами, зачастую умышленно очень разных по возможностям и машинному языку программирования.

щие значения 0 либо 1. Для чисел транслятор выделит ячейку, а для логических переменных — разряд в какой-либо ячейке. Таким образом, каждая ячейка памяти может участвовать в программе либо целиком, либо имеет смысл каждый ее разряд в отдельности. Не случайно АЛГОЛ не предусматривает логических операций.

Можно привести и примеры вычислительных программ, требующих информации в виде группы разрядов ячейки. В § 22 уже упоминался пример расчета магнитного поля. Существуют стандартные программы решения краевых задач для дифференциальных уравнений с частными производными, в которых информация о границе области задается также с помощью растра.

Надо сказать, что АЛГОЛ допускает написание процедур непосредственно в коде машины. Хотя таким образом язык АЛГОЛ превращается в универсальный, использование этой возможности означает отказ от АЛГОЛа.

Размещение информации в ячейке зависит от числа разрядов. Например, предложенный в § 22 способ кодирования шахматной позиции хорош для машины, ячейка которой имеет 44 разряда, но едва ли его можно рекомендовать, если ячейка имеет лишь 30 разрядов. Изменение разрядности машины может поэтому потребовать полного пересмотра способа расположения информации. Но дело не только в числе разрядов.

При выборе способа размещения информации необходимо учитывать также способы ее переработки, которые в свою очередь существенно зависят от особенностей различных операций машины, в частности, например, от особенностей фиксированных действий. Совершенно очевидно, что переработка информации на машине М-2, где при фиксированном сложении все содержимое ячейки воспринимается как число с фиксированной запятой, будет происходить иначе, чем на машине БЭСМ, на которой, как и на описанной нами условной машине, фиксированное сложение есть сложение адресных частей, а передача единицы из адресной части в кодовую блокирована. И совсем иначе придется организовывать переработку, а значит, и запись информации в машине «Стрела», где фиксированное сложение

является п о а д р е с н ы м с блокировкой передачи единиц из одного адреса в другой.

Так как на машинах БЭСМ и М-2 фиксированные действия происходят быстрее, чем плавающие, то все величины, принимающие лишь целые значения, в особенности индексы и счетчики, выгодно представлять в фиксированной форме. С другой стороны, на машине «Стрела» фиксированное сложение потребует во много раз больше времени, чем плавающее. Действительно, фиксированное число может быть бóльшим, чем 2^{12} (один адрес). Поэтому придется программным путем обеспечивать проверку переполнения адреса и, если нужно, передачу единицы в следующий адрес.

Посмотрим теперь, что произойдет с программой, написанной на АЛГОЛе, если программист имел в виду машину БЭСМ, а программа должна будет работать на «Стреле». Транслятор, разумеется, переработает ее в программу для «Стрелы», но это обойдется вычислителю довольно дорого: потеря скорости будет уже измеряться п о р я д к а м и.

Другим серьезным недостатком АЛГОЛа является чрезвычайная трудность отладки написанных на нем программ.

Для нахождения и устранения ошибок программы нельзя не владеть языком машины, так как в конечном счете машиной выполняется программа, составленная транслятором на языке машины. Конечно, когда программа идет верно, программисту нет никакого дела до того, какой вид имеет программа в машине. Если же программа «не идет» или работает неверно, то далеко не всегда удается быстро обнаружить ошибку, рассматривая лишь напечатанные машиной числа и не глядя на выполняемые ею команды.

Особенно трудно обнаружить ошибки в перфорации информации для транслятора; во всяком случае исправить найденную ошибку программы при программировании в содержательных обозначениях с помощью сервисной программы *Поправка с пульта* занимает около 20 *сек*, тогда как при программировании на АЛГОЛе требуется, по меньшей мере, повторная работа транслятора, что отнимает не менее полутора минут.

Следующий недостаток АЛГОЛа, на который мы хотим указать, является, по-видимому, наиболее существенным. Для математика, решающего задачу на вычислительной машине, если оставить в стороне вопрос о выборе метода решения, основная трудность заключается вовсе не в программировании как таковом, а в приведении алгоритма к удобному для программирования виду.

Написание команд или тем более кодирование и даже распределение памяти не требуют и десятой доли того напряженного внимания, которое уходит на разбиение алгоритма на блоки, выписывание формул в нужной последовательности и в удобной для программирования форме, продумывание системы контроля правильности написанной программы (отладки), организацию ручного просчета и т. д.

Все эти вопросы не только не снимаются при использовании любой из современных систем автоматического программирования, но, наоборот, встают еще более остро. Система автоматизации программирования в том виде, как она существует сейчас, не экономит сил программиста в преодолении ни одной из этих трудностей. Если сравнивать АЛГОЛ с системой программирования в содержательных обозначениях, то он экономит только кодирование программы, которое при правильной организации работы все равно делается не программистом *).

*) Воспользовавшись этим замечанием, а также сделанным выше замечанием о необходимости повторной работы транслятора при исправлении ошибки, можно легко подсчитать экономический эффект использования АЛГОЛа. Для того чтобы обеспечить всю кодировку программ для большой электронной счетной машины, достаточно иметь трех кодировщиц, месячная зарплата которых соответствует стоимости примерно трех-четырех часов работы машины. Если считать, что при исправлении каждой ошибки требуется повторная работа транслятора, т. е. лишняя минута машинного времени, а на нахождение одной ошибки уходит около 10 минут, то АЛГОЛ удлиняет отладку на 10%. Так как на отладку уходит не менее 50% машинного времени, то такое удлинение отладки означает уже не один десяток часов! Сразу же вспоминается то место из «Одноэтажной Америки» И. Ильфа и Е. Петрова, где описывается завод Форда и рабочий, вручную кисточкой проводящий цветную полоску на борту автомашины.

Можно указать еще ряд недостатков, имеющих самостоятельный характер или вытекающих из сказанного выше. Даже по признанию горячих сторонников АЛГОЛа программа, составленная транслятором, работает, по самым скромным подсчетам, в три-четыре раза медленнее и требует во много раз больше ячеек памяти, чем программа, составленная вручную средним программистом. Еще более важно то, что программа, написанная без учета специфики данной машины и особенностей ее кодов, будет плохой «сама по себе» даже при самом хорошем трансляторе. Можно, конечно, утешать себя тем, что при миллионных скоростях, которые будут достигнуты в ближайшем будущем, такое замедление не будет играть никакой роли. Однако такое самоутешение не более, чем самообман. Это легко понять, если вспомнить, что достигнутые в настоящее время скорости в десятки тысяч операций в секунду казались десять — пятнадцать лет назад столь же фантастическими и позволяющими пренебрегать десятикратным замедлением, как нам сейчас миллионные.

К сожалению, перечисленные недостатки и трудности не являются недосмотром авторов АЛГОЛа, но свойственны каждой системе автоматизации программирования, пригодной для использования на любой машине. Нужно еще добавить, что создание транслятора является столь сложной, дорогой, высококвалифицированной и трудоемкой работой, что для машин, выпускаемых промышленностью в небольшом числе экземпляров, создание транслятора экономически не окупается.

Из сказанного естественно вытекает, что направление и формы, в которых развиваются современные языки автоматизации программирования, нельзя считать прогрессивными. Эти языки вполне приемлемы для решения сравнительно простых вычислительных задач, очень затрудняют работу при решении сложных вычислительных задач и вовсе не приспособлены для решения логических, игровых и подобных задач, в которых и лежит центр тяжести современного программирования.

Стремление создать язык программирования, независимый от конкретной машины, на которой данный ал-

горитм должен быть реализован, как было показано выше, само по себе неразумно. Оно тем более неразумно, если учесть, что число различных типов трехадресных электронных счетных машин у нас совсем не так велико и переход от одного типа к другому совсем не так сложен, чтобы всерьез считаться с необходимостью создания «машинного эсперанто», а не знакомиться с языком каждой машины. Стремиться же к увеличению числа типов универсальных трехадресных цифровых машин нет никаких оснований.

Всему сказанному выше не стоило бы уделять так много места, если бы не два важных обстоятельства.

Во-первых, это стремление сторонников АЛГОЛа представить его в виде универсального языка, на который следует перевести все программирование. Как видно из предыдущего, АЛГОЛ не является универсальным языком и полный перевод программирования на АЛГОЛ, если бы этот перевод и удался, мог бы принести только вред.

Во-вторых, — и это самое важное — стремление представить АЛГОЛ как шаг к полной автоматизации вычислительного процесса и процесса программирования, когда «машина все будет делать сама», т. е. шаг к осуществлению той идиллической картины, которая была нарисована в начале § 23.

Автоматизацию программирования в таком смысле можно было бы только приветствовать, но АЛГОЛ не представляет собою никакого шага в этом направлении. Чтобы сделать такой шаг, нужны принципиальные решения ряда сложных проблем. Быть может, будущее принадлежит автоматическому программированию в указанном смысле. Но даже при современных темпах развития науки трудно ожидать, что это будущее совсем близко.

Рафаил Самойлович Гутер,
Владимир Львович Арлазаров,
Анатолий Васильевич Усков

ПРАКТИКА ПРОГРАММИРОВАНИЯ

Справочник

М., 1965 г., 212 стр. с илл.

Редактор *П. Т. Резниковский*
Техн. редактор *С. Я. Шкляр*
Корректор *А. С. Бакулова*

Сдано в набор 24/IX 1964 г. Подписано к печати 19/I 1965 г. Бумага $84\times108^{1}/_{32}$. Физ. печ. л. 6,63. Условн. печ. л. 10,86. Уч.-изд. л. 10,41. Тираж 30 000 экз. Т-01628. Цена книги 67 коп. Зак. № 739.

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

Ленинградская типография № 2
имени Евгении Соколовой
Главполиграфпрома
Государственного комитета
Совета Министров СССР по печати.
Измайловский проспект, 29.

Цена 67к.

Справочник содержит подробный разбор практических приемов программирования для трехадресных электронных счетных машин. Большое внимание уделяется методам организации и отладки программ, а также составлению и использованию библиотеки стандартных программ. Материал изложен таким образом, что он может быть использован программистами, работающими на любой из современных трехадресных электронных счетных машин.

В основу книги положен простой и удобный метод программирования, предложенный А. Л. Брудно.

Предварительных сведений по программированию у читателя справочника может не быть. Вводная первая глава кратко знакомит с требуемыми первоначальными понятиями. Последующие основные главы будут интересны и программистам, имеющим опыт работы на электронных счетных машинах. В книге рассмотрено большое число примеров, иллюстрирующих рекомендуемые приемы программирования.