

АКАДЕМИЯ НАУК СССР
ИНСТИТУТ ТОЧНОЙ МЕХАНИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Л. А. ЛЮСТЕРНИК, А. А. АБРАМОВ,
В. И. ШЕСТАКОВ, М. Р. ШУРА-БУРА

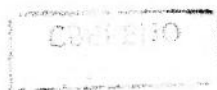
РЕШЕНИЕ МАТЕМАТИЧЕСКИХ ЗАДАЧ НА АВТОМАТИЧЕСКИХ ЦИФРОВЫХ МАШИНАХ

ПРОГРАММИРОВАНИЕ
ДЛЯ БЫСТРОДЕЙСТВУЮЩИХ
ЭЛЕКТРОННЫХ
СЧЕТНЫХ МАШИН

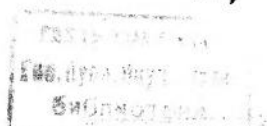


ИЗДАТЕЛЬСТВО АКАДЕМИИ НАУК СССР
1952

Ответственный редактор
член-корр. АН СССР Л. А. Люстерник



10
1011 67



ПРЕДИСЛОВИЕ

Настоящая книга посвящена вопросам вычислений на автоматических машинах, выполнения операций и программирования.

Первая глава носит вводный характер.

Вторая глава посвящена операциям над числами в цифровых машинах (в основном, для случая двоичной системы). Рассматриваются системы задания чисел с фиксированной и плавающей запятой с различными формами задания отрицательных чисел (прямой, дополнительный и обратный коды). Рассматриваются операции, встречающиеся в машинах, начиная с простейших. Описываются разные варианты выполнения арифметических действий при разных формах задания и хранения чисел. Особенно подробно рассмотрены вопросы умножения и деления при разных кодах. Дается также анализ точности при этих действиях. Читатель, интересующийся специально программированием, но не реализацией арифметических действий, может эту главу опустить.

Третья глава посвящена стандартным задачам — преобразованиям систем счисления, действиям с удвоенным числом знаков, контрольным вычислениям, вычислениям значений функций разными алгоритмами и т. д.

Четвертая глава посвящена вопросам программирования. Рассматриваются разные системы программирования, отличающиеся числом адресов команды, порядком выполнения команд, набором элементарных операций и т. д. После выяснения связи между разными системами программирования программы для определенности даются в одной системе трехадресного кода.

В пятой главе приводятся примеры программ для решения математических задач — обыкновенных дифференциальных уравнений, систем линейных алгебраических уравнений и т. д.

§§ 21, 28, 29 и первые три пункта § 27 написаны И. М. Степиным, § 32 написан М. Л. Бродским. Важная помощь была оказана О. А. Червоненкисом, проверившим почти все приведенные программы и примеры.

Настоящая книга написана в результате работы Отдела приближенных вычислений Института точной механики и вычислительной техники в 1950—1951 гг. над вопросами решения математических задач на цифровых машинах и, в частности, специального семинара по программированию, работавшего в Отделе в 1950 г.

При написании книги были использованы материалы отчетов по работам, выполненным в 1950—1951 гг. сотрудниками отдела приближенных вычислений ИТМ и ВТ АН СССР.

Ряд ценных замечаний, учтенных авторами, сделал В. М. Курочкин, которому авторы выражают свою благодарность.

Глава I

ЦИФРОВЫЕ МАШИНЫ И АВТОМАТИЗАЦИЯ ВЫЧИСЛЕНИЙ

§ 1. СИСТЕМЫ СЧИСЛЕНИЯ

Цифровые машины. С развитием науки и техники усложняются математические задачи, которые приходится решать, увеличивается объем вычислительной работы, связанной с решением таких задач. Современная техника позволила построить сложные машины, выполняющие чрезвычайно быстро большое количество вычислений по заранее разработанному плану. На таких машинах можно реализовать алгоритмы численного решения сложных математических задач.

Цифровыми математическими машинами называются такие, в которых числа задаются последовательностью своих цифр в некоторой позиционной системе — десятичной, двоичной и т. п. Количество разрядов и, следовательно, относительная точность чисел, над которыми оперирует машина, определяются ее конструкцией и могут быть, по крайней мере теоретически, сделаны сколь угодно большими. В этом преимущество цифровых машин перед приборами и машинами „непрерывного действия“ (например, логарифмическая линейка, планиметры, механические и электрические интеграторы и т. п.), в которых числа задаются непрерывно меняющимися величинами — длинами, углами, напряжениями, силами токов и т. п.; в них точность вводимых и получаемых чисел зависит от точности задания и измерения этих величин и является поэтому фактически ограниченной.

Изображать цифру какого-либо разряда в p -ичной системе счисления может любая физическая система, способная принимать p устойчивых состояний, которым отвечают соответственно p цифр данного разряда. Так, цифровое колесо обычного счетчика, обладающее десятью устойчивыми положениями, может изображать цифры десятичной системы 0, 1, 2, ..., 9. Цифра может изображаться числом костяшек, отложенных на счетах, числом электрических импульсов, положением отверстия, пробитого в столбце перфокарты, одним из возможных состояний цепи электронных ламп и т. д.

Двоичная и троичная системы. С точки зрения реализации в машинах особенно простой является двоичная система. Для изображения цифры какого-нибудь двоичного разряда можно воспользоваться любой физической системой, способной принимать два значения (одно относим цифре 0, другое — цифре 1), например, двумя положениями электромагнитного реле или двумя устойчивыми состояниями триггерной ячейки и т. п. Цифры двоичной системы могут отмечаться соответственно наличием или отсутствием какого-либо признака, например, наличием или отсутствием на определенном месте перфокарты отверстия и т. п. Вообще цифры 0 и 1 двоичной системы можно обозначать двумя из возможных ответов „да“ и „нет“ логической дизъюнкции.

Числа в двоичной системе имеют более длинную запись чем в десятичной (большее число разрядов), например, число $999 = 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2 + 1$ имеет двоичную запись в виде десятизначного числа 1111100011.

Но с точки зрения числа цифро-разрядов (разных цифр разных разрядов), с помощью которых можно изображать числа в данном диапазоне, двоичная система примерно в $1\frac{1}{2}$ раза экономнее десятичной. Для изображения 1000 целых чисел от 0 до 999 в десятичной системе нужны $3 \times 10 = 30$, в двоичной же $10 \times 2 = 20$ цифро-разрядов.

Наиболее экономной (с этой точки зрения) является троичная система: в двоичной системе с помощью $3 \times 2 = 6$ цифро-разрядов можно изобразить $8 = 2^3$ чисел (0, 1, 2, ..., 7),

в троичной с помощью $2 \times 3 = 6$ цифро-разрядов уже $9 = 3^2$ чисел (0, 1, 2, ..., 8).

Троичная система довольно проста с точки зрения оперирования, особенно если пользоваться тем ее вариантом, когда в качестве цифр фигурируют 1, 0 и $\bar{1} = -1$. В ней, например, $19 = 27 - 9 + 1 = 1 \cdot 3^3 - 1 \cdot 3^2 + 0 \cdot 3^1 + 1$ изображается в виде $1\bar{1}01$, а число -19 изображается в виде $\bar{1}10\bar{1}$. Таблица умножения в такой системе проста. В ней совершенно одинаково записываются положительные и отрицательные числа (они различаются тем, равна ли 1 или $\bar{1}$ первая отличная от нуля цифра). Но до сих пор троичная система не получила применения, так как из систем, отличных от десятичной, двоичная проще всех с точки зрения реализации.

Заметим, что отрицательные цифры $\bar{1}$ ($= -1$), $\bar{2}$ ($= -2$) и т. д. часто применяются и в вычислительной практике с обычными десятичными числами. Например, умножение на 48 на арифмометре заменяется умножением на $5\bar{2} = 50 - 2 = 48$ (быстрее сложить 5 удесятеренных множимых и вычесть два, чем сложить 4 удесятеренных множимых и 8 множимых).

Двоично-десятичная и двоично-пятиричная системы. При использовании в машине системы, отличной от десятичной, например, двоичной, требуется вводимые в машину числа преобразовывать из десятичной системы в двоичную, а при выводе обратно переводить в десятичную. Чтобы избежать таких преобразований и в то же время использовать хотя бы частично преимущества двоичной системы, некоторые машины, главным образом релейные, например Mark II, используют так называемую двоично-десятичную систему. Это система с основанием 10, каждая из цифр которой задается своим двоичным разложением: цифры 1, 2, 3, ..., 9 записываются в виде двоичных четырехзначных чисел 0001, 0010, 0011, ..., 1001. Употребляются и другие записи десятичных цифр комбинациями из 4-х двоичных.

Двоично-десятичная система менее экономна, чем двоичная, так как 4 двоичных разряда использованы для записи

лишь 10 чисел из 16 возможных; запись числа по двоично-десятичной системе на 20% длиннее его чисто двоичной записи. Например, число $643 = 512 + 128 + 2 + 1 = 2^9 + 2^7 + 2^1 + 1$ имеет двоичную запись 1010000011 (10 цифр) и двоично-десятичную: 0110 0100 0011 (12 цифр).

Упомянем еще о так называемой „двоично-пятиричной“ системе, при которой каждому десятичному разряду отвечает 5 двоичных; так как число сочетаний из 5 по 2 равно 10, то существует 10 пятизначных двоичных чисел, у которых две цифры 1 и остальные три 0: 11000, 10100, 10010, 10001, 01100, 01010, 01001, 00110, 00101, 00011, ими можно изображать 10 десятичных цифр. Каждая десятичная цифра может, например, изображаться системой из 5 реле, из которых 2 находятся в возбужденном состоянии. Такой способ изображения еще менее экономен, чем двоично-десятичный. Он нашел, однако, применение в телефонной аппаратуре и в счетных машинах, основанных на использовании этой аппаратуры (бэлловские релейные машины) ввиду удобства контроля передачи числа. Ошибка передачи чаще всего выражается в том, что в одной пятерке реле появятся не два, а три или одно возбужденное реле.

Чисто двоичная система нашла в последние годы широкое применение в так называемых быстродействующих машинах (см. § 5).

§ 2. АРИФМЕТИЧЕСКИЕ ДЕЙСТВИЯ И ИХ АВТОМАТИЗАЦИЯ

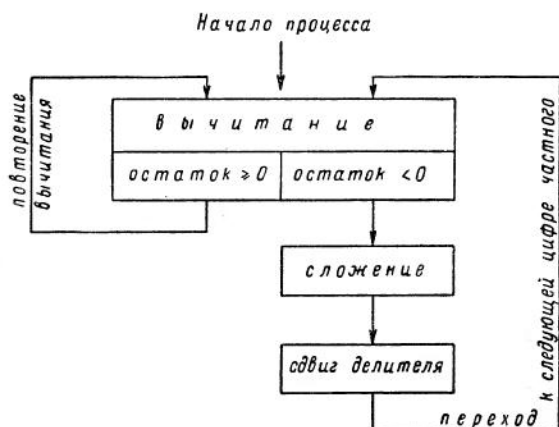
Долгое время единственными цифровыми машинами были суммирующие машины и арифмометры. Последние выполняют две операции: суммирование и сдвиг числа, т. е. его умножение на целую степень десяти. С помощью этих операций реализуются на арифмометре операции умножения и деления. Остановимся сначала на выполнении операции умножения.

Процесс умножения на однозначный множитель состоит из серии сложений (добавлений множимого), число которых равно множителю. Процесс умножения на многозначное число распадается на этапы или циклы, отвечающие умно-

жению на каждую цифру множителя и состоящие поэтому из серий сложений. После выполнения цикла, отвечающего умножению на k -ю цифру множителя, производится сдвиг множимого, после чего переходим к выполнению этапа, отвечающего умножению на $(k + 1)$ -ю цифру множителя.

Процесс деления также распадается на совершенно одинаковые циклы операций, с помощью которых определяются последовательно все цифры частного, а именно для получения его l -й цифры исходят из остатка, полученного после определения $(l - 1)$ -й цифры (для первой цифры этот остаток есть само делимое); из этого остатка последовательно вычитают делитель, сдвинутый на соответственное число разрядов. Вычитание производится, пока не получится на счетчике отрицательное число — пусть это случилось после $(k + 1)$ -го вычитания. После этого производят „переходные операции“, заканчивающие вычисление l -й цифры частного и подготавливающие переход к вычислению $(l + 1)$ -й: добавляют к числу на сумматоре делитель (так что всего мы фактически вычтем k раз — это и есть искомая l -я цифра частного); число, которое стоит после этого на сумматоре, будет $(l + 1)$ -м остатком; далее сдвигают делитель на один разряд вправо и повторяют весь цикл операций уже для определения $(l + 1)$ -й цифры (схема А).

Схема А



Мы остановились на реализации алгоритмов умножения и деления на обычном арифмометре, так как мы здесь имеем довольно типичный пример реализации алгоритма решения задач. Процесс вычисления распадается на циклы: в случае умножения — на циклы умножения на одну цифру множителя, в случае деления — на циклы для определения одной цифры частного. Каждый цикл, в свою очередь, распадается на повторяющиеся подциклы, состоящие из отдельных сложений или, соответственно, вычитаний. В случае деления число подциклов (вычитаний) в каждом цикле заранее неизвестно.

Первой задачей в области автоматизации вычислений была задача автоматизации выполнения операций умножения и деления. Впервые они были осуществлены в арифмометре Чебышева. Этот арифмометр, в отличие от обычных, обладал специальным колесным регистром множителя: если l -я цифра множителя равнялась a_l , то l -е цифровое колесо устанавливалось в положении a_l . Пусть мы уже умножили на все предыдущие цифры множителя и перешли к умножению на l -ю цифру (т. е. к серии добавлений соответственно сдвинутого множимого). При каждом таком добавлении (повороте ручки арифмометра) производится вычитание 1 из l -й цифры множителя, т. е. l -е разрядное колесо позиции занимает последовательно положения $s = a_l, a_l - 1, a_l - 2, \dots, 1, 0$.

Когда это колесо (после a_l -кратного добавления множимого) занимает позицию $s = 0$, машина автоматически прекращает добавлять множимое и сдвигает его на один разряд, после чего начинается повторение процесса уже для следующего $(l + 1)$ -го разряда множителя.

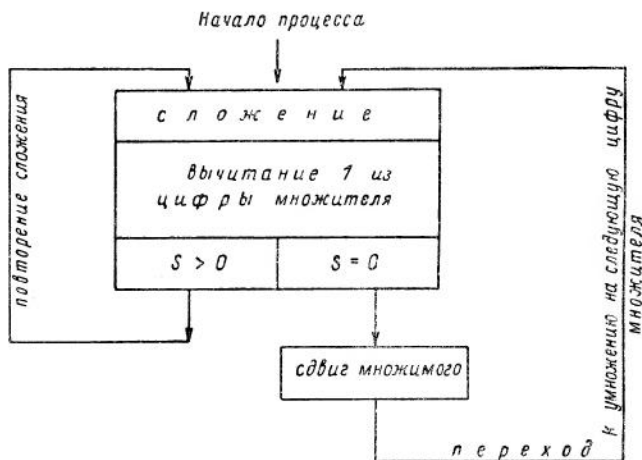
Для того чтобы выполнять умножение, машина должна уметь реагировать на позицию s соответственного разрядного регистра множителя: в зависимости от того, имеем ли мы $s > 0$ или $s = 0$, машина переходит к тому или иному продолжению хода операции (схема В).

Аналогично, автоматизация деления происходит на основе схемы А. Машина должна уметь реагировать на знак остатка, чтобы выбирать то или другое продолжение хода операции:

если остаток положительный, продолжаем производить вычитание, если остаток отрицательный, производим сложение, сдвиг, после чего переходим к следующему циклу вычитаний.

Здесь мы встречаемся с *разветвлением вычислительного процесса*, столь характерным для решений многих математических задач. Именно наличие этих разветвлений и составляло ту трудность, которую в свое время пришлось преодолеть при создании автоматических арифмометров.

Схема В



Заметим, что производство операций умножения и деления чрезвычайно упрощается, если перейти к двоичной системе. Умножение сводится к последовательности сложений и сдвигов. Каждый разряд множителя может принимать два значения: 0 и 1. В первом случае мы сразу сдвигаем множимое на один разряд, во втором — сначала добавляем множимое, затем сдвигаем. При делении искомая цифра частного определяется сразу после вычитания из предыдущего остатка соответственно сдвинутого делителя: если новый остаток положителен, то цифра равна 1, если отрицателен — равна 0. Подробнее об этом см. § 11.

Упомянем о других встречающихся приемах умножения многозначных чисел в десятичной системе.

1. Поразрядное умножение, применяемое в ручном счете, основанное на применении таблицы умножения однозначных чисел. Сложность его реализации вытекает из громоздкости таблицы умножения в десятичной системе. Такой способ умножения был принят в некоторых ручных вычислительных машинах (машина „миллионер“) и в так называемом „мультиплеере“ (машине из комплекта счетно-аналитических машин, см. § 3).

В двоичной системе метод поразрядного умножения и метод последовательных сложений со сдвигами совпадают.

2. Умножение, основанное на тех или иных формах представления множителя или отдельных его цифр по двоичной и трончной системе.

Если число A представлено по двоичной системе в виде

$$A = \sum_{k=0}^n a_k 2^k = ((\dots((a_n \cdot 2 + a_{n-1}) \cdot 2 + a_{n-2}) \cdot 2 + \dots + a_2) \cdot 2 + a_1) \cdot 2 + a_0$$

то умножение на A сводится к серии сложений и удвоений, т. е. тоже сложений (числа с самим собой), осуществляемым на чисто суммирующем устройстве.

Один из вариантов трончного представления цифр множителя осуществлен в машине ENIAC: каждая десятичная цифра представлена в виде $\epsilon_0 \cdot 1 + \epsilon_1 \cdot 3 + \epsilon_2 \cdot 6$, где $\epsilon_0, \epsilon_1, \epsilon_2$ равны 0, 1 или —1. Каждое число A представимо в виде $A = A_0 + 3A_1 + 6A_2$, где в десятичном изображении A_0, A_1, A_2 встречаются лишь цифры 0, 1 и —1; умножение на A_0, A_1, A_2 сводится к серии добавлений или вычитаний множимого.

§ 3. СЧЕТНО-АНАЛИТИЧЕСКИЕ МАШИНЫ

Следующим этапом в деле автоматизации вычислений было создание (в начале настоящего столетия) перфорационных или счетно-аналитических машин. Предыдущие счетные машины служили для выполнения индивидуальных действий — они требовали каждый раз задания на машине тех чисел, над которыми производилось действие. В перфорационных машинах процесс задания чисел, над которыми совершается операция, был впервые отделен от процесса операции. Числа кодировались системой пробивок на перфокартах — каждой цифре некоторого разряда отвечала своя позиция пробивки в соответственном столбце перфокарты. Все числовые данные заранее кодировались на перфокартах; точно так же отверстиями на них кодировались и особые сигналы управления машиной. Массив перфокарт закладывался в машину, перфокарты последовательно поступали в воспринимающие части машины, которая соответственным

образом реагировала на наличие или отсутствие пробивки в перфокарте: например, в электромеханических перфорационных машинах прохождение отверстия перфокарты в воспринимающем устройстве вызывало появление электрического сигнала, поступающего в рабочие части машины. Каждая перфокарта несла таким образом систему „команд“ машине.

В счетно-аналитических машинах была осуществлена схема управления посредством коммутаций. В основной, например, машине из цикла счетно-аналитических — многосчетчиковой суммирующей машине-табуляторе — сигналы, получаемые от прощупывания отверстий в каждом столбце перфокарты, могут быть направлены в любой разряд любого из счетчиков, что достигается соответствующей коммутацией (которая осуществлялась перед началом вычислений). Такая неизменная в ходе всего процесса вычислений коммутация оказалась недостаточно гибкой: пришлось дополнительно ввести приспособление, которое могло бы реализовать разные варианты направления сигналов, получаемых от перфокарты, в зависимости от наличия или отсутствия специальных пробивок. Осуществив все желаемые коммутации и направив в машину массив перфокарт, заставляют машину выполнять предусмотренную последовательность операций.

Счетно-аналитические машины возникли как машины, обслуживающие нужды бухгалтерии, статистики, учета. Поэтому в связи с потребностью этих областей практики раньше возникли суммирующие машины и лишь потом умножающие.

Одновременно с машинами для выполнения арифметических действий возникли перфорационные машины (сортировальные, раскладочные), выполняющие логические операции: классификации, расположения перфокарт в определенном порядке, выборки и т. п. Без выполнения реализуемых на таких машинах операций, связывающих между собой арифметические действия, невозможно механизировать решение даже сравнительно несложных задач, с которыми приходится иметь дело в указанных выше областях практики.

Как было выяснено выше, для автоматического производства вычислений машины должны производить „развет-

вления“ процесса, т. е. выбирать, реагируя на тот или иной признак, соответственно то или другое продолжение вычислительного процесса.

Рассмотрим, например, процесс образования сумм на счетчиках табулятора, складывающих числа, поступающие с перфокарт. После прохождения перфокарты в табуляторе возможны два продолжения работы его счетчика — или к образованному в нем числу добавится число, которое поступит со следующей перфокарты (продолжение процесса накапливающегося суммирования), или же счетчик погашается (переходит в положение нуля), причем стоящее в нем число — итог процесса суммирования — передается на выход (в печатающее или перфорирующее устройство). Признаком, определяющим то или иное продолжение работы, является совпадение или несовпадение пробивок в определенном столбце перфокарт, которыми отличаются разные серии перфокарт. Этот признак воспринимается машиной при синхронном прощупывании двух перфокарт — уже сработанной и поступающей в рабочую часть машины. При совпадении пробивок в соответственном месте обеих перфокарт процесс накапливающего суммирования продолжается, при несовпадении он заканчивается и начинается новый процесс такого суммирования.

Для управления машиной используются также специальные позиции перфокарт: присутствие или отсутствие этих пробивок вызывает то или другое состояние подвижной коммутации.

Таким образом, „читающие“ машины комплекта счетно-аналитических машин выполняют некоторые логические операции.

Однако, в большинстве случаев, этих логических операций оказывается недостаточно для проведения нужного расчета и приходится прибегать к специальным машинам для выполнения логических операций. Так, например, упомянутая раньше сортировальная машина распределяет массивы перфокарт на части; признаком при распределении является место пробивки в определенном столбце перфокарты. Эти машины выполняют основную работу при

обработке статистических данных и важную вспомогательную роль при других расчетах.

Раскладочная машина производит операцию сравнения многозначных чисел на разных перфокартах в определенном месте и *результат этого сравнения* (одно из противоположных неравенств или равенство) *является тем признаком, который определяет дальнейшее движение этих перфокарт.*

Далее возник вопрос о фиксации промежуточных ответов в такой форме, чтобы они снова могли поступать в машину для дальнейшей работы. В связи с этим машины были снабжены приспособлениями, выдающими ответы в виде серии пробивок на так называемых „итоговых перфокартах“. Итоговые перфокарты являются одной из форм сохранения чисел (промежуточных результатов работы) в машине.

Процесс вычислений на комплекте счетно-аналитических машин не является, как правило, вполне механизированным, он требует на отдельных этапах вмешательства человека. Такое вмешательство нужно не только при переходе от этапа вычисления, осуществляемого одной машиной комплекта к другой, но и при вводе итоговых перфокарт для дальнейшей работы над содержащимися в них числами.

Более поздние счетно-аналитические машины обладали запоминающими устройствами для хранения чисел в самой машине: такими были устройства для хранения константы (импульсаторы), а также счетчики табулятора, не участвующие непосредственно в вычислениях — они используются для хранения в них чисел, поступающих в процессе вычислений и в нужный момент передаваемых в другие счетчики.

Комплект счетно-аналитических машин мог выполнять все те операции — арифметические, логические и фиксации (запоминания) промежуточных результатов, к серии которых сводится решение весьма большого числа математических задач. Однако эффективность использования этих машин для решения более сложных задач снижалась необходимостью дробить процесс вычисления часто на очень короткие подпроцессы, выполняемые отдельными машинами, а также малым объемом внутреннего запоминающего устройства. Счетно-

аналитические машины применялись главным образом при массовом решении однотипных задач. Решение каждой из них сводилось к серии этапов вычислений, выполняемых отдельными машинами комплекта, причем каждая из однотипных задач одинаковым образом разбивалась на однотипные этапы. Весь массив решаемых задач проходил последовательно через эти этапы вычислений.

Для того чтобы одна машина могла полностью решать математические задачи, она должна, во-первых, быть агрегатом, способным производить все те элементарные арифметические и логические операции, к серии которых сводится решение задачи и которые выполняются различными счетно-аналитическими машинами; во-вторых, обладать достаточно объемным запоминающим устройством, чтобы хранить все введенные и возникающие в процессе вычислений числа, нужные для продолжения хода вычислений; в третьих, необходимо, чтобы можно было „настроить“ машину на решение этой задачи, т. е. заставить ее выполнить последовательно все операции, на которые решение задачи распадается.

§ 4. АВТОМАТИЧЕСКИЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

Первая по времени такая машина, так называемая машина Mark I, запроектированная в 1937 г., представляла собой агрегат, состоящий в основном из стандартных блоков счетно-аналитических машин, содержащий 72 23-разрядных суммирующих устройства, множительное и делительное устройства, блоки для специальных операций (логарифмирование, потенцирование, вычисление синуса, интерполяция) и блок управления. Счетчики суммирующих устройств выполняли также роль запоминающих устройств для хранения промежуточных результатов вычислений: кроме того, машина имела несколько десятков специально запоминающих устройств.

Решение задачи сводилось к последовательному выполнению операций разными блоками машины. Управление должно было, следовательно, заключаться в последовательном включении этих блоков и передаче результатов с одних

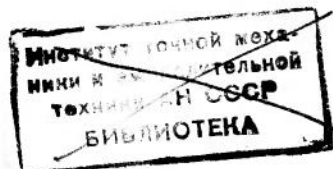
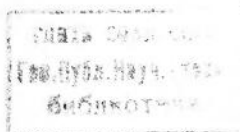
1011 67

блоков на другие. Поэтому для решения задачи необходимо было скомутировать соответственно блоки между собой. Однако эта коммутация не могла оставаться неизменной во все время процесса вычислений. Например, множительное устройство могло неоднократно включаться, причем в одном случае полученное произведение могло посылаться в делительное устройство в качестве делимого, а в другом, скажем, в суммирующее. Поэтому необходимо было обеспечить гибкость коммутаций, которые устанавливались бы и менялись в процессе вычислений. Это достигалось введением центрального управления, через которое связывались между собой отдельные блоки. Подаваемая в машину команда, содержащая код операции и коды („адреса“) двух блоков, воспринималась машиной, что приводило к соответствующей коммутации, обеспечивающей посылку числа из первого блока во второй и выполнение последним закодированной операции, результат которой остается во втором блоке. Команда имела запись $S\alpha\beta$, где S — код операции, α и β — адреса первого и второго блоков, и означала: направить число из блока α в блок β , выполнить над ним операцию S и результат сохранить в блоке β .

Кодами операций и кодами („адресами“) блоков были числа, записанные на перфоленге в виде серии пробивок. Здесь мы имеем пример так называемого двухадресного кода, когда команда содержит два адреса.

Все решение задачи распределено на элементарные операции, выполняемые блоками машины, и „программа“ этого решения заключалась в последовательности команд, записанных на перфоленге и вызывавших посылки чисел из одних блоков (частей машины) в другие для выполнения последними соответствующих операций. При движении перфоленга команды последовательно воспринимались и выполнялись машиной.

Составление программы, а затем запись ее в форме, воспринимаемой машиной, например, в виде серии пробивок на перфоленге, для некоторых задач является делом столь трудоемким, что может пропасть вся выгода от автоматического выполнения самих вычислений. Более того, во многих



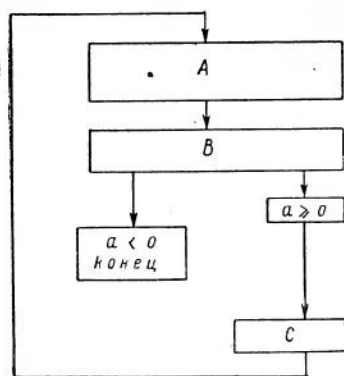
случаях тот или иной ход продолжения вычислительного процесса зависит от характера промежуточных результатов, заранее неизвестных.

Выше мы указали, что чаще всего весь алгоритм решения задачи распадается на циклы повторяющихся серий операций, которые в свою очередь могут разбиваться на более мелкие циклы, и т. д.; число циклов, на которые распадается весь алгоритм, или подциклов в цикле и т. д. может иногда определяться лишь в ходе решения задачи. Приведем в качестве нового примера решение уравнения

$$x = f(x)$$

методом итерации. Исходя из начального значения $x = x_0$, следует строить последовательно значения $x = x_1, x_2, \dots, x_k, x_{k+1}, \dots$, где $x_{k+1} = f(x_k)$. Этот процесс следует продолжать до тех пор, пока не совпадут с точностью до заданного числа $\epsilon > 0$ два последовательно полученные значения x_n и $x_{n+1} = f(x_n)$, т. е. пока $|f(x_n) - x_n|$ не станет меньше ϵ , или пока $a = |f(x_n) - x_n| - \epsilon$ не станет меньше 0. Число n итераций, которые надо произвести, заранее, следовательно, неизвестно.

Полагая первоначально $x = x_0$, мы должны проводить вычисления по следующей схеме:



A — по заданному значению x найти $f(x)$

B — найти $a = |f(x) - x| - \epsilon$, знак a определит дальнейший ход процесса: при $a < 0$ процесс вычислений заканчивается, при $a \geq 0$ готовится переход к следующей итерации

C — x заменяется на $f(x)$

Далее происходит возвращение к началу, т. е. повторение всего цикла вычислений, но с другими начальными

данными. Если вначале исходным значением было $x = x_0$, то после первого процесса итерации мы придем к значению $x = x_1 = f(x_0)$, после второго — к значению $x = x_2 = f(x_1)$ и т. д., пока не получим $|f(x) - x| - \epsilon < 0$, на чем вычисление остановится.

Программа должна предусматривать: проведение последовательности вычисления этапов A и B ; далее, в зависимости от знака $a = |f(x) - x| - \epsilon$, она должна предусматривать два возможных дальнейших хода процесса. В одном случае — прекратить вычисления, в другом — перейти к этапу C (замена исходных данных), после чего вернуться к повторению процесса с уже новыми исходными данными. Для того чтобы иметь возможность осуществить такую схему вычислений, машина должна реагировать на знак числа a , определяя то или иное продолжение процесса (появление того или другого знака должно вызвать ту или другую коммутацию). Далее должно быть предусмотрено возвращение от определенного этапа программы к более раннему. Это даст возможность реализовать решение задач, распадающихся на циклы.

Для реализации циклических повторений в первых универсальных цифровых машинах вводились специальные устройства. Так, в упомянутой машине Гарвардского университета Mark I наряду с основной программой вычислений, закодированной в виде пробивок на перфоленте, вводились местные программы, закодированные на других перфолентах. Для повторяющихся циклов операций местные программы кодировались на замкнутой ленте. Особый код на основной программе вводил в действие такую местную программу.

В первой электронной машине ENIAC для аналогичной цели служило специальное электронное устройство, реализующее возвращение к началу вычисления циклического процесса; другое устройство реализовало разветвления вычислительного процесса в зависимости от выработанного на определенном этапе вычислений признака — знака числа. В этой же машине специальное устройство заставляло повторять выполнение цикла заданное число раз.

Дальнейшим естественным развитием автоматических вычислительных машин было увеличение объема запоминающих устройств и усовершенствование управления в сторону большей гибкости. Большая гибкость управления последующих машин позволила, между прочим, освободиться от наличия блоков для таких операций, как, например, логарифмирование и т. д., которые можно свести к серии более простых операций.

Заметим в заключение, что большинство электромеханических автоматических машин является релейными машинами, в которых числа представлены по двоично-десятичной или двоично-пятеричной системе.

§ 5. БЫСТРОДЕЙСТВУЮЩИЕ АВТОМАТИЧЕСКИЕ МАШИНЫ

В то время как совершенствовалась автоматика производства вычислений, появился принципиально новый тип счетчиков в совершенно другой области техники. Речь идет о технике физического эксперимента. В ядерной физике для подсчета элементарных частиц требовались чрезвычайно быстро работающие счетчики.

В двадцатых годах появились электронные счетчики. В то время как во всех предыдущих счетчиках переход от состояния, отвечающего числу n , к числу $n + 1$ требовал механического движения — перемещения инерционных масс (вращение колес, перемещения якоря электромеханического реле и т. п.), в электронных счетчиках такой переход связан с изменением состояния некоторой системы электронных ламп, т. е. процессом практически безинерционным, время „срабатывания“ которого есть величина совершенно другого порядка. Электронные счетчики работают в тысячи раз быстрее инерционных.

Элементом такого быстродействующего счетчика явилась так называемая триггерная ячейка, система из пары электронных ламп (триодов), которая может находиться в двух устойчивых состояниях, в каждом из которых проводит ток одна из ламп. Под воздействием поступившего (достаточно большого) электрического импульса ячейка переходит из одного устойчивого состояния в другое. Если обозначать

эти состояния через 0 и 1, то под влиянием поступающих импульсов ячейка будет последовательно принимать состояния 0,1,0,1,0,1,...

Таким образом, триггерная ячейка выполняет функции одноразрядного двоичного счетчика (счетчика по модулю 2) поступающих импульсов. При переходе ячейки из состояния 1 в состояние 0 возникает импульс; если этот импульс передается в следующую ячейку, то он будет играть роль двоичной передачи. Серия ячеек u_1, u_2, \dots, u_k , где u_i соединено с u_{i+1} указанным образом, выполняет функции k -разрядного двоичного счетчика импульсов, поступающих в ячейку u_1 . Время срабатывания триггерной ячейки, т. е. перехода ее из состояния 0 в 1 или обратно, измеряется долей микросекунды. Поэтому такие счетчики работают с большой скоростью; они широко применялись, начиная с двадцатых годов текущего столетия, в экспериментальной физике для подсчета элементарных частиц.

Из этих же элементов можно сконструировать сумматор для сложения по двоичной (и по любой другой) системе, а следовательно, и арифметическое устройство для выполнения арифметических операций. Такое арифметическое устройство с безинерционными элементами работает в тысячи раз быстрее устройств с инерционными элементами и получило название быстродействующего.

Сочетание принципа автоматического управления вычислений с быстродействующим арифметическим устройством привело к созданию быстродействующих вполне автоматических машин, которые работают соответственно в тысячи раз быстрее обычных. Такие машины выполняют в секунду тысячи арифметических действий.

Быстрота выполнения арифметических действий требовала соответственной быстроты проведения других операций, например, посылки числа из арифметического устройства в запоминающее и обратно. В связи с этим на других более „тонких“ физических основах строились запоминающие устройства, блок управления и т. д.

Сама быстрота выполнения операции не только далеко раздвинула область практически разрешимых задач, она

отразилась на самой методике вычислений. Так, именно в быстродействующих машинах десятичная система счисления стала вытесняться двоичной. Последняя, как было указано выше, обладает рядом преимуществ, но зато требует предварительного перевода вводимых чисел, заданных в десятичной системе, в двоичную и, обратно, окончательных ответов из двоичной системы в десятичную. На быстродействующих машинах этот перевод совершается столь быстро, что необходимость его выполнения уже не является решающим фактором при выборе системы счисления; к тому же такие машины предназначены для решения задач, в которых ответы получаются в результате весьма длинной серии операций; поэтому дополнительные операции в начале и в конце вычислений, связанные с переводом чисел из одной системы в другую, не являются относительно обременительными.

Ряд машин предшествующих типов обладал серией параллельно работающих устройств (серией сумматоров), чем достигалось ускорение решения задач. В быстродействующих же машинах перешли, в основном, на последовательное выполнение операций на одном и том же устройстве. Этим достигалась экономия оборудования, а исключительная скорость выполнения каждой операции делала решение всей задачи достаточно быстрым и при последовательном выполнении операций. Более сложные операции сводятся к серии арифметических действий, выполняемых по специальным программам арифметическим устройством. Точно так же в машинах, работающих по двоичной системе, это же устройство обычно выполняет перевод чисел из одной системы в другую. В некоторых машинах (например, EDSAC) даже операция деления не является элементарной операцией машины и сводится к серии других арифметических действий.

В дальнейшем мы будем иметь в виду именно быстродействующие машины.

§ 6. ЗАПОМИНАЮЩЕЕ УСТРОЙСТВО

Во всех случаях, когда вычисления производятся вручную, все промежуточные результаты либо записываются, либо запоминаются.

Если же вычисления должны выполняться машиной совершенно автоматически, то функция фиксации промежуточных результатов и хранения их до момента использования в процессе вычислений должна также перейти от человека к этой автоматической машине. Эту задачу выполняют различные виды *запоминающих устройств* (сокращенно ЗУ).

Первые автоматические машины — счетно-аналитические — фиксировали и хранили лишь исходные данные и окончательные результаты вычислений. Исходные данные фиксировались посредством пробивок на перфокартах. Полученные окончательные результаты фиксировались в виде показаний счетчиков, в виде отпечатанных табулограмм или же в виде массива итоговых перфокарт.

При этом массив итоговых перфокарт может служить массивом исходных данных для другой серии вычислений. Но для этого требуется вмешательство человека.

Таким образом, для автоматического выполнения сколь угодно сложных и разнообразных вычислений совершенно недостаточно, чтобы автоматическая машина обладала только так называемым *внешним ЗУ*, т. е. могла хранить лишь исходные данные и фиксировать окончательные результаты. *Вполне автоматическая* универсальная вычислительная машина должна хранить внутри себя промежуточные результаты до их использования для дальнейших вычислений, т. е. должна обладать *внутренним ЗУ*.

В случае, когда скорость выполнения вычислительных устройств во много раз больше наивысшей технически достижимой скорости ввода чисел в машину и наибольшей скорости печатания или перфорирования получаемых в машине окончательных результатов, внутреннее ЗУ отличается от внешнего не только его назначением — хранить промежуточные результаты, — но также и скоростью его работы, т. е. скоростью, с какой фиксируются в нем промежуточные результаты, и скоростью, с какой эти результаты могут быть получены из него и посланы в дальнейшие вычисления. В этом случае внутреннее ЗУ является в то же время и *быстродействующим ЗУ*, могущим работать с той же

скоростью, с какой производятся внутри машины основные вычислительные операции.

Высокая скорость производства вычислительных операций, достигаемая в современных быстродействующих вычислительных машинах, оказывает влияние также и на конструкцию устройств ввода в машину и вывода из нее числовых данных.

Для сравнительно медленно работающих электромеханических и релейных автоматических вычислительных машин оказываются вполне удовлетворительными (с точки зрения скорости их работы) такие технические средства ввода исходных данных, как перфоленты и перфокарты и такие технические средства вывода получаемых результатов, как электромеханические пишущие машины или перфораторы. Введение постоянных величин в такие машины может осуществляться и посредством установки их от руки на специальных переключателях. Настройка таких машин на решение какой-либо задачи также производится в значительной мере вручную посредством осуществления требуемых для данной задачи соединений на коммутационных досках.

В первой электронной автоматической вычислительной машине (машине ENIAC) были применены такие же средства ввода исходных данных и вывода получаемых результатов. Но их применение в быстродействующих автоматических вычислительных машинах не является удовлетворительным с точки зрения эффективного использования этих машин.

Вследствие медленности процессов ввода и вывода эффективность от быстроты производства арифметических операций в значительной мере терялась. Для ускорения процесса ввода чисел в машину начали применяться намагниченные ленты, на которых вводимые в машину данные фиксированы посредством магнитной записи импульсов, ленты с фотозаписью импульсов, фотоэлектрический метод съема данных с перфоленты и т. д.

Скорость работы выходных устройств автоматических быстродействующих машин увеличивают также посредством применения магнитной записи получаемых результатов.

Однако и до сих пор ввод и вывод данных является наиболее медленным этапом работы.

Увеличение объема внутреннего ЗУ уменьшает число остановок машины, необходимых при переходе от решения одной задачи, или одной серии задач, к другой, облегчает программирование и вообще увеличивает эффективность использования быстродействующей вычислительной машины.

В современных быстродействующих автоматических вычислительных машинах объем внутреннего ЗУ достигает 10^5 и даже более двоичных позиций. Так, например, машина Raytheon строится с расчетом возможного расширения емкости внутреннего ЗУ до $2^{12} = 4096$ наборов цифр, каждый из которых имеет 45 двоичных разрядов, т. е. емкость внутреннего ЗУ этой машины может быть доведена до 184320 двоичных позиций.

Для решения некоторых задач, например, для решения систем линейных алгебраических уравнений с большим числом неизвестных, или же для приближенного решения уравнений в частных производных, даже такая емкость внутреннего ЗУ может оказаться недостаточной. Для того чтобы сделать возможным непрерывный автоматический процесс решения таких задач, вводится так называемое *промежуточное* ЗУ, скорость работы которого является промежуточной между скоростью работы устройств внешнего и внутреннего ЗУ.

Технически промежуточное ЗУ ныне чаще всего осуществляется посредством магнитной записи на стальную проволоку, ленту или барабан (наивысшая могущая быть использованной при магнитной записи частота импульсов в настоящее время равна 30000 гц).

В первых вполне автоматических вычислительных машинах (Mark I, ENIAC) функция внутреннего ЗУ еще не была специализирована и конструктивно отделена от других функций, выполняемых суммирующими устройствами машины. Каждое суммирующее устройство в этих машинах является накапливающим счетчиком, хранящим находящееся в нем число до тех пор, пока в него не поступит новое число, после чего счетчик будет уже сохранять сумму только что получен-

ного числа с числом, которое в нем находилось ранее. Каждый из накапливающих счетчиков сумматора машины является одновременно и ячейкой внутреннего ЗУ этой машины.

Каждая ячейка суммирующего ЗУ, являясь по существу суммирующим устройством, требует большого количества оборудования, что неизбежно приводит к тому, что из-за экономических соображений не могут быть осуществлены машины с достаточно большим внутренним ЗУ такого вида.

Как уже отмечалось, первая электромеханическая автоматическая вычислительная машина Mark I имела лишь 72 двадцатитрехразрядных десятичных счетчика, а первая электронная вычислительная машина ENIAC обладала лишь 20 десятиразрядными счетчиками. Но даже при столь ограниченной емкости внутреннего ЗУ машина ENIAC имела непомерно большое количество электронных ламп — 18000.

Естественно поэтому, что в построенных позднее и строящихся в настоящее время вычислительных быстродействующих машинах суммирующее ЗУ не применяется.

В современных автоматических быстродействующих машинах применяется новый тип внутреннего ЗУ, ячейки которого осуществляют более простую операцию — *операцию замещения* или *подстановки*. В этих машинах функции внутреннего ЗУ переходят уже к устройствам, конструктивно отделенным от устройств, выполняющих собственно вычисления и управления. Такое разделение функций позволило упростить конструкцию ЗУ настолько, что машины с большим внутренним ЗУ стали технически и экономически осуществимыми.

Ячейка ЗУ хранит число, записанное набором цифр, или команду, обычно также кодируемую набором цифр, и т. п. Каждая ячейка сохраняет содержащийся в ней набор лишь до тех пор, пока он не будет „вытеснен“ или заменен новым, посланным в него набором. Передаваемый из какой-либо ячейки ЗУ набор остается без всякого изменения в этой ячейке до тех пор, пока в эту ячейку не будет послан новый набор; таким образом набор может быть передан из одной ячейки в любую другую неограниченное число раз.

Такой тип ЗУ, естественно называть *вытесняющим*.

Глава II

ОПЕРАЦИИ НАД ЧИСЛАМИ В ЦИФРОВЫХ МАШИНАХ

§ 7. ПРЕДСТАВЛЕНИЕ ЧИСЕЛ

Каждая машина имеет дело с числами, представленными в некоторой p -ичной системе счисления ($p = 2, 3, 10$ и т. д.) с данным фиксированным числом n разрядов, т. е. с числами вида

$$x = \pm \sum_{k=1}^n x_k p^{m-k} = \pm p^m \left(\sum_{k=1}^n x_k p^{-k} \right) \quad (1)$$

здесь x_k ($k = 1, 2, \dots, n$) — цифры $0, 1, \dots, p-1$, а число m показывает место запятой (запятая перед $(m+1)$ -й от начала цифрой). Если показатель m одинаков для всех хранящихся в машине чисел, то машина называется машиной с *фиксированной запятой*; в этом случае нет необходимости особым образом отмечать положение запятой. Если же число m переменное, то машина называется машиной с *плавающей запятой*. В этом случае набор, изображающий число x по формуле (1), должен содержать не только изображение знака и набора цифр числа x , но и изображение показателя m .

Изображение знаков. Удобно отмечать знаки плюс и минус цифрами 0 и 1 и рассматривать, следовательно, знак как цифру первого разряда числа. Конечно, можно принять любой из двух возможных способов отметки знака двоичными цифрами. Мы остановимся на том, при котором

знак плюс отмечается цифрой 0, а знак минус — цифрой 1, т. е. знак числа x выражается функцией $\bar{1}(x)$, где

$$\bar{1}(x) = \begin{cases} 0 & \text{при } x > 0 \\ 1 & \text{при } x < 0 \end{cases}$$

При таком обозначении знаков в силу формул

$$\bar{1}(xy) \equiv \bar{1}(x) + \bar{1}(y) \pmod{2}, \quad \bar{1}\left(\frac{x}{y}\right) \equiv \bar{1}(x) + \bar{1}(y) \pmod{2}$$

знак произведения и частного определяется сложением по модулю 2 изображений знаков сомножителей или делимого и делителя, причем это сложение производится на обычном одноразрядном двоичном сумматоре. В этом и заключается преимущество такого изображения знака.

Изображение чисел в машинах с фиксированной запятой. Число x в машинах с фиксированной запятой изображается набором

$$x_0 x_1 x_2 \dots x_n$$

где x_0 — двоичная цифра, изображающая знак $x_0 = \bar{1}(x)$, а $x_1 x_2 \dots x_n$ — последовательные цифры изображения числа x . Если показатель m в формуле (1) равен нулю, то запятая фиксирована перед первой цифрой x_1 числа, и все числа, представленные в машине, суть правильные дроби. В машинах с фиксированной запятой чаще всего реализован этот случай. Дело в том, что при $m > 0$ произведение двух чисел часто становится больше наибольшего числа, которое можно представить в машине; при $m < 0$ происходит всегда большая потеря знаков; так что случай $m = 0$ наиболее удобен с точки зрения производства операции умножения.

Мы сейчас остановимся на машинах, работающих по двоичной системе при $m = 0$. Другие системы и другие значения m будут рассмотрены ниже.

В машинах рассматриваемого типа положительное число $x = 0, x_1 x_2 \dots x_n$ изображается в виде $x_0 x_1 x_2 \dots x_n$ ($x_0 = \bar{1}(x) = 0$). Условимся здесь набору $x_0 x_1 \dots x_n$ относить число $x_0, x_1 x_2 \dots x_n$ и сам набор отождествлять с этим числом; набор, изображающий положительное число $x = 0, x_1 x_2 \dots x_n$, совпадает с самим числом.

Для отрицательных чисел в машинах применяют три способа изображения, или, как говорят, три кода; изображение числа x в этих кодах будем обозначать соответственно $[x]_1$, $[x]_2$, $[x]_3$. Для положительного числа $x > 0$, в согласии с выше сказанным, будем считать

$$[x]_1 = [x]_2 = [x]_3 = x \quad (2)$$

Изображение отрицательных чисел и нуля. Для изображения отрицательных чисел применяются следующие коды:

1) прямой код — представление $[x]_1$ отрицательного числа $x = -0, x_1 x_2 \dots x_n = -|x|$ имеет вид:

$$[x]_1 = 1, x_1 x_2 x_3 \dots x_n = 1 + |x| = 1 - x \quad (3)$$

При прямом коде ноль может иметь два представления, которые мы будем называть соответственно положительным и отрицательным нулем.

$$[+0]_1 = 0,000\dots 0; [-0]_1 = 1,00\dots 0 \quad (4)$$

Прямым кодом представимы числа x из отрезка $[-(1-2^{-n}), 1-2^{-n}]$

2) дополнительный код $[x]_2$ представления отрицательного числа $x = -0, x_1 x_2 \dots x_n$ имеет вид:

$$[x]_2 = 1, x'_1 x'_2 \dots x'_n$$

где после запятой стоит дополнение $|x|$ до 1, т. е. число $1 - |x| = 1 + x$. Итак, при $x < 0$

$$[x]_2 = 2 - |x| = 2 + x \quad (5)$$

Пример: при $x = -0,11001$, $[x]_2 = 1,00111$.

При дополнительном коде ноль может иметь только одно изображение:

$$[0]_2 = 0,00\dots 0 \quad (6)$$

Число же $1,00\dots 0$ изображает отрицательную единицу: $[-1]_2 = 1,00\dots 0$. В дополнительном коде можно представить все числа отрезка $[-1, 1-2^{-n}]$.

Равенства (2), (5), (6) можно объединить записью:

$$[x]_2 \equiv x \pmod{2} \quad (7)$$

отсюда следует

$$[x + y]_2 \equiv x + y \equiv [x]_2 + [y]_2 \pmod{2} \quad (8)$$

Сложение по модулю 2 реализуется на сумматоре, в котором отсутствуют разряды старше разряда единиц и нет передачи с высшего разряда на низший. Такой сумматор производит, следовательно, суммирование чисел, представленных дополнительным кодом.

3) обратный код $[x]_3$ представления отрицательного числа $x = -0, x_1 x_2 \dots x_n$ заключается в изображении числа поразрядным дополнением $[x]_3 = 1, \bar{x}_1 \bar{x}_2 \dots \bar{x}_n$, где $\bar{x}_k = 1$, если $x_k = 0$ и $\bar{x}_k = 0$, если $x_k = 1$. Иными словами:

$$\begin{aligned} [x]_3 + |x| &= 1,111 \dots 1 = 2 - 2^{-n} \\ [x]_3 &= 2 - 2^{-n} - |x| = x + (2 - 2^{-n}) \end{aligned} \quad (9)$$

При обратном коде так же, как и в прямом, имеются два изображения нуля:

$$[+0]_3 = 0,00 \dots 0; \quad [-0]_3 = 1,11 \dots 1 = 2 - 2^{-n} \quad (10)$$

В силу формулы (2) и (9) в результате сложения и вычитания $y - y = y + (-y)$ всегда возникает -0 : если $y > 0$, то $[y]_3 = y$, $[-y]_3 = 2 - 2^{-n} - y$; $[y]_3 + [-y]_3 = 2 - 2^{-n} = [-0]_3$.

Наибольшее по абсолютной величине изобразимое обратным кодом отрицательное число есть число $-0,111 \dots 1 = -(1 - 2^{-n})$, для которого $[-(1 - 2^{-n})]_3 = 1,000 \dots 0$, т. е. при обратном, как и при прямом коде, можно представить все числа сегмента $[-(1 - 2^{-n}), 1 - 2^{-n}]$.

Из (2), (9) и (10) следует, что для любого x

$$[x]_3 \equiv x \pmod{2 - 2^{-n}} \quad (11)$$

При обратном коде эквивалентными являются числа, сравниваемые по модулю $2 - 2^{-n}$; так как, в частности, $2 \equiv 2^{-n} \pmod{2 - 2^{-n}}$, то 2 эквивалентно 2^{-n} , т. е. единица

второго разряда перед запятой, нереализованного в машине, эквивалентна единице последнего разряда. Эта эквивалентность реализуется тем, что *между разрядом целых единиц (разрядом знаков) и последним разрядом осуществляется в сумматоре двоичная передача (циклическая двоичная передача)*. Так как из формулы (11) следует:

$$[x + y]_2 \equiv x + y \equiv [x]_2 + [y]_2 \pmod{(2 - 2^{-n})} \quad (12)$$

то сумматор с циклической двоичной передачей осуществляет суммирование чисел, заданных обратным кодом.

Прямой код для представления чисел удобен при произведении действий умножения и деления, поскольку там перемножаются и делятся абсолютные величины чисел. Для операций сложения и вычитания удобны дополнительный или обратный код. Для обратного кода операция перемены знака числа, которая заключается в замене каждой цифры на дополнительную, проще, чем эта же операция для дополнительного кода.

Напомним, что все числа, которые можно представить в машине, должны быть правильными дробями, поэтому в машинах рассматриваемого типа в результате сложения не должно появляться неправильных дробей (по той же причине запрещено деление большего по абсолютной величине числа на меньшее). Об автоматической реализации запретов см. ниже § 10.

Изображение n -значных чисел в виде бесконечнозначных. Положительное n -значное двоичное число

$$x = 0, x_1 x_2 \dots x_n \quad (13)$$

может рассматриваться как бесконечнозначное

$$x = 0, x_1 x_2 \dots x_n 00 \dots 0 \dots \quad (14)$$

Для отрицательных n -значных чисел разность между изображениями дополнительным и обратным кодами, равная 2^{-n} , стремится к 0 при $n \rightarrow \infty$. Для бесконечнозначных чисел изображения обратным и дополнительным кодами равны.

Обратный n -значный код числа $-x = -0, x_1 x_2 \dots x_n$ получается из (13) поразрядным дополнением

$$[-x]_3 = 1, \bar{x}_1 \bar{x}_2 \dots \bar{x}_n \quad (15)$$

Соответственно обратный бесконечнозначный код числа $-x$ получается из (14) поразрядным дополнением. Он равен

$$[-x]_3^\infty = 1, \bar{x}_1 \bar{x}_2 \dots \bar{x}_n 11 \dots 1 \dots \quad (16)$$

т. е. получается из n -значного обратного кода (15) прибавлением бесконечного „хвоста“ единиц.

Так как $0, \underbrace{00 \dots 01}_{n} 11 \dots = 0, \underbrace{00 \dots 10}_{n} 00 \dots = 2^{-n}$, то

$$[-x]_3^\infty = [-x]_3 + 2^{-n} = [-x]_2$$

Дополнительный бесконечнозначный код $[-x]_2^\infty$ числа $-x$ равен его бесконечнозначному обратному коду, так как

$$[-x]_2^\infty = [-x]_3^\infty = [-x]_2$$

Бесконечнозначный дополнительный код $[-x]_2^\infty$ числа $-x$ равен его n -значному дополнительному коду $[-x]_2$ и получается из последнего добавлением бесконечного хвоста нулей.

Таким образом, дополнительный и обратный коды отрицательных чисел отвечают двум способам представления n -значных чисел в виде бесконечнозначной дроби.

Другие случаи машин с фиксированной запятой. Если запятая фиксирована перед k -й от начала цифрой двоичного разложения числа, то все предыдущие рассмотрения, в частности изображения чисел разными кодами, сохраняют силу, если заменить сравнения по модулю 2 и $2 - 2^{-n}$ соответственно сравнениями по модулю 2^k и $2^k - 2^{k-1-n}$. Отрицательный ноль в обратном коде имеет тогда запись

$$[-0]_3 = 2^k - 2^{k-1-n} = \underbrace{111 \dots 1}_{k-1}, \underbrace{11 \dots 1}_{n-k+1}$$

(первая цифра 1 означает знак минус).

Если, в частности, $k = n + 1$, т. е. запятая фиксирована после всех цифр, машина изображает только целые числа.

В машинах с десятичной системой счисления двоичный разряд знака отличается от следующих за ним десятичных знаков. Поразрядное дополнение в десятичной системе заключается в замене каждой десятичной цифры k ее дополнением до девятки, $9 - k$: например, при фиксации запятой перед первой цифрой обратный код (поразрядное дополнение) трехзначного числа $-0,327$ есть число $1,672$ (1 определяет знак минус); отрицательный ноль в обратном коде с такой фиксацией запятой имеет вид $1,99 \dots 9$.

Некоторые из машин, работающих по двоично-десятичной системе, пользуются такой двоичной записью десятичных цифр, при которой поразрядное десятичное дополнение сводится к двоичному. Для этого достаточно, чтобы дополнительные десятичные цифры k и $9 - k$ изображались четырехзначными двоичными числами, дополняющими друг друга до $1111 (= 15)$. Например, если каждая десятичная цифра k ($k = 0, 1, \dots, 9$) изображается двоичным разложением числа $k + 3$, то пара дополнительных цифр k и $9 - k$ имеет двоичные изображения, равные $k + 3$ и $12 - k$, дополняющие друг друга до числа $1111 (= 15)$. Так, трехзначное число $0,723$ будет иметь двоично-десятичное изображение $R = 0,1010 \ 0101 \ 0110$, а число $-0,723$ в обратном коде имеет двоично-десятичную запись $R_1 = 1,0101 \ 1010 \ 1001$, получаемую обычным двоичным поразрядным дополнением из R .

При решении задач на машинах с фиксированной запятой приходится иметь дело с числами, выходящими за пределы диапазона чисел, которые можно представить в машине с запятой, фиксированной перед первой цифрой. С этой целью числа, хранящиеся в некоторой группе ячеек ЗУ, считаются взятыми в некотором масштабе, т. е. снабженными некоторым постоянным для этой группы множителем. Иногда таких групп ячеек со своим масштабным множителем вводится несколько. Группа, к которой принадлежит ячейка, служит указателем величины множителя. Дальнейшим развитием представления чисел с множителями явилось появление машин с плавающей запятой.

Представление чисел в машинах с плавающей запятой. В машинах с плавающей запятой числа заданы в виде $x = p^m u$, где m — целое число, называемое порядком числа x , а u — число по абсолютной величине меньшее единицы. Ограничимся случаем $p = 2$. Число называется нормализо-

ванным, если оно представлено в виде $x = 2^m y$, где $y = \pm 0,1 x_2 x_3 \dots x_n$; первая цифра после запятой есть 1, т. е. $1/2 \leq |y| < 1$. В этом случае y называется цифровой частью числа x .

Часть разрядов набора, изображающего число, служит для изображения порядка, остальные разряды набора — для изображения цифровой части. Порядки изображаются как в машинах с запятой, фиксированной после всех цифр; цифровые части — как в машинах с запятой, фиксированной перед первой цифрой.

Если для изображения порядка служат, например, первые разряды, то набор, изображающий число в машине с плавающей запятой, имеет вид

$$y_0 y_1 y_2 \dots y_r, x_0 x_1 x_2 \dots x_n$$

Здесь $y_0 y_1 \dots y_r$ — изображение порядка (y_0 — знак порядка), $x_0 x_1 x_2 \dots x_n$ — изображение цифровой части (x_0 — ее знак).

При $r = 3$, $n = 6$, набор

$$\begin{array}{c} \underline{0\ 1\ 0\ 1}, \quad \underline{0\ 1\ 0\ 1\ 1\ 1\ 0} \\ \text{порядок} \quad \text{цифровая часть} \end{array}$$

изображает число с порядком $+101 (=5)$ и цифровой частью $+0,101110$, т. е. число $2^5 \cdot 0,101110$.

Пусть для изображения порядков даются r разрядов, а также разряд знака. Тогда порядки принимают целые значения от $-(2^r - 1)$ до $2^r - 1$. Например, при $r = 4, 5, 6, \dots$ порядки могут принимать целые значения, соответственно, от -15 до $+15$, от -31 до $+31$, от -63 до $+63$ и т. д. Соответственно машина может изображать числа в диапазоне примерно от 2^{-15} до 2^{15} , от 2^{-31} до 2^{31} , от 2^{-63} до 2^{63} и т. д.

В машинах с фиксированной запятой одинаковые разряды всех чисел занимают одинаковое положение. Это удобно для выполнения действий сложения и вычитания, поэтому выполнение этих операций на машинах с фиксированной запятой проще, чем на машинах с плавающей запятой.

Основным же преимуществом машин с плавающей запятой является большой диапазон чисел, которые можно пред-

ставить в машине, как весьма малых по абсолютной величине, так и больших, причем все нормализованные числа можно представить в машинах с плавающей запятой с примерно *одинаковой относительной точностью*. Между тем, в машинах с фиксированной запятой меньшие числа представляются с меньшей относительной точностью; отсюда проистекает потеря знаков при умножении таких чисел. Это вызывает неудобство при решении задач, требующих большого количества умножений, например, при решении систем алгебраических уравнений методом исключения. Кроме того, в таких машинах часто приходится учитывать запреты в смысле появления „слишком больших“ чисел в результате операций. Например, в машинах наиболее удобных с других точек зрения, в которых запятая фиксируется перед первым разрядом, все числа, которые можно представить, меньше 1 по абсолютной величине; здесь нужно поэтому предусматривать, чтобы в результате арифметических операций не появилось чисел, больших 1. В машинах же с плавающей запятой верхняя граница представимых чисел обычно достаточно велика и с запретом превосходить ее приходится считаться значительно реже. Поэтому реже приходится прибегать и ко всяким предосторожностям при планировании вычислений, например, к изменению масштаба и т. д.

Для изображения цифровых частей употребляется как прямой, так и дополнительный и обратный коды. Что касается порядков, то с ними производятся, в основном, операции сложения и вычитания, для которых удобнее пользоваться обратным или дополнительным кодами. Поэтому они, естественно, и применяются при изображении порядков.

Пусть $-m = -(2^r - 1) = -111...1$ наименьший из возможных порядков. Тогда число $x < 2^{-m} \cdot 0,1 = 2^{-(m+1)}$ уже не может быть представлено в нормализованном виде. Эти числа обычно принимаются за 0: они меньше наименьшего нормализованного числа, представленного в машине, и к ним „обычные операции“ в машинах с плавающей запятой, связанные с нормализацией, неприменимы. Таким образом, наличие наименьшего порядка $-m$ и первой цифры 0 после

запятой в цифровой части числа есть указатель нуля. Для простоты можно отказаться и от чисел вида $x = 2^{-m} \cdot 0,1\alpha \dots$ ($2^{-(m+1)} \leq x < 2^{-m}$), т. е. принять их тоже равными нулю и считать указателем нуля наличие наименьшего порядка $-m$.

§ 8. ПОРАЗРЯДНЫЕ ОПЕРАЦИИ

Рассмотрение различных операций, совершаемых над числами и, вообще, над наборами, хранящимися в машине, мы начнем с „поразрядных“ операций, т. е. тех, в которых операции над цифрами каждого разряда совершаются независимо друг от друга. Сюда относятся, например, такие операции, как преобразования числа из прямого кода в обратный и из обратного в прямой, взятие абсолютной величины числа и т. д.

Передача числа из одной части машины в другую, например, из арифметического устройства в ячейку ЗУ и обратно, в том случае, когда число не подвергается никаким преобразованиям, есть также поразрядная операция. Передача числа, заданного прямым (обратным) кодом, с преобразованием его в обратный (прямой) также является поразрядной операцией. Мы ограничимся рассмотрением этих операций в двоичной системе.

Поразрядное дополнение. Эта операция преобразует набор $x_0 x_1 x_2 \dots x_n$ в набор $\bar{x}_0 \bar{x}_1 \dots \bar{x}_n$, где $\bar{x}_k = 1 - x_k$. В машинах с фиксированной запятой такая операция превращает число x , заданное обратным кодом, в число $-x$, заданное тем же кодом, что вытекает из равенства (9), § 7. При таком задании чисел операция изменения знака является поразрядной операцией.

Чаще встречается аналогичная операция, при которой поразрядному дополнению подвергается лишь фиксированная часть разрядов набора. Например, поразрядное дополнение всех цифровых разрядов числа (без разрядов знака) превращает прямой код отрицательного числа в обратный и наоборот. Последняя операция фигурирует, как часть операции сложения, в машинах, в которых числа в ЗУ заданы прямым

кодом, а сумматор работает по принципу обратного кода (см. стр. 31).

В машинах с плавающей запятой изменение знака числа, заданного обратным кодом, получается поразрядным дополнением всех разрядов цифровой части числа, включая разряд знака.

Передача числа из одной части машины в другую в виде его поразрядного дополнения называется *инверсной*.

Поразрядное сложение. Одноразрядный двоичный сумматор совершает сложение по модулю 2. Если такое сложение обозначить знаком \oplus , то мы имеем следующую таблицу

$$0 \oplus 0 = 0, 0 \oplus 1 = 1 \oplus 0 = 1, 1 \oplus 1 = 0$$

Поразрядное сложение $x \oplus y$ двух наборов x и y заключается в сложении по модулю 2 цифр каждого разряда этих наборов: если $x = x_0 x_1 \dots x_n$, $y = y_0 y_1 \dots y_n$, то

$$x \oplus y = z_0 z_1 \dots z_n, z_k = x_k \oplus y_k$$

Поразрядное сложение значительно проще обычного, так как оно не требует передачи единиц из низших разрядов в высшие.

Операцию поразрядного дополнения числа x можно осуществить путем поразрядного сложения. Для этого нужно осуществить поразрядное сложение $x \oplus y$, где набор y имеет цифру 1 во всех разрядах, для которых осуществляется поразрядное дополнение, и цифру 0 в остальных разрядах.

Поразрядное логическое сложение. Логическое сложение двоичных цифр определяется таблицей:

$$0 \vee 0 = 0, 0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$$

Поразрядное логическое сложение двух наборов

$$x = x_0 x_1 \dots x_n, y = y_0 y_1 \dots y_n$$

заключается в получении набора

$$x \vee y = z_0 z_1 \dots z_n, z_k = x_k \vee y_k$$

Операция логического сложения может служить для модификации команд и чисел. Пользуясь этой операцией,

можно легко, например, получить из числа x , заданного прямым кодом, его отрицательный модуль $-|x|$. Для этого надо набор $x_0, x_1, x_2, \dots, x_n$, являющийся прямым кодом числа x , логически сложить с набором $y = 1, 00 \dots 0 \dots 0$.

Если в двух наборах $x = x_0, x_1, \dots, x_n$ и $y = y_0, y_1, \dots, y_n$ не встречается одновременного равенства единице цифр одинакового разряда, то поразрядное логическое сложение x и y совпадает с поразрядным сложением, а также и с обычной операцией сложения $x + y$ в машинах с фиксированной запятой.

Поразрядное (логическое) умножение. Поразрядное умножение двух наборов $x = x_0, x_1, x_2, \dots, x_n$, $y = y_0, y_1, y_2, \dots, y_n$ заключается в образовании набора

$$x \wedge y = z_0, z_1, \dots, z_n, \quad z_k = x_k y_k$$

где операция $x_k \cdot y_k$ есть обычная операция умножения цифр 0 и 1:

$$(0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0, \quad 1 \cdot 1 = 1)$$

Операция эта называется иногда операцией *сличения* или *выборки*, так как она позволяет из набора $x = x_0, x_1, x_2, \dots, x_n$ выбрать любые его разряды посредством сличения этого набора с таким набором y , у которого в разрядах, соответствующих тем, которые надо выбрать из числа x , стоят единицы, а во всех остальных разрядах — нули.

Пользуясь этой операцией, легко можно выполнить выделение знака числа x , его абсолютной величины, цифровой его части или только модуля этой части, порядка числа x , знака или же модуля порядка, шифра операции какой-нибудь команды или любого ее адреса (см. ниже, § 19). Она может быть использована также для выделения целой или дробной части числа, а также для гашения содержимого какого-либо места ЗУ.

Для выделения знака x_0 числа x , представленного набором x_0, x_1, \dots, x_n , надо сличить этот набор с набором $y = 1, 0 \dots 0$, у которого во всех разрядах стоят нули, кроме разряда знака, в котором стоит единица. Если же требуется получить абсолютную величину $|x|$ числа x , представленного прямым кодом, то достаточно положить $y = 0, 1 \dots 1$.

Аналогично может быть произведено выделение знака или модуля порядка или, вообще, любой части набора.

Перемена знака числа может быть осуществлена либо как частный случай некоторых арифметических операций (именно, вычитания числа x из нуля), либо посредством специальной операции. Для чисел, представленных прямым кодом, для осуществления этой операции достаточно изменить цифру знака в наборе. Для чисел, представленных обратным кодом, достаточно произвести операцию поразрядного дополнения. Для чисел, заданных дополнительным кодом, перемена знака не является поразрядной операцией: она сводится к поразрядному дополнению и добавлению единицы в последний разряд.

Образование абсолютной величины числа x есть тождественная операция при $x \geq 0$ и сводится к операции перемены знака числа при $x < 0$. Особенно просто техническое осуществление этой операции для чисел, представленных прямым кодом. Для реализации этой операции будет достаточно передать в арифметическое устройство машины все разряды числа x , кроме разряда знака.

Изменение знака числа x в зависимости от знака числа y . Сюда относятся операция *умножения числа x на знак числа y* , т. е. образование числа $\text{sign } y \cdot x$, и операция *передачи числу x знака числа y* , т. е. образование числа $\text{sign } y \cdot |x|$. Эти операции могут быть сведены к предыдущим операциям, а могут фигурировать и как самостоятельные операции. Например, для реализации второй из этих операций, т. е. для образования числа $\text{sign } y \cdot |x|$, если x задано прямым кодом, достаточно осуществить устройство, которое посылало бы по каналу разряда знака знак числа y , а по остальным каналам соответственные значения цифр числа $|x|$.

Передача числа из одних частей машины в другие. При передаче числа из внутреннего ЗУ в арифметическое устройство число остается в ячейке ЗУ совершенно в том же виде, в каком оно было до этой операции, так что операция эта заключается просто в „прочтении“ или „восприятии“ арифметическим устройством набора, находящегося

в той ячейке ЗУ, откуда он поступает в арифметическое устройство.

Но в арифметическом устройстве набор, поступивший из ячейки ЗУ, в общем случае преобразуется при передаче чисел из одних частей арифметического устройства в другие; при этом широко применяется не только прямая, но и *инверсная передача*, т. е. передача кода числа поразрядным дополнением. Если, например, x задано обратным кодом, то инверсная передача числа x означает передачу его с изменением знака. Если число x в ячейке ЗУ представлено прямым кодом $[x]_1$, то в сумматор оно попадает представленным либо обратным кодом $[x]_3$, либо дополнительным кодом $[x]_2$, в зависимости от типа сумматора (т. е. в зависимости от отсутствия или наличия в сумматоре циклической передачи из разряда знака в младший цифровой разряд)¹. Если же в ячейках ЗУ числа хранятся в виде дополнений, т. е. представлены дополнительным или обратным кодом, то в суммирующий блок числа передаются из ячеек ЗУ без всяких изменений. В множительное устройство числа поступают без всяких изменений или с изменением кода, в зависимости от того, совпадает или не совпадает код, применяемый в ячейках ЗУ, с кодом, по которому работает множительное устройство. При передаче числа из арифметического устройства в ЗУ коды чисел подвергаются операциям, обратным предыдущим.

Как указано в § 6, в современных машинах ячейки ЗУ автоматически очищаются (устанавливаются на нуль) перед тем, как в них поступает новое число².

В одних машинах арифметическое устройство после послышки из него числа всегда гасится, т. е. устанавливается на нуль, в других машинах³ оно может как гаситься, так и сохранять посылаемое из него число.

¹ Передача дополнительным кодом требует добавления 1 в последний разряд обратного кода, что требует связи между разрядами.

² В машинах с суммирующим ЗУ для замены в ячейке ЗУ числа x числом y нужно было послать в эту ячейку разность $y - x$.

³ Например, в машинах одноадресного кода (см. § 19).

Операция передачи числа со входа машины или из внешнего ЗУ во внутреннее ЗУ и обратно сводится к простой операции пересылки в случае, когда числа во внутреннем ЗУ машины представлены в той же системе счисления, что и числа, поступающие со входа машины, как это имеет место, например, в машинах, вычисляющих в десятичной системе.

В случае же, когда числа представлены во внутреннем ЗУ машины в системе счисления, отличной от той, в какой числа вводятся в машину или в какой системе счисления числа хранятся во внешнем ЗУ, то эти операции становятся весьма сложными, так как они включают в себя операции, необходимые для преобразования чисел из одной системы в другую (см. стр. 122).

§ 9. ОПЕРАЦИИ СДВИГА И НОРМАЛИЗАЦИИ

Сдвиг. Операция сдвига заключается в смещении цифр набора на фиксированное число разрядов вправо или влево. Сдвиг на s разрядов вправо заменяет набор $x_0x_1x_2\dots x_n$ на набор $\underbrace{0\dots 0}_s x_0x_1\dots x_{n-s}$. Сдвиг на s разрядов влево заменяет набор $x_0x_1\dots x_n$ на набор $x_sx_{s+1}\dots x_n \underbrace{00\dots 0}_s$. Иногда операции сдвига подвергается не весь набор $x_0x_1\dots x_n$, а часть $x_kx_{k+1}\dots x_m$ этого набора; например, в машинах с плавающей запятой можно говорить о сдвиге в цифровой части числа.

Операции сдвига являются элементами операции умножения и деления, а в машинах с плавающей запятой — операции нормализации, а также сложения и вычитания. Эти же операции позволяют выделить знак и модуль числа, порядок и цифровую часть в машинах с плавающей запятой и вообще стереть (т. е. сделать равными нулю) заданное число высших или низших разрядов набора. Таким образом, операции сдвига во многих наиболее важных случаях заменяют операцию сличения.

Умножение на 2^k и 2^{-k} . Для операции умножения на 2^{-k} в машинах с фиксированной запятой употребляется

модификация операции сдвига вправо на k разрядов, заменяющая набор $x = x_0x_1x_2 \dots x_n$ набором $\underbrace{x_0x_0 \dots x_0}_k x_0x_1 \dots x_{n-k}$.

Если $x > 0$, $x = 0, x_1x_2 \dots x_n$, то

$$2^{-k} \cdot x = 0, \underbrace{0 \dots 0}_k x_1x_2 \dots x_{n-k}$$

Если $y < 0$ и его обратный (дополнительный) код равен $1, y_1y_2 \dots y_n$, то обратный код $2^{-k} \cdot y$ равен $1, \underbrace{1 \dots 1}_k y_1y_2 \dots y_{n-k}$.

Операция умножения числа $x = 0, \underbrace{0 \dots 0}_k x_1x_2 \dots x_{n-k}$ на 2^k заключается в сдвиге на k разрядов влево, т. е. $2^k \cdot x = 0, x_1x_2 \dots x_{n-k} \underbrace{0 \dots 0}_k$. Аналогично, если обратный (дополнительный) код числа y равен $1, \underbrace{1 \dots 1}_k y_1y_2 \dots y_{n-k}$, то соответственный код числа $2^k \cdot y$ равен $1, y_1y_2 \dots y_{n-k} \underbrace{1 \dots 1}_k$ ($1, y_1 \dots y_{n-k} \underbrace{0 \dots 0}_k$). Итак, умножение на 2^k числа, заданного

дополнительным кодом, сводится к сдвигу кода этого числа на k разрядов влево с появлением в освободившихся младших k разрядах цифры 0; умножение на 2^k числа, заданного обратным кодом, сводится к k последовательным циклическим сдвигам влево на один разряд, т. е. сдвигам, при которых в освободившийся младший разряд поступает цифра из разряда знака.

Операция нормализации числа. Эта операция сводится, в основном, к операции сдвига. Выше указывалось, что числа, представленные в машине с плавающей запятой в нормализованном виде, имеют цифровую часть, заключенную по абсолютной величине между $1/2$ и 1. В результате выполнения арифметических действий в арифметическом блоке могут образоваться числа, представленные в ненормализованном виде. При сложении чисел противоположного знака или при умножении могут образоваться числа вида:

$$x = 2^m \cdot y, \text{ где } y = \pm 0,00 \dots 01 \alpha \dots$$

т. е. абсолютная величина у меньше половины. Например,

$$2^m \cdot 0,1 \cdot 2^n \cdot 0,1 = 2^{m+n} \cdot 0,01; 2^m \cdot 0,111 - 2^m \cdot 0,110 = 2^m \cdot 0,001$$

Такое нарушение нормализации мы назовем нарушением нормализации *вправо*. Признаком нарушения нормализации вправо для положительного числа является наличие двух нулей в разряде знака и первом разряде после запятой. Признаком нарушения нормализации вправо у отрицательного числа, заданного дополнительным или обратным кодом, является наличие двух единиц в разряде знака и в первом разряде после запятой, вообще, признаком такого нарушения нормализации является совпадение цифр этих двух разрядов; например, $-2^m \cdot 0,0011$ в обратном коде имеет представление $2^m \cdot 1,1100$. Для приведения такого числа к нормализованному виду применяется операция, которую назовем нормализацией влево; эта операция превращает число вида $2^m \cdot 0, \underbrace{00 \dots 0}_k 1 x_1 x_2 \dots$ в равное ему нормализованное число

$2^{m-k} \cdot 0,1 x_1 x_2 \dots \underbrace{00 \dots 0}_k$. Соответственно отрицательное число, представленное в обратном коде в виде $2^m \cdot 1,11 \dots 10 y_1 y_2 \dots$, превращается в равное ему число, представленное в виде $2^{m-k} \cdot 1,0 y_1 y_2 \dots \underbrace{1 \dots 1}_k$. Операция нормализации влево заклю-

чается в последовательном выполнении сдвигов влево цифровой части числа с вычитанием каждый раз 1 из порядка до тех пор, пока число не примет нормализованный вид.

В результате сложения чисел одного знака может получиться число, равное

$$\pm 2^m \cdot 1, x_1 x_2 \dots x_n \quad (1)$$

(сумма цифровых частей превзошла по абсолютной величине 1). Например,

$$2^m \cdot 0,11 + 2^m \cdot 0,10 = 2^m \cdot 1,01$$

В этом случае необходимо произвести нормализацию *вправо*, т. е. заменить число (1) равным ему нормализованным числом $\pm 2^{m+1} \cdot 0,1 x_1 x_2 \dots$; это достигается сдвигом цифровой

части на один разряд вправо и увеличением порядка на 1. Такую операцию назовем операцией нормализации вправо. Подробнее о ней в следующем параграфе.

§ 10. СЛОЖЕНИЕ И ВЫЧИТАНИЕ

Сложение положительных чисел в машинах с фиксированной запятой. Будем рассматривать сложение чисел, заданных в двоичной системе, и начнем со сложения положительных чисел в машинах с фиксированной запятой.

Одноразрядный двоичный сумматор производит сложение по модулю 2 по следующей таблице сложения:

$$0+0=0; 0+1=1; 1+0=1; 1+1=0$$

Многоразрядный сумматор состоит из серии одноразрядных сумматоров, связанных двоичной передачей, складывающих цифры одинакового разряда. При сложении двух единиц $1+1$ образуется импульс двоичной передачи: в следующий разряд передается 1.

Мы рассмотрим следующие способы производства сложения:

1) **Последовательный**, когда сложение $x+y$ производится для разных разрядов последовательно, начиная с младшего. После производства сложения в разряде $(k-1)$ -м с конца переходим к сложению в k -м разряде. Мы складываем при этом фактически 3 цифры — 2 цифры этого разряда обоих слагаемых и цифру 0 или 1 передачи из предыдущего разряда (отсутствие или появление передачи). Это сложение ведется по таблице, где первые два слагаемых суть цифры x_k и y_k данного разряда, а третье — цифра переноса из предыдущего разряда c_k .

Все сложение осуществляется в n тактов, где n — число разрядов.

Этот способ является самым подходящим в случае динамического представления чисел в машине, т. е. когда каждое число представляется цепочкой импульсов, непрерывно циркулирующих в машине все время, пока сохраняется в машине это число. В этом случае суммирующее устройство получается чрезвычайно простым — достаточно иметь сумми-

рующее устройство всего лишь для одного разряда. Подавая одновременно на два входа одноразрядного сумматора последовательного действия два числа x и y поразрядно, начиная с низших разрядов x_n и y_n , на выходе будем получать последовательно все разряды s_k суммы $s = x + y$, начиная с низшего разряда s_n .

Слагаемые		Перенос из низшего раз- ряда c_k	Цифра k -го разряда суммы s_k	Цифра, переда- ваемая в высший разряд суммы d_k
x_k	y_k			
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Фактически, в этом сумматоре осуществляется одновременное сложение трех чисел: x_k , y_k , c_k , так что сумматор последовательного действия должен иметь три входа для величин x_k , y_k и c_k и два выхода: для величин s_k и d_k . Но выход переноса d_k должен быть соединен с входом переноса c_{k+1} , причем это соединение должно быть осуществлено через некоторое устройство, реализующее запаздывание импульса переноса на один такт.

2) Параллельное сложение цифр в каждом разряде с последовательной двоичной передачей в следующие разряды там, где появляются импульсы передачи. Если в результате поразрядного сложения в группе соседних разрядов стоят цифры $\underbrace{011 \dots 1}_k$ и в последний из этих разрядов происходит передача единицы из предыдущего, то она превратит эту цифру 1 в 0; при этом появится передача в следующий разряд и т. д. В результате k последовательных двоичных передач в следующие разряды в этой группе разрядов будут стоять

цифры $1 \underbrace{00 \dots 0}_k$. Наиболее длинная цепь передач имеет

место, если на сумматоре в результате поразрядного сложения появилась серия цифр $0111 \dots 110$, причем из младшего разряда имеет место двоичная передача.

Легко показать, что в данный разряд импульс двоичной передачи может поступать только один раз. В самом деле, пусть складываются числа $0, x_1 x_2 \dots x_n$ и $0, y_1 y_2 \dots y_n$. Так как $0, \underbrace{00 \dots 0}_l x_{l+1} \dots x_n \leq 0, \underbrace{00 \dots 011 \dots 1}_l = 2^{-l} - 2^{-n}$ и

$0, \underbrace{00 \dots 0}_l y_{l+1} \dots y_n \leq 2^{-l} - 2^{-n}$ (части чисел x и y , образо-

ванные последними $(n - l)$ разрядами, не больше $2^{-l} - 2^{-n}$), то сумма этих частей не превосходит $2 \cdot 2^{-l} - 2^{-(n-1)} < 2 \cdot 2^{-l}$, т. е. меньше двух цифр l -го разряда, и в l -й разряд (а также из l -го разряда) может поступать из младших разрядов только одна единица l -го разряда. Более того, если в младший разряд числа поступает дополнительная единица (что имеет место при циклической передаче, см. стр. 49), то и тогда в результате сложения цифр в разрядах после l -го окажется число, не превосходящее $2 \cdot 2^{-l} - 2^{-(n-1)} + 2^{-n} < 2 \cdot 2^{-l}$, т. е. и в этом случае двоичная передача в данный разряд и из данного разряда может иметь место только один раз. Таким образом, *передачи, идущие из младших разрядов, не накапливаются.*

3) Параллельное сложение цифр с одновременной двоичной передачей отличается от предыдущего тем, что если в группе соседних разрядов стоят цифры $0 \underbrace{11 \dots 1}_k$, и в последний из них поступает импульс

двоичной передачи, то он вызывает одновременное превращение этой группы цифр в группу $1 \underbrace{00 \dots 0}_k$.

В силу того, что передача в каждый разряд имеет место только один раз, такой процесс „групповой“ передачи будет иметь место только один раз.

Естественно, эта схема связана с усложнением конструкции сумматора.

Прямое вычитание. Как известно, машины с десятичными колесными счетчиками выполняют как „прямое“, так и „непрямое“ вычитание. Прямое вычитание осуществляется вращением разрядных колес в обратном направлении. Обычные арифмометры являются машинами прямого вычитания.

Непрямое вычитание заключается в том, что вычитание положительного числа заменяется добавлением его дополнения. Оно применяется, например, в автоматических арифмометрах, в табуляторах и т. д.

Дело в том, что отрицательная разность двух положительных чисел задается в машинах прямого вычитания в виде дополнения. Таким образом, прямое вычитание не избавляет от операций с дополнительными числами. В то же время машины непрямого вычитания имеют дело фактически только с операцией сложения, которую проще автоматизировать, чем две различные операции сложения и вычитания при прямом вычитании.

По этой же причине машины, работающие с двоичными числами, являются машинами непрямого вычитания. Для полноты, однако, рассмотрим и прямое вычитание двоичных чисел.

Для целых чисел a и b , $a + b_2^* \equiv a - b \pmod{2}$. Поэтому одноразрядный двоичный сумматор выполняет также функции одноразрядного „вычитателя“. Многоразрядное устройство прямого вычитания должно состоять из одноразрядных двоичных сумматоров, связанных двоичной передачей, причем передача *отрицательной единицы в следующий разряд* происходит при вычитании 0—1 в данном разряде, т. е. при переходе одноразрядного сумматора от позиции 0 к позиции 1. Но 1 и —1 совпадают по модулю 2. Таким образом, при переходе „вычитателя“ от позиции 0 к позиции 1 в данном разряде происходит двоичная передача в следующий разряд (в то время, как при обычном суммировании такая передача происходит при обратном переходе в данном разряде сумматора от позиции 1 к позиции 0).

Сложение чисел произвольного знака в машинах с фиксированной запятой. Рассмотрим для определенности случай запятой, фиксированной перед первой цифрой. Числа x

и y по абсолютной величине меньше единицы, а сумма $x + y$ должна быть по абсолютной величине тоже меньше единицы.

Числа поступают на сумматор дополнительным или обратным кодом. В сумматоре, работающем по дополнительному коду, разряд первой после запятой цифры связан двоичной передачей с разрядом целых единиц — разрядом знака. Из разряда же целых единиц двоичной передачи нет. Такой сумматор осуществляет суммирование по модулю два. Сумма дополнительных кодов $[x]_2$ и $[y]_2$, полученная на таком сумматоре, совпадает с дополнительным кодом $[x + y]_2$ суммы $x + y$ (если $|x + y| < 1$). В самом деле, в силу (8) § 7 $[x + y]_2 = [x]_2 + [y]_2$.

$$\begin{array}{rcl} \text{Пример 1. } x = [x]_2 = 0,101000; & & y = [y]_2 = 0,010100 \\ & + & \\ & 0,101000 & \\ & 0,010100 & \\ \hline & 0,111100 & \end{array} \quad \begin{array}{l} x + y = [x + y]_2 = 0,111100 \end{array}$$

$$\begin{array}{rcl} \text{Пример 2. } x = [x]_2 = 0,110111; & & y = 0,010010 \\ x > |y|; \quad x + y > 0; & & [y]_2 = 2 + y = 1,101110 \\ & + & \\ & 0,110111 & \\ & 1,101110 & \\ \hline & 0,100101 & \end{array}$$

Так как разряд целых двоек отсутствовал, то на сумматоре остается число $0,100101 \equiv [x]_2 + [y]_2 \pmod{2}$. Это есть дополнительный код $[x + y]_2$ равной ему суммы $x + y$.

$$\begin{array}{rcl} \text{Пример 3. } x = [x]_2 = 0,010010; & & y = -0,110111 \\ (x < |y|, x + y < 0) & & [y]_2 = 2 + y = 1,001001 \\ & + & \\ & 0,010010 & \\ & 1,001001 & \\ \hline & 1,011011 & \\ [x + y]_2 = 1,011011; & & x + y = -0,100101 \end{array}$$

$$\begin{array}{rcl} \text{Пример 4. } x = -0,100010; & & y = -0,010011 \\ [x]_2 = 2 + x = 1,011110; & & [y]_2 = 2 + y = 1,101101 \\ & + & \\ & 1,011110 & \\ & 1,101101 & \\ \hline & 1,001011 & \end{array}$$

На сумматоре получаем дополнительный код $[x + y]_2$ суммы $x + y$; $x + y = -0,110101$.

Сумматор, работающий с числами, заданными обратным кодом, отличается от сумматора, работающего дополнительным кодом, наличием циклической двоичной передачи, т. е. передачи из разряда знаков (целых единиц) в младший цифровой разряд. Такой сумматор осуществляет сложение по модулю $2 - 2^{-n}$. Поэтому (см. (12) § 7), на сумматоре при сложении обратных кодов $[x]_3$ и $[y]_3$ слагаемых $x + y$ получится обратный код $[x + y]_3$ суммы $x + y$.

Сложение положительных чисел (пример 1) для обоих кодов одинаково. Проведем сложения в примерах 2, 3, 4, где фигурируют отрицательные слагаемые при обратном коде.

$$\text{Пример 2'. } x = [x]_3 = 0,110111; \quad y = -0,010010 \\ [y]_3 = 1,101101$$

$$\begin{array}{r} + \quad 0,110111 \\ \quad 1,101101 \\ \hline 10,100100 \\ \quad \quad \uparrow \text{ Циклическая передача} \\ \hline 0,100101 \end{array}$$

На сумматоре стоит обратный код $[x + y]_3$ суммы $x + y = 0,100101$.

$$\text{Пример 3'. } x = [x]_3 = 0,010010; \quad y = -0,110111 \\ [y]_3 = 1,001000$$

$$\begin{array}{r} + \quad 0,010010 \\ \quad 1,001000 \\ \hline 1,011010 \\ [x + y]_3 = 1,011010; \quad x + y = -0,100101 \end{array}$$

$$\text{Пример 4'. } x = -0,100010; \quad y = -0,010011 \\ [x]_3 = 1,011101; \quad [y]_3 = 1,101100$$

$$\begin{array}{r} + \quad 1,011101 \\ \quad 1,101100 \\ \hline 11,001001 \\ \quad \quad \uparrow \text{ Циклическая передача} \\ \hline 1,001010 \\ [x + y]_3 = 1,001010; \quad x + y = -0,110101 \end{array}$$

Указатель запрета. При сложении двух положительных или двух отрицательных чисел может получиться число по

абсолютной величине большее единицы. На такие результаты наложен запрет.

Если $x > 0$, $y > 0$, $x + y > 1$, то при сложении на сумматоре в разряде знаков появится 1.

Пример 5. При $x = 0,101101$, $y = 0,110110$

$$\begin{array}{r} 0,101101 \\ + 0,110110 \\ \hline 1,100011 \end{array}$$

На сумматоре получим 1,100011.

Рассмотрим теперь случай, когда $x < 0$, $y < 0$ и $|x + y| > 1$; так как $x > -1$, $y > -1$, то $-2 < x + y < -1$. При дополнительном коде $[x]_2 = 2 + x$, $[y]_2 = 2 + y$, $[x]_2 + [y]_2 = 4 + (x + y)$, значит $2 < [x]_2 + [y]_2 < 3$. Целая часть $[x]_2 + [y]_2$ равна 2; в двоичной системе она равна 10. Так как разряд целых двоек в сумматоре отсутствует, то в разряде целых единиц (разряде знаков) при сложении кодов $[x]_2$ и $[y]_2$ получится 0.

Пример 6. При $x = -0,101101$, $y = -0,110110$

$$[x]_2 = 1,010011; [y]_2 = 1,001010$$

$$\begin{array}{r} 1,010011 \\ + 1,001010 \\ \hline 0,011101 \end{array}$$

На сумматоре получим 0,011101. Появление в разряде знаков цифры 1 при сложении положительных чисел или цифры 0 при сложении отрицательных чисел является указателем того, что сумма превзошла по абсолютной величине 1, т. е. что мы имеем случай запрета. Такой указатель требует сохранения знаков слагаемых в сумматоре для сравнения их со знаком суммы. Удобнее другой указатель, который мы сейчас приведем.

Рассмотрим сначала случай дополнительного кода. На сумматоре вводится дополнительный разряд знака. Положительные числа представляются в виде $00, x_1 x_2 \dots x_n$; отрицательные — в виде $11, y_1 y_2 \dots y_n$ (дополнительный разряд есть

разряд целых двоек). Такое модифицированное представление числа x дополнительным кодом обозначим $[\bar{x}]_2$. Очевидно, при $x > 0$, $[\bar{x}]_2 = [x]_2 = x$; при $x < 0$, $[\bar{x}]_2 = 2 + [x]_2 = = 4 + x$ (напомним, что $[\bar{x}]_2$ отличается от $[x]_2$ наличием цифры 1 в разряде двоек). Вообще $[\bar{x}] \equiv x \pmod{4}$. Далее $[\bar{x} + y]_2 = [\bar{x}]_2 + [\bar{y}]_2 \pmod{4}$. Сложение по модулю 4 есть обычное сложение, если все разряды сумматора, включая оба старших разряда (разряды знака), связаны между собой двоичной передачей, а из старшего разряда (целых двоек) нет двоичной передачи. На таком сумматоре складываются модифицированные дополнительные коды слагаемых x и y , и если $|x + y| < 1$, то на сумматоре появится $[\bar{x} + y]_2$, т. е. модифицированный дополнительный код суммы $(x + y)$.

Целая часть представления любого числа z , $|z| < 1$, есть 00 или 11 ($= 3$), в зависимости от знака z . Если при сложении кодов $[\bar{x}]_2$ и $[\bar{y}]_2$ на сумматоре целая часть оказывается равной 01 или 10, то это означает, что $|x + y| > 1$ и сложение незаконно.

Пример 7. Пусть $x = -0,110011$, $y = -0,101001$

$$\begin{array}{r} 11,001101 \text{ (изображение числа } x) \\ + 11,010111 \text{ (" " } y) \\ \hline 10,100100 \end{array}$$

В разрядах знаков стоит пара неравных цифр 10.

Итак, указателем запрета является появление в разрядах знаков сумматора пары неравных цифр 01 или 10.

Случай *обратного кода* совершенно аналогичен предыдущему. И здесь можно ввести дополнительный разряд знаков, так что положительные числа x имеют модифицированное обратное представление $[\bar{x}]_3 = x$, а отрицательные числа — представление $[\bar{x}]_3 = [x]_3 + 2 = (4 - 2^{-n}) + x$.

Изображение отрицательного числа обратным кодом попрежнему на 2^{-n} , т. е. на единицу последнего разряда, меньше его дополнительного изображения.

Вообще, $[\bar{x}]_3 \equiv x \pmod{4 - 2^{-n}}$.

При модифицированном обратном коде также необходима на сумматоре циклическая двоичная передача: старший разряд (на этот раз разряд дополнительного знака, т. е. разряд целых двоек) связан двоичной передачей с самым младшим разрядом. Это отвечает сравнению $4 \equiv 2^{-n} \pmod{(4-2^{-n})}$.

Как и в случае дополнительного кода появление на сумматоре в разрядах знаков пары разных цифр 01 или 10 есть указатель запрета.

Пример 7'. Вернемся к примеру (7) сложения чисел $-0,110011$ и $-0,101001$, проводя его в модифицированном обратном коде.

$$\begin{array}{r} 11,001100 \text{ (изображение числа } x) \\ + 11,010110 \text{ (" " } y) \\ \hline 110,100010 \text{ циклическая передача} \\ \hline \end{array}$$

На сумматоре оказывается число 10,100011. Наличие пары цифр 10 указывает на запрет.

Вычитание. Операция вычитания $x - y$ сводится к изменению знака числа y и к сложению $x + (-y)$. Для чисел, заданных обратным кодом, вычитание $x - y$ сводится к сложению кода числа x с инвертированным кодом числа y ; для этой цели внутри арифметического устройства предусматривается как прямая, так и инверсная передача кода числа.

Пример 8. $x = 0,1010$; $y = 0,1101$

$$\begin{array}{r} \text{Сумматор} \\ 0,1010 \text{ на сумматор подается } [x]_3 \\ + 1,0010 \text{ на сумматор инверсно подается } [y]_3 \\ \hline 1,1100 = [x - y]_3 \\ x - y = -0,0011 \end{array}$$

При вычитании чисел, заданных дополнительным кодом, нужно не только код вычитаемого подать инверсной передачей, но и послать в сумматор единицу младшего разряда.

Пример 8'. $x = 0,1010$; $y = 0,1101$

$$\begin{array}{r} \text{Сумматор} \\ 0,1010 \text{ В сумматор подается } [x]_2 \\ + 1,0010 \text{ В сумматор инверсно подается } [y]_2 \\ \hline 1,1100 \text{ Добавление 1 в младший разряд} \\ 1,1101 = [x - y]_2 \\ x - y = -0,0011 \end{array}$$

Сложение и вычитание чисел в машинах с плавающей запятой. Для чисел с равными порядками $A = 2^p x$ и $B = 2^p y$, где $|x| < 1$, $|y| < 1$, сложение сводится к сложению их цифровых частей x и y : $A + B = 2^p(x + y)$. Причем, поскольку цифровые части по модулю меньше 1, сложение их производится так же, как и в машинах с запятой, фиксированной перед первым знаком. Однако в случае, когда $|x + y| > 1$, т. е. в случае, когда в машинах с фиксированной запятой имеется запрет, в машинах с плавающей запятой появляется нарушение нормализации влево. Если же $|x + y| < 1/2$, результат получается с нарушением нормализации вправо. Таким образом, после операции сложения требуется дополнительная операция нормализации. Так как указатель необходимости нормализации вправо совпадает здесь с указателем запрета в машинах с фиксированной запятой, то и здесь удобно пользоваться модифицированным обратным или дополнительным кодом, т. е. кодом с двумя разрядами знака. Появление пары разных цифр 01 или 10 в разрядах знака (см. стр. 51) является в машинах с плавающей запятой с таким кодом указателем необходимости нормализации вправо. Вычитание $x - y$ попрежнему сводится к сложению $x + (-y)$.

Пример 9. Сложить $2^4 \cdot 0,110011$ и $2^4 \cdot 0,101001$.

При сложении цифровых частей получаем:

$$\begin{array}{r} 00,110011 \\ + 00,101001 \\ \hline 01,011100 \end{array}$$

Наличие пары цифр 01 в разрядах знака есть указатель нарушения нормализации влево. Сдвиг на 1 разряд вправо даст на сумматоре число 00,101110. Сумма равна $2^5 \cdot 0,101110$.

Пример 10. Сложить числа: $-2^4 \cdot 0,110011$ и $-2^4 \cdot 0,101001$.

Воспользуемся сначала модифицированным дополнительным кодом. На сумматоре имеем:

$$\begin{array}{r} 11,001101 \\ + 11,010111 \\ \hline 10,100100 \end{array}$$

Появление пары цифр 10 в разряде знаков есть указатель нарушения нормализации влево. Сдвинем число, стоящее на сумматоре, на 1 разряд вправо (сохранив 1 в первой цифре знака), получим на сумматоре 11,010010. Это есть изображение числа $-0,101110$. Сумма равна $-2^5 \cdot 0,101110$.

При обратном коде имеем на сумматоре:

$$\begin{array}{r}
 11,001100 \\
 + 11,010110 \\
 \hline
 110,100010 \\
 \hline
 10,100011 \uparrow
 \end{array}$$

После нормализации получим на сумматоре 11,010001. Это есть изображение числа $-0,101110$. Сумма равна $-2^5 \cdot 0,101110$.

При сложении чисел противоположного знака может получиться нарушение нормализации вправо, и, следовательно, необходимость нормализации влево. Указателем необходимости такой нормализации для чисел, заданных обратным или дополнительным кодом, обычным или модифицированным, является наличие пары одинаковых цифр 00 или 11 в разряде знака и первом разряде после запятой. Каждый этап нормализации влево заключается в сдвиге и в добавлении к порядку (-1) . Приведем пример, предполагая, что порядки изображены обратным кодом, цифровые части — модифицированным обратным кодом.

Пример 11. Найти $A + B$, где $A = 2^5 \cdot 0,110011$, $B = 2^5 \cdot (-0,100100)$

	Порядок	Цифровая часть	
	0101	00,110011	
	0101	+ 11,011011	
		<u>100,001110</u>	Циклическая передача
		<u>100,001110</u>	\uparrow
Добавление (-1) к порядку	+ 0101	00,001111	
	1110		Сдвиг влево на 1 разряд
Циклическая передача	<u>10011</u>		\uparrow
Добавление (-1) к порядку	+ 0100	00,011110	
	1110		Сдвиг влево на 1 разряд
Циклическая передача	<u>10010</u>		\uparrow
	0011	00,111100	

Порядок суммы + 11(3), цифровая часть + 0,111100

$$A + B = 2^3 \cdot 0,1111$$

Пример 12. $A = 2^{-2} \cdot (-0,110011)$, $B = 2^{-2} \cdot 0,100100$

	Порядок	Цифровая часть	
	1101	11,001100	
	1101	+ 00,100100	
Добавление (-1)	1101	11,110000	
к порядку	+ 1110		Сдвиг влево на 1 разряд
Циклическая пе-	11011		
редача	↑		
Добавление (-1)	1100	11,100001	
к порядку	+ 1110		Сдвиг влево на 1 разряд
Циклическая пе-	11010		
редача	↑		
	1011	11,000011	

Порядок суммы $-100(-4)$, цифровая часть $-0,111100$

Если порядки слагаемых $A = 2^p x$, $B = 2^q y$ не равны между собой ($p \neq q$), то прежде всего порядки надо *выровнять*. Пусть, например, $p > q$, т. е. $p = q + k$, где $k > 0$. Тогда слагаемое B заменяется ненормализованным числом $B_1 = 2^p \cdot y_1$, где y_1 образовано сдвигом числа y на k разрядов вправо. B_1 с точностью до 2^{p-n} совпадает с числом B . После этого сложение чисел A и B_1 проходит как описанное выше сложение чисел с равными порядками.

Пример 13. Сложим числа $A = 2^5 \cdot 0,100111$ и $B = 2^3 \cdot 0,100110$. Разность порядков здесь равна 2, и число B имеет меньший порядок. Поэтому $B_1 = 2^5 \cdot 0,001001$ и сумма $A + B$, получаемая в машине, равна $2^5 \cdot 0,110000$.

§ 11. УМНОЖЕНИЕ И ДЕЛЕНИЕ ЧИСЕЛ, ЗАДАННЫХ ПРЯМЫМ КОДОМ, И ИЗВЛЕЧЕНИЕ КВАДРАТНОГО КОРНЯ

Умножение при фиксированной запятой. Рассмотрим сначала умножение чисел в машинах с фиксированной запятой. Ограничимся для простоты случаем, когда запятая фиксирована перед первой цифрой.

Рассмотрим сначала умножение чисел, представленных прямым кодом. Для множимого x и множителя y рассмотрим представления

$$[x]_1 = x_0, x_1 x_2 \dots x_n, [y]_1 = y_0, y_1 y_2 \dots y_n$$

Здесь $x_0 = \bar{1}(x)$, $y_0 = \bar{1}(y)$ — знаки чисел x и y . Знак произведения xy определяется по формуле: $\bar{1}(xy) \equiv \bar{1}(x) + \bar{1}(y) \pmod{2}$, т. е. получается сложением по модулю 2 двух знаков сомножителей. Для получения знака произведения достаточно иметь одноразрядный двоичный сумматор, осуществляющий такое сложение.

Правило умножения цифровых частей в любой системе счисления аналогично обычному правилу умножения в десятичной системе: множимое $A = 0, x_1 x_2 \dots x_n$, соответственным образом сдвинутое, последовательно умножается на все цифры множителя $B = 0, y_1 y_2 \dots y_n$ с посылкой произведений в сумматор. Особенно просто производство умножения в двоичной системе: каждая цифра множителя может принимать лишь два значения 1 и 0; если k -я цифра после запятой $y_k = 1$, то множимое, сдвинутое на k разрядов вправо, посылается в сумматор, если же $y_k = 0$, то такой посылки не производится.

Мы должны иметь регистр (хранитель) множимого, откуда числа могут передаваться на сумматор, регистр множителя и сумматор, он же — регистр произведения. Как мы увидим ниже, регистр множителя может быть соединен с сумматором в некоторый общий регистр.

Можно производить умножения множимого на все цифры множителя y_1, y_2, \dots, y_n , начиная с цифры его старшего разряда y_1 , каждый раз сдвигая множимое на 1 разряд *вправо*; можно производить умножения множимого на цифры множителя $y_n, y_{n-1}, \dots, y_2, y_1$, начиная с его последней n -й цифры y_n , каждый раз сдвигая множимое на один разряд *влево*. Приведем пример умножения двух двоичных чисел обоими способами.

1-й способ

$$\begin{array}{r}
 \times 0,110101 \\
 0,111010 \\
 \hline
 11010 \mid 1 \\
 1101 \mid 01 \\
 110 \mid 101 \\
 \text{Сдвиг} \quad 1 \mid 10101 \\
 \text{Сдвиг} \quad \hline
 0,110000 \mid 000010
 \end{array}$$

$$\begin{array}{r}
 \times 0,110101 \\
 0,111010 \\
 \hline
 1 \mid 10101 \text{ Сдвиг} \\
 110 \mid 101 \text{ Сдвиг} \\
 1101 \mid 01 \\
 11010 \mid 1 \\
 \hline
 0,110000 \mid 000010
 \end{array}$$

При умножении по первому способу на сумматоре должны складываться все $2n$ разрядов, входящих в произведение. Умножение по второму способу, как будет указано ниже, можно осуществлять так, чтобы в сложении участвовали лишь первые n разрядов регистра произведения, а остальные его n разрядов, не участвуя в сложении, выполняли лишь функции передачи цифр в следующий разряд при соответствующем сдвиге вправо. Поэтому, при таком способе умножения разрядами сумматора должны быть лишь первые n разрядов регистра произведения; если последние n разрядов регистра произведения отсутствуют, то умножение по второму способу даст первые n цифр произведения.

Остановимся подробнее на таком осуществлении второго способа. Выше мы все время сдвигали множимое влево и сдвинутое множимое последовательно прибавляли к стоящему на сумматоре числу. Можно оставить множимое неподвижным, но зато каждый раз на один разряд вправо сдвигать стоящее в регистре произведения число: последовательно рассматривая каждую цифру множителя, начиная с последней, $y_n, y_{n-1}, \dots, y_2, y_1$, мы каждый раз или добавляем множимое к числу, стоящему в этом регистре, если $y_k = 1$ ($k = n, n-1, \dots, 1$), или оставляем число неизменным, если $y_k = 0$, после чего сдвигаем число в регистре на один разряд вправо. Таким образом, лишь первые n разрядов, а также стоящий впереди дополнительный разряд выполняют суммирующие функции.

Возьмем к последнему примеру умножения множимого $A = 0,110101$ и множителя $B = 0,101101$.

С х е м а 1 (умножение)

Цифры множителя (начиная с последней)	Сумматор	Регистр произведения
1	+ 0,000000 0,110101	Передача множимого на сумматор
	0,110101 0,011010	Сдвиг вправо на 1 разряд
0	0,011010	Множимое на сумматор не передается Сдвиг вправо на 1 разряд
	+ 0,001101 0,110101	Передача множимого на сумматор
1	1,000010 0,100001	Сдвиг вправо на 1 разряд Передача множимого на сумматор
	+ 0,110101 1,010110	Сдвиг вправо на 1 разряд
1	0,101011 0,101011	Сдвиг вправо на 1 разряд
0	0,101011 0,010101	Сдвиг вправо на 1 разряд Передача множимого на сумматор
	+ 0,110101 1,001010	Сдвиг вправо на 1 разряд
1	0,100101	Результат

В процессе умножения цифры $y_n, y_{n-1}, \dots, y_2, y_1$ множителя выполняют последовательно функции управления (множимое посылается или не посылается в сумматор в зависимости от того, равна ли эта цифра 1 или 0). Для упрощения процесса управления можно каждый раз после посылки множимого в сумматор сдвигать множитель на один разряд вправо, в результате чего в n -м разряде регистра множителя будут стоять последовательно цифры $y_n, y_{n-1}, \dots, y_2, y_1$; таким образом, упомянутые функции управления выполняет каждый раз цифра, стоящая в n -м (последнем) разряде регистра множителя.

Напомним, что в регистре произведения последние n разрядов в начале процесса умножения свободны и занимаются последовательно, начиная с $(n+1)$ -го разряда в результате сдвигов вправо из первых (суммирующих) разрядов этого регистра. В то же время в регистре множителя, в результате его сдвигов, последовательно освобождаются разряды, начиная с первого. Это позволяет обойтись без специального регистра множителя, передав функции хране-

ния множителя последним n разрядам регистра произведения. При этом функции управления процессом умножения несет последняя $2n$ -я цифра регистра произведения.

Рассмотренное только что умножение чисел $A = 0,110101$ и $B = 0,101101$ будет производиться по следующей схеме.

С х е м а 2 (умножение)

Множимое 0,110101	Множитель 0,101101
Регистр произведения	
Сумматор	Множи- тель
+ 0,000000 101101	Передача множимого в сумматор
+ 0,110101	
0,110101 101101	Сдвиг вправо на 1 разряд
0,0110101 10110	Множимое на сумматор не передается
	Сдвиг вправо на 1 разряд
+ 0,00110101 1011	Передача множимого в сумматор
+ 0,110101	
1,00001001 1011	Сдвиг вправо на 1 разряд
+ 0,100001001 101	Передача множимого в сумматор
+ 0,110101	
1,010110001 101	Сдвиг вправо на 1 разряд
0,1010110001 10	Множимое в сумматор не передается
	Сдвиг вправо на 1 разряд
+ 0,01010110001 1	Передача множимого в сумматор
+ 0,110101	
1,00101010001 1	Сдвиг вправо на 1 разряд
0,100101010001	Результат

Умножение чисел в машинах с плавающей запятой.

Пусть нужно умножить числа $A = \pm 2^p \cdot 0, a_1 a_2 \dots a_n$ и $B = \pm 2^q \cdot 0, b_1 b_2 \dots b_n$. Умножение заключается:

- 1) в сложении знаков сомножителей;
- 2) в сложении их порядков;
- 3) в умножении цифровых частей.

Последнее умножение ничем не отличается от умножения в машинах с фиксированной запятой.

В результате умножения двух нормализованных чисел может получиться число ненормализованное, которое потом нужно нормализовать, например, $2^5 \cdot 0,1001 \times 2^{-2} \cdot 0,1001 = 2^3 \cdot 0,01010001 = 2^2 \cdot 0,1010001$.

Частные случаи умножения. а. Если один из сомножителей есть нуль, то произведение есть нуль. Машина, обладающая способностью реагировать на равенство одного из сомножителей нулю, сразу выдает в качестве произведения нуль.

б. Умножение на 2^s (s — целое положительное число) сводится в машинах с фиксированной запятой к сдвигу множимого на s разрядов соответственно влево или вправо, а в машинах с плавающей запятой — прибавлению к порядку множимого числа s .

Деление в машинах с фиксированной запятой. Рассмотрим сначала случай деления в машинах с запятой, фиксированной перед первой цифрой числа. В таких машинах делимое должно быть меньше делителя для того, чтобы целая часть была нуль.

Пусть делимое и делитель представлены прямым кодом. Знак частного получается сложением на одноразрядном сумматоре знаков делимого и делителя. В дальнейшем, говоря о делимом и делителе, мы будем иметь в виду их абсолютные величины.

Операция деления в двоичной системе совершается, в основном, по схеме А (см. стр. 9) с тем упрощающим обстоятельством, что каждая цифра частного может принимать лишь значения 0 и 1: при делении A на b образуются последовательные остатки $A_0 = A, A_1, A_2, \dots, A_n$.

Для определения k -й после запятой цифры d_k частного из k -го остатка A_k вычитают сдвинутый на k цифр вправо делитель b , т. е. $2^{-k}b$. Если разность положительна, то цифра $d_k = 1$. Эта разность есть $(k+1)$ -й остаток.

Если же разность $A_k - 2^{-k}b$ отрицательная, то цифра $d_k = 0$ и $(k+1)$ -й остаток A_{k+1} совпадает с A_k ; $A_{k+1} = A_k$.

Для его восстановления нужно к разности $A_k - 2^{-k}b$ снова добавить сдвинутый делитель $2^{-k}b$.

После этого мы сдвигаем делитель еще на один разряд вправо. Этот цикл операций повторяем n раз. Если сдвигать делитель каждый раз на один разряд вправо, то его последние цифры будут последовательно выходить за пределы

рабочей части сумматора. Такое деление есть деление по правилу „сокращенных действий“.

Вместо сдвига делителя на разряд вправо можно производить каждый раз сдвиг соответственного остатка на один разряд влево. Тогда для определения каждой цифры используются все цифры делителя. Это есть деление по правилу *точного* деления или деления с остатком. После вычисления последней цифры d_n частного мы получим остаток \bar{A}_n , который и будет сдвинутый на n разрядов влево точный остаток A_n от деления A на b : $A_n = 2^{-n}\bar{A}_n$

$$A = bd + A_n = bd + 2^{-n}\bar{A}_n$$

Произведение bd берется здесь с полным числом $2n$ знаков.

Остановимся на этом приеме деления, отвечающем умножению по схеме 1, стр. 58; мы имеем регистр произведения, состоящий из разрядов знаков и $2n$ цифровых разрядов; в этот регистр посылается делимое (точнее его абсолютная величина). Последние n разрядов регистра произведения пока свободны — в них будут последовательно появляться цифры частного¹.

Делимое $A = \bar{A}_0$ меньше делителя b . Сдвигаем A на 1 разряд влево, из полученного числа $2A_0$ вычитаем делитель b , посылая на сумматор — b , или соответственно обратный код числа b . Если разность $2A_0 - b > 0$, то первая цифра частного $d_1 = 1$. Если же эта разность $2A_0 - b$ отрицательна, то цифра частного $d_1 = 0$, вычитание b было лишним, и мы его компенсируем, вводя в сумматор делитель b . В обоих случаях набор, стоящий в регистре произведения, сдвигается на 1 разряд влево и в последний освободившийся разряд регистра посылается цифра d_1 . Такой цикл повторяется n раз. После n -го цикла в последних n разрядах регистра произведения будут стоять цифры $d_1 d_2 \dots d_n$ частного, а на сумматоре (левой половине регистра) — умноженный на 2^n остаток от деления.

¹ Эти же свободные пока разряды могли бы быть использованы в случае $2n$ -значного делимого для его дополнительных цифр.

С х е м а 3 (деление)

Делимое $x = 0,100111$ Делитель $y = 0,110001$

Сумматор	Регистр частного	
00,100111	000000	Сдвиг влево на 1 разряд
+ 01,001110	00000—	На сумматор посылается $[-y]_2$
<u>11,001111</u>		
00,011101	000001	Сдвиг влево на 1 разряд
+ 00,111010	00001—	На сумматор посылается $[-y]_2$
<u>11,001111</u>		
00,001001	000011	Сдвиг влево на 1 разряд
00,010010	00011—	На сумматор посылается $[-y]_2$
+ 11,001111		
<u>11,100001</u>	000110	На сумматор посылается код делителя
+ 00,110001		
00,010010		Сдвиг влево на 1 разряд
00,100100	00110—	На сумматор посылается $[-y]_2$
+ 11,001111		
<u>11,110011</u>	001100	На сумматор посылается код делителя
+ 00,110001		
00,100100		Сдвиг влево на 1 разряд
01,001000	01100—	На сумматор посылается $[-y]_2$
+ 11,001111		
<u>00,010111</u>	011001	Сдвиг влево на один разряд
00,101110	11001—	На сумматор посылается $[-y]_2$
+ 11,001111		
<u>11,111101</u>	110010	На сумматор посылается код делителя
+ 00,110001		
<u>00,101110</u>		

$$x/y = 0,110010$$

$$\text{Остаток} = 2^{-6} \cdot 0,101110$$

При описанном способе деления для определения одной цифры частного, когда эта цифра равна нулю, приходится употреблять две операции — вычитания и сложения. Укажем другой вариант процесса деления, требующий лишь одного добавления сложения или вычитания для определения каждой цифры частного.

В этом способе допускаются как положительные, так и отрицательные остатки. Если очередной k -й остаток положителен, то следующий цикл заключается в вычитании делителя из сдвинутого на один разряд влево остатка, если

же очередной k -й остаток отрицателен, то следующий цикл состоит не в вычитании, а в добавлении делителя к сдвинутому на один разряд влево остатку.

Если в результате цикла возникает положительный остаток, то определяемая циклом цифра есть 1, если же возникает отрицательный остаток, то определяемая циклом цифра есть 0.

Для обоснования этого способа воспользуемся двоичной системой с цифрами 1 и $\bar{1} = -1$ (см. ниже § 13). Если в такой системе $(k+1)$ -я цифра положительного числа ($k > 0$) есть $\bar{1}$, соответственно 1, то в обычном двоичном представлении k -я цифра есть 0 или, соответственно, 1. Если пользоваться для изображения частного цифрами 1 и $\bar{1}$, то в описанном выше способе деления вычитание делителя из сдвинутого k -го остатка означает, что $(k+1)$ -я цифра частного есть 1, а прибавление делителя к этому остатку означает, что эта цифра есть $\bar{1}$. Согласно правилу перехода от такого двоичного изображения к обычному k -я цифра частного равна 1, в случае положительного k -го остатка, и равна нулю, в случае отрицательного k -го остатка. Если после определения n цифр на сумматоре оказывается положительное число, то оно совпадает с умноженным на 2^n остатком от деления. Если же число на сумматоре оказывается отрицательным, то для определения умноженного на 2^n остатка от деления необходимо послать в сумматор код делителя.

С х е м а 4 (деление)

		Делимое $x = 0,100111$	Делитель $y = 0,110001$
Сумматор	Регистр частного		
00,100111	000000	Сдвиг влево на 1 разряд	
+ 01,001110	000000—	На сумматор посылается $[-y]_2$	
<u>11,001111</u>			
00,011101	000001	Сдвиг влево на 1 разряд	
+ 00,111010	00001—	На сумматор посылается $[-y]_2$	
<u>11,001111</u>			
00,001001	000011	Сдвиг влево на 1 разряд	
+ 00,010010	00011—	На сумматор посылается $[-y]_2$	
<u>11,001111</u>			
11,100001	000110	Сдвиг влево на 1 разряд	

+	11,000010	00110—	На сумматор посылается код делителя
	00,110001		
	11,110011	001100	Сдвиг влево на 1 разряд
+	11,100110	01100—	На сумматор посылается код делителя
	00,110001		
	00,010111	011001	Сдвиг влево на 1 разряд
+	00,101110	11001—	На сумматор посылается $[-y]_2$
	11,001111		
+	11,111101	110010	На сумматор посылается код делителя (восстанов-
	00,110001		ление остатка)
	00,101110		

$$x/y = 0,110010$$

$$\text{Остаток} = 2^{-6} \cdot 0,101110$$

В процессе деления критерием для определения следующей цифры частного служит цифра знака числа, оказавшегося на сумматоре после очередного вычитания (или быть может прибавления во втором способе) делимого. Очевидно, что 0 следовало бы отнести к числам с цифрой знака 0. Если сумматор работает по принципу дополнительного кода, то цифра знака нуля есть всегда 0 и при делении не возникает никаких затруднений. Однако, если сумматор работает по принципу обратного кода, то число ноль появляется в сумматоре после сложения всегда как „—0“ (см. § 7). Поэтому при нашей схеме деления в частном вместо цифры 1 в этом случае появится цифра 0. Все цифры, следующие за этой цифрой, будут единицами, так как все остатки после сдвига на 1 разряд влево будут равны удвоенному делителю. Таким образом, вместо частного $0, z, \dots z_{k-1} \underbrace{10 \dots 0}_{n-k}$ и остатка, равного нулю, мы получим частное $0, z, \dots z_{k-1} \underbrace{01 \dots 1}_{n-k}$ и остаток, равный умноженному

на 2^{-n} делителю. Если деление проводить неограниченно, то оба результата совпадают. Каждый из них соответствует двум разным представлениям двоично-рационального числа в виде бесконечной двоичной дроби (с „хвостом“ нулей и с „хвостом“ единиц). Оба результата совпадают также, если деление проводится с округлением без вывода остатка.

Для того, чтобы было возможным и точное деление при сумматоре обратного кода, достаточно в описанные выше способы деления внести следующие изменения:

а) делимое посылать на сумматор инверсной передачей;
 б) все прибавления делителя заменить вычитаниями, а вычитания — прибавлениями, т. е. инвертировать все передачи делителя на сумматор;

в) в последний разряд регистра частного посылать цифру, равную цифре знака числа, оказавшегося на сумматоре, т. е. инвертировать правило посылки очередной цифры частного.

После окончания процесса деления стоящее на сумматоре число будет остатком, умноженным на 2^n . Правило определения остатка при делении по второму способу соответствующим образом видоизменяется.

Деление будет проводиться по следующим схемам.

С х е м а 5 (деление)

Делимое $x = 0,100111$ Делитель $y = 0,110001$

Сумматор	Регистр частного	
11,011000	000000	Сдвиг влево на 1 разряд
+ 10,110001	00000—	На сумматор посылается код делителя
00,110001		
11,100010	000001	Сдвиг влево на 1 разряд
+ 11,000101	00001—	На сумматор посылается код делителя
00,110001		
11,110110	000011	Сдвиг влево на 1 разряд
+ 11,101101	00011—	На сумматор посылается код делителя
00,110001		
100,011110		
↑		
+ 00,011111	000110	На сумматор посылается код делителя инверсной передачей
11,001110		
11,101101	000110	Сдвиг влево на 1 разряд
+ 11,011011	00110—	На сумматор посылается код делителя
00,110001		
100,001100		
↑		
+ 00,001101	001100	На сумматор посылается код делителя инверсной передачей
11,001110		
11,011011	001100	Сдвиг влево на 1 разряд

+ 10,110111	01100—	На сумматор посылается код делителя
00,110001		
11,101000	011001	Сдвиг влево на 1 разряд
11,010001	11001—	На сумматор посылается код делителя
+ 00,110001		
100,000010		
↑		
+ 00,000011	110010	На сумматор посылается код делителя инверсной передачей
11 001110		
11,010001		

$$x/y = 0,110010$$

$$\text{Остаток} = 2^{-6} \cdot 0,101110$$

С х е м а 6 (деление)

Делимое $x = 0,100111$ Делитель $y = 0,110001$

Сумматор	Регистр частного	
11,011000	000000	Сдвиг влево на 1 разряд
+ 10,110001	00000—	На сумматор посылается код делителя
+ 00,110001		
11,100010	000001	Сдвиг влево на 1 разряд
+ 11,000101	00001—	На сумматор посылается код делителя
+ 00,110001		
11,110110	000011	Сдвиг влево на 1 разряд
+ 11,101101	00011—	На сумматор посылается код делителя
+ 00,110001		
100,011110		
↑		
00,011111	000110	Сдвиг влево на 1 разряд
+ 00,111110	00110—	На сумматор посылается код делителя инверсной передачей
+ 11,001110		
100,001100		
↑		
00,001101	001100	Сдвиг влево на 1 разряд
+ 00,011010	01100—	На сумматор посылается код делителя инверсной передачей
+ 11,001110		
11,101000	011001	Сдвиг влево на 1 разряд
+ 11,010001	11001—	На сумматор посылается код делителя
+ 00,110001		
100,000010		
↑		
+ 00,000011	110010	

$$\begin{array}{r} + \quad 11,001110 \\ 11,010001 \end{array}$$

На сумматор посылается код делителя инверсной передачей (восстановление остатка)

$$x/y = 0,110010$$

$$\text{Остаток} = 2^{-6} \cdot 0,101110$$

Деление в машинах с плавающей запятой. Пусть делимое A и делитель B заданы в нормализованном виде:

$$A = 2^p \cdot \pm 0,1a_1 \dots a_n; \quad B = 2^q \cdot \pm 0,1b_1 b_2 \dots b_n$$

Тогда $A : B = 2^{p-q} C$, где C — частное от деления цифровых частей A и B , $C = (\pm 0,1a_1 \dots a_n) : (\pm 0,1b_1 \dots b_n)$. Это деление проводится, в основном, как в машинах с запятой, фиксированной перед первым знаком, с той лишь разницей, что частное C может оказаться по абсолютной величине больше 1, однако мы всегда имеем $1/2 < |C| < 2$; деление следует начинать с определения цифры целых единиц делителя. Определение этой цифры совершается в начале процесса (до производства сдвигов) так же, как и остальных цифр частного. Регистр частного должен содержать не n , а $(n+1)$ цифру. Если цифра целых единиц частного оказывается равной 1, то после деления необходима нормализация вправо. Процесс деления дает верный результат и в том случае, когда делимое A задано с нарушением нормализации вправо: $A = 2^p \cdot \pm 0,0 \dots 0 a_1 a_2 \dots$. При этом в частном может также оказаться нарушенной нормализация вправо. Нарушение нормализации делителя может привести при таком способе деления к неверным результатам.

Извлечение квадратного корня. Обычный „школьный“ алгоритм извлечения квадратного корня значительно упрощается при переходе к двоичной системе, поскольку искомые цифры корня могут принимать лишь два значения 0 и 1.

Пусть мы нашли первые $(k-1)$ цифры b_1, b_2, \dots, b_{k-1} корня из $A = 0, a_1 a_2 \dots a_n$; очевидно, $0, b_1 \dots b_{k-1}$ есть наибольшее число вида $0, x_1 x_2 \dots x_{k-1}$, квадрат которого не превосходит A . Обозначим через A_{k-1} соответствующий остаток $A - (0, b_1 b_2 \dots b_{k-1})^2$. Следующая k -я цифра b_k может

равняться 1 или 0. Очевидно, если $A - (0, b_1 b_2 \dots b_{k-1} 1)^2 \geq 0$, то $b_k = 1$, если же $A - (0, b_1 \dots b_{k-1} 1)^2 < 0$, то $b_k = 0$; но

$$\begin{aligned} A - (0, b_1 b_2 \dots b_{k-1} 1)^2 &= A - (0, b_1 b_2 \dots b_{k-1} + 2^{-k})^2 = \\ &= [A - (0, b_1 b_2 \dots b_{k-1})^2] - 2 \cdot 2^{-k} 0, b_1 b_2 \dots b_{k-1} - 2^{-2k} = \\ &= A_{k-1} - 2^{-(k-1)} \cdot 0, b_1 b_2 \dots b_{k-1} 01 \end{aligned}$$

Для получения этой разности нужно к числу $0, b_1 b_2 \dots b_{k-1}$ приписать пару цифр 01, сдвинуть образованное число на $(k-1)$ разрядов вправо и вычесть из $(k-1)$ -го остатка A_{k-1} . Если при вычитании получится число ≥ 0 , то цифра $b_k = 1$ и эта разность совпадает со следующим остатком A_k . Если же эта разность меньше нуля, то $b_k = 0$ и $A_k = A_{k-1}$. Для восстановления остатка A_{k-1} нужно к разности снова добавить прежнее вычитаемое $0, b_1 b_2 \dots b_{k-1} 01$. Процедура нахождения цифр корня близка к обычной процедуре нахождения цифр частного, только роль делителя, постоянного в процессе деления, играет переменное число $0, b_1 b_2 \dots b_{k-1} 01$, которое можно называть переменным делителем. k -й переменный делитель сдвигается на k разрядов вправо, т. е. каждый следующий переменный делитель сдвинут по сравнению с предыдущим на 1 разряд вправо. Первый переменный делитель, служащий для определения цифры b , есть 0,01.

Вместо того, чтобы сдвигать переменный делитель вправо, можно, как это делалось в случае деления, каждый раз после нахождения соответствующей цифры корня сдвигать остаток на 1 разряд влево.

С х е м а 7 (извлечение корня)

$$z = \sqrt{x}, \text{ где } x = 0,100101$$

Сумматор		
+ 00,100101	000000	На сумматор посылается код числа x
+ 11,110000		На сумматор посылается код числа $-0,01$
00,010101	000000	
00,010101	000001	Сдвиг влево на 1 разряд
+ 00,101010	000011	На сумматор посылается код числа $-0,101$
+ 11,011000		
00,000010	000011	Сдвиг влево на 1 разряд

00,000100	00011	На сумматор посылается код числа — 0,1101
+ 11,001100		
11,010000	000110	Восстановление остатка
+ 00,110100		
00,000100	000110	Сдвиг влево на 1 разряд
00,001000	00110—	На сумматор посылается код числа — 0,11001
+ 11,001110		
11,010110	001100	Восстановление остатка
+ 00,110010		
00,001000	001100	Сдвиг влево на 1 разряд
00,010000	01100—	На сумматор посылается код числа — 0,110001
+ 11,001111		
11,011111	011000	Восстановление остатка
+ 00,110001		
00,010000	011000	Сдвиг влево на 1 разряд
+ 00,100000	11000—	На сумматор посылается код числа — 0,110000
11,010000		
11,110000	110000	Восстановление остатка
+ 00,110000		
00,100000	110000	

$$z = \sqrt{x} = 0,110000$$

Так же как и в случае деления, для того чтобы было осуществимо точное извлечение корня из точного квадрата, следует инвертировать все подачи на сумматор, а также правило определения очередной цифры корня:

С х е м а 8 (извлечение корня)

$$z = \sqrt{x}, \text{ где } x = 0,100101$$

Сумматор		
11,011010	000000	На сумматор посылается код числа
+ 00,010000		На сумматор посылается код числа 0,01
11,101010	000000	
11,101010	000001	Сдвиг влево на 1 разряд
11,010101	00001—	На сумматор посылается код числа 0,101
+ 00,101000		
11,111101	000011	Сдвиг влево на 1 разряд
+ 11,111011	00011—	На сумматор посылается код числа 0,1101
00,110100		
100,101111		
_____↑		

00,110000	000110	Восстановление остатка
+ 11,001011		
11,111011	000110	Сдвиг влево на 1 разряд
11,110111	00110—	На сумматор посылается код числа 0,11001
+ 00,110010		
100,101001		
↑		
00,101010	001100	Восстановление остатка
+ 11,001101		
11,110111	001100	Сдвиг влево на 1 разряд
11,101111	01100—	На сумматор посылается код числа 0,110001
+ 00,110001		
100,100000		
↑		
00,100001	011000	Восстановление остатка
+ 11,001110		
11,101111	011000	Сдвиг влево на 1 разряд
11,011111	11000—	На сумматор посылается код числа 0,110000
+ 00,110000		
100,001111	110000	
↑		
00,010000		Восстановление остатка
+ 11,001111		
11,011111		

$$z = \sqrt{x} = 0,110000$$

$$\text{Остаток} = 2^{-6} \cdot 0,100000$$

По поводу извлечения корня в машинах с плавающей запятой можно добавить следующее. Пусть $M = 2^m \cdot 0, a_1 a_2 \dots a_n$. Если порядок m четный, то \sqrt{M} находим по формуле: $\sqrt{M} = 2^{m/2} \sqrt{0, a_1 a_2 \dots a_n}$. Если же порядок m нечетный, то \sqrt{M} находим по формуле: $\sqrt{M} = 2^{\frac{m+1}{2}} \sqrt{0, 0 a_1 a_2 \dots}$. Деление на 2 порядка m (или суммы $m + 1$) получается сдвигом изображения числа $m + 1$ на 1 разряд в сторону младших разрядов.

Операции с блокировкой нормализации. В машинах с плавающей запятой операции совершаются над числами, заданными в нормализованном виде, результат также автоматически нормализуется. Иногда бывает удобно оперировать с числами, заданными в ненормализованном виде. Так, например, при серийном сложении выгодно иметь порядки

всех чисел, приведенные к порядку максимального по абсолютной величине числа, тогда сложение займет меньше времени.

Можно предусмотреть в машине блокировку нормализации, которая заключается в следующем:

а) числа употребляются в виде $2^q \cdot 0, a_1 a_2 \dots a_n$, где a_1 не обязательно единица;

б) по окончании арифметической операции окончательная нормализация влево не производится.

Так, при сложении или вычитании чисел с порядками q_1 и q_2 ответ будет иметь порядок $q = \max(q_1, q_2)$, при умножении $q_1 + q_2$, при делении $q_1 - q_2$. Разумеется, появление цифровых частей больших единицы все же исключено. Если в результате операции требуется нормализация вправо, то машина должна автоматически остановиться.

При операциях с блокировкой нормализации уменьшается время, требующееся для производства операции, но понижается точность. Операции с блокировкой нормализации могут широко использоваться при действиях с удвоенным числом разрядов (см. § 16, 27).

§ 12. УМНОЖЕНИЕ И ДЕЛЕНИЕ ЧИСЕЛ, ЗАДАННЫХ ДОПОЛНИТЕЛЬНЫМ И ОБРАТНЫМ КОДОМ

Умножение чисел, заданных дополнительным кодом.

Если числа в машине заданы их дополнительными или обратными кодами, то для выполнения операции умножения машина должна уметь кодам $[x]_2$ и $[y]_2$ или $[x]_3$ и $[y]_3$ двух чисел x и y поставить в соответствие код $[xy]_2$ или $[xy]_3$ их произведения xy . Схему такого эквивалента операции умножения можно составить различными способами, например, путем преобразования в прямой код, умножения в прямом коде и преобразования произведения в дополнительный или обратный коды. Остановимся сейчас на другом варианте умножения чисел, заданных дополнительным кодом.

Регистр произведения отличается от описанного выше регистра тем, что он имеет два разряда знака, а эти разряды вместе с первыми n разрядами после запятой образуют сумматор чисел, заданных модифицированным дополнительным

кодом (см. § 10). Сдвиг на один разряд вправо в этом регистре есть модифицированный сдвиг, т. е. он переводит набор $x'_0x_0x_1x_2 \dots x_{2n-1}x_{2n}$ в набор $x'_0x'_0x_0x_1x_2 \dots x_{2n-1}$ (при сдвиге вправо в крайнем левом разряде сохраняется прежняя цифра).

Рассмотрим сначала процесс умножения множимого x на положительный множитель $y = 0, u_1u_2 \dots u_n$. Пусть $[x]_2 = x_0x_0, x_1x_2 \dots x_n$. Процесс умножения совершенно аналогичен второму способу умножения по схеме 2 в прямом коде, с тем лишь отличием, что сдвиг является модифицированным сдвигом и что на сумматор посылается весь набор, составляющий код множимого, включая оба разряда знака. Именно такой сдвиг и такое сложение есть, соответственно, умножение на 2^{-1} и сложение чисел произвольного знака, заданных в дополнительном коде.

Перейдем к случаю, когда множитель y отрицателен. Пусть $[y]_2 = 2 + y = 1, u_1u_2 \dots u_n$. Отсюда $y = 0, u_1u_2 \dots u_n - 1$; $xu = x \cdot 0, u_1u_2 \dots u_n - x$. Умножение числа x на отрицательный множитель y сводится к умножению x на положительное число $0, u_1u_2 \dots u_n = 1 + y$ (дробную часть изображения y) и добавлению числа $-x$. Таким образом, при умножении x на множитель y вслед за последним сдвигом происходит или не происходит посылка дополнительного кода числа $-x$.

С х е м а 9 (умножение)

Пример 1.

Пусть $x = -0,110101$, $y = 0,101101$, $[x]_2 = 11,001011$

Регистр произведения

Знак множи- теля	Сумматор	Дробная часть кода множителя	
0	00,000000	101101	Передача кода множимого в сумматор
+	11,001011		
	11,001011	101101	Сдвиг вправо на 1 разряд
	11,1001011	10110	Код множимого в сумматор не передается
			Сдвиг вправо на 1 разряд
	11,11001011	1011	Передача кода множимого в сумматор
+	11,001011		
	10,11110111	1011	Сдвиг вправо на 1 разряд

+	11,01111011	101		101	Передача кода множимого в сумматор		
	11,001011						
	10,101001111	101		101	Сдвиг вправо на 1 разряд		
	11,0101001111	10					
	11,10101001111	1		1	Код множимого в сумматор не передается		
	11,001011						
+	10,11010101111	1		1	Сдвиг вправо на 1 разряд		
	11,011010101111						
	11,0110101011				Посылка $[-x]_2$ в сумматор не производится		
	11,0110101011						
					Код результата		
$xy = -0,100101010001$							

Пример 2. Пусть $x = -0,110101$; $y = -0,101101$

$$[x]_2 = 11,001011, \quad [-x]_2 = 00,110101, \quad [y]_2 = 11,010011$$

Регистр произведения			
Знак множи- теля	Сумматор	Дробная часть кода множителя	
1	+	00,000000	010011
		11,001011	
		11,001011	010011
		11,100101	01001
+		11,001011	
		10,110000	01001
		11,011000	0100
		11,101100	010
		11,110110	01
+		11,001011	
		11,000001	01
		11,100000	010010
		11,110000	10001
+		00,110101	
		00,100101	010001

Сдвиг вправо на 1 разряд

Передача кода множимого в сумматор

Сдвиг вправо на 1 разряд

Код множимого в сумматор не передается

Сдвиг вправо на 1 разряд

Код множимого в сумматор не передается

Сдвиг вправо на 1 разряд

Передача кода множимого в сумматор

Сдвиг вправо на 1 разряд

Код множимого в сумматор не передается

Сдвиг вправо на 1 разряд

Передача $[-x]_2$ в сумматор

Код результата

$xy = +0,100101010001$

Мы получаем в регистре произведения набор $x_0x_0, x_1x_2 \dots x_{2n}$ — дополнительный код точного $2n$ -значного произведения xy . Набор $x_0x_0, x_1x_2 \dots x_n$, стоящий в сумматоре, дает модифицированный дополнительный код главной части этого произведения, т. е. приближенного значения с точностью до 2^{-n} .

В последних n разрядах регистра произведения стоит набор $x_{n+1}x_{n+2} \dots x_{2n}$. Число $2^{-n} \cdot 0, x_{n+1}x_{n+2} \dots x_{2n}$ есть поправка к этому приближенному значению до точного, причем эта поправка всегда *положительная*. В самом деле, $x_0x_0, x_1x_2 \dots x_nx_{n+1} \dots x_{2n} = x_0x_0, x_1x_2 \dots x_n \underbrace{00 \dots 0}_n + + 00, \underbrace{00 \dots 0}_n x_{n+1}x_{n+2} \dots x_{2n}$.

Второе слагаемое и есть поправка. В случае, если нужно знать точное произведение, поправка со знаком 0 (+) посылается в отдельную ячейку ЗУ.

Умножение чисел, заданных обратным кодом. Перейдем теперь к умножению чисел, заданных обратным кодом, по аналогичной схеме. Естественно, что сумматор в этом случае есть сумматор обратного кода, т. е. его старший разряд знаков связан циклической передачей с последним разрядом сумматора (если, как в предыдущей схеме, сумматор имеет $n+2$ разряда, включая 2 разряда знака, то старший разряд знака связан с n -м разрядом после запятой). В случае, если множитель y отрицательный и $[y]_3 = = 1, y_1y_2 \dots y_n = y + 2 - 2^n$, то $y = 0, y_1y_2 \dots y_n - 1 + 2^n$.

Поэтому $xu = x \cdot 0, y_1y_2 \dots y_n - x + 2^{-n}x$. Если мы ищем главную часть произведения, т. е. произведение с точностью до 2^{-n} , то членом $2^{-n}x < 2^{-n}$ можно пренебречь, и умножение совершается так же, как в случае дополнительного кода.

Пример. $x = -0,1101, y = -0,1010$

$$[x]_3 = 11,0010, [y]_3 = 11,0101, [-x]_3 = 00,1101$$

Регистр произведения			
Знак множителя	Сумматор	Дробная часть кода множителя	
1		0101	Передача кода множимого в сумматор
	11,0010	0101	Сдвиг вправо на 1 разряд ¹
	11,1001	010	Код множимого в сумматор не передается Сдвиг вправо на 1 разряд

¹ Поскольку мы ищем первые n разрядов произведения, нас не интересуют цифры произведения, выходящие за пределы сумматора.

+	11,1100	—	—	01	Передача кода множимого в сумматор
	11,0010	—	—		
	110,1110	—	—		
			↑		
	10,1111	—	—	01	Сдвиг вправо на 1 разряд
	11,0111	—	—	0	Код множимого в сумматор не передается
					Сдвиг вправо на 1 разряд
					Передача $[-x]_3$ в сумматор
+	11,1011				
	00,1101				
	100,1000				
			↑		
	00,1001				Результат

Точное значение произведения с полным числом (8) знаков будет 0,10000010.

Для того чтобы получить точное $2n$ -значное произведение xu , нужно учесть, во-первых, что циклическая передача в наш сумматор соединяет разряд знаков с n -м, а не с $2n$ -м разрядом после запятой; во-вторых, ввести в случае отрицательного множителя слагаемое $2^{-n}x$.

Заметим, что если обратный код n -значного числа x добавить к коду числа A того же знака, стоящему в регистре произведения, то результат сложения не зависит от того, будут ли суммирующие функции выполнять все $2n$ разрядов или только n разрядов после запятой и будет ли, следовательно, осуществлена циклическая передача в n -й или $2n$ -й разряд после запятой.

Для случая положительных x и A это очевидно, так как при сложении положительных чисел не возникает циклической передачи. Если x и A отрицательны, n -значное изображение x в обратном коде есть $11, x_1 x_2 \dots x_n$, то $2n$ -значное изображение x будет $11, x_1 x_2 \dots x_n \underbrace{111 \dots 1}_n$, т. е.

получается из его n -значного изображения прибавлением числа $0, \underbrace{0 \dots 00}_n \underbrace{11 \dots 1}_n = 2^n - 2^{2n}$.

В случае добавления кода n -значного отрицательного числа x в регистр произведений, в котором стоит код отрицательного числа A , возникает циклическая передача. Циклическая передача в n -й разряд после запятой эквивалентна

передаче в $2n$ -й разряд и добавлению 1 во все разряды регистра после n -го.

Таким образом, результат прибавления кода числа x при n -значном и $2n$ -значном сумматоре одинаков:

Пример ($n = 4$). $[A]_3 = 11,11100110$, $x = -0,1011$

n -значный сумматор		$2n$ -значный сумматор	
Код A	11,1110 1110	Код A	11,11101110
Код x	11,0100	Код x	11,01001111
<hr/>		<hr/>	
111,0010 1110		111,00111101	
<hr/>		<hr/>	
↑		↑	
Циклическая передача		Циклическая передача	
11,0011 1110		11,00111110	

Таким образом, n -значный сумматор будет выполнять функции $2n$ -значного сумматора при добавлении кода множимого x , если в сумматоре будет число того же знака. Поэтому перед процессом умножения в сумматор посылается множитель, взятый со знаком множимого (см. стр. 39). Модифицированный сдвиг полученного набора на n разрядов вправо переведет дробную часть изображения множителя в дополнительные n разрядов регистра, а в сумматоре оставит $+0$ или -0 , в зависимости от знака множимого x . Последовательные вводы кода множимого x в n -значный сумматор будут в силу предыдущего эквивалентны вводу в $2n$ -значный сумматор. Затем, в зависимости от знака 1 или 0 множителя вводится или не вводится в сумматор код числа x . В результате последующих n сдвигов, этот ввод оказывается эквивалентным добавлению слагаемого $2^{-n} \cdot x$.

Далее происходит процесс умножения кода числа x на дробную часть кода y так же, как в случае дополнительного кода. После этого в регистре произведения оказывается набор $x_0x_0, x_1x_2 \dots x_nx_{n+1} \dots x_{2n}$, где x_0 — цифра знака множимого, $x_0, x_1 \dots x_n$ — обратный код главной части изображаемого набором числа и $x_0, x_{n+1} \dots x_{2n}$ — код добавки, умноженной на 2^n .

Наконец, в зависимости от знака множителя 0 или 1, в сумматор вводится или не вводится код $[-x]_3$. В случае

ввода этого кода в n -значном сумматоре складываются числа разных знаков, и для такого процесса сложения n -значный сумматор не эквивалентен $2n$ -значному. Добавка не участвует в суммировании и остается неизменной. Обратный код главной части произведения стоит на сумматоре; код добавки, умноженной на 2^n , получается присоединением знака множимого x к группе n цифр регистра произведения.

С х е м а 10 (умножение)

Пример 1. Пусть $x = -0,110101$, $y = 0,101100$, $[x]_3 = 11,001010$

Знак множи- теля	Регистр произведения	
	Сумматор	Регистр множи- теля
0	11,111111	101100
		В сумматор посылается „0·sign x “, а в регистр множителя — дробная часть кода множителя Сдвиг вправо на один разряд
	11,111111	101110
		Код множимого в сумматор не передается Сдвиг вправо на 1 разряд
+	11,111111 11,001010	1011
		Передача кода множимого в сумматор
	111,00100	
	11,00101011	1011
		Сдвиг вправо на 1 разряд
+	11,100101011	101
		Передача кода множимого в сумматор
	11,001010	
	110,101111	
	10,110000011	101
		Сдвиг вправо на 1 разряд
	11,0110000011	110
		Код множимого в сумматор не передается Сдвиг вправо на 1 разряд
+	11,10110000011	1
		Передача кода множимого в сумматор
	11,001010	
	110,110110	
	10,11011100011	1
		Сдвиг вправо на один разряд
	11,011011100011	
		$[-x]_3$ в сумматор не передается
	11,011011100011	
		Код главной части произведения 11,011011, код добавки $2^{-6} \cdot 11,100011$

Главная часть $xу$: $-0,100100$, добавка $-2^{-6} \cdot 0,011100$

$$xy = -0,100100011100$$

Пример 2. Пусть $x = 0,110101$; $y = -0,101100$
 $[y]_3 = 11,010011$; $[-x]_3 = 11,001010$

Регистр произведения		
Знак множителя	Сумматор	Регистр множителя
1	00,000000	010011
+	00,110101	
	00,110101	010011
+	00,110101	
	01,101010	010011
	00,110101	010011
+	00,110101	
	01,101010	010011
	00,110101	010011
	00,011010	010011
	00,001101	010011
+	00,110101	
	01,000100	010011
	00,100010	010011
	00,010000	010011
+	11,001010	
	11,011010	010011

В сумматор посылается „0·sign x“, а в регистр множителя — дробная часть кода множителя
 Передача кода множимого в сумматор
 Передача кода множимого в сумматор
 Сдвиг вправо на 1 разряд
 Передача кода множимого в сумматор
 Сдвиг вправо на 1 разряд
 Код множимого в сумматор не передается
 Сдвиг вправо на 1 разряд
 Код множимого в сумматор не передается
 Сдвиг вправо на 1 разряд
 Передача кода множимого в сумматор
 Сдвиг вправо на 1 разряд
 Код множимого в сумматор не передается
 Сдвиг вправо на 1 разряд
 Передача $[-x]_3$ в сумматор
 Код главной части произведения:
 11,011010, код добавки 00,100100
 Главная часть xu : $-0,100101$, добавка $+2^{-6} \cdot 0,100100$
 $xu = -0,100101 + 2^{-6} \cdot 0,100100 = -0,100100011100$

Деление чисел, заданных дополнительным и обратным кодом. Операция деления заключается в серии операций, обратных тем, к которым сводится операция умножения. При этом делимое отвечает произведению, делитель — множимому, частное — множителю. Такое соответствие удобно потому, что в описанных выше процессах умножения множимое поступает на сумматор все целиком, вплоть до конца процесса, между тем в каждом этапе умножения фигурирует лишь одна цифра множителя, начиная с младшей. При обращении этого процесса мы будем получать цифры частного, начиная со старшей.

Начнем с деления в дополнительном коде по схеме, обратной схеме 9. Знак частного (множителя) определяется

совпадением или несовпадением знаков делимого или делителя. Делимое посылается в сумматор, являющийся первой частью регистра произведения. В зависимости от цифры знака 1 или 0 частного, делитель прибавляется или не прибавляется к делимому. Далее мы определяем последовательно все цифры дробной части частного, причем процесс идет по следующей схеме (отвечающей схеме 3).

1. Число, стоящее в регистре произведения, сдвигается на 1 разряд влево.

2. Из числа, стоящего в регистре, вычитается делитель.

3. Если знак разности совпадает со знаком делителя, то в младший разряд регистра произведения посылается 1; после этого возвращаются к операции (1).

4. Если знак разности обратен знаку делителя, то в последний разряд регистра произведения посылается цифра 0; к числу, стоящему на сумматоре, добавляется код делителя, после чего возвращаются к операции (1).

В конце процесса деления (после n циклов) на сумматоре стоит умноженный на 2^n остаток от деления, а в регистре множителя—дробная часть дополнительного кода частного.

С х е м а 11 (деление)

Пример 1. Делимое $x = -0,100111$ Делитель $y = -0,110001$
 $[x]_2 = 11,011001$ $[y]_2 = 11,001111$

Знак частного $+$ (0)

Сумматор	Регистр частного	
11,011001	000000	Сдвиг влево на 1 разряд
$+$ 10,110010	00000—	Передача $[-y]_2$ в сумматор
00,110001		
11,100011	000001	Сдвиг влево на 1 разряд
$+$ 11,000110	00001—	Передача $[-y]_2$ в сумматор
00,110001		
11,110111	000011	Сдвиг влево на 1 разряд
$+$ 11,101110	00011—	Передача $[-y]_2$ в сумматор
00,110001		
$+$ 00,011111	000110	Передача $[y]_2$ в сумматор
11,001111		
11,101110	000110	Сдвиг влево на 1 разряд

+	<u>11,011100</u>	00110—	Передача $[-y]_2$ в сумматор
	00,110001		
+	<u>00,001101</u>	00100—	Передача $[y]_2$ в сумматор
	11,001111		
	<u>11,011100</u>	001100	Сдвиг влево на 1 разряд
	10,111000	01100—	Передача $[-y]_2$ в сумматор
+	<u>00,110001</u>		
	11,101001	011001	Сдвиг влево на 1 разряд
+	<u>11,010010</u>	11001—	Передача $[-y]_2$ в сумматор
	00,110001		
+	<u>00,000011</u>	110010	Передача $[y]_2$ в сумматор
	11,001111		
	<u>11,010010</u>		

$$\text{Частное } \frac{x}{y} = 0,110010$$

$$\text{Остаток} = -2^{-6} \cdot 0,101110$$

Пример 2. Делимое $x = -0,100111$ Делитель $y = +0,110001$
 $[x]_2 = 11,011001$ $[y]_2 = 00,110001$

Знак делителя 0

$$-y = -0,110001$$

Знак частного 1

$$[-y]_2 = 11,001111$$

Сумматор	Регистр частного	
+	<u>11,011001</u>	000000 В сумматор передается код делителя
	00,110001	
	<u>00,001010</u>	000000 Сдвиг влево на 1 разряд
	00,010100	00000— В сумматор передается $[-y]_2$
+	<u>11,001111</u>	
	11,100011	000000 В сумматор передается код делителя
+	<u>00,110001</u>	
	00,010100	Сдвиг влево на 1 разряд
	00,101000	00000— В сумматор передается $[-y]_2$
+	<u>11,001111</u>	
	11,110111	000000 В сумматор передается код делителя
+	<u>00,110001</u>	
	00,101000	Сдвиг влево на 1 разряд
	01,010000	00000— В сумматор передается $[-y]_2$
+	<u>11,001111</u>	
	00,011111	000001 Сдвиг влево на 1 разряд
	00,111110	00001— В сумматор передается $[-y]_2$
+	<u>11,001111</u>	
	00,001101	000011 Сдвиг влево на 1 разряд

+	00,011010	00011—	В сумматор передается $[-y]_2$
	11,001111		
+	11,101001	000110	В сумматор передается код делителя
	00,110001		
	00,011010		Сдвиг влево на 1 разряд
+	00,110100	00110—	В сумматор посылается $[-y]_2$
	11,001111		
	00,000011	001101	
		Частное $\frac{x}{y} = -0,110011$	
		Остаток $= +2^{-6} \cdot 0,000011$	

Деление в обратном коде отличается от деления в дополнительном коде, во-первых, тем, что сумматор является сумматором обратного кода с циклической передачей из разряда знака в младший разряд сумматора, во-вторых, тем, что для получения остатка от деления, в случае отрицательного частного, надо после определения n -й цифры вычесть делитель из числа, стоящего в сумматоре.

С х е м а 12 (деление)

Пр и м е р 1.

Делимое $x = -0,100111$ Делитель $y = -0,110001$

$[x]_3 = 11,011000$ $[y]_3 = 11,001110$

Сумматор	Регистр частного	
11,011000	000000	Сдвиг влево на 1 разряд
10,110001	00000—	Передача y в сумматор инверсной передачей
00,110001		
11,100010	000001	Сдвиг влево на 1 разряд
11,000101	00001—	Передача y в сумматор инверсной передачей
00,110001		
11,110110	000011	Сдвиг влево на 1 разряд
11,101101	00011—	Передача y в сумматор инверсной передачей
+	00,110001	
100,011110		
	↑	
+	00,011111	000110 Передача $[y]_3$ в сумматор
	11,001110	
	11,101101	000110 Сдвиг влево на 1 разряд
+	11,011011	00110— Передача y в сумматор инверсной передачей
	00,110001	
100,001100		
	1	

+	00,001101	001100	Передача $[y]_3$ в сумматор
	<u>11,001110</u>		
	11,011011	001100	Сдвиг влево на 1 разряд
+	10,110111	01100—	Передача y в сумматор инверсной передачей
	<u>00,110001</u>		
	11,101000	011001	Сдвиг влево на 1 разряд
	11,010001	11001—	Передача y в сумматор инверсной передачей
+	<u>00,110001</u>		
	100,000010		
	↑		
+	00,000011	110010	В сумматор посылается $[y]_3$
	<u>11,001110</u>		
	11,010001		

$$\text{Частное } \frac{x}{y} = 0,110010$$

$$\text{Остаток} = -2^{-6} \cdot 0,101110$$

Пример 2.

Делимое $x = -0,100111$ Делитель $y = +0,110001$

$$[x]_3 = 11,011000 \quad [y]_3 = 00,110001$$

$$\text{Знак частного: } 1 (-) \quad [-y]_3 = 11,001110$$

$$\text{Знак делителя: } 0 (+)$$

Сумматор	Регистр частного	
+	11,011000	000000 В сумматор передается код делителя
	<u>00,110001</u>	
	100,001001	
	↑	
	00,001010	000000 Сдвиг влево на 1 разряд
	00,010100	00000— В сумматор передается $[-y]_3$
+	<u>11,001110</u>	
	11,100010	000000 В сумматор передается код делителя
+	<u>00,110001</u>	
	100,010011	
	↑	
	00,010100	000000 Сдвиг влево на 1 разряд
	00,101000	0000— В сумматор передается $[-y]_3$
+	<u>11,001110</u>	
	11,110110	000000 В сумматор передается код делителя
+	<u>00,110001</u>	
	100,100111	
	↑	

	00,101000	000000	Сдвиг влево на 1 разряд
+	01,010000	000000—	В сумматор передается $[-y]_3$
	11,001110		
	100,011110		
		↑	
	00,011111	000001	Сдвиг влево на 1 разряд
+	00,111110	00001—	В сумматор передается $[-y]_3$
	11,001110		
	100,001100		
		↑	
	00,001101	000011	Сдвиг влево на 1 разряд
+	00,011010	00011—	В сумматор передается $[-y]_3$
	11,001110		
+	11,101000	000110	В сумматор передается код делителя
	00,110001		
	100,011001		
		↑	
	00,011010	000110	Сдвиг влево на 1 разряд
+	00,110100	00110—	В сумматор передается $[-y]_3$
	11,001110		
	100,000010		
		↑	
+	00,000011	001101	В сумматор передается $[-y]_3$
	11,001110		
	11,010001		

$$\text{Частное } \frac{x}{y} = -0,110010$$

$$\text{Остаток} = -2^{-6} \cdot 0,101110$$

Другие варианты умножения и деления чисел, заданных обратным кодом. Рассмотрим другой вариант умножения чисел, заданных обратным кодом, основанный на формуле

$$xy = \text{sign } y \cdot x |y|$$

Схема умножения, основанная на этой формуле, по существу совпадает со схемой умножения числа, заданного обратным кодом, на положительный множитель (см. стр. 74). Схема умножения следующая.

Передача в сумматор происходит всегда прямым или инверсным образом, в зависимости от цифры 0 или 1 знака множителя. Вначале на все разряды сумматора подается

(прямым или инверсным образом) знак множимого (т. е. подается код $+0$ или -0).

В дополнительные n разрядов подается (прямым образом) дробная часть обратного кода множителя. Знак множителя фиксируется для управления процессом умножения. При несовпадении последней цифры регистра произведения с цифрой знака множителя производится передача в сумматор прямым или инверсным образом кода множимого. При совпадении этих цифр такой передачи нет. Затем производится сдвиг набора, стоящего в регистре произведения на один разряд вправо. После повторения этого процесса n раз, в регистре произведения оказывается обратный код $2n$ -значного произведения.

В самом деле, прямая или инверсная передача кода множимого x в зависимости от знака множителя y означает прямую передачу кода числа $A = \text{sign } y \cdot x$, т. е. именно это число фактически фигурирует в качестве множимого. Далее, если $[y]_3 = y_0, y_1 y_2 \dots y_n$, то $|y| = 0, \alpha_1 \alpha_2 \dots \alpha_n$, где $\alpha_k = |y_k - y_0|$, $k = 1, 2, \dots, n$. Последовательное умножение числа A на цифры α_k ($k = n, n-1, \dots, 1$) числа $|y|$ обеспечивается посылкой или не посылкой кода этого числа в сумматор, в зависимости от различия или совпадения цифр y_k и y_0 , так как при $y_k \neq y_0$, $\alpha_k = 1$, а при $y_k = y_0$, $\alpha_k = 0$.

С х е м а 13 (умножение)

Пример 1.

Множимое $x = -0,110101$ Множитель $y = -0,101010$

$[x]_3 = 11,001010$

$[y]_3 = 11,010101$

Регистр произведения

Знак множителя	Сумматор	Регистр множителя	
1	00,000000	010101	Сдвиг на 1 разряд вправо
	00,000000	0 01010	В сумматор посылается код множимого инверсной передачей
+	00,110101		
	00,110101	0 01010	Сдвиг на 1 разряд вправо
	00,011010	10 0101	Сдвиг на 1 разряд вправо
	00,001101	010 010	В сумматор посылается код множимого инверсной передачей
+	00,110101		
	01,000010	010 010	Сдвиг на 1 разряд вправо

	00,100001	0010 01	Сдвиг на 1 разряд вправо
+	00,010000	10010 0	В сумматор посылается код множимого
	00,110101		инверсной передачей
	01,000101	10010 0	Сдвиг вправо на 1 разряд
	00,100010	110010	

$$xy = +0,100010110010$$

Пример 2.

Множимое $x = -0,110101$ Множитель $y = +0,101010$
 $[x]_3 = 11,001010$ $[y]_3 = 00,101010$

Регистр произведения			
Знак множителя	Сумматор	Регистр множителя	
0	11,111111	101010	Сдвиг вправо на 1 разряд
+	11,111111	110101	В сумматор посылается код множимого
	11,001010		
	111,001001		
	↑		
	11,001010	110101	Сдвиг вправо на 1 разряд
	11,100101	011010	Сдвиг вправо на 1 разряд
+	11,110011	101101	В сумматор посылается код множимого
	11,001010		
	110,111100		
	↑		
	10,111101	101101	Сдвиг вправо на 1 разряд
	11,011110	110110	Сдвиг вправо на 1 разряд
+	11,101111	011011	В сумматор посылается код множимого
	11,001010		
	110,111001		
	↑		
	10,111010	011011	Сдвиг вправо на 1 разряд
	11,011101	001101	

$$xy = -0,100010110010$$

Рассмотрим деление чисел, заданных обратным кодом, по принципу сведения к делению чисел одинакового знака с переменной в случае необходимости знака частного. Как всегда, делимое соответствует произведению, делитель — множимому, а частное — множителю. Согласно этой схеме делимое должно делиться на делитель, умноженный на знак частного. Цифра знака частного равна 0 или 1, в зависимости от

совпадения или несовпадения знаков делимого и делителя. Следовательно, при несовпадении этих знаков мы должны заменить код делителя его поразрядным дополнением (что отвечает изменению знака). Цифра знака частного определяется совпадением или несовпадением цифр знаков делимого и делителя. Эта цифра выполняет функции управления процессом деления.

В зависимости от значения 0 или 1 этой цифры код делимого посылается на сумматор прямой или инверсной передачей, т. е. мы фактически меняем знак делимого, если цифра знака частного равна 1. Далее процесс деления числа, стоящего на сумматоре, на делитель совершается по обычной схеме деления с той разницей, что в случае, когда цифра знака частного равна 1, цифры, посылаемые в регистр частного (последний разряд регистра произведения), попадают туда своим дополнением. Таким способом мы фактически меняем знак частного, что компенсирует перемену знака делимого при посылке его в сумматор.

С х е м а 14 (деление)

Пример.

Делимое $x = -0,100111$ Делитель $y = +0,110001$

$[x]_3 = 11,011000$ $[y]_3 = 00,110001$

Знак частного: 1 (—)

Знак делителя: 0 (+) $[-y]_3 = 11,001110$

Сумматор	Регистр частного	
00,100111	000000	Сдвиг влево на 1 разряд
+ 01,001110	000000—	В сумматор передается $[-y]_3$
11,001110		
100,011100		
	↑	
00,011101	000000	Сдвиг влево на 1 разряд
+ 00,111010	000000—	В сумматор передается $[-y]_3$
11,001110		
100,001000		
	↑	
00,001001	000000	Сдвиг влево на 1 разряд

+	00,010010	00000—	В сумматор передается $[-y]_3$
	11,001110		
	11,100000	000001	На сумматор передается код делителя
+	00,110001		
	100,010001		
		↑	
	00,010010	000001	Сдвиг влево на 1 разряд
+	00,100100	00001—	В сумматор передается $[-y]_3$
	11,001110		
	11,110010	000011	В сумматор передается код делителя
+	00,110001		
	100,100011		
		↑	
	00,100100	000011	Сдвиг влево на 1 разряд
+	01,001000	00011—	В сумматор передается $[-y]_3$
	11,001110		
	100,010110		
		↑	
	00,010111	000110	Сдвиг влево на 1 разряд
+	00,101110	00110—	В сумматор передается $[-y]_3$
	11,001110		
	11,001110	001101	В сумматор передается код делителя (восстанов-
+	00,110001		ление остатка)
	100,101101		
		↑	
	00,101110		

$$\text{Частное } \frac{x}{y} = -0,110010$$

$$\text{Остаток} = -2^{-6} \cdot 0,101110$$

Вместо того, чтобы инвертировать делимое, можно было бы инвертировать делитель.

Обычно, однако, передача из ячейки ЗУ делимого в сумматор и делителя в его регистр (он же регистр множимого) совершается без инвертирования. Такое инвертирование требует дополнительных устройств или дополнительных передач в арифметическом устройстве.

Удобнее, не производя фактически инвертирования делителя, вести вычисления так, как будто у делителя был изменен знак, т. е. заменить вычитание делителя его прибавлением: в случае совпадения цифр знаков делителя и числа,

стоящего на сумматоре, продолжать действия, как прежде при несовпадении, и наоборот. Тогда процесс деления происходит по следующей схеме, являющейся обращением схемы 13 умножения.

а) На сумматор посылается код делимого, в регистр множимого — код делителя; если цифра знака делимого и делителя совпадают (не совпадают), в ячейку знака частного посылается 0 (1).

б) Число, стоящее в регистре произведения, сдвигается на один разряд влево.

в) В сумматор посылается код делителя инверсной (прямой) передачей.

г) Если цифра знака числа, стоящего на сумматоре, совпадает (не совпадает) с цифрой знака делителя, то в последний разряд регистра произведения посылается цифра, дополнительная к цифре знака частного, после чего повторяется б).

г') Если цифра знака числа, стоящего на сумматоре, не совпадает (совпадает) с цифрой знака делителя, то в последний разряд регистра произведения посылается цифра знака частного. В сумматор посылается прямой (инверсной) передачей код делителя, после чего повторяется б).

После n -кратного выполнения операций г) или г') деление заканчивается. В дополнительных разрядах регистра произведения оказывается дробная часть кода частного. В отведенной для нее ячейке находится цифра знака частного, а в сумматоре — код остатка, умноженного на 2^n .

С х е м а 15 (деление)

Делимое $x = -0,100111$ Делитель $y = +0,110001$
 $[x]_3 = 11,011000$ $[y]_3 = 00,110001$
 Знак частного: 1 (—) $[-y]_3 = 11,001110$
 Знак делителя: 0 (+)

Знак частного	Сумматор	Регистр частного	
1	11,011000	000000	Сдвиг влево на 1 разряд
	10,110001	000000—	В сумматор посылается код делителя
+	00,110001		
	11,100010	000000	Сдвиг влево на 1 разряд

11,000101	00000—	В сумматор посылается код делителя
+ 00,110001		
11,110110	000000	Сдвиг влево на 1 разряд
11,101101	00000—	В сумматор посылается код делителя
+ 00,110001		
100,011110		
↑		
00,011111	000001	В сумматор посылается $[-y]_2$
+ 11,001110		
11,101101	000001	Сдвиг влево на 1 разряд
11,011011	00001—	В сумматор посылается код делителя
+ 00,110001		
100,001100		
↑		
00,001101	000011	В сумматор посылается $[-y]_2$
+ 11,001110		
11,011011	000011	Сдвиг влево на 1 разряд
10,110111	00011—	В сумматор посылается код делителя
+ 00,110001		
11,101000	000110	Сдвиг влево на 1 разряд
11,010001	00110—	В сумматор посылается код делителя
+ 00,110001		
100,000010		
↑		
00,000011	001101	В сумматор посылается $[-y]_2$
+ 11,001110		
11,010001		

$$\text{Частное } \frac{x}{y} = -0,110010$$

$$\text{Остаток} = -2^{-6} \cdot 0,101110$$

Все описанные нами методы деления чисел, заданных в дополнительном или обратном коде, являются обобщением первого из способов деления положительных чисел. Каждый из таких методов допускает изменения, отвечающие второму способу деления положительных чисел. Этих изменений мы здесь приводить не будем, а в следующем параграфе приведем способ деления чисел, заданных в дополнительном или обратном коде, являющийся непосредственным обобщением второго способа деления положительных чисел. Кроме того, следует иметь в виду, что в описанных способах

деления при некоторых комбинациях знаков делимого и делителя оказывается невозможным точное деление (см. стр. 64). Каждую из приведенных схем нетрудно видоизменить так, чтобы такое деление стало возможным.

§ 13. ПРИМЕНЕНИЕ ДВОИЧНОЙ СИСТЕМЫ С ЦИФРАМИ 1, $\bar{1}$ ДЛЯ УМНОЖЕНИЯ, ДЕЛЕНИЯ И ИЗВЛЕЧЕНИЯ КВАДРАТНОГО КОРНЯ

Двоичная система (1, $\bar{1}$). Рассмотрим сейчас вариант двоичной системы, в которой применяются две цифры 1 и $\bar{1}$

$\bar{1} = -1$. Так как $2^k - \sum_{l=0}^{k-1} 2^l = 1$, то

$$\underbrace{1\bar{1}\bar{1} \dots \bar{1}}_k = \underbrace{00 \dots 01}_k$$

Тем самым устанавливается связь между обычной двоичной системой и этой новой системой — системой (1, $\bar{1}$): группа из $k+1$ цифр — цифра 1, за которой следуют k цифр $\bar{1}$, заменяется в обычной системе группой из k нулей и следующей за ними цифрой 1.

Переход от группы цифр системы (1, $\bar{1}$) к группе цифр обычной системы и обратно задается следующей таблицей:

Номер цифры	...	r	$r+1$...	s	$(s+1)$	$(s+2)$...	$s+k$	$s+k+1$	$s+k+2$
Цифра системы (1, $\bar{1}$)	...	1	1	...	1	$\bar{1}$	$\bar{1}$...	$\bar{1}$	1	1
Цифра обычной системы	1	1	1	...	0	0	0		1	1

Если в l -м разряде обычной системы стоит цифра 0 или 1, то в $(l+1)$ разряде системы (1, $\bar{1}$) стоит соответственно $\bar{1}$ или 1. Это соответствие не относится пока к первой цифре числа, заданного в двоичной системе (1, $\bar{1}$).

Пусть x в обычной системе равно

$$x = 0, x_1 x_2 \dots x_n$$

где x_1, x_2, \dots, x_n равны 0 или 1.

Тогда в двоичной системе $(1, \bar{1})$ число x можно представить в виде

$$x = 1, \bar{1} \tilde{x}_1 \tilde{x}_2 \dots \tilde{x}_n$$

где $\tilde{x}_k = \bar{1}$, если $x_k = 0$, $\tilde{x}_k = 1$, если $x_k = 1$.

Вообще, если a есть обычная двоичная цифра (0 или 1), то через \tilde{a} будем обозначать соответственно цифры $\bar{1}$ или 1 системы $(1, \bar{1})$:

$$\tilde{a} = 2a - 1 = \begin{cases} 1 & \text{при } a = 1 \\ -1 & \text{при } a = 0 \end{cases} \quad (1)$$

Установим общее правило записи числа x , ($|x| < 1$), в системе $[1, \bar{1}]$. Рассмотрим дополнительный бесконечно-значный код числа x :

$$[x]_2^\infty = x_0, x_1 x_2 \dots x_n \dots \quad (2)$$

При $x \geq 0$, $x_0 = 0$, $\bar{x}_0 = 1$, $[x]_2^\infty = x$; при $x \leq 0$, $x_0 = 1$, $\bar{x}_0 = 0$, $[x]_2^\infty = x + 2$. Поэтому всегда $x = [x]_2^\infty - 2 + 2\bar{x}_0$.

Поставив перед записью (2) числа $[x]_2^\infty$ в разряде целых двоек цифру \bar{x}_0 (дополнительную к цифре знака), получим число

$$\bar{x}_0 x_0, x_1 x_2 \dots x_n \dots = [x]_2^\infty + 2\bar{x}_0 = x + 2 \quad (3)$$

Для того, чтобы получить бесконечнозначную запись числа x в системе $(1, \bar{1})$, нужно заменить каждую цифру a в правой части (3) ($a = \bar{x}_0, x_0, x_1 \dots$) соответственной цифрой \tilde{a} (т. е. сохранить без изменения цифры, равные 1, и заменить каждую цифру 0 цифрой $\bar{1}$) и полученное число сдвинуть на один разряд вправо: $x = \tilde{\bar{x}_0} \tilde{x}_0 \tilde{x}_1 \tilde{x}_2 \dots$

В самом деле, в силу (1), (3)

$$\begin{aligned} \tilde{\bar{x}_0} \tilde{x}_0 \tilde{x}_1 \tilde{x}_2 \dots &= 2 \cdot (\bar{x}_0, x_0 x_1 \dots x_n \dots) - 1, 111 \dots = \\ &= \bar{x}_0 x_0, x_1 x_2 x_3 \dots - 2 = (x + 2) - 2 = x \end{aligned}$$

Иными словами, при переходе от обычной двоичной системы к системе $(1, \bar{1})$ каждая цифра, включая цифру знака в бесконечнозначном коде числа x , сдвигается на 1 разряд вправо, причем цифра 0 заменяется цифрой $\bar{1}$. В разряде целых единиц ставится цифра 1 при $x \geq 0$ и цифра $\bar{1}$ при $x < 0$. Например, $1/3 = 0,10101 \dots$. В системе $(1, \bar{1})$ число $1/3$ имеет запись $1, \bar{1} \bar{1} \bar{1} \bar{1} \dots$; число $-1/3$, имеющее дополнительный код $1,010101 \dots$, запишется в виде $\bar{1}, \bar{1} \bar{1} \bar{1} \bar{1} \dots$.

n -значное число, изображенное дополнительным кодом, можно рассматривать как бесконечнозначное, в котором все цифры, начиная с $(n+1)$ -й, равны 0. Изображение этого числа в системе $(1, \bar{1})$ должно иметь все цифры, начиная с $(n+2)$ -й цифры, равными $\bar{1}$. Но $-\sum_{k=n+2}^{\infty} 2^{-k} = 2^{-(n+1)}$. Поэтому такой бесконечный хвост из цифры $\bar{1}$ эквивалентен добавлению цифры $\bar{1}$ в $(n+1)$ -й разряд.

Таким образом, n -значное число x , дополнительный код которого равен $x_0 x_1 x_1 \dots x_n (\bar{x}_0 x_0 x_1 x_1 \dots x_n)$, равно $x = \tilde{x}_0 \tilde{x}_0 \tilde{x}_1 \dots \tilde{x}_n - 2^{-(n+1)}$.

Дополнительный и обратный код положительных чисел совпадают. В случае отрицательного числа, заданного обратным кодом, предполагается, что во всех разрядах кода после n -го стоит цифра 1. В соответственном изображении в системе $(1, \bar{1})$ мы должны были бы считать все цифры после $(n+1)$ -й равными 1. Этот бесконечный „хвост“ цифр 1 эквивалентен дополнительной единице в $(n+1)$ -м разряде (числу $2^{-(n+1)}$). Таким образом, при переходе от обратного кода $y_0, y_1 y_2 \dots y_n$ числа y к его изображению в системе $(1, \bar{1})$ имеем: $y = \tilde{y}_0 \tilde{y}_0 \tilde{y}_1 \dots \tilde{y}_n \mp 2^{-(n+1)}$ (знак $-$ или $+$ в зависимости от положительности или отрицательности числа y).

Приведенное простое правило перехода от дополнительного или обратного кода числа к его изображению в системе $(1, \bar{1})$ может быть использовано для производства умножения и деления.

Умножение. Умножение числа x на множитель y , заданный в системе $(1, \bar{1})$, состоит в последовательном умножении x на цифры 1 или $\bar{1}$ множителя, т. е. в посылке на сумматор кодов чисел x или $-x$ и соответственных сдвигов.

Рассмотрим случай дополнительного кода. Пусть множитель y имеет код $[y]_2 = y_0 y_1 \dots y_n$. Дополним этот код обратной цифрой знака \bar{y}_0 и поместим это изображение $\bar{y}_0 y_0 y_1 y_2 \dots y_n$ множителя в регистр множителя, совпадающий с дополнительными цифрами регистра произведений¹. Так как $y = \bar{\tilde{y}}_0 \tilde{y}_0 \tilde{y}_1 \dots \tilde{y}_n - 2^{-(n+1)}$, то $xy = x \cdot \bar{\tilde{y}}_0 \tilde{y}_0 \dots \tilde{y}_n - 2^{(n+1)}x$.

Процесс умножения начинается с посылки кода числа $-x$, которая эквивалентна (из-за последующих $(n+1)$ сдвигов) добавлению слагаемого $-2^{(n+1)}x$ к результату. Далее следует обычный процесс умножения на цифры $\tilde{y}_n, \tilde{y}_{n-1}, \dots, \tilde{y}_0, \bar{\tilde{y}}_0$. Фактически же мы имеем цифры $y_n, y_{n-1}, \dots, y_0, \bar{y}_0$. Поэтому если последняя цифра y_k после $(n-k)$ сдвигов регистра произведения равна 0 (т. е. $\tilde{y}_k = \bar{1}$), в сумматор посылается код числа $-x$. Если эта цифра равна 1, в сумматор посылается код числа x . В результате в регистре произведения получится дополнительный код $2n$ -значного произведения.

С х е м а 16 (умножение)

Множимое $x = 0,110101$ Множитель $y = 0,101101$

$[-x]_2 = 11,001011$ $[y]_2 = 0,101101$

$[y]_2' = 10,101101$

Регистр произведения
Сумматор Множитель

В системе $(1, \bar{1})$: $y = 1, \bar{1} \bar{1} \bar{1} \bar{1} \bar{1} - 2^{-7}$

$ \begin{array}{r} 00,000000 \\ + 11,001011 \\ \hline 11,001011 \\ + 00,110101 \\ \hline 100,000000 \end{array} $	<p>10101101 В сумматор посылается код $[-x]_2$</p> <p>10101101 В сумматор посылается код числа x</p> <p>10101101 Сдвиг вправо на 1 разряд</p>
---	---

¹ В дальнейшем такое видоизменение дополнительного или обратного кода будет отмечаться добавлением штриха к прежним обозначениям: $[x]_2'$ для дополнительного и $[x]_3'$ для обратного кодов.

$$\begin{array}{rcl}
& 00,000000 & 01010110 \quad \text{В сумматор посылается код } [-x]_2 \\
+ & 11,001011 & \\
\hline
& 11,001011 & 01010110 \quad \text{Сдвиг вправо на 1 разряд} \\
& 11,100101 & 10101011 \quad \text{В сумматор посылается код числа } x \\
+ & 00,110101 & \\
\hline
& 100,011010 & 10101011 \quad \text{Сдвиг вправо на 1 разряд} \\
& 00,001101 & 01010101 \quad \text{В сумматор посылается код числа } x \\
+ & 00,110101 & \\
\hline
& 01,000010 & 01010101 \quad \text{Сдвиг вправо на 1 разряд} \\
& 00,100001 & 00101010 \quad \text{В сумматор посылается код } [-x]_2 \\
+ & 11,001011 & \\
\hline
& 11,101100 & 00101010 \quad \text{Сдвиг вправо на 1 разряд} \\
& 11,110110 & 00010101 \quad \text{В сумматор посылается код числа } x \\
+ & 00,110101 & \\
\hline
& 100,101011 & 00010101 \quad \text{Сдвиг вправо на 1 разряд} \\
& 00,010101 & 10001010 \quad \text{В сумматор посылается код } [-x]_2 \\
+ & 11,001011 & \\
\hline
& 11,100000 & 10001010 \quad \text{Сдвиг вправо на 1 разряд} \\
& 11,110000 & 01000101 \quad \text{В сумматор посылается код числа } x \\
+ & 00,110101 & \\
\hline
& 100,100101 & 01000101 \\
& 00,100101 & 01000101
\end{array}$$

$$xy = 0,100101010001$$

Пусть теперь сумматор есть сумматор обратного кода, множимое x и множитель y заданы обратными кодами. Если $[y]_3 = y_0, y_1 y_2 \dots y_n$, то $y = \tilde{y}_0 \tilde{y}_1 \tilde{y}_2 \dots \tilde{y}_n \mp 2^{-(n+1)}$. Поэтому $xy = x \cdot \tilde{y}_0 \tilde{y}_1 \tilde{y}_2 \dots \tilde{y}_n \mp 2^{-(n+1)}x$. Помещаем в регистр множителя, образованный последними $(n+2)$ цифрами регистра произведения, изображение множителя $(\bar{y}_0 y_0 y_1 y_2 \dots y_n)$; в сумматор посылает код числа $-x$ или x , в зависимости от равенства 0 или 1 цифры знака y_0 множителя. Далее переходим к умножению x на цифры $\bar{y}_n, \bar{y}_{n-1}, \dots, \bar{y}_1, \bar{y}_0, \bar{y}_0$ и суммированию произведений с соответственными сдвигами: если последняя цифра регистра произведения равна 0, то посылает на сумматор код числа $-x$; если она равна 1, то посылает код числа x ; если такая операция повторена меньше, чем $n+2$ раза, то на регистре произведения происходит общий сдвиг на 1 разряд вправо, после чего сле-

дует повторение процесса. После того, как эта операция повторена $n + 2$ раза, на сумматоре оказывается обратный код n -значного произведения.

Так как цифры, оказавшиеся в процессе умножения на дополнительных к сумматору разрядах регистра произведения, попадают из последнего разряда сумматора на дополнительные разряды с тем знаком, который был на сумматоре в момент сдвига, и в дальнейшем суммировании фактически не участвуют, то процесс сдвига для получения $2n$ -значного произведения приходится изменить. Удобнее иметь для этой цели $n + 3$ дополнительных разряда регистра произведения, причем старший из этих дополнительных разрядов несет специальные функции управления сдвигом и процессом циклической передачи. В начале процесса умножения в этом разряде стоит цифра 0. При сдвиге цифра из последнего разряда сумматора переходит в этот разряд без изменения или своим дополнением в зависимости от равенства 0 или 1 первой цифры знака на сумматоре. Кроме того, если в этом дополнительном разряде стоит цифра 1, то выключается циклическая передача на сумматоре. Сама цифра, стоящая в этом разряде, передается в следующие разряды без изменения. При таких сдвигах в регистре произведения в конце процесса окажется код $2n$ -значного произведения, причем добавка, стоящая в дополнительных разрядах, всегда положительна.

В самом деле, n -я цифра положительного числа, стоящая на сумматоре, передается в добавку без изменения. Цифра 1 кода отрицательного числа, стоящего на сумматоре, эквивалентная цифре 0, передается в добавку в виде цифры 0. Цифра 0 кода отрицательного числа, эквивалентная цифре -1 , передается в добавку в виде цифры 1. При этом происходит увеличение числа, стоящего в добавке, на 2 единицы дополнительного разряда, т. е. на $2 \cdot 2^{-(n+1)} = 2^{-n}$. Как сейчас будет доказано, это увеличение компенсируется выключением циклической передачи при следующем сложении.

Прежде всего докажем методом конечной индукции, что после каждого k -го ($k = 1, 2, \dots, n + 1$) сдвига на сумматоре стоит код числа, не превосходящего по абсолютной

величине x ; если же в первом дополнительном разряде стоит число 1, то оно не превосходит по абсолютной величине $|x| - 2^{-n}$.

В самом деле, это имеет место при $k = 1$, т. е. после первого сдвига: в начальный момент на сумматоре стоит $\pm x$; добавляя число $\pm x$, получаем на сумматоре 0 или $2x$ (число, оканчивающееся нулем); после сдвига в дополнительном разряде будет 0, а на сумматоре — число, не превосходящее x по абсолютной величине.

Пусть это имеет место после $(k - 1)$ -го сдвига; докажем, что при $k \leq n + 1$ это имеет место и после k -го сдвига. Если в дополнительном разряде после $(k - 1)$ -го сдвига стоит 0, то на сумматоре стоит число, не превосходящее $|x|$ по абсолютной величине. Добавление x или $-x$ на сумматоре происходит по обычному правилу сложения на сумматоре обратного кода, и результат сложения по абсолютной величине не превосходит $2x$. Если же в дополнительном разряде стоит 1, то на сумматоре стоит число, по абсолютной величине не превосходящее $|x| - 2^{-n}$. От добавления числа x или $-x$ при выключении циклической передачи появится число, по абсолютной величине не превосходящее $|x| + (|x| - 2^{-n}) + 2^{-n} = 2|x|$ (выключение циклической передачи может изменить абсолютную величину результата не больше чем на 2^{-n}). Итак, после k -го сложения стоящее на сумматоре число не превосходит по абсолютной величине $2|x|$. Сдвиг на один разряд вправо числа, стоящего на сумматоре, означает его деление на 2 (точное или с недостатком по абсолютной величине). Итак, после k -го сдвига на сумматоре стоит число, не превосходящее числа x по абсолютной величине. Если же в дополнительном разряде оказалась 1, то сдвиг был делением с недостатком по абсолютной величине; число, стоящее на сумматоре, по абсолютной величине меньше x , т. е. не превосходит $|x| - 2^{-n}$.

Таким образом, если в дополнительном разряде стоит 1, то число, стоящее на сумматоре, всегда меньше по абсолютной величине добавляемого слагаемого x или $-x$. Если это число положительное, то циклическая передача не должна иметь места и ее выключение не влияет на результат. Если

же число, стоящее на сумматоре, отрицательно, то при добавлении числа, большего по абсолютной величине, должна иметь место циклическая передача. Ее выключение уменьшает результат на 2^{-n} и компенсирует результат сдвига, произведенного по указанному выше правилу.

З а м е ч а н и е. Описанный сдвиг вместе с выключением циклической передачи оказывается эквивалентным вычитанию числа $z_0 \cdot 2^{-n}$ из числа z , стоящего на сумматоре (z_0 — цифра знака числа z), с последующим обычным сдвигом вправо на один разряд.

С х е м а 17 (умножение)

Множимое $x = 0,110101$ Множитель $y = -0,101101$

$[-x]_3 = 11,001010$ $[y]_3 = 1,010010$

$[x]_3 = 00,110101$ $[y]'_3 = 01,010010$

$y_0 = 1$

Регистр произведения		
Сумматор	Множитель	
00,000000	01010010	В сумматор посылается код числа x
+ 00,110101		
00,110101	01010010	В сумматор посылается код $[-x]_3$
+ 11,001010		
11,111111	01010010	Сдвиг вправо на 1 разряд
+ 11,111111	10101001	В сумматор посылается код числа x
+ 00,110101		
100,110100		
↑		
00,110101	10101001	Сдвиг вправо на 1 разряд
+ 00,011010	11010100	В сумматор посылается код $[-x]_3$
+ 11,001010		
11,100100	11010100	Сдвиг вправо на 1 разряд
+ 11,110010	01101010	В сумматор посылается код $[-x]_3$
+ 11,001010		
110,111100		
↑		
10,111101	01101010	Сдвиг вправо на 1 разряд
+ 11,011110	10110101	В сумматор посылается код числа x
+ 00,110101		
100,010011		
↑		
00,010100	10110101	Сдвиг вправо на 1 разряд
+ 00,001010	01011010	В сумматор посылается код $[-x]_3$

+	11,001010	
	11,010100	01011010 Сдвиг вправо на 1 разряд
+	11,101010	00101101 В сумматор посылается код числа x
	00,110101	
	100,011111	
		↑
	00,100000	00101101 Сдвиг вправо на 1 разряд
+	00,010000	00010110 В сумматор посылается код $[-x]_3$
	11,001010	
	11,011010	00010110

$$xy = -0,100101$$

С х е м а 18 (умножение)

Множимое $x = 0,110101$ Множитель $y = -0,101101$
 $[-x]_3 = 11,001010$ $[y]_3 = 1,010010$
 $[y]'_3 = 01,010010$

Регистр произведения		
Сумматор	Множитель	
+	00,000000	0 01010010 В сумматор посылается код числа x
	00,110101	
+	00,110101	0 01010010 В сумматор посылается код $[-x]_3$
	11,001010	
	11,111111	0 01010010 Сдвиг вправо на 1 разряд
+	11,111111	0 00101001 В сумматор посылается код числа x
	00,110101	
	100,110100	0 00101001
		↑
	00,110101	0 00101001 Сдвиг вправо на 1 разряд
	00,011010	1 00010100 В сумматор посылается код $[-x]_3$
+	11,001010	
	11,100100	1 00010100 Сдвиг вправо на 1 разряд
	11,110010	1 10001010 В сумматор посылается код $[-x]_3$
+	11,001010	
	10,111100	1 10001010 Циклическая передача выключена. Сдвиг
	11,011110	1 11000101 В сумматор посылается код числа x
+	00,110101	
	100,010011	1 11000101 Циклическая передача выключена
	00,010011	1 11000101 Сдвиг вправо на 1 разряд
+	00,001001	1 11100010 В сумматор посылается код $[-x]_3$
	11,001010	
	11,010011	1 11100010 Сдвиг вправо на 1 разряд

$$\begin{array}{r}
+ \begin{array}{l} 11,101001 \\ 00,110101 \end{array} \quad 0 \quad 11110001 \quad \text{В сумматор посылается код числа } x \\
\hline
\begin{array}{l} 100,011110 \\ \uparrow \end{array} \quad 0 \quad 11110001 \\
\begin{array}{l} 00,011111 \\ 00,001111 \\ 11,001010 \end{array} \quad \begin{array}{l} 0 \\ 1 \end{array} \quad \begin{array}{l} 11110001 \\ 01111000 \end{array} \quad \begin{array}{l} \text{Сдвиг вправо на 1 разряд} \\ \text{В сумматор посылается код } [-x]_3 \end{array} \\
+ \begin{array}{l} 11,001010 \\ 11,011001 \end{array} \quad \begin{array}{l} 0 \\ 1 \end{array} \quad \begin{array}{l} 11110001 \\ 01111000 \end{array}
\end{array}$$

Главная часть произведения: $-0,100110$, добавка: $+2^{-6} \cdot 0,101111$
 $xy = -0,100101010001$

При умножении таким способом цифровых частей в машинах с плавающей запятой после произведения умножения может возникнуть необходимость в нормализации влево. Такая нормализация проводится вычитанием 1 из порядка произведения, сдвига всего произведения на один разряд влево и посылки после сдвига в последний разряд сумматора цифры знака произведения.

С х е м а 19 (умножение)

Множимое $x = -2^3 \cdot 0,1011$ Множитель $y = 2^2 \cdot 0,1001$

$[x]_3 = 11,0100$ $[y]_3 = 0,1001$

$[-x]_3 = 00,1011$ $[y]'_3 = 10,1001$

Сумматор порядка произведения	Регистр произведения		
	Сумматор	Множитель	
$ \begin{array}{r} + \\ + \frac{3}{2} \\ + \frac{5}{5} \end{array} $	00,0000	0 101001	На сумматор посылается код $[-x]_3$
	00,1011		
	00,1011	0 101001	На сумматор посылается код $[x]_3$
	11,0100		
	11,1111	0 101001	Сдвиг вправо на 1 разряд
	11,1111	0 010100	На сумматор посылается код $[-x]_3$
	00,1011		
	100,1010	0 010100	Циклическая передача
	00,1011	0 010100	Сдвиг вправо на 1 разряд
	00,0101	1 001010	На сумматор посылается код $[-x]_3$
	00,1011		
	01,0000	1 001010	Сдвиг вправо на 1 разряд

+	00,1000	0	100101	На сумматор посылается код $[x]_3$
	11,0100			
	11,1100	0	100101	Сдвиг вправо на 1 разряд
	11,1110	1	010010	На сумматор посылается код $[-x]_3$
+	00,1011			
	100,1001	1	010010	Циклическая передача выключена. Сдвиг вправо на 1 разряд
+	00,0100	1	101001	На сумматор посылается код $[x]_3$
	11,0100			
-	11,1000	1	101001	Из порядка произведения вычитается единица
	5			
-	1			
	4			
	11,1000	1	101001	Сдвиг влево на 1 разряд
	11,0001	1	010010	Передача в последний разряд сумматора
	11,0010			цифры знака произведения

Главная часть: $-0,1101$, добавка: $\cdot 2^{-4} \cdot 0,1010$

Цифровая часть произведения:

$$\begin{array}{r}
 -0,1101 \\
 +0,1010 \\
 \hline
 -0,1100110
 \end{array}$$

Произведение равно $-2^4 \cdot 0,11000110$

Деление. Процесс деления может быть организован как обращение описанного только что процесса умножения, причем попрежнему делимое отвечает произведению, делитель — множимому, частное — множителю. Частное, в зависимости от типа сумматора, получается в дополнительном или обратном коде с добавочной инвертированной цифрой знака впереди. Для машины с запятой, фиксированной перед первой цифрой, делитель должен быть по абсолютной величине больше делимого. В машине с плавающей запятой делитель должен быть нормализован, в частном может быть нарушение нормализации влево, причем признаком нарушения нормализации является совпадение обеих цифр знака частного — инвертированной и прямой (сравни стр. 51). Для нормализации частное подается на сумматор с инверсией дополнительного разряда знака, после чего нормализация вправо протекает обычным путем. Перед делением на сумматор посылается код делимого. Если знак делителя у совпадает со знаком числа, стоящего на сумматоре, то в последнюю ячейку регистра произведения посылается цифра 1,

если же эти знаки не совпадают, то цифра 0. В первом случае на сумматор посылается код числа $-y$, а во втором — код делителя y . Если предыдущая операция повторена меньше, чем $n + 2$ раза, то производится сдвиг набора, стоящего в регистре произведений, на один разряд влево и операция повторяется снова. После $n + 2$ повторений этой операции деление кончено. На сумматоре стоит код остатка; на $n + 2$ последних разрядах регистра произведения стоит соответственно дополнительный или обратный код частного, с дополнительной инвертированной цифрой знака впереди. Остаток, умноженный на $2^{(n+1)}$, получается на сумматоре после посылки туда кода делителя в случае дополнительного кода и делителя прямой или инверсной передачей (в зависимости от равенства 0 или 1 цифры знака частного) в случае обратного кода.

Для случая положительных делимого и делителя этот способ деления совпадает по существу с описанным на стр. 62.

С х е м а 20 (деление)

Делимое	$x = -$	0,100111	Делитель	$y = +$	0,110001
	$[x]_2 =$	11,011001		$[y]_2 =$	00,110001
	$[-x]_2 =$	0,100111		$-y = -$	0,110001
				$[-y]_2 =$	11,001111
				Знак делителя 0	

Сумматор	Регистр частного	
11,011001	-----	Знаки не совпадают. Посылаем 0
+ 11,011001	-----0	На сумматор передается код y
00,110001		
00,001010	-----0	Сдвиг влево на 1 разряд
00,010100	-----0	Знаки совпадают. Посылаем 1
+ 00,010100	-----01	На сумматор передается код $[-y]_2$
11,001111		
11,100011	-----01	Сдвиг влево на 1 разряд
11,000110	-----01	Знаки не совпадают. Посылаем 0
+ 11,000110	-----010	На сумматор передается код y
00,110001		
11,110111	-----010	Сдвиг влево на 1 разряд
11,101110	-----010	Знаки не совпадают. Посылаем 0

+	11,101110	----	0100	На сумматор передается код y
	00,110001			
	00,011111	----	0100	Сдвиг влево на 1 разряд
	00,111110	---	0100 -	Знаки совпадают. Посылаем 1
+	00,111110	---	01001	На сумматор посылается код $[-y]_2$
	11,001111			
	00,001101	---	01001	Сдвиг влево на 1 разряд
	00,011010	--	01001 -	Знаки совпадают. Посылаем 1
+	00,011010	--	010011	На сумматор передается код $[-y]_2$
	11,001111			
	11,101001	--	010011	Сдвиг влево на 1 разряд
	11,010010	-	010011 -	Знаки не совпадают. Посылаем 0
+	11,010010	-	0100110	На сумматор передается код y
	00,110001			
	00,000011	-	0100110	Сдвиг влево на 1 разряд
	00,000110	0100110 -		Знаки совпадают. Посылаем 1
+	00,000110	01001101		На сумматор передается код $[-y]_2$
	11,001111			
+	11,010101			На сумматор посылается код делителя (восстановление остатка)
	00,110001			
	00,000110			
				Частное: $-0,110011$
				Остаток: $+2^{-7} \cdot 0,000110$

С х е м а 21 (деление)

Делимое	$x = -$	0,100111	Делитель	$y = -$	0,110001
	$[x]_3 =$	11,011000		$[y]_3 =$	11,001110
				$[-y]_3 =$	00,110001
				Знак делителя	1

Сумматор	Регистр частного	
11,011000	-----	Знаки совпадают. Посылаем 1
+	11,011000	-----1
	00,110001	На сумматор передается код $[-y]_3$
	100,001001	
	↑	
	00,001010	-----1
	00,010100	-----1 -
+	00,010100	-----10
	11,001110	
	11,100010	-----10
	11,000101	-----10 -
		Сдвиг
		Знаки совпадают. Посылаем 1

+	11,000101	----- 101	На сумматор передается код $[-y]_3$
	00,110001		
	11,110110	----- 101	Сдвиг
	11,101101	---- 101 -	Знаки совпадают. Посылаем 1
+	11,101101	---- 1011	На сумматор передается код $[-y]_3$
	00,110001		
	100,011110		
		↑	
	00,011111	---- 1011	Сдвиг
	00,111110	--- 1011 -	Знаки не совпадают. Посылаем 0
+	00,111110	--- 10110	На сумматор передается код y
	11,001110		
	100,001100		
		↑	
	00,001101	--- 10110	Сдвиг
	00,011010	-- 10110 -	Знаки не совпадают. Посылаем 0
+	00,011010	-- 101100	На сумматор передается код y
	11,001110		
	11,101000	-- 101100	Сдвиг
	11,010001	- 101100 -	Знаки совпадают. Посылаем 1
+	11,010001	- 1011001	На сумматор передается код $[-y]_3$
	00,110001		
	100,000010		
		↑	
	00,000011	- 1011001	Сдвиг
	00,000110	1011001 -	Знаки не совпадают. Посылаем 0
+	00,000110	10110010	На сумматор передается код y
	11,001110		
+	11,010100		На сумматор передается код y
	11,001110		
	110,100010		
		↑	
	10,100011		Нормализация
	11,010001		

Частное: $+ 0,110010$

Остаток: $- 2^{-6} \cdot 0,101110$

Извлечение квадратного корня. Рассмотрим операцию извлечения квадратного корня в системе $(1, \bar{1})$. Пусть требуется найти корень квадратный из числа x . Укажем алгоритм последовательного получения цифр $\tilde{y}_0, \tilde{y}_1, \dots$ разложения корня в системе $(1, \bar{1})$. Код числа x подается на сум-

матор и происходит сдвиг кода вправо на один разряд. Допустим, что уже определены цифры $\tilde{y}_0, \tilde{y}_1, \dots, \tilde{y}_{s-1}$ разложения корня в системе $(1, \bar{1})$. Цифра \tilde{y}_s определяется знаком остатка $z^{(s)}$ (числа на сумматоре): $\tilde{y}_s = 1$, если $z^{(s)} \geq 0$ и $\tilde{y}_s = \bar{1}$, если $z^{(s)} < 0$. Затем из остатка $z^{(s)}$ вычитается произведение $\tilde{y}_s \left(\sum_{l=0}^s \tilde{y}_l 2^{-l} - \tilde{y}_s 2^{-(s+1)} \right)$, т. е. вычитается или прибавляется в зависимости от знака остатка $z^{(s)}$ уже полученное приближенное значение корня $y^{(s)} = \sum_{l=0}^s \tilde{y}_l 2^{-l}$ с поправкой $-\tilde{y}_s 2^{-(s+1)}$, т. е. величина, разложение которой в системе $(1, \bar{1})$ будет

$$\tilde{y}_0, \tilde{y}_1, \tilde{y}_2, \dots, \tilde{y}_{s-1}, \tilde{y}_s (-\tilde{y}_s) 1 \bar{1} \bar{1} \bar{1} \dots$$

После операции вычитания следует сдвиг на сумматоре на один разряд влево. Полученный на сумматоре код является кодом следующего остатка $z^{(s+1)}$, знак которого определяет следующую цифру \tilde{y}_{s+1} корня и т. д.

Перейдем к обоснованию приведенного алгоритма. Пусть

$$y = \tilde{y}_0, \tilde{y}_1, \tilde{y}_2, \dots, \tilde{y}_k, \dots = \sum_{k=0}^{\infty} \tilde{y}_k 2^{-k}$$

— точный корень и его разложение в системе $(1, \bar{1})$. Очевидно

$$\begin{aligned} \frac{1}{2} y^2 &= \frac{1}{2} \sum_{k=0}^{\infty} \sum_{l=0}^{\infty} \tilde{y}_k \tilde{y}_l 2^{-(k+l)} = \sum_{k=0}^{\infty} \tilde{y}_k 2^{-k} \sum_{l=0}^k \tilde{y}_l 2^{-l} - \frac{1}{2} \sum_{k=0}^{\infty} 2^{-2k} = \\ &= \sum_{k=0}^{\infty} \tilde{y}_k 2^{-k} \left(\sum_{l=0}^k \tilde{y}_l 2^{-l} - \tilde{y}_k 2^{-(k+1)} \right) \end{aligned}$$

Если в процессе извлечения корня цифры $\tilde{y}_0, \dots, \tilde{y}_{s-1}$ определены правильно, то

$$z^{(s)} = 2^s \sum_{k=s}^{\infty} \tilde{y}_k 2^{-k} \left(\sum_{l=0}^k \tilde{y}_l 2^{-l} - \tilde{y}_k 2^{-(k+1)} \right)$$

Действительно, $z^{(0)} = \frac{1}{2} y^2$ и, если

$$z^{(s-1)} = 2^{(s-1)} \sum_{k=s-1}^{\infty} \tilde{y}_k 2^{-k} \left(\sum_{l=0}^k \tilde{y}_l 2^{-l} - \tilde{y}_k 2^{-(k+1)} \right)$$

и цифра \tilde{y}_{s-1} определена верно, то

$$\begin{aligned} z^{(s)} &= 2 \left(z^{(s-1)} - \tilde{y}_{s-1} \left(\sum_{l=0}^{s-1} \tilde{y}_l 2^{-l} - \tilde{y}_k 2^{-(k+1)} \right) \right) = \\ &= 2^s \sum_{k=s}^{\infty} \tilde{y}_k 2^{-k} \left(\sum_{l=0}^k \tilde{y}_l 2^{-l} - \tilde{y}_k 2^{-(k+1)} \right) \end{aligned}$$

Таким образом, и цифра \tilde{y}_s будет определена правильно, если знак суммы $\sum_{k=s}^{\infty} \tilde{y}_k 2^{-k} \left(\sum_{l=0}^k \tilde{y}_l 2^{-l} - \tilde{y}_k 2^{-(k+1)} \right)$ определяется знаком ее первого члена $\tilde{y}_s 2^{-s} \left(\sum_{l=0}^s \tilde{y}_l 2^{-l} - \tilde{y}_s 2^{-(s+1)} \right)$. Для того, чтобы доказать это, достаточно показать, что

$$\begin{aligned} 2^{-s} \left(\sum_{l=0}^s \tilde{y}_l 2^{-l} - \tilde{y}_s 2^{-(s+1)} \right) &= \sum_{k=s+1}^{\infty} 2^{-k} \left(\sum_{l=0}^s \tilde{y}_l 2^{-l} - \tilde{y}_s \sum_{l=s+1}^k 2^{-l} + \right. \\ &\quad \left. + \tilde{y}_s 2^{-(k+1)} \right) \end{aligned} \quad (A)$$

В самом деле, так как $\sum_{l=0}^k \tilde{y}_l 2^{-l} - \tilde{y}_k 2^{-(k+1)} > 0$, из выписанного равенства следует, что абсолютная величина первого члена в ряде, определяющем $z^{(s)}$, больше суммы любого конечного числа следующих за ним членов противоположного знака. Отсюда очевидно следует, что знак остатка $z^{(s)}$ совпадает со знаком \tilde{y}_s . Равенство (A) сводится к равенству

$$2^{-2s-1} = \sum_{k=s+1}^{\infty} 2^{-k} \sum_{l=s+1}^k 2^{-l} - \sum_{k=s+1}^{\infty} 2^{-2k-1}$$

Но

$$\begin{aligned} \sum_{k=s+1}^{\infty} 2^{-k} \sum_{l=s+1}^k 2^{-l} - \sum_{k=s+1}^{\infty} 2^{-2k-1} &= 2^{-2s} \sum_{l=1}^{\infty} 2^{-l} \sum_{k=l}^{\infty} 2^{-k} - \\ - 2^{-2s-1} \sum_{k=1}^{\infty} 2^{-2k} &= 2^{-2s+1} \sum_{l=1}^{\infty} 2^{-2l} - 2^{-2s-1} \sum_{k=1}^{\infty} 2^{-2k} = \\ &= 3 \cdot 2^{-2s-1} \cdot \frac{2^{-2}}{1-2^{-2}} = 2^{-2s-1} \end{aligned}$$

что и требовалось доказать.

Замечания. 1. Получаемые последовательно цифры \tilde{y}_k разложения корня в системе $(1, \bar{1})$ можно считать цифрами y_{k-1} обычного двоичного разложения, воспринимая цифру $\bar{1}$ как 0 и 1 как 1. Для определения следующей цифры в соответствии с правилом перехода от системы $(1, \bar{1})$ к обычной двоичной системе из остатка вычитается полученное уже приближенное значение корня с приписанными сзади цифрами 01, если остаток положителен, и прибавляется полученное уже приближенное значение корня с приписанными сзади цифрами 11, если остаток отрицателен.

2. Цифры \tilde{y}_0 и \tilde{y}_1 известны заранее и поэтому можно в сущности начинать извлечение корня с определения цифры \tilde{y}_2 . Для получения остатка $z^{(2)}$, как нетрудно показать, достаточно вычесть из подкоренного выражения 0,01 и сдвинуть полученный на сумматоре код на один разряд влево. Дальнейший процесс извлечения корня может продолжаться по описанной выше схеме.

§ 14. ДРУГИЕ ОПЕРАЦИИ

Взятие целой части числа. Будем обозначать через $E(x)$ и $\{x\}$ соответственно целую и дробную части числа x ; например, $E(\pi) = 3$, $\{\pi\} = 0,14159 \dots$, $E(-\pi) = -4$, $\{-\pi\} = 0,85840 \dots$. Во многих задачах возникает необходимость расщепить число на целую и дробную части. Например, при вычислении значений показательной функции 2^x , целая и дробная части x играют разную роль в вычислении. При нахождении значений функции $f(x)$ по таблице с шагом 1,

целая часть x служит для поисков в таблице, а дробная часть x служит аргументом интерполяционного многочлена. Операция выделения целой части играет существенную роль при переводе чисел из двоичной системы в десятичную (см. § 16). Часто целая часть нужна именно как набор цифр — например, в последнем случае.

В машинах с фиксированной запятой расщепление числа на целую и дробную части или, общее, на часть, образованную первыми цифрами, и часть, образованную последними цифрами, может быть сведена к операции сличения. Кроме того, для выделения группы первых p цифр можно воспользоваться сдвигом вправо на $(n - p)$ разрядов. Аналогично может быть выделена любая группа цифр, следующих друг за другом.

В машине с плавающей запятой числа $x = 2^p \cdot 0, x_1 x_2 \dots x_n$ имеют целую часть, равную нулю, если порядок $p \leq 0$. Если порядок $p > 0$, то целая часть x записывается первыми p цифрами его цифровой части $E(x) = x_1 x_2 \dots x_p$. В этом случае порядок p выполняет функции управления процессом выделения группы первых p цифр.

Если эта целая часть должна фигурировать как нормализованное число, то выделенная группа цифр должна занимать первые p разрядов после запятой, а в качестве порядка удерживается число p . Если порядок больше числа n цифр в машине, то такая операция обычно не имеет смысла.

Операция выделения целой и дробной частей может быть сведена к арифметическим действиям, выполняемым машиной.

В машинах с фиксированной запятой при умножении числа x на число 2^{-k} ($0 < k < n$) способом, описанным на стр. 58, мы получим на сумматоре первые k цифр числа x , а в дополнительных разрядах регистра произведения — остальные $(n - k)$ цифр.

В машинах с плавающей запятой, если наибольший возможный порядок n_1 не меньше числа n цифр после запятой, операция выделения целой части x в нормализованном виде сводится к сложению $s = x + 2^{n-1}$ и к вычитанию $E(x) = s - 2^{n-1}$. В самом деле, 2^{n-1} записывается в нормали-

зованном виде: $2^{n-1} = 2^n \cdot 0,1$. Если $x = 2^p \cdot 0, x_1 x_2 \dots x_n, p < n$, то

$$s = 2^n \cdot (0,1 + \underbrace{0,0 \dots 0}_{n-p} x_1 x_2 \dots x_p) = 2^n \cdot (0,1 \underbrace{00 \dots 0}_{n-p-1} x_1 x_2 \dots x_p)$$

$$\text{далее, } s - 2^{n-1} = 2^n (0,1 \underbrace{00 \dots 0}_{n-p-1} x_1 x_2 \dots x_p - 0,1) =$$

$$= 2^n \cdot 0, \underbrace{00 \dots 0}_{n-p} x_1 x_2 \dots x_p = 2^p \cdot 0, x_1 x_2 \dots x_p 00 \dots 0$$

Если наибольший порядок есть число $n - \alpha < n$, то при $x = 2^p \cdot 0, x_1 x_2 \dots x_n$

$$E(x) = [(2^{-\alpha} x + 2^{n-\alpha-1}) - 2^{n-\alpha-1}] : 2^{\alpha}$$

Операции с порядками. Выделение порядка в виде числа в нормализованном виде заключается в преобразовании числа $x = y \cdot 2^m$ в число $z = m$, причем целое число m в свою очередь может быть представлено в нормализованной форме. Рассмотрим подробнее эту операцию, предполагая, что и цифровая часть и порядок числа x представлены прямым кодом и притом так, что разряды знака и цифр порядка являются низшими разрядами набора $[x] = y_0 y_1 y_2 \dots y_k \dots y_n m_0 m_1 m_2 \dots m_r$, представляющего число x .

При таком представлении числа x передача знака и цифр его порядка в цифровую часть может быть осуществлена очень просто — посредством лишь одной операции сдвига влево на $n + 1$ разряд. В результате этой операции сдвига получим набор $m_0 m_1 m_2 \dots m_r 00 \dots 0$, представляющий прямой код числа $m \cdot 2^{-r}$ (в порядке стоит 0).

Для получения набора, представляющего искомое число $z = m$, необходимо еще, очевидно, в разряды порядка поместить код $\rho_0 \rho_1 \dots \rho_r$ числа r (заданный раз навсегда в данной машине), после чего получим код числа z : $m_0 m_1 m_2 \dots m_r 00 \dots 0 \rho_0 \rho_1 \dots \rho_i \dots \rho_r$. Полученное таким образом число z не является, вообще говоря, нормализованным. Нормализация числа $z = m$ производится совершенно так же, как и всех других чисел. Если разряды знака и цифр порядка являются высшими разрядами набора $m_0 m_1 m_2 \dots m_r y_0 y_1 \dots y_k \dots y_n$, представляющего число x , то операция выделения порядка

m числа x несколько усложняется из-за необходимости предварительного гашения цифровой части числа x . Это гашение можно осуществить посредством операции сдвига изображения числа x на $n + 1$ разряд вправо. Получим набор

$$00 \dots 0 m_0 m_1 m_2 \dots m_r$$

Сдвигая его на $(n - r)$ разрядов влево, получим набор

$$\underbrace{00 \dots 0}_{r+1} m_0 m_1 m_2 \dots m_r 00 \dots 0$$

представляющий код числа $m \cdot 2^{-r}$.

Если в первые $(r + 1)$ разрядов посылается изображение $\rho_0 \rho_1 \dots \rho_r$ числа r , то в результате описанных операций получим набор

$$\rho_0 \rho_1 \rho_2 \dots \rho_r m_0 m_1 m_2 \dots m_r 00 \dots 0$$

представляющий число $z = m$, которое остается только нормализовать¹.

Рассмотренная операция используется при логарифмировании, а также при некоторых других операциях.

Операция преобразования высших разрядов цифровой части числа в порядок другого числа заключается в преобразовании числа $x = u \cdot 2^m$ в число $z = u \cdot 2^{\bar{y}}$, где \bar{y} — число, представляемое r высшими разрядами цифровой части u числа x , u — цифровая часть числа z , причем величина ее безразлична.

Наиболее просто осуществляется эта операция в том случае, когда и порядок и цифровая часть числа x представлены одним кодом. Пусть разряды знака и цифр порядка являются высшими разрядами набора

$$m_0 m_1 m_2 \dots m_r u_0 u_1 \dots u_r \dots u_n \quad (1)$$

представляющего число x .

В этом случае для осуществления передачи всех $r + 1$ разрядов $u_0 u_1 u_2 \dots u_r$ цифровой части числа в его порядок

¹ Применяя обратный код и модифицированный сдвиг, мы после первого сдвига получим набор $m_0 m_0 \dots m_0 m_1 m_2 \dots m_r$, представляющий число $2^{-n} \cdot m$. Умножив его на 2^n с нормализацией результата, получим m .

достаточно просто сдвинуть все разряды набора (1) на $r + 1$ место влево. Тогда получим набор:

$$y_0 y_1 y_2 \dots y_r y_{r+1} \dots y_n 00 \dots 0$$

первые $r + 1$ разрядов которого представляют порядок \bar{y} , а следующие разряды $y_{r+1} \dots y_n 00 \dots 0$ — цифровую часть u числа z . Так как цифровая часть числа z безразлична, то нормализация не нужна и полученное число z является окончательным результатом.

Если разряды знака и цифр порядка являются низшими разрядами набора $y_0 y_1 y_2 \dots y_r \dots y_n m_0 m_1 m_2 \dots m_r$, представляющими число x , то операция передачи $r + 1$ высших разрядов $y_0 y_1 y_2 \dots y_r$ набора (1) в порядок числа x может быть произведена посредством сдвига всех разрядов этого набора на $n + 1$ место вправо.

В результате такого сдвига получим набор

$$000 \dots 0 \dots 0 y_0 y_1 y_2 \dots y_r$$

представляющий число z с „нулевой“ цифровой частью:

$$z = 0 \cdot 2^{\bar{y}}$$

т. е. представляющее число нуль.

Так как значение цифровой части безразлично, то нормализация, как и во всех случаях применения этой операции, производиться не должна.

Операция передачи цифровой части числа в его порядок может использоваться при операции потенцирования.

Из других операций с порядками укажем на следующие: прибавление к порядку или вычитание из него порядка другого числа; присоединение к цифровой части одного числа порядка другого числа.

Операция сравнения. Даны два числа a и b ; требуется указать на большее (меньшее) из этих чисел. Очевидно, для этой цели достаточно произвести вычитание. Знак разности $a - b$ покажет, какое число больше: если $a - b > 0$, то $a > b$; если $a - b < 0$, то $b > a$.

Если в сумматоре применяется дополнительный код для

изображения отрицательных чисел, то (см. § 7) нуль изображается как $+0$: $[0]_2 = 0,00 \dots 0$.

Если применяется обратный код для изображения отрицательных чисел, то 0 , полученный на сумматоре при вычитании равных чисел, есть отрицательный нуль:

$$[-0]_3 = 1,11 \dots 1$$

Таким образом, случай равенства $a = b$ воспринимается при дополнительном коде так же, как неравенство $a > b$, а при обратном коде так же, как неравенство $a < b$. Другими словами, знак $a - b$ определяет следующие дизъюнкции: при дополнительном коде $a \geq b$ ($a - b$ имеет знак $+$) или $a < b$ ($a - b$ имеет знак $-$); при обратном коде $a > b$ ($a - b$ имеет знак $+$) или $a \leq b$ ($a - b$ имеет знак $-$).

Для того, чтобы получить первую из этих дизъюнкций при обратном коде и вторую при дополнительном коде, надо произвести вычитание $b - a$. Можно в ряде случаев избежать вычитания. Именно, если $a > 0$, а $b < 0$, то всегда $a > b$. Если a и b нормализованы, имеют одинаковые знаки, но разные порядки (в машинах с плавающей запятой), то $a > b$, если порядок числа a больше порядка числа b и $a > 0, b > 0$ или если порядок числа b больше порядка числа a и $a < 0, b < 0$. Наконец, при одинаковых знаках и равных порядках для сравнения a и b требуется произвести вычитание $a - b$ (без выравнивания порядков).

В некоторых машинах (например, в раскладочной машине из комплекта счетно-аналитических) сравнение чисел заключается в последовательном сравнении их цифр, начиная со старших разрядов. Из двух положительных чисел больше то, у которого первая неодинаковая цифра больше (при двоичном счислении — то, у которого первая неодинаковая цифра есть 1). Для отрицательной пары чисел это правило очевидным образом изменяется.

§ 15. ТОЧНОСТЬ ПРОИЗВОДСТВА ОПЕРАЦИЙ И ОКРУГЛЕНИЕ РЕЗУЛЬТАТА

Операция сдвига. Операция сдвига, рассматриваемая как арифметическая операция, есть умножение числа на степень

двойки (см. § 9). Арифметической операцией является по существу лишь модифицированный сдвиг, и поэтому здесь мы остановимся только на нем, называя его в дальнейшем просто сдвигом. Говоря о сдвиге влево, мы будем иметь в виду допустимый сдвиг, т. е. сдвиг, не выводящий за рас- полагаемые разряды. При таком сдвиге влево не происходит потери значащих цифр и поэтому такой сдвиг есть точная операция умножения на положительную степень двойки. Напротив, при сдвиге вправо на p разрядов происходит потеря цифр последних p разрядов и результат операции отличен от результата точного деления на 2^p . Абсолютная ошибка меньше единицы последнего разряда, однако может в наиболее неблагоприятном случае отличаться от единицы последнего разряда лишь коэффициентом $(1 - 2^{-p})$.

В машинах с сумматором, работающим по дополнительному коду, сдвиг вправо есть деление на степень двух, как правило, с недостатком. В машинах с сумматором, работающим в обратном коде, такой сдвиг есть деление, вообще говоря, с недостатком в абсолютной величине результата.

Для того, чтобы уменьшить ошибку сдвига вправо, можно применять процесс округления. Одним из возможных способов такого округления является следующий. На сумматоре вводится дополнительный разряд, т. е. число цифровых разрядов сумматора, на единицу больше числа цифр чисел, хранящихся в машине. При передаче числа на сумматор в дополнительный разряд передается цифра 0 для сумматора дополнительного кода и цифра знака числа для сумматора обратного кода. Введение такого дополнительного разряда эквивалентно приписыванию последней цифры 0 ко всем числам, хранящимся в ЗУ машины. Если после сдвига вправо послать единицу в дополнительный разряд сумматора в случае дополнительного кода, а в случае обратного кода добавлять или вычитать (в зависимости от цифры 0 или 1 знака числа на сумматоре) единицу в дополнительном разряде, а затем отбросить дополнительный разряд, то максимально возможная ошибка деления на степень двух с помощью сдвига вправо

уменьшается до половины единицы последнего разряда результата.

Для дополнительного кода результат сдвига с таким округлением отличается от результата сдвига без округления только тогда, когда в дополнительном разряде оказывается цифра 1. При этом результат сдвига с округлением на единицу последнего разряда больше результата сдвига без округления, а последний содержит отрицательную ошибку, абсолютная величина которой не меньше половины единицы последнего двоичного разряда. Таким образом ошибка результата сдвига с округлением не превосходит половины единицы последнего разряда. Если же в дополнительном разряде оказывается цифра ноль, то ошибка результата сдвига без округления не превосходит половины единицы последнего двоичного разряда, а так как в этом случае результат сдвига без округления совпадает с результатом сдвига с округлением, то ошибка в последнем и в этом случае не превосходит половины единицы последнего разряда результата.

В случае обратного кода, если цифра знака числа совпала с цифрой в дополнительном разряде, то результат сдвига с округлением совпадает с результатом сдвига без округления, но в этом случае ошибка в последнем не превосходит половины единицы последнего разряда. Если цифра знака числа отлична от цифры в дополнительном разряде, то абсолютная величина результата сдвига без округления на единицу последнего разряда меньше результата сдвига с округлением, но в этом случае абсолютная величина результата сдвига без округления получается с недостатком, равным, по крайней мере, половине последнего разряда. Так что и в этом случае ошибка результата сдвига с округлением не превосходит половины последнего разряда. При таком способе округления результата сдвига два последовательно проведенных сдвига вправо на p и q разрядов не эквивалентны одному сдвигу вправо на $p + q$ разрядов. Как для сдвигов без округлений, так и для сдвигов с округлениями ошибка результата нескольких последовательно проведенных сдвигов вправо не превосходит единицы последнего двоичного разряда результата.

Если один из двух последовательно проведенных сдвигов есть сдвиг вправо на p разрядов, а второй — сдвиг влево на q разрядов, то такие два сдвига есть деление на 2^{p-q} . Если сдвиг влево происходит первым, то ошибка может появиться только при $p > q$. Такая ошибка не превосходит единицы последнего разряда для сдвигов без округления и половины этого разряда для сдвигов с округлением. Если сдвиг вправо предшествует сдвигу влево, то ошибка для сдвигов без округлений может достигнуть единицы $(q+1)$ -го двоичного разряда, умноженной на $(1 - 2^{-p})$, считая с конца. Для сдвигов на сумматоре с дополнительным разрядом такое переползание ошибки на старшие двоичные разряды может произойти только при $p > 1$. Наличие дополнительного разряда сводит к нулю ошибку при $p = 1$ и $q > 0$. Но и при $p > 1$ дополнительный разряд уменьшает максимально возможную ошибку вдвое.

Изменение порядка и нормализация. В машинах с плавающей запятой операция изменения порядка числа и, в частности, нормализация есть сдвиг, сопровождаемый уменьшением (при сдвиге влево) или увеличением (при сдвиге вправо) порядка на число, равное числу разрядов, на которое произошел сдвиг. Так как прибавление к порядку числа x числа p эквивалентно точному умножению числа x на 2^p , то сдвиг, сопровождаемый указанным изменением порядка, должен оставлять число, подвергаемое операции, без изменений. Однако сдвиг вправо есть деление с ошибкой и поэтому сдвиг вправо на p разрядов, сопровождаемый компенсирующим увеличением порядка, дает ошибку, абсолютная величина которой в 2^p раз больше ошибки одного сдвига. Относительная ошибка, конечно, остается прежней. Это значит, что ошибка при увеличении порядка числа на p не превосходит умноженной на $(1 - 2^{-p})$ единицы последнего двоичного разряда результата, если нет округления, и не превосходит половины этой единицы для операции с округлением. Так как нормализация вправо есть увеличение порядка на $p = 1$, то процесс округления не уменьшает максимум возможной ошибки при нормализации n -значных чисел. Однако опера-

ция нормализации обычно применяется к числам, у которых есть цифра в дополнительном разряде сумматора, а следующие цифры могли быть отброшены. В этих условиях процесс округления результата нормализации оказывается целесообразным. Кроме этого, в пользу процесса округления даже при нормализации n -значных чисел говорит то, что без округления ошибки всех нормализаций будут складываться, в то время как при округлении возможна компенсация ошибок нормализаций.

Сложение и вычитание. В машинах с фиксированной запятой операции сложения и вычитания являются точными операциями. В машинах с плавающей запятой эти операции содержат сдвиги и, следовательно, не являются точными. В процессе сложения и вычитания чисел в машинах с плавающей запятой сдвиги возникают, во-первых, при выравнивании порядков, во-вторых, при нормализации результата. Если в сумматоре нет дополнительного разряда для округления, то выравнивание порядков дает ошибку, меньшую единицы последнего знака, и если не происходит нормализация результата влево, то ошибка останется попрежнему меньше единицы последнего знака. Однако, если результат нормализуется влево, то ошибка может превратиться даже в единицу первого знака, т. е. перейдет в порядок числа. Например, пусть $n = 6$, $\alpha = 2^3 \cdot 0,100000$, $\beta = 2^2 \cdot 0,111111$ и требуется найти $\alpha - \beta$. В результате выравнивания порядков β заменяется числом $\beta_1 = 2^3 \cdot 0,011111$, $\alpha - \beta_1 = 2^3 \cdot 0,000001 = = 2^{-2} \cdot 0,100000$, точное значение разности есть $2^{-3} \cdot 0,1$. Наличие в сумматоре дополнительного разряда мешает такому росту ошибки. Действительно, нормализация после сложения больше чем на один разряд влево может возникнуть лишь в том случае, когда порядки слагаемых отличались не больше чем на единицу. Но в этом случае дополнительный разряд обеспечивает сохранение всех цифр и, следовательно, точность результата. Если после сложения происходит нормализация влево лишь на один разряд, то ошибка результата не превосходит единицы последнего двоичного знака, каков бы ни был сдвиг вправо при выравнивании порядков.

При сложении из-за сдвига вправо того из слагаемых,

порядок которого меньше, происходит отбрасывание последних цифр его кода, а затем суммирование. Для сумматора, работающего по дополнительному коду, результат получается такой же, как если бы мы провели суммирование с учетом всех цифр, т. е. точно, и только после этого отбросили бы последние цифры. Это означает, что возникает такая же ошибка, как ошибка при сдвиге, метод уменьшения которой при помощи округлений мы рассмотрели выше, говоря о сдвигах. Поэтому для сумматора, работающего по дополнительному коду, округление результата посылкой единицы в дополнительный разряд сумматора в том случае, когда нормализации влево не происходит¹, приводит к тому, что максимально возможная ошибка не может превзойти половины единицы последнего разряда результата.

Аналогичные соображения применимы к сумматору, работающему в обратном коде, только в том случае, когда складываются числа одинакового знака (ср. § 12 стр. 75). При сложении чисел разных знаков возникшая ошибка складывается из значений группы цифр, отброшенной при выравнивании порядков, и значения цифры дополнительного разряда. Эти ошибки имеют в рассматриваемом случае противоположные знаки и каждая из них по абсолютной величине не превосходит половины единицы последнего двоичного разряда результата. Следовательно, обычное округление при помощи прибавления или вычитания единицы дополнительного разряда является в этом случае излишним и может привести к увеличению ошибки, если цифра дополнительного разряда сумматора отлична от цифры знака результата.

Так как вычитание чисел $\alpha - \beta$ заменяется в машине сложением кодов чисел α и $(-\beta)$, то все сказанное может быть отнесено и к операции вычитания с той, однако, оговоркой, что в случае дополнительного кода операции сдвига и перемены знака числа не коммутативны и поэтому следует сперва получить код числа $(-\beta)$, а затем производить выравнивание порядков. В связи с тем, что операцию перемены

¹ Такую посылку можно и удобно производить во всех случаях, но после нормализации результата влево такая посылка не может повлиять на результат.

знака в дополнительном коде удобно производить при помощи инверсной передачи числа на сумматор с посылкой дополнительной единицы в младший разряд сумматора, при операции вычитания первым числом на сумматор следует подавать вычитаемое. В случае обратного кода это безразлично.

Умножение. В большинстве приведенных выше схем умножения чисел в результате операции появляется точное $2n$ -значное произведение. Однако чаще требуется иметь возможно более точное n -значное произведение. Рассмотрим методы получения такого произведения, исходя из произведения $2n$ -значного.

Если в результате умножения на регистре произведения (сумматоре и разрядах регистра множителя) получается соответствующий код $2n$ -значного произведения, то ошибка, возникающая при отбрасывании последних n цифр, такая же, как ошибка сдвига вправо, и, следовательно, дополнительный разряд сумматора и тот же процесс округления, что и при сдвигах, приведет к тому, что ошибка n -значного результата не превзойдет половины единицы последнего разряда. Конечно, если полученное в машине с плавающей запятой $2n$ -значное произведение оказалось ненормализованным, то нормализация при помощи общего сдвига на регистре произведения должна быть проведена до округления и отбрасывания последних n цифр. К схемам умножения, при которых в регистре произведения появляется код $2n$ -значного произведения, относятся следующие:

а) умножение чисел, заданных прямым кодом, (§ 11, схема 2);

б) умножение чисел, заданных дополнительным кодом (§ 12, схема 9; § 13, схема 16);

в) умножение чисел, заданных обратным кодом, по схеме 13, § 12.

В некоторых схемах умножения чисел, заданных обратным кодом, $2n$ -значное произведение получается не просто в виде $2n$ -значного кода, а состоящим из кода главной части на сумматоре и добавки на разрядах регистра множителя, которая может иметь знак, отличный от знака главной части. Обратный код $2n$ -значного произведения не получается при

этом простой припиской цифр добавки к коду главной части и операция получения округленного произведения становится несколько более сложной. Наличие дополнительного разряда на сумматоре приводит к тому, что добавка, оказывающаяся вне сумматора, становится по абсолютной величине меньше половины единицы последнего разряда n -значного результата и поэтому, если ее знак противоположен знаку результата, эта добавка может быть отброшена вместе с цифрой дополнительного разряда сумматора. При совпадении знаков главной части и добавки следует произвести обычное округление, так как в этом случае на регистре произведения оказывается обратный код $2n$ -значного произведения.

При умножении по схеме 10 § 12 и наличии дополнительного разряда сумматора операция округления сводится к тому, что после умножения инвертированная цифра знака множителя прибавляется или вычитается в дополнительном разряде сумматора в зависимости от цифры 0 или 1 знака произведения, а затем отбрасываются все цифры, начиная с $(n + 1)$ -й.

При умножении по схеме 18 § 13 та же операция сводится к прибавлению в дополнительный разряд сумматора инвертированной цифры знака произведения с последующим отбрасыванием всех цифр, начиная с $(n + 1)$ -й. Описанное округление для машин с плавающей запятой следует, как всегда, производить после нормализации. Нормализацию влево произведения, полученного по схеме 10 § 12, можно осуществить следующим способом. После общего сдвига кода на один разряд влево на регистре произведения к последнему (дополнительному) разряду сумматора прибавляется цифра знака множителя, если знак произведения отрицателен, и вычитается эта цифра, если знак произведения положителен. Процесс нормализации произведения для схемы 19 § 13 описан на стр. 99.

Для всех перечисленных схем применение дополнительного разряда сумматора и соответствующей операции округления приводит к тому, что ошибка n -значного произведения не превосходит половины единицы последнего разряда результата.

Рассмотрим теперь схемы умножения, по которым в результате получается укороченное произведение. К таким способам умножения относятся:

а) умножение чисел, заданных обратным кодом, по схеме на стр. 74 § 12;

б) умножение чисел, заданных обратным кодом, по схеме 17 § 13.

Умножение чисел, заданных обратным кодом по схеме на стр. 74 § 12, даст при отрицательном множителе ошибку, не превосходящую двух единиц последнего двоичного разряда n -значного результата, если после умножения не происходит нормализации влево, и не превосходящую двух единиц предпоследнего двоичного разряда n -значного результата, если такая нормализация происходит. Максимально возможная ошибка умножения по той же схеме на положительный множитель вдвое меньше. Ошибка при умножении по этой схеме всегда имеет знак множимого и, следовательно, при отрицательном множителе — знак противоположный знаку произведения. Поэтому ошибка в случае отрицательного множителя не возрастает, если мы отбросим последнюю цифру результата. Итак, применение дополнительного разряда на сумматоре с последующим округлением в случае положительного множителя и отбрасыванием цифры дополнительного разряда при отрицательном множителе уменьшает максимально возможную ошибку до половины единицы последнего разряда результата при положительном множителе и единицы этого разряда при отрицательном множителе, если после умножения не происходит нормализации влево. Наличие такой нормализации увеличивает ошибку вдвое. При умножении чисел, заданных обратным кодом по схеме 17 § 13, ошибка не превосходит единицы последнего разряда результата. В связи с тем, что знак этой ошибки может быть произвольным, назначение дополнительного разряда сводится к функции хранения последней цифры результата в том случае, когда после умножения происходит нормализация влево. Если такой нормализации не происходит, цифра дополнительного разряда просто отбрасывается.

В обоих случаях ошибка не превосходит единицы последнего разряда результата.

Деление. Во всех рассмотренных выше схемах деления определялся конечный n -значный отрезок бесконечного кода частного. Возникающая при этом ошибка, очевидно, такая же, как и ошибка сдвига вправо. Отсюда следует, что применение дополнительного разряда в регистре частного и определение одной лишней цифры кода частного, с последующей посылкой частного на сумматор для нормализации вправо и округления, дает возможность получить n -значный результат, отличающийся от истинного частного не более чем на половину единицы последнего разряда.

Глава III

НЕКОТОРЫЕ СТАНДАРТНЫЕ ПРОЦЕССЫ ПРИ АВТОМАТИЧЕСКИХ ВЫЧИСЛЕНИЯХ

§ 16. СТАНДАРТНЫЕ ЗАДАЧИ

Мы сейчас остановимся коротко на некоторых стандартных арифметических процессах, являющихся составной частью многих более сложных математических задач.

Нахождение сумм произведений. Если процесс умножения производится по схеме сдвигов множимого (см. стр. 56), а числа на сумматоре остаются неподвижными, то такой сумматор может накапливать произведения: если результат умножения a_1b_1 остается на сумматоре, а затем происходит процесс умножения a_2b_2 , то после его окончания на сумматоре оказывается число $a_1b_1 + a_2b_2$. Продолжая такой

процесс, можно найти сумму произведений $\sum_{k=1}^n a_k b_k$ без фиксации результатов отдельных умножений (таким способом находятся суммы произведений на арифмометре).

Как выше было указано, во многих случаях предпочтительнее схема умножения, при котором сдвигам подвергаются числа на сумматоре. Если в машине принята такая схема умножения, то описанный выше способ нахождения сумм произведений неприменим, так как перед каждым умножением сумматор очищается от результатов предыдущих операций.

Процесс нахождения сумм произведений упрощается, если машина обладает специальным накапливающим сумма-

тором, не участвующим в умножении; если результат каждого умножения $a_1b_1, a_2b_2, \dots, a_nb_n$ посылать в такой сумматор, то в результате в нем появится искомая сумма произведений. После посылки на сумматор очередного произведения a_kb_k начинается процесс следующего умножения $a_{k+1}b_{k+1}$. К окончанию этого умножения должен закончиться процесс сложения на сумматоре; так как операция умножения связана с многократным суммированием, то ясно, что накапливающий сумматор может работать медленнее (особенно в машинах с фиксированной запятой) основного сумматора, участвующего в умножении. А это означает, что дополнительный сумматор может быть устроен проще.

Такие накапливающие сумматоры имеются в некоторых машинах с фиксированной запятой.

К нахождению сумм произведений сводится численный гармонический анализ, операции перемножения векторов и матриц и т. д. В некоторых случаях нахождение сумм произведений может сводиться к последовательному суммированию (см. ниже, стр. 126).

Перевод чисел из одной системы в другую. Если машина работает по двоичной системе, то вводимые в машину числа должны предварительно подвергаться преобразованию из десятичной системы в двоичную, а поступающие ответы — обратному преобразованию. Эти преобразования связаны с вводом чисел в машину и с выдачей чисел из машины. Поэтому возможно производить эти операции как вне основной машины на специальных блоках, связанных с блоками ввода и вывода чисел, так и внутри машины на ее арифметическом устройстве.

Если преобразования производит само арифметическое устройство машины, то в качестве промежуточного этапа выступает запись чисел в двоично-десятичной системе.

Пусть некоторое число x , имеющее десятичную запись

$$x = \sum_{k=1}^m a_k 10^{m-k} \quad (1)$$

должно подвергнуться преобразованию в двоичную систему.

Мы запишем число x в виде набора

$$\alpha_{11}\alpha_{12}\alpha_{13}\alpha_{14}\alpha_{21}\alpha_{22}\alpha_{23}\alpha_{24} \dots \alpha_{k1}\alpha_{k2}\alpha_{k3}\alpha_{k4} \dots \alpha_{m1}\alpha_{m2}\alpha_{m3}\alpha_{m4}$$

четверок двоичных цифр, составляющих двоичную запись цифр a_1, a_2, \dots, a_m . Эта последовательность воспринимается как некоторое двоичное число u . Например, при $x = 0,967$, $u = 0,1001\ 0110\ 0111$. Четверка цифр $\alpha_{k1}\alpha_{k2}\alpha_{k3}\alpha_{k4}$, изображающая цифру a_k , есть коэффициент при степени $(m - k)$ числа десять (1010 в двоичной записи). Чтобы найти двоичную запись числа x нужно выделять последовательно четверки цифр числа u , т. е. коэффициенты a_k в (1), и произвести вычисление значения многочлена $\sum_{k=1}^m a_k t^{m-k}$ при $t = 10$. Как

было отмечено в § 1, двоично-десятичная запись числа на 20% „длиннее“ двоичной. Например, машина, действующая с 40 двоичными знаками, что отвечает 12-значным десятичным числам, может воспринимать в двоично-десятичной записи 40-значные числа, что отвечает лишь 10-значным десятичным. Ввод полного 12-значного десятичного числа в двоично-десятичной записи может быть выполнен в два приема.

Перейдем теперь к преобразованию двоичного числа в десятичное. Пусть $x_1 = 0, \alpha_1 \alpha_2 \dots \alpha_m$, где $\alpha_i = 0$ или 1, и надо найти его десятичную запись $x_1 = \sum_{k=1}^l a_k 10^{-k}$. Очевидно, $a_1 =$

$= E(10x_1)$; соответственно, для $x_2 = \{10x_1\} = \sum_{k=2}^l a_k 10^{-k+1}$ (x_2 — дробная часть $10x_1$) будем иметь $a_2 = E(10x_2)$ и т. д.

Итак, получение последовательности a_1, a_2, \dots, a_l цифр десятичной записи совершается по схеме:

$$a_k = E(10x_k), \quad x_k = \{10x_{k-1}\} \quad (2)$$

В машинах с плавающей запятой двоичное число $x = 2^m u$, где $1/2 \leq u < 1$, мы превращаем в десятичное число с порядком s : именно, $x = 10^s v$, где $1/10 \leq v < 1$; порядок s опреде-

ляется как единственное целое число, удовлетворяющее неравенствам

$$10^s > x \geq 10^{s-1} \quad (3)$$

Отыскание этого числа можно произвести разными способами (поиском в таблице степеней 10, подбором и т. д.). Тогда $v = 10^{-s}x < 1$ и его десятичная запись найдется по формулам (2).

Таким образом, перевод чисел из одной системы в другую сводится к серии более элементарных операций, выполняемых машиной.

Операции с удвоенным числом знаков. Машины, работающие с n -значными числами, могут быть использованы и для работы с произвольным числом цифр; подобно тому, как действия над многозначными числами могут свестись к серии действий над однозначными числами, так и действия над m -значными числами сводятся к действиям над n -значными числами. Ясно, что произведение n -значных чисел надо брать при этом с полным (удвоенным) числом знаков.

Рассмотрим случай действий над $2n$ -значными числами, заключенными по абсолютной величине между 0 и 1. Каждое такое $2n$ -значное число имеет вид $x_1 + 2^{-n}x_2$, где x_1 — число, образованное его первыми n знаками, $|x_1| < 1$, x_2 — число, образованное следующими n знаками (x_1 и x_2 не обязаны иметь равные знаки), x_1 и x_2 хранятся в разных ячейках ЗУ, x_1 берется в ненормализованном виде с порядком 0.

Рассмотрим, например, операцию умножения. Ищем число $z_1 + 2^{-n}z_2 = (x_1 + 2^{-n}x_2)(y_1 + 2^{-n}y_2) \approx x_1y_1 + 2^{-n}(x_1y_2 + x_2y_1)$ (член $2^{-2n}x_2y_2$ мы откидываем). Произведение x_1y_1 надо брать с $2n$ знаками: $x_1y_1 = (x_1y_1)_1 + 2^{-n}(x_1y_1)_2$, где $(x_1y_1)_1$ и $(x_1y_1)_2$ — числа, образованные первыми и последними n цифрами произведения x_1y_1 .

Итак, искомое произведение равно

$$z_1 + 2^{-n}z_2 = (x_1y_1)_1 + 2^{-n}[(x_1y_1)_2 + x_1y_2 + x_2y_1]$$

Если выражение в квадратных скобках меньше 1, то мы уже получили произведение в виде двух n -значных чисел; если же оно больше 1, то его целая часть, умноженная на

2^{-n} , должна быть прибавлена к первой части $(x_1 y_1)_1$. Дробная его часть составит вторую часть z_2 произведения. В целях единообразия это выделение целой части можно проводить всегда.

Рассмотрим теперь операцию деления. Пусть требуется определить $2n$ знаков частного

$$z_1 + 2^{-n} z_2 = \frac{x_1 + 2^{-n} x_2}{y_1 + 2^{-n} y_2}$$

В знаменателе первую двоичную часть будем считать отличной от 0, $|x_1| < |y_1| < 1$. Тогда

$$\begin{aligned} z_1 + 2^{-n} z_2 &= \frac{x_1 + 2^{-n} x_2}{y_1 + 2^{-n} y_2} = \frac{x_1 + 2^{-n} x_2}{y_1 \left(1 + 2^{-n} \frac{y_2}{y_1}\right)} = \\ &= \left(\frac{x_1}{y_1} + \frac{2^{-n} x_2}{y_1}\right) \left(1 - 2^{-n} \frac{y_2}{y_1} + \dots\right) \approx \frac{x_1}{y_1} + 2^{-n} \frac{x_2 - \frac{x_1}{y_1} y_2}{y_1} \end{aligned} \quad (4)$$

(мы отбрасываем члены порядка 2^{-2n}). Произведем деление x_1 на y_1 с остатком. Пусть полученное n -значное частное есть $\left(\frac{x_1}{y_1}\right)$, а остаток $0 \left(\frac{x_1}{y_1}\right)$. Имеем:

$$z_1 + 2^{-n} z_2 = \left(\frac{x_1}{y_1}\right)_1 + 2^{-n} \cdot \frac{0 \left(\frac{x_1}{y_1}\right) + x_2 - \left(\frac{x_1}{y_1}\right)_1 y_2}{y_1}$$

Деление числителя на y_1 производится приближенно (получаем n -значное частное без вывода остатка). Выделяя из полученного числа целую часть и добавляя ее к $\left(\frac{x_1}{y_1}\right)_1$, получим первую часть ответа; дробная часть даст вторую часть ответа.

При выводе окончательной формулы мы отбросили члены, начиная с $2^{-2n} \left(\frac{y_2}{y_1}\right)^2$. Если y_1 начинается с нескольких нулей (и $y_2 \neq 0$), то точность формулы может сильно понизиться. Поэтому перед делением целесообразно привести делитель

к виду $y'_1 + 2^{-n} y'_2$, $1/2 \leq |y'_1| < 1$ путем деления числителя и знаменателя на 2^{-s} (s — число равных нулю цифр y_1 после запятой).

Вычисление многочленов. Вычисление значения многочлена $P(x) = \sum_{k=0}^n a_k x^k$ при заданных коэффициентах обычно целесообразно вести по схеме Горнера

$$P(x) = (\dots ((a_n x + a_{n-1}) x + a_{n-2}) x + \dots + a_1) x + a_0$$

В тех случаях, когда коэффициент a_n образуется последовательно рекуррентным образом (например, для многочленов, составленных из первых n членов степенных разложений e^x , $\sin x$, $\cos x$, $J_n(x)$ и т. п.), целесообразно последовательно образовывать члены $a_k x^k$.

Серию значений многочлена $P(x)$ в точках $x = a + lh$, $l = 0, 1, \dots$, удобно вычислять n -кратным суммированием констант $\Delta^n P(x)$ (см. ниже стр. 127).

В § 27 даны программы вычисления многочленов и степенных рядов, отвечающие различным приемам вычислений.

Вычисление значений элементарных функций. Для вычислений значений элементарных функций пользуются обычно их разложением в степенные ряды; весьма удобным с точки зрения быстроты сходимости и простоты является алгоритм разложения их в непрерывные дроби. Подробнее об этом см. следующий параграф.

Вычисление значений $\sin x$ и $\cos x$ в точках $x_k = a + kh$, $k = 0, 1, \dots$ получается последовательно из формул сложения:

$$\begin{aligned} \cos(a + kh) &= \alpha \cos[a + (k-1)h] - \beta \sin[a + (k-1)h] \\ \sin(a + kh) &= \beta \cos[a + (k-1)h] + \\ &+ \alpha \sin[a + (k-1)h], \quad \alpha = \cos h, \quad \beta = \sin h \end{aligned}$$

Аналогично, для функции b^x : $b^{a+kh} = cb^{a+(k-1)h}$, $c = b^h$.

Нахождение последовательных сумм и разностей. Стандартными операциями являются также операции нахож-

дения последовательных сумм

$$s_n^{(1)} = \sum_{i=1}^n a_i, \quad s_n^{(2)} = \sum_{i=1}^n s_i^{(1)}, \dots, \quad s_n^{(N)} = \sum_{i=1}^n s_i^{(N-1)} \quad (1)$$

и разностей

$$\Delta_n^{(1)} = a_{n+1} - a_n, \quad \Delta_n^{(2)} = \Delta_{n+1}^{(1)} - \Delta_n^{(1)} \text{ и т. д.} \quad (2)$$

Нахождение разностей применяется, в частности, весьма широко при контрольных операциях.

Операция последовательного суммирования применяется, например, при

- а) нахождении кратных интегралов;
- б) нахождении интегралов вида

$$s = \int_a^b f(x) P_N(x) dx, \quad \text{где } P_N(x) = \sum_{k=0}^N b_k x^k \quad (3)$$

Такие интегралы встречаются при вычислении моментов $\int_a^b f(x) x^n dx$, при нахождении коэффициентов Фурье для разложения функции по ортогональным многочленам, при квадратической интерполяции по многочленам Чебышева для функции, заданной в дискретном ряде точек, и т. д.

Интегрируя в формуле (3) последовательно по частям и обозначая

$$f^{(-1)}(x) = \int_a^x f(x) dx, \quad f^{(-2)}(x) = \int_a^x f^{(-1)}(x) dx, \dots, \\ f^{(-N)}(x) = \int_a^x f^{(-N+1)}(x) dx$$

получаем

$$\begin{aligned} s &= \int_a^b f(x) P_N(x) dx = f^{(-1)}(x) P_N(x) \Big|_a^b - \int_a^b f^{(-1)}(x) P'_N(x) dx = \\ &= [f^{(-1)}(x) P_N(x) - f^{(-2)}(x) P'_N(x)]_a^b + \int_a^b f^{(-2)}(x) P''_N(x) dx = \\ &= \dots \dots \dots = \\ &= [f^{(-1)}(x) P_N(x) - f^{(-2)}(x) P'_N(x) + \dots + \\ &+ (-1)^{N-1} f^{(-N)}(x) P_N^{(N-1)}(x)]_a^b - (-1)^N \int_a^b f^{(-N)}(x) P_N^{(N)}(x) dx \end{aligned}$$

или, так как $P_N^{(N)}(x) = b_N N!$, получаем

$$s = f^{(-1)}(b) P_N(b) - f^{(-2)}(b) P'_N(b) + \dots + \\ + (-1)^{N-1} f^{(-N)}(b) P_N^{(N-1)}(b) + (-1)^N \cdot b_N N! \cdot f^{-(N+1)}(b) \quad (4)$$

Числа $f^{(-1)}(b), \dots, f^{-(N+1)}(b)$ находятся последовательным суммированием по формуле (1).

В Институте точной механики и вычислительной техники при массовом вычислении интегралов вида

$$a_\alpha = \int f(x) \cos \alpha x dx, \quad b_\alpha = \int f(x) \sin \alpha x dx$$

применялся аналогичный прием (с предварительной заменой каждой полуволны синусоиды аппроксимирующей ее параболой k -го порядка; вычисления проводились для $k=2$). Это сводит нахождение таких интегралов в основном к $(k+1)$ -кратному последовательному интегрированию функции $f(x)$ и знакопеременному суммированию значений $f^{-(n+1)}(x)$ в точках арифметической прогрессии.

Решение уравнений. Для численного решения уравнения $f(x) = 0$ (или системы уравнений $f_i(x_1 \dots x_n) = 0, i = 1, 2, \dots, n$) удобны обычные способы численного решения, носящие итерационный характер (метод Ньютона, метод Лобачевского для случая действительных корней и т. п.).

Особенно возрастает ценность способов, быть может и продолжительных, но приводящих к серии совершенно однотипных несложных этапов. Остановимся подробнее на методе проб (сравнений) решения уравнения $f(x) = 0$. Пусть $f(a)f(b) < 0$. Вычислим $f\left(\frac{a+b}{2}\right)$. Если $\left|f\left(\frac{a+b}{2}\right)\right|$ достаточно мало, вычисления¹ оканчиваются. В противном случае при $f(a)f\left(\frac{a+b}{2}\right) < 0$ переходим к рассмотрению интервала $\left(a, \frac{a+b}{2}\right)$, а при $f(a)f\left(\frac{a+b}{2}\right) > 0$ переходим к рассмотрению интервала $\left(\frac{a+b}{2}, b\right)$.

¹ Здесь речь идет о вычислении одного корня.

Один этап этого процесса решения очевидно очень легко реализовать на автоматической машине, а следующие этапы суть повторения первого с новыми данными (интервал $\left(a, \frac{a+b}{2}\right)$ или $\left(\frac{a+b}{2}, b\right)$ вместо (a, b)). Этот же метод деления пополам применим и в том случае, когда надо найти место перемены знака или обращение в нуль последовательности a_1, \dots, a_n : при $a_1 a_n < 0$ нужно взять $a_{\frac{n+1}{2}}$ и в зависимости от знака $a_1 \cdot a_{\frac{n+1}{2}}$ перейти к последовательности $a_1, \dots, a_{\frac{n+1}{2}}$ или $a_{\frac{n+1}{2}}, \dots, a_n$.

Этот способ может быть применен для отыскания в таблице $f(x_i)$ значения x_i , для которого $(f(x_i) - c) (f(x_{i+1}) - c) \leq 0$. Для этого случая можно предложить также способ последовательного рассмотрения знака $f(x_i) - c$ и перехода к $f(x_{i+1}) - c$. Первый способ требует меньше действий (максимальное их число $\sim \log_2 n$ вместо $\sim n$).

В главе IV в § 27 приведены программы рассмотренных здесь задач.

§ 17. ЗАДАНИЕ ФУНКЦИЙ В МАШИНЕ

Табличное задание функции. В запоминающем устройстве хранятся значения функции $f(x)$ для некоторой системы значений аргумента $x = x_1, x_2, \dots, x_m$.

Если значение аргумента x не совпадает ни с одним из табличных значений x_1, x_2, \dots, x_m , причем $x_p < x < x_{p+1}$, то $f(x)$ представляется с нужной нам степенью точности соответствующим интерполяционным многочленом n -й степени.

Для меньшей нагрузки ЗУ и облегчения поисков можно рекомендовать пользование более редкой табличной сетью и, соответственно, интерполяционным многочленом более высокой степени.

Процесс вычисления значений функции существенно упрощается, если табличные значения аргумента x даются с постоянным шагом h , т. е. даются в точках арифметической прогрессии $x_0 + ph, p = 1, 2, \dots, m$. Можно преобразовать аргумент так, чтобы табличные значения аргумента превра-

тились в целые числа $1, 2, 3, \dots, m$. Сохраняя для аргумента и функции прежние обозначения $x, f(x)$, мы получаем: данному x отвечает в качестве табличного значения аргумента целая часть $p = E(x)$ аргумента x , а аргументом интерполяционного многочлена является дробная часть $x; y = \{x\} = x - p$.

Если значения $f(p)$ находятся в ячейках $\alpha + lp$ (l — натуральное число), то сразу, найдя для данного x число $\alpha + lp = \alpha + lE(x)$, мы найдем адрес ячейки, в которой хранится $f(p)$.

Интерполяционный многочлен $P_n^{(p)}(y)$ в качестве коэффициентов $a_k^{(p)}$, $k = 0, 1, \dots, n$, имеет линейные комбинации значений функции $f(x)$ и ее разностей в точке $x = p$ или значений функции в табличных точках, соседних с точкой p . Само значение $f(p)$ играет роль свободного члена $a_0^{(p)}$ многочлена $P_n^{(p)}(y)$.

Можно хранить вместе с значением $f(p) = a_0^{(p)}$ и другие (зависящие от p) значения коэффициентов $a_k^{(p)}$; $k = 1, 2, \dots, n$ многочлена $P_n^{(p)}(y)$. Нагрузка ЗУ при этом увеличивается, однако вычисление значения $P_n^{(p)}(y)$ чрезвычайно упростится и программа такого вычисления сведется к программе вычисления многочленов, приведенной в § 27.

Процесс вычисления сведется:

- 1) к нахождению $p = E(x)$ и $y = x - p = \{x\}$
- 2) к вычислению многочлена

$$P_n^{(p)}(y) = \sum_{k=0}^n a_k^{(p)} y^k = (\dots ((a_n^{(p)} y + a_{n-1}^{(p)}) y + \\ + a_{n-2}^{(p)}) + \dots) y + a_0^{(p)}$$

З а м е ч а н и е 1. Коэффициент $a_n^{(p)}$ при старшей степени y бывает обычно мал. С точностью до $\frac{1}{2^{2n-1}}$ можно заменить на отрезке $[0,1]$ степень y^n многочленом $(n-1)$ -й степени — $L_{n-1}(y)$, где $L_{n-1}(y)$ получается отбрасыванием члена y^n в многочлене Чебышева $T_n\left(\frac{1+y}{2}\right)$.

Если $\left| \frac{a_n}{2^{2n-1}} \right| + |R_n|$ (R_n — остаточный член) меньше ϵ , где $\epsilon > 0$ — допустимая при выбранной точности величина отброшенных членов, то мы можем заменить многочлен n -й степени $P_n^{(p)}(y)$ многочленом $(n-1)$ -й степени

$$P_n^{(p)}(y) = a_n^{(p)} y^n + L_{n-1}(y) \cdot a_n^{(p)}$$

Замечание 2. Можно уменьшить количество запоминаемых чисел, если пользоваться интерполяционными формулами, типа формулы Эверетта. В эту формулу входят лишь разности четного порядка, но взятые в двух смежных табличных точках.

Замечание 3. При составлении программы интерполяционных вычислений следует иметь в виду, что хотя результат интерполяции укладывается в диапазон чисел, с которыми оперирует машина, результаты промежуточных вычислений могут выходить за пределы этого диапазона.

Ввод функций, заданных аналитическими процессами. Аналитические функции, в частности элементарные функции, обычно бывают заданы некоторым аналитическим процессом: бесконечными степенными и иными рядами, бесконечными произведениями, интегральными представлениями, бесконечными непрерывными дробями, процессом решения дифференциальных и других уравнений и т. д. Эти процессы, реализуемые с определенной точностью, могут быть использованы для ввода функций в машину.

Ряды. Наиболее широко распространено вычисление функций с помощью степенных рядов

$$\sum a_k x^k$$

Можно заранее предусмотреть число членов ряда, нужное для получения суммы с требуемой точностью, или определять его в процессе вычислений. Мы приводим ниже (гл. V) программы для вычислений синусов, косинусов и бесселевых функций по рядам

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \quad (1)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots \quad (2)$$

$$J_\nu(x) = \frac{x^\nu}{2^\nu \Gamma(\nu)} \left(1 - \frac{x^2}{4 \cdot 1 \cdot (\nu + 1)} + \frac{x^4}{4^2 \cdot 1 \cdot 2 \cdot (\nu + 1)(\nu + 2)} - \dots \right) \quad (3)$$

Вычисление бесселевых функций ведется по формуле (3) при значениях $|x|$, меньших некоторого числа X . Для значений $|x|$, больших X , употребляется представление $J_\nu(x)$ асимптотическим рядом

$$J_\nu(x) = \left(\frac{2}{\pi x}\right)^{1/2} \left[\cos\left(x - \frac{1}{2}\nu\pi - \frac{1}{4}\pi\right) \sum_{m=0}^{\infty} \frac{(-1)^m (\nu, 2m)}{(2x)^{2m}} - \right. \\ \left. - \sin\left(x - \frac{1}{2}\nu\pi - \frac{1}{4}\pi\right) \sum_{m=0}^{\infty} \frac{(-1)^m (\nu, 2m+1)}{(2x)^{2m+1}} \right] \\ \text{где } (\nu, m) = \frac{\Gamma\left(\nu + m + \frac{1}{2}\right)}{m! \Gamma\left(\nu - m + \frac{1}{2}\right)} \quad (3')$$

Степенные ряды (1), (2), (3) и асимптотический ряд (3') дают нам примеры рядов, для которых каждый следующий коэффициент a_{k+1} при x^{k+1} (или x^{-k-1}) образован умножением или делением предыдущего коэффициента a_k на значение некоторого многочлена $P(k)$ от индекса k .

Изображение с помощью непрерывных дробей. В XIX веке для изображения функций широко пользовались непрерывными дробями

$$s = \frac{a_1}{b_1 + \frac{a_2}{b_2 + \dots}} \quad (4)$$

n -я подходящая дробь

$$\frac{P_n}{Q_n} = \frac{a_1}{a_1 + \frac{a_2}{b_2 + \dots} + \frac{a_{n-1}}{b_{n-1} + \frac{a_n}{b_n}}} \quad (4')$$

может быть найдена вычислением „с конца“, т. е. последовательным нахождением чисел $\frac{a_n}{b_n}$, $b_{n-1} + \frac{a_n}{b_n}$, $\frac{a_{n-1}}{b_{n-1} + \frac{a_n}{b_n}}$ и т. д.

Такой способ требует для вычисления значения $\frac{P_n}{Q_n}$ n делений, $(n-1)$ сложений и весьма прост с точки зрения программирования.

Числители и знаменатели P_n и Q_n подходящих дробей можно находить также по рекуррентным формулам

$$P_{n+1} = b_n P_n + a_n P_{n-1}, \quad Q_{n+1} = b_n Q_n + a_n Q_{n-1}$$

Из этих формул выводится соотношение

$$\frac{P_{n+1}}{Q_{n+1}} - \frac{P_n}{Q_n} = \frac{(-1)^n b_1 b_2 \dots b_n}{Q_n Q_{n+1}}$$

Для бесконечной сходящейся непрерывной дроби (4) имеем:

$$s = \lim_{n \rightarrow \infty} \frac{P_n}{Q_n}, \text{ или } s = \sum_{n=0}^{\infty} \left(\frac{P_{n+1}}{Q_{n+1}} - \frac{P_n}{Q_n} \right) = \sum_{n=1}^{\infty} \frac{(-1)^n b_1 b_2 \dots b_n}{Q_n Q_{n+1}}$$

Отсюда и вытекает оценка точности n -го приближения

$$\left| s - \frac{P_n}{Q_n} \right| = \left| \sum_{m=n}^{\infty} \frac{(-1)^m b_1 b_2 \dots b_m}{Q_m Q_{m+1}} \right| \leq \sum_{m=n}^{\infty} \left| \frac{b_1 b_2 \dots b_m}{Q_m Q_{m+1}} \right| \quad (5)$$

Если a_n и b_n суть многочлены по x , то подходящая дробь $\frac{P_n}{Q_n}$ есть рациональная функция x , дающая в некотором смысле оптимальное рациональное приближение функции $s = f(x)$, изображаемой сходящейся бесконечной дробью (4). В качестве примеров приведем разложения в непрерывную дробь гиперболического и обычного тангенса:

$$\operatorname{th} x = \frac{x}{1 + \frac{x^2}{3 + \frac{x^2}{5 + \dots}}} \quad \operatorname{tg} x = \frac{x}{1 - \frac{x^2}{3 - \frac{x^2}{5 - \dots}}} \quad (6)$$

Формулой (6') можно пользоваться для нахождения

$$\operatorname{cosec} x = \frac{1}{2} \left(\operatorname{tg} \frac{x}{2} + \frac{1}{\operatorname{tg} \frac{x}{2}} \right)$$

Из формулы (5) в силу $\operatorname{th} \frac{x}{2} = \frac{e^x - 1}{e^x + 1}$ находим

$$e^x = -1 + \frac{2}{1 - \operatorname{th} \frac{x}{2}} = -1 + \frac{2}{1 - \frac{x}{2 + \frac{x^2}{6 + \frac{x^2}{10 + \frac{x^2}{14 + \frac{x^2}{18 + \dots}}}}} \quad (7)$$

Формулы (5—7) дают весьма быструю сходимость. Если удержать в разложении (7) в непрерывную дробь приведенные 6 знаменателей, получаем дробь, дающую значения e^x при $0 < x < 1$ с 11 точными десятичными знаками. Для нахождения e^x с такой точностью нужно произвести одно возвышение в квадрат (нахождение x^2), 6 делений и 6 добавлений целых чисел. Каждый дополнительный знаменатель дает увеличение точности на 2—3 десятичных знака ценою одного деления и одного сложения. Преимущество этого представления не только в скорости сходимости, но и в простоте его реализации на автоматической машине, так как все промежуточные результаты просто и единообразно получаются один из других. С помощью непрерывных дробей хорошо получается рациональное приближение функций вблизи ее полюсов.

Итерационные методы. Для вычислений значений функций широко применяются итерационные методы.

Если искомое значение y получается как предел последовательности y_n , где $y_n = \psi(y_{n-1})$, то нахождение y совершается по схеме, указанной в § 4. Приведем один из распространенных примеров быстро сходящегося итерационного процесса для нахождения значений функций. Пусть уравнение $y = f(x)$ эквивалентно уравнению

$$\varphi(x, y) = 0 \quad (8)$$

Для решения (8) применим итерационный метод Ньютона: строим последовательность y_n , исходя из выбранного y_0 по рекуррентной формуле

$$y_{n+1} = y_n - \frac{\varphi(x, y_n)}{\varphi'_y(x, y_n)} \quad (9)$$

Если ньютоновский процесс сходится, то начиная с некоторого места он сходится очень быстро — каждая следующая погрешность примерно пропорциональна квадрату предыдущей.

Пример 1. В некоторых машинах, например EDSAC, деление не входит в число элементарных операций и оно сводится к умножению на обратное число, находимое по методу Ньютона (применением операций сложения и умножения). Для нахождения $y = \frac{1}{x}$ решаем уравнение $\varphi(x, y) = \frac{1}{y} - x = 0$. Получаем частный случай формулы (9):

$$y_{n+1} = y_n - \frac{\frac{1}{y_n} - x}{-\frac{1}{y_n^2}} = y_n(2 - y_n x) \quad (10)$$

При $x = 2^n x_1$, $\frac{1}{2} \leq x_1 < 1$ можно принять за первое приближение $y_0 = 2^{-n}$. После 7 итераций получается ответ с точностью до 26 десятичных знаков.

Пример 2. Извлечение квадратного корня. Для нахождения $y = \sqrt{x}$ можно использовать, например, следующие уравнения

$$\varphi_1(x, y) = y^2 - x = 0; \quad \varphi_2(x, y) = 1 - \frac{x}{y^2} = 0$$

первое из них приводит к известной итерационной формуле

$$y_{n+1} = \frac{1}{2} \left(y_n + \frac{x}{y_n} \right) \quad (11)$$

второе приводит к формуле $y_{n+1} = y_n \left(\frac{3}{2} - \frac{y_n^2}{2x} \right)$ (12)

выполняемой удобно и на машинах без делений: требуется

нахождение лишь одной обратной величины $\frac{1}{2x}$ (см. предыдущий пример).

В качестве y_0 можно взять при $x = 2^n x_1$

$$y_0 = 2^E \left(\frac{n}{2} \right)$$

Существуют аналогичные методы (не требующие деления) для извлечения корня любой данной степени, а также для решения уравнений n -й степени.

Пример 3. Для вычисления $y = \ln x$ надо решить уравнение

$$\varphi(x, y) = e^y - x = 0$$

Это приводит к ньютоновскому процессу

$$y_{n+1} = y_n - 1 + x e^{-y_n} \quad (13)$$

Разложением $\ln(1+x)$ в ряд по многочленам Чебышева на отрезке $0 \leq x \leq 1$ можно построить многочлен 6-й степени, приближающий $\ln(1+x)$ с точностью до 10^{-6} (см. § 27). Если это значение рассматривать как начальное приближение y_0 к $y = \ln(1+x)$, то можно путем нахождения значения e^{-y_0} найти y , равный $\ln(1+x)$ с точностью до 10^{-12} . Второе приближение y_2 требует нахождения

$$e^{-y_1} = e^{-y_0} e^{y_1 - y_0} = e^{-y_0} \left[1 + (y_1 - y_0) + \frac{(y_1 - y_0)^2}{2} + \frac{(y_1 - y_0)^3}{6} + \varepsilon \right]$$

где $|\varepsilon| < 10^{-25}$, и дает значение $y = \ln(1+x)$ с точностью до 10^{-24} . Для получения логарифма с такой точностью необходимо с соответственной точностью найти e^{-y_0} . Но вычисление показательной функции разложением в степенной ряд или непрерывную дробь требует значительно меньшего объема вычислений, чем вычисление такими же методами логарифма, так как для него указанные процессы сходятся значительно медленнее.

Аппроксимация функций. Во многих случаях представляется удобным аппроксимировать функции просто вычисляемыми выражениями, например, многочленами. Если удастся с нужной точностью аппроксимировать некоторую

заданную таблично функцию многочленом, то вместо хранения в ЗУ серии значений этой функции нужно хранить набор коэффициентов многочлена, что, как правило, дает большую экономию в загрузке ЗУ.

В некоторых случаях целесообразно разбивать отрезок, на котором задана функция, на частичные отрезки, в которых функция с нужной степенью точности аппроксимируется многочленами невысокой степени. Так, при интегрировании уравнения внешней баллистики оказалось возможным хорошо аппроксимировать функцию сопротивления $G(v)$ многочленами 3-й степени на 13 различных интервалах.

Следует отметить, что процесс разложения функции $f(x)$ в ряд по ортогональным многочленам $P_n(x)$ с весом $\rho(x)$ очень хорошо проводится на автоматических машинах, так как нахождение коэффициентов Фурье

$$\int f(x) P_n(x) \rho(x) dx$$

сводится к нахождению последовательных неопределенных интегралов от функции $f(x) \rho(x)$, т. е. хорошо автоматизируемому процессу последовательного суммирования.

Для функции от двух или более переменных следует искать аппроксимацию, получаемую арифметическими действиями из функций одного переменного.

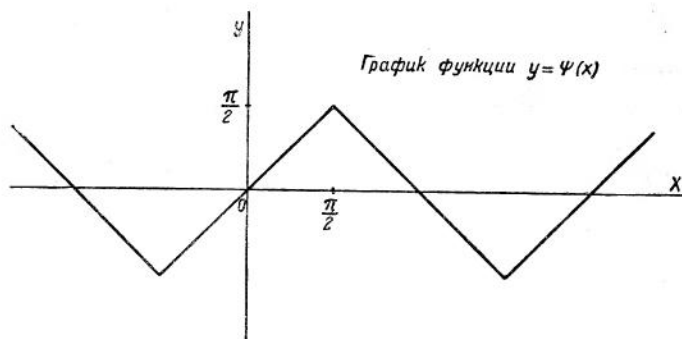
Разрывные функции. При помощи операции сравнения и вызванного ею „разветвления процесса“ можно производить вычисление значений функций, задаваемых на разных интервалах различными аналитическими процессами (подробнее об этом см. §. 20), и, в частности, некоторых классов разрывных функций.

Такие применяемые в машинах операции, как взятие абсолютной величины числа, дробной и целой его части и т. п., означают вычисление значений функций $|x|$, $\{x\}$, $E(x)$ и т. п. С их помощью можно также вычислять значения суперпозиций этих функций. Например, при вычислении функции $\sin x$ для произвольного x удобно использовать функцию

$$\psi(x) = \left| \left\{ \frac{x}{2\pi} - \frac{1}{4} \right\} 2\pi - \pi \right| - \frac{\pi}{2}$$

образованную из x при помощи 6 операций: 4-х арифметических действий, операции $||$ — взятия абсолютной величины и операции $\{ \}$ — взятия дробной части. Имеем:

$$\sin x = \sin \psi(x), \quad |\psi(x)| \leq \frac{\pi}{2}$$



§ 18. КОНТРОЛЬНЫЕ ВЫЧИСЛЕНИЯ

При вычислении на автоматических машинах существенную роль играют вопросы контроля.

Задачи, решаемые на таких машинах, требуют выполнения большого количества операций, а ошибка при выполнении одной из них может вызвать ошибку в результатах последующих операций и обесценить всю работу. Мы коснемся здесь контроля, осуществляемого в процессе вычислений — контроля правильности выполнения программы. Правильность выполнения программы заключается в правильности следования команд, правильности истолкования каждой команды и в правильности выполнения передач и действий, предусмотренных каждой командой.

Мы не будем касаться здесь предварительного контроля правильности работы отдельных узлов машины, осуществляемого при помощи специальных тестов, и будем рассматривать лишь контроль по ходу вычислений. Основной формой такого контроля является выполнение некоторых дополнительных контрольных вычислений, которые должны быть предусмотрены в программе решения задачи. Эти вы-

числения выполняются обычными рабочими устройствами машины и не требуют специального оборудования.

Примером программы, предусматривающей контрольные вычисления, является программа решения систем линейных алгебраических уравнений, приведенная в гл. V, § 30. В некоторых машинах предусмотрены специальные автоматические формы контроля отдельных операций, требующие дополнительного оборудования.

Контроль, предусмотренный программой. Существуют различные системы такого контроля. Большинство их основано на том, что некоторая функция f вычисленных величин x, y, \dots, u , если операции выполнены правильно, должна принимать значение нуль. Например, если $x = ab$, $y = ba$, то $x - y = 0$. При контроле длительных вычислений эта величина $f(x, y, \dots, u)$ даже при отсутствии ошибок может быть отлична от нуля. Устанавливается допустимое, предельное отклонение $\varepsilon > 0$ абсолютной величины $|f|$ от нуля — „допуск“, и производится операция сравнения $|f|$ и ε . Если выполняется неравенство $|f(x, y, \dots, u)| > \varepsilon$, мы считаем, что допущена ошибка, вызванная неправильностью составления программы, в частности, недостаточной точностью вычислений, или это ошибка машины. В этом случае, обычно, предусматривается автоматическая остановка машины.

Примеры: 1) Для проверки правильности корня x_0 уравнения $f(x) = 0$ мы вычисляем $|f(x_0)|$ и сравниваем его с ε .

2) При решении системы дифференциальных уравнений

$$\frac{dx_i}{dt} = f_i(x_1, x_2, \dots, x_n, t); i = 1, 2, \dots, n \quad (1)$$

можно внести дополнительную переменную $x_{n+1}(t)$, связанную с x_1, x_2, \dots, x_n соотношением

$$\psi[x_1(t), x_2(t), \dots, x_n(t), x_{n+1}(t), t] = 0 \quad (2)$$

составить для нее дополнительное уравнение

$$\frac{dx_{n+1}}{dt} = f_{n+1}(x_1, x_2, \dots, x_{n+1}, t) \quad (3)$$

и решать совместно систему (1), (3) из $n + 1$ уравнения, на каждом этапе проверяя выполнение тождества (2) с точностью до заданного $\varepsilon > 0$.

Так, при интегрировании систем уравнений механики часто удобно в качестве дополнительной функции взять кинетическую энергию или полную энергию системы.

3) В некоторых методах решения систем алгебраических уравнений основные операции заключаются в линейных операциях над строками (или столбцами) матрицы $\|a_{ij}\|$ — n -мерной матрицы системы, например, в замене каждого элемента

a_{ij} i -й строки элементом $a'_{ij} = \sum_{k=1}^n \beta_k^{(i)} a_{kj}$; введем в этом слу-

чае дополнительный столбец элементов $a_{i,n+1}$ ($i = 1, 2, \dots, n$) и подвергнем элементы этого столбца тем же линейным

операциям — заменим $a_{i,n+1}$ на $a'_{i,n+1} = \sum_{k=1}^n \beta_k^{(i)} a_{k,n+1}$. Если,

например, каждый дополнительный элемент $a_{i,n+1}$ выбирается равным сумме элементов той же строки, $a_{i,n+1} = \sum_{j=1}^n a_{ij}$, то

и после нашего преобразования выполняется аналогичное равенство $a'_{i,n+1} = \sum_{j=1}^n a'_{ij}$. В проверке выполнения этого равенства с заданной точностью $\varepsilon > 0$ и заключается контроль (см. § 30, гл. V).

4) Применяется также контроль в виде выхода на контрольное значение — заранее известно, что вычисленная на определенном этапе величина x равна числу a .

Пусть, например, считаются каким-нибудь рекуррентным методом последовательно значения функции $f(x)$ для $x_i = x_0 + ih$, $i = 1, 2, \dots$, причем известно заранее значение $a = f(x_0 + nh)$. Контроль заключается в сравнении вычисленного значения $f(x_n)$ с заданным его значением a .

5) При составлении таблиц или решении дифференциальных уравнений применяется контроль по разностям — разности определенного порядка k должны быть малы (по абсо-

лютной величине меньше заданного $\epsilon > 0$). Такой контроль совершенно аналогичен предыдущим формам контроля. И здесь критерием правильности решения является малость некоторой функции вычисляемых величин.

6) Вычисление в „две руки“ заключается в одновременном двукратном решении задачи и систематическом сличении промежуточных результатов. При этом необходимо чтобы один и тот же источник ошибок не мог производить одинаковый эффект в обоих решениях. Можно использовать, например, разные части машины (в частности, разные ячейки ЗУ) или вести вычисления с разными схемами выполнения отдельных их этапов (например, менять местами множимое и множитель).

Существуют машины (Mark III), специально приспособленные для вычислений в две руки. Они состоят из двух независимых устройств, синхронно производящих одинаковые операции и сличающих их результаты. Было предложено даже вычисление в „три руки“ — если два из ответов совпадают, а третий отличается, то правильным считается общий ответ двух, так как, очевидно, менее вероятно совпадение двух ошибок одинакового действия; машина должна автоматически продолжать вычисления, приняв в случае разных ответов, ответ „большинства“.

Некоторые автоматические формы контроля: а) **Контроль арифметических операций.** Для контроля индивидуальной арифметической операции можно употреблять свойства переместительности сложения и умножения (при умножении множимое и множитель играют существенно разную роль), получение одного из исходных чисел путем обратной операции и т. п. В некоторых машинах применяются специальные формы автоматического контроля. Сумма цифр целого десятичного числа сравнима с самим числом по модулю 9; общее, сумма цифр целого числа, заданного в p -ичной системе, сравнима с самим числом по модулю $p-1$. В самом деле, $p^k - 1$ делится на $p - 1$, откуда $p^k \equiv 1 \pmod{p-1}$ и

$$\sum_{k=0}^n a_k p^k \equiv \sum_{k=0}^n a_k \pmod{p-1}.$$

На этом основан контроль, применявшийся в машине ENIAC, действовавшей по десятичной системе счисления.

Одноразрядный десятичный счетчик с десятичной передачей на самого себя осуществляет сложение по модулю 9: при прибавлении десятки в счетчик вводится единица $10 \equiv 1 \pmod{9}$.

Пусть A, B — десятичные числа, a и b — сумма их цифр по модулю 9, полученная на таком одноразрядном счетчике

$$A \equiv a \pmod{9}$$

$$B \equiv b \pmod{9}$$

Имеем: $AB \equiv ab \pmod{9}$

$$A + B \equiv a + b \pmod{9}$$

Таким образом, при операциях умножения и сложения аналогичные операции (по модулю 9) совершаются над их суммами цифр.

Пример. $576767 \times 2852 = 1464493484$

Сумма цифр по модулю 9 числа 5756767 равна 2

" " " " " 9 " 2852 " 8

$2 \times 8 = 16$, сумма цифр = 7

Сумма цифр по модулю 9 произведения 1644933484 равна 7.

Для производства этого контроля нужно дополнить сумматор одноразрядным десятичным сумматором, осуществляющим сложение цифр по модулю 9. Мы получаем контроль операции сложения и умножения.

При таком способе контроля взаимное замещение цифр 0 и 9 есть элементарная неуловимая ошибка. Так как

$$p^k \equiv (-1)^k \pmod{p+1}, \text{ то } \sum_{k=0}^n a_k p^k \equiv (-1)^k a_k \pmod{p+1}$$

т. е. знакопеременная сумма цифр p -ичного числа сравнима с самим числом по модулю $p+1$; в частности, для десятичного числа по модулю 11. Все предыдущие рассуждения сохраняют свою силу, если заменить сумму цифр их знакопеременной суммой и модуль 9 — модулем 11. При таком способе контроля ошибка в одной цифре может быть обнаружена. Аналогично, соединяя, например, по 3 (4,5) цифры двоичной системы, будем ее рассматривать как восьмиричную (16-ричную, 32-ричную).

Трехзначный (четырёхзначный, пятизначный и т. д.) двоичный сумматор с циклической двоичной передачей есть сумматор по модулю 7 (15, 31). Складывая на этом сумматоре все восьмиричные цифры данного числа (образованные тройками его двоичных цифр), получим сумму его цифр, сравнимую с самим числом по модулю 7 (15, 31). Предыдущее сохранит силу с заменой модуля 9 модулем 7 (15, 31) и модуля 11 — модулем 9 (17, 33).

б) Контроль передач. Ошибки могут возникать при передаче чисел (например, при передаче из ЗУ в арифметический блок и т. п.).

Мы уже указывали (§ 1) на контрольные функции двоично-пятиричного представления десятичных чисел. Для чисто двоичных чисел предложены разные формы контроля, основанные на том, что наборы двоичных знаков, изображающих числа, команды и т. п., должны обладать определенными признаками. Элементарная ошибка, заключающаяся в замене в каком-либо разряде цифры нуль на единицу или обратно, приводит к нарушению одного из этих признаков. Взаимно компенсирующая комбинация элементарных ошибок имеет гораздо меньшую вероятность. Усложнением схем такого контроля можно обнаружить появление двух, трех и т. д. элементарных ошибок.

Можно, например, потребовать, чтобы сумма цифр наборов, фигурирующих в машине, была четной. Это можно реализовать путем добавления дополнительного двоичного разряда, в котором стоит число 1 или 0, равное сумме по модулю 2 цифр всех разрядов данного числа (эта сумма находится одноразрядным сумматором). Тогда при добавлении этого разряда к разрядам набора сумма цифр всех разрядов по модулю 2 становится равной 0. Если после передачи сумма цифр (по модулю 2) станет равной 1, то это указывает на ошибку в передаче.

В работе [7] предложена более сложная схема контроля, позволяющая не только обнаружить элементарную ошибку передачи, но и исправить ее.

Пусть наборы (чисел, команд и т. д.), фигурирующие в машине, занимают l двоичных разрядов; добавим еще k дополнительных разрядов,

несущих функции контроля так, чтобы $2^{k-1} < 1 + l + k \leq 2^k$. Например, при $l = 40$ можно взять $k = 6$, так как $2^5 < 47 < 2^6$.

Перенумеруем все $(k + l)$ разрядов числами $1, 2, \dots, (k + l)$ так, чтобы дополнительные разряды имели номера 2^s , $s = 0, 1, 2, \dots, (k - 1)$. Обозначим через E_s , $s = 0, 1, 2, \dots, (k - 1)$, совокупность разрядов, номера которых имеют цифру 1 на s -м месте в своем двоичном представлении. Каждый дополнительный разряд входит в одно и только одно из таких множеств.

В дополнительных разрядах набора должны стоять такие цифры, чтобы сумма цифр разрядов каждого множества E_s была четной (равнялась 0 по модулю 2). Пусть при передаче имела место лишь элементарная ошибка в p -м разряде набора ($1 \leq p \leq k + l$). Сумма цифр x_s по модулю 2 разрядов из E_s равна теперь 0 или 1, в зависимости от того, входит ли разряд с ошибкой в множество E_s или нет, т. е. от того, равна ли 0 или 1 s -я с конца цифра в двоичном представлении номера p . Иными словами, s -я с конца цифра номера p совпадает с суммой по модулю 2 цифр разрядов E_s . Определив эти суммы $x_1, x_2, \dots, x_{k-1}, x_k$, мы получим двоичное разложение $x_k, x_{k-1}, \dots, x_2, x_1$ номера разряда, в котором произошла ошибка передачи. Для исправления ошибки достаточно изменить цифру разряда номера на дополнительную.

Усложнением этой схемы можно исправить ошибки передачи в разрядах, число которых не больше заданного m ($m = 2, 3$ и т. д.).

Пример. Пусть $l = 13$, $k = 4$ ($2^3 < 11 + 4 + 1 = 2^4$), 11 разрядов передают информации, 4 разряда с номерами 1, 2, 4, 8 выполняют контрольные функции.

$$\begin{aligned} E_1 &= \{ 1, 3, 5, 7, 9, 11, 13, 15 \} & E_2 &= \{ 2, 3, 6, 7, 10, 11, 14, 15 \} \\ E_3 &= \{ 4, 5, 6, 7, 12, 13, 14, 15 \} & E_4 &= \{ 8, 9, 10, 11, 12, 13, 14, 15 \} \end{aligned}$$

Пусть передается 15-значный набор 101001001001011, 11-разрядный набор — — 1 — 010 — 1001011 является передаваемой информацией. Цифры в контрольных разрядах 10 — 0 — — — 0 — — — — — подобраны так, чтобы сумма цифр в разрядах по каждому множеству E_1, E_2, E_3, E_4 была четной. Пусть наш 15-значный набор был передан в виде 10101001001011; сумма цифр в разрядах

$$\begin{aligned} \text{из } E_1 &\text{ равна } 1 + 1 + 1 + 0 + 1 + 0 + 0 + 1 = 5 \equiv 1 \pmod{2} \\ \text{„ } E_2 &\text{ „ } 0 + 1 + 1 + 0 + 0 + 0 + 0 + 1 = 4 \equiv 0 \pmod{2} \\ \text{„ } E_3 &\text{ „ } 0 + 1 + 1 + 0 + 1 + 0 + 1 + 1 = 5 \equiv 1 \pmod{2} \\ \text{„ } E_4 &\text{ „ } 0 + 1 + 0 + 0 + 1 + 0 + 1 + 1 = 4 \equiv 0 \pmod{2} \end{aligned}$$

Номер разряда, в котором допущена элементарная ошибка, 0101 = 5.

В самом деле, в 5-м разряде стояла цифра 0, а после передачи стоит цифра 1.

Глава IV

ПРОГРАММИРОВАНИЕ РЕШЕНИЙ МАТЕМАТИЧЕСКИХ ЗАДАЧ

§ 19. ПРОГРАММЫ И КОМАНДЫ

Команды. При решении математической задачи на вычислительных машинах необходимо после выбора метода решения составить программу для реализации принятого алгоритма решения. Программа состоит из последовательности арифметических и логических операций, к которым сводится алгоритм, причем каждая из операций, равно как и вся их последовательность, выполняема машиной.

Для таких машин, как счетно-аналитические, в которых решение выполняется комплектом машин, программа состоит соответственно из серии частичных программ, последовательно выполняемых на разных машинах комплекта. На вполне автоматических машинах осуществляется единая программа.

Программа состоит из серии команд, каждая из которых заставляет машину выполнить определенную операцию и которые должны быть определенным образом зашифрованы для ввода в машину.

Команда, заставляющая выполнить определенную операцию, должна содержать, во-первых, шифр (код) самой операции; во-вторых, она должна содержать указание об объектах, над которыми выполняется операция.

В счетно-аналитических машинах каждая вводимая в машину перфокарта содержит команду (а иногда и серию команд). Числа, над которыми совершается операция, при-

ведены в самой перфокарте, т. е. в самой команде обычно содержатся числа, над которыми совершается операция.

Адреса. Для того, чтобы автоматически выполнить серию операций, к которой сводится решение какой-нибудь задачи, нельзя ограничиться командами, содержащими заранее вводимые числа; большинство операций производится над результатами предыдущих операций. Поэтому числа, над которыми совершаются операции, приходится посылать в арифметическое устройство из ячеек ЗУ, а результаты посылать, вообще говоря, в другие ячейки ЗУ. Во вполне автоматических машинах команды содержат большей частью не числа, над которыми совершаются операции, а „адреса“ этих чисел, т. е. коды тех ячеек ЗУ, в которых эти числа хранятся и куда эти числа направляются. Чаще всего в качестве адреса ячейки берется номер этой ячейки.

В дальнейшем мы будем обозначать через (α) число, хранящееся в ячейке ЗУ, адрес которой есть α .

Заметим, что даже для операций над заранее заданными числами бывает удобнее приводить в командах не сами эти числа, а их адреса. Пусть, например, машина действует над двенадцатизначными десятичными числами, а номера ячеек ее ЗУ не более чем трехзначные десятичные числа. Выгоднее задать в команде три десятичные цифры адреса числа α , чем 12 цифр самого числа (α) .

Для выполнения каждой операции надо знать некоторое количество k ($k = 1, 2, 3, \dots$) адресов чисел, участвующих в этой операции. Соответственно, такую операцию можно назвать k -адресной, а команду о выполнении такой операции — k -адресной командой. Операция записывается обычно набором, который мы будем называть *шифром* или *кодом* операции. k -адресная операция записывается в виде набора $S\alpha_1\alpha_2\dots\alpha_k$, где S — шифр операции, $\alpha_1, \alpha_2, \dots, \alpha_k$ — адреса чисел, участвующих в операции. Так, операция посылки чисел из арифметического устройства в ячейку ЗУ есть одноадресная операция, так как для ее выполнения нужно знать адрес α ячейки, куда число направляется, и код такой операции записывается в виде набора $S\alpha$, где S — шифр операции (в данном случае — посылки), α — адрес.

Арифметические действия — сложение, вычитание, умножение и деление — естественнее всего выступают, как трехадресные операции: арифметическое действие S выполняется над числами, хранящимися в ячейках α и β , и результат посылается в ячейку γ . Код такой трехадресной операции будет $S\alpha\beta\gamma$. Деление же с остатком может выступать как четырехадресная операция, код которой должен иметь вид $S\alpha\beta\gamma\delta$, где S — код операции деления, α и β — адреса делимого и делителя, γ и δ — адреса ячеек, в которые нужно отправить частное и остаток.

Количество k адресов данной операции зависит также от характера и условия выполнения операции. Сложение может быть не только трехадресной операцией, но (в машинах с накапливающим сумматором) и двухадресной операцией $S\alpha\beta$ (см. § 4) и одноадресной операцией $S\alpha$: добавить число (α), хранящееся в ячейке α , к числу, хранящемуся в сумматоре, и оставить в нем ответ (сумматор выполнил функции ячейки ЗУ, хранящей одно из слагаемых, и ячейки, которая должна хранить сумму).

Ввод набора, означающего команду, не отличается от ввода набора, означающего число. Команды в определенной последовательности воспринимаются машиной (см. ниже, стр. 150); восприятие шифра адреса вызывает коммутацию, в результате которой в арифметическое устройство машины поступает число именно из данного адреса, или обратно — из арифметического устройства посылается число именно в данный адрес; восприятие шифра операции вызывает настройку машины на выполнение именно данной операции.

k -адресные коды. В разных машинах применяются разные k -адресные команды. Соответственно, машина называется машиной k -адресного кода или просто k -адресной машиной. В настоящее время осуществлены k -адресные машины для $k = 1, 2, 3, 4$.

Стандартная команда k -адресного кода $S\alpha_1\alpha_2\dots\alpha_k$ имеет следующий смысл:

Шифр операции	1-й адрес	2-й адрес	. . .	k -й адрес
S	α_1	α_2	. . .	α_k

Мы будем обозначать эту команду или $S\alpha_1\alpha_2\dots\alpha_k$ или

S	α_1	α_2	\dots	α_k
-----	------------	------------	---------	------------

Для l -адресной операции, где $l < k$, в k -адресном коде $k-l$ адресов отсутствуют. Если нет ячейки ЗУ с номером 0, то мы можем помещением числа 0 в месте для одного из адресов указывать на то, что в данной команде этот адрес опускается. Например, команда операции v — взятия абсолютной величины — над числом (α) и послыки результата в ячейку γ имеет в трехадресном коде вид:

Шифр операции	1-й адрес	2-й адрес	3-й адрес
v	α		γ

Второй адрес опускается, что может быть отмечено помещением цифры 0 во всех разрядах, служащих для хранения 2-го адреса.

Заметим, что в команде $S\dots\alpha\dots$ адрес α ячейки, из которой берется число, может быть адресом ячейки как постоянного, так и переменного ЗУ; адрес ячейки, в которую посылают число, есть всегда адрес ячейки переменного ЗУ.

Удобно считать, что ячейка с адресом 0 хранит постоянно число 0: $(0) = 0$. При таком условии ячейка может быть фактически не реализована. Таким образом, например, команда $S_0\alpha\beta\gamma$ (сложить (α) и (β) и переслать сумму в γ) при $\beta = 0$ превращается в $S_0\alpha 0\gamma$ (переслать (α) в γ). При записи мы будем просто опускать (оставлять пустым) место нулевого адреса. Например,

$$\begin{array}{|c|c|c|c|} \hline S_0 & \alpha & 0 & \gamma \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline S_0 & \alpha & & \gamma \\ \hline \end{array}$$

Принципиально каждую k -адресную операцию ($k = 2, 3, 4$) можно расщепить на k одноадресных, поэтому и машины с одноадресным кодом могут быть универсальными. Например, трехадресную операцию $S\alpha\beta\gamma$ умножения S чисел (α) и

(β) с посылкой ответа в ячейку γ можно разбить на три одноадресные операции:

1) Операцию $S_0\alpha$ — посылки числа (α) из ячейки α в арифметическое устройство, где оно должно храниться.

2) Операцию $S_1\beta$ — умножения числа, хранящегося в арифметическом устройстве, на число (β), хранящееся в ячейке β , с сохранением произведения в арифметическом устройстве.

3) Операцию $S_2\gamma$ — посылки произведения из арифметического устройства в ячейку γ .

Отсюда видно, что *арифметическое устройство машин одноадресного кода должно обладать способностью хранить посылаемые в него и образованные в нем числа*. Такие машины снабжаются накапливающими сумматорами.

Одной команде машины с трехадресным кодом соответствуют в некоторых случаях три команды машины с одноадресным кодом, в среднем же менее, чем три, так как, во-первых, не все операции, выполняемые машиной трехадресного кода, суть трехадресные, и, во-вторых, то, что арифметическое устройство одноадресной машины также выполняет функции ЗУ, экономит число пересылок из арифметического устройства в ячейки ЗУ и обратно. Можно считать, что число команд программы в одноадресном коде примерно в два раза больше, чем в соответственной программе трехадресного кода (см. пример, приведенный в конце параграфа). Зато каждая команда вида $S\alpha$ одноадресного кода „короче“ команды $S\alpha\beta\gamma$ трехадресного кода.

Первые вполне автоматические машины с суммирующим ЗУ являлись, как указывалось выше (§ 4), машинами двухадресного кода. В более новых машинах двухадресный код не применяется, так как, будучи „длиннее“ одноадресного, он, в отличие от трехадресного, все-таки не позволяет одной командой осуществлять обычные арифметические операции.

О машинах четырехадресного кода см. ниже. Среди более новых машин встречаются машины одноадресного кода (EDSAC, BINAC, UNIVAC и др.), трехадресного кода (ACE, Mark II, Mark III), четырехадресного кода (EDVAC, Raytheon и др.).

Последовательность выполнения команд. В современных автоматических вычислительных машинах совокупность команд, образующих программу, вводится в машину и хранится в ячейках ЗУ с определенными адресами. Существуют два способа установления порядка выполнения команд. Первый способ заключается в том, что после выполнения команды, хранящейся в ячейке с адресом α , следует выполнение команды, хранящейся в ячейке ЗУ с адресом $\alpha + 1$. При втором способе адрес следующей команды указан в предыдущей команде. Ясно, что второй метод гибче, так как отсутствует принудительная последовательность команд, которую надо учитывать при программировании. Однако, каждая команда должна содержать дополнительный адрес — адрес следующей команды.

Существуют машины, которые, в основном, работают по первому способу — машины „последовательного выполнения команд“ и машины, работающие по второму способу — машины „свободного выполнения команд“.

Первый способ применяется в машинах одно-, двух- и трехадресного кода, а второй — в машинах четырехадресного кода. Четвертый адрес команд в машинах последнего типа используется обычно для указания адреса команды, которая должна быть выполнена сразу после того, как будет выполнена рассматриваемая команда. Такая четырехадресная команда (стандартная четырехадресная команда) представляется обычно в виде набора $S_0\alpha\beta\gamma\delta$, где S_0 — шифр операции, α и β — адреса объектов операции, γ — адрес результата операции, δ — адрес следующей команды.

При работе по первому способу в наборе команды в большинстве случаев достаточно указать не более трех адресов: α , β и γ , вследствие чего и не существует машин, работающих по этому способу и имеющих более трех адресов в команде. Для работы по второму способу пригодны машины также двух- и трехадресного кода. Действительно, если в первом адресе команды мы будем указывать номер α объекта операции, а во втором адресе — номер β команды, которая должна выполняться сразу после выполнения рассматриваемой команды, то мы получим набор $S_0\alpha\beta$ команды

двухадресной машины, работающей вторым способом. Аналогично, двухадресной машине, работающей по первому способу, отвечали бы трехадресные машины, работающие по второму способу.

Машины одноадресного кода, естественно, могут работать только по первому способу.

Элементарные операции. Число основных операций, выполняемых машиной и кодируемых в командах, различно в различных машинах. Для каждой машины существует свой набор элементарных операций, т. е. операций, осуществляемых с помощью одной команды. Как известно, все арифметические действия сводятся к операциям сложения или вычитания и сдвига. Однако выполнять арифметические действия путем стандартных программ, описывающих это сведение по схемам §§ 11—13, затруднило бы эксплуатацию машины. В универсальных машинах чаще всего все арифметические действия относятся к числу основных операций; в некоторых машинах (например, EDSAC) операция деления не является основной и выполняется по особой программе (см. § 17).

Иногда в качестве самостоятельных операций фигурируют те или иные варианты этих действий (сложение в накапливающем сумматоре с сохранением суммы в сумматоре или с выдачей суммы; умножение с получением ординарного или удвоенного числа цифр произведения, деления без остатка или с остатком и т. д.). Операция извлечения квадратного корня может быть самостоятельной операцией или же выполняться по особой программе (см. § 17).

Многие операции могут быть сведены к операции сличения. Однако такие часто встречающиеся операции, как взятие абсолютной величины и т. п., обычно фигурируют как элементарные операции, даже если они сводимы к операциям сличения. Сама операция сличения встречается не во всех машинах.

В одноадресных машинах обязательны, как самостоятельные операции, операции посылки числа в арифметическое устройство и в ЗУ. В машинах с плавающей запятой в числе элементарных операций могут быть те или иные операции

с порядком, операция взятия целой части и т. д. Что касается перевода из одной системы счисления в другую, то ввиду сложности этой операции она выполняется обычно по специальной программе.

Иногда в машинах трех- и четырехадресного кода две двухадресные операции над одним и тем же числом объединяются в одну элементарную операцию: например, операцию выделения целой части $E(x)$ и дробной части $\{x\}$ числа $x = (\alpha)$, с посылкой $E(x)$ в ячейку β и $\{x\}$ в ячейку γ , можно объединить в трехадресную операцию $S\alpha\beta\gamma$ (целую и дробную части содержимого ячейки α послать, соответственно, в ячейки β и γ).

Ясно, что кроме команд о выполнении операций должны быть предусмотрены команды управления машиной, например, остановки машины, послышки чисел в ЗУ. Кроме того предусматриваются команды, позволяющие в машинах последовательного выполнения команд нарушать порядок выполнения команд и позволяющие выбирать то или иное продолжение процесса вычислений в случае его разветвления.

Замечание. Если стандартная ячейка машины может хранить n -разрядные наборы, шифр операции состоит из l разрядов, каждый адрес записывается m -значным набором и число адресов в команде равно k , то для того, чтобы команды могли храниться в стандартных ячейках ЗУ, должно быть выполнено неравенство

$$n \geq mk + l$$

Например, если $n = 35$, число операций равно 20, т. е. $l = 5$ ($2^4 < 20 < 2^5$), число ячеек ЗУ равно $2^{10} = 1024$, $m = 10$, то стандартная ячейка ЗУ может хранить команды трехадресного кода ($k = 3$): $35 = 3 \cdot 10 + 5$. Однако одна стандартная ячейка не могла бы хранить команды четырехадресного кода. Одна такая стандартная ячейка могла бы хранить две команды одноадресного кода (как в машине EDSAC).

Пример программы. Программа решения математической задачи на вполне автоматической машине должна содержать серию команд, выполнение которых приводит к решению

этой задачи. Программа определяется не только выбранным алгоритмом решения задачи, но и особенностями машины: принятым для данной машины k -адресным кодом ($k = 1, 2, 3, 4, \dots$), списком операций, выполняемых машиной, и т. д.

Мы приведем пример простых программ, а именно, программ деления комплексных чисел для машин трехадресного и одноадресного кода с последовательным выполнением команд.

Деление комплексных чисел сводится к отысканию действительной части $x = \frac{ac + bd}{c^2 + d^2}$ и мнимой части $y = \frac{bc - ad}{c^2 + d^2}$ частного $x + iy = \frac{a + bi}{c + di}$. Выберем следующий естественный порядок действий на машине. Находим квадраты c^2 и d^2 , их сумму $c^2 + d^2$, образуем произведения ac, bd, bc, ad , образуем сумму $ac + bd$ и разность $bc - ad$ и, наконец, делим $ac + bd$ и $bc - ad$ на $c^2 + d^2$. Пусть, например, в машине адреса ячеек ЗУ, хранящих команды, имеют номера $K + 1, K + 2, \dots$, адреса ячеек „постоянного“ ЗУ, т. е. ячеек, содержание которых вводится в ЗУ до начала вычислений и остается неизменным в процессе вычислений, имеют номера A_1, A_2, A_3, \dots , адреса ячеек вытесняющего ЗУ — B_1, B_2, \dots и выше. Операции сложения, вычитания, умножения и деления будем обозначать здесь, как и в дальнейшем, символами $+$, $-$, \times , $:$.

Программа для машины трехадресного кода и последовательного выполнения команд выглядит следующим образом.

Содержимое ячеек с исходными и вспомогательными величинами: $(A_1) = a, (A_2) = b, (A_3) = c, (A_4) = d; (B_1), (B_2), (B_3), (B_4)$ — что угодно (эти ячейки понадобятся в процессе вычисления).

Адреса команд	Операция	1 адрес	2 адрес	3 адрес	Примечание
$K + 1$	\times	A_3	A_3	B_1	В ячейке B_1 , получим c^2
$K + 2$	\times	A_4	A_4	B_2	„ „ B_2 „ d^2
$K + 3$	$+$	B_1	B_2	B_1	„ „ B_1 „ $c^2 + d^2$

Адрес команд	Операция	1 адрес	2 адрес	3 адрес	Примечание
$K + 4$	\times	A_1	A_3	B_2	В ячейке B_2 получим ac
$K + 5$	\times	A_2	A_4	B_3	" " B_3 " bd
$K + 6$	$+$	B_2	B_3	B_2	" " B_2 " $ac + bd$
$K + 7$	\times	A_2	A_3	B_3	" " B_3 " bc
$K + 8$	\times	A_1	A_4	B_4	" " B_4 " ad
$K + 9$	$-$	B_3	B_4	B_3	" " B_3 " $bc - ad$
$K + 10$	$:$	B_2	B_1	B_2	" " B_2 " $x = \frac{ac + bd}{c^2 + d^2}$
$K + 11$	$:$	B_3	B_1	B_3	" " B_3 " $y = \frac{bc - ad}{c^2 + d^2}$
$K + 12$	остановка				

Составим теперь программу этого деления в одноадресном коде. При этом будем пользоваться следующей символикой (обозначения, принятые, в основном, для машины EDSAC).

An — послать число, хранящееся в n -й ячейке ЗУ в накапливающий сумматор, т.е. прибавить это число.

Sn — вычесть число, хранящееся в n -й ячейке ЗУ из числа b в накапливающем сумматоре, т.е. послать это число с обратным знаком в накапливающий сумматор.

Hn — передать число, хранящееся в n -й ячейке ЗУ в регистр множителя.

Vn — умножить число, хранящееся в n -й ячейке ЗУ, на число, содержащееся в регистре множителя, и послать произведение в накапливающий сумматор.

Nn — умножить число, хранящееся в n -й ячейке ЗУ, на число, содержащееся в регистре множителя, и послать с обратным знаком в накапливающий сумматор.

Wn — разделить число, хранящееся в n -й ячейке ЗУ, на число, содержащееся в регистре множителя, и результат послать в накапливающий сумматор.

Tn — послать содержимое накапливающего сумматора в n -ю ячейку ЗУ и очистить накапливающий сумматор.

Программу деления мы составим путем разбивки трех-адресных команд уже составленной нами программы на команды одноадресного кода.

Содержимое ячеек: $(A_1)=a$, $(A_2)=b$, $(A_3)=c$, $(A_4)=d$; (B_1) , (B_2) , (B_3) — что угодно.

Адрес команд	Операция	Адрес	Примечания
$K+1$	H	A_3	
$K+2$	V	A_3	В накапливающем сумматоре получим c^2
$K+3$	H	A_4	
$K+4$	V	A_4	В накапливающем сумматоре получим $c^2 + d^2$
$K+5$	T	B_1	В ячейке ЗУ B_1 получим $c^2 + d^2$ (накапливающий сумматор очищается)
$K+6$	H	A_3	
$K+7$	V	A_1	В накапливающем сумматоре получим ac
$K+8$	H	A_4	
$K+9$	V	A_2	В накапливающем сумматоре получим $ac + bd$
$K+10$	T	B_2	В ячейке ЗУ B_2 получим $ac + bd$ (накапливающий сумматор очищается)
$K+11$	H	A_3	
$K+12$	V	A_2	В накапливающем сумматоре получим bc
$K+13$	H	A_4	
$K+14$	N	A_1	В накапливающем сумматоре получим $bc - ad$
$K+15$	T	B_3	В ячейке ЗУ B_3 получим $bc - ad$ (накапливающий сумматор очищается)
$K+16$	H	B_1	
$K+17$	W	B_2	В накапливающем сумматоре получим $x = \frac{ac + bd}{c^2 + d^2}$
$K+18$	T	B_2	В ячейке ЗУ B_2 получим x (накапливающий сумматор очищается)
$K+19$	W	B_3	В накапливающем сумматоре получим $y = \frac{bd - ac}{c^2 + d^2}$
$K+20$	T	B_3	В ячейке ЗУ B_3 получим y
$K+21$	остановка		

В данном случае 11 команд трехадресного кода оказались эквивалентными 20 командам одноадресного.

В приведенных примерах каждой элементарной операции соответствует своя команда, выполняемая не более одного раза в процессе вычислений. Такие программы будем называть прямыми. Другие типы программ будут рассмотрены далее.

§ 20. КОМАНДЫ БЕЗУСЛОВНОГО И УСЛОВНОГО ПЕРЕХОДА

Безусловный переход. Как было указано в предыдущем параграфе, последовательный способ выполнения команд менее гибок, чем свободный способ. Для устранения этого недостатка в машинах, работающих по первому способу, предусматривается возможность нарушения последовательности выполнения команд. Для этой цели служит так называемая команда *безусловного перехода*.

Операция безусловного перехода есть одноадресная операция вида $T\alpha$, где T — шифр операции, а α означает адрес команды, к выполнению которой следует перейти.

Пусть, например, в одноадресном коде после выполнения команды (20) следует перейти к выполнению команды (50). Тогда в качестве команды (21) вводят команду безусловного перехода $T\ 50$. После выполнения команд... (18), (19), (20) будет выполнена команда безусловного перехода (21), а затем команды (50), (51), (52)... В трехадресном коде команда безусловного перехода записывается в виде $T - \alpha$, первый и второй адреса свободны, третий адрес α есть адрес команды, к выполнению которой следует перейти.

Примеры использования команд безусловного перехода будут даны на стр. 161.

Команда условного перехода. Как неоднократно указывалось, при реализации алгоритмов возникают разветвления процесса: в зависимости от результатов предыдущих вычислений необходимо перейти на данном этапе к одному из двух возможных продолжений. В машинах, управляемых командами, это означает переход к одной из двух команд (α) или (β). Такое разветвление реализуется с помощью так называемой команды *условного перехода*.

Команда условного перехода отличается от команды безусловного перехода тем, что переход к выполнению команды, адрес которой указан в данной команде, совершается лишь в случае выполнения некоторого условия P .

В том случае, когда условие P не выполняется, в машине последовательного выполнения команд нарушения последовательности выполнения не происходит: машина переходит к выполнению команды, непосредственно следующей за командой условного перехода.

В машинах свободного выполнения команд команда условного перехода должна содержать адреса двух команд. В зависимости от выполнения или невыполнения условия P следующей оказывается та или другая из этих двух команд. Условием P обычно является наличие определенного знака у некоторого числа, образованного или в процессе предыдущих вычислений, или в результате операций над числами, адреса которых указаны в самой команде условного перехода. В машинах одноадресного кода речь идет о знаке числа, стоящего в накапливающем сумматоре. Число, знак которого определяет выполнение или невыполнение команды перехода, образуется чаще всего с помощью вычитания двух чисел в некоторых ячейках ЗУ α и β . Таким образом, два возможных результата сравнения чисел (α) и (β) дают тот или иной ход процессу вычислений. Так как содержание ячеек ЗУ α и β меняется в процессе вычислений, то меняется, вообще говоря, результат их сравнения и, следовательно, результат команды условного перехода.

Команды условного перехода дают, таким образом, возможность изменять течение вычислительного процесса в зависимости от полученных в процессе вычисления результатов и обеспечивают тем самым полную автоматичность работы вычислительной машины.

Как было указано выше (§ 14), машины могут воспринимать при сравнении (α) и (β) как дизъюнкцию ($\alpha \geq (\beta)$) или ($\alpha < (\beta)$), так и дизъюнкцию ($\alpha > (\beta)$) или ($\alpha \leq (\beta)$). Соответственно, в качестве условия P может фигурировать выполнение неравенства ($\alpha \geq (\beta)$) или ($\alpha > (\beta)$). Символом операции

условного перехода будет у нас служить знак неравенства \leq или $>$, в зависимости от той или иной формы условия.

Будем обозначать через \geq (соответственно, через $>$) операцию условного перехода, в которой признак P означает неравенство $(\alpha) \geq (\beta)$ (соответственно $(\alpha) > (\beta)$).

Команда условного перехода в машинах четырехадресного кода

$$\boxed{\begin{array}{|c|c|c|c|c|} \hline > & \alpha & \beta & \gamma & \delta \\ \hline \end{array}} \quad (1)$$

означает: произвести сравнение чисел (α) и (β) и если $(\alpha) \geq (\beta)$, то перейти к выполнению команды (γ) , а если $(\alpha) < (\beta)$, то перейти к выполнению команды (δ) .

В трехадресном коде команда условного перехода

$$\boxed{\begin{array}{|c|c|c|c|} \hline > & \alpha & \beta & \gamma \\ \hline \end{array}} \quad (2)$$

означает: сравнить числа (α) и (β) и если $(\alpha) \geq (\beta)$, то перейти к выполнению команды (γ) , а если $(\alpha) < (\beta)$, то перейти к выполнению следующей за командой условного перехода команды. Если $\alpha = \beta$, то команда (2) условного перехода превращается, очевидно, в команду безусловного перехода. Проще всего, как мы это будем делать в дальнейшем, считать $\alpha = \beta = 0$. Тогда команда в машине трехадресного кода

$$\boxed{\begin{array}{|c|c|c|c|} \hline > & & & \gamma \\ \hline \end{array}} = \boxed{\begin{array}{|c|c|c|c|} \hline > & 0 & 0 & \gamma \\ \hline \end{array}}$$

где адреса α и β отсутствуют (равны нулю), означает безусловный переход к выполнению команды γ .

Если сама команда условного перехода имеет адрес ε , то трехадресная команда (2) эквивалентна четырехадресной команде

$$\boxed{\begin{array}{|c|c|c|c|c|} \hline > & \alpha & \beta & \gamma & \varepsilon + 1 \\ \hline \end{array}}$$

Полностью четырехадресную команду (1) можно, очевидно, реализовать в виде двух последовательно следующих друг

за другом трехадресных команд условного и безусловного переходов.

$(\varepsilon) =$	$>$	α	β	γ
$(\varepsilon+1) =$	$>$			δ

Отмеченная только что эквивалентность четырехадресной команды двум трехадресным командам имеет место не только в случае команды условного перехода, но и в общем случае: одна четырехадресная команда

$(\varepsilon) =$	S_0	α	β	γ	δ
-------------------	-------	----------	---------	----------	----------

эквивалентна двум трехадресным командам

$(\varepsilon) =$	S_0	α	β	γ
$(\varepsilon+1) =$	$>$			δ

выполняемым в указанной последовательности.

Одна команда условного перехода в машинах трехадресного кода, осуществляющих команды в принудительной последовательности,

$>$	α	β	γ
-----	----------	---------	----------

эквивалентна трем следующим командам машины одноадресного кода:

$(\varepsilon - 2) = A\alpha$ — взять из ячейки ЗУ α число (α) и поместить его в предварительно очищенный счетчик A ;

$\{\varepsilon - 1\} = S\beta$ — вычесть из числа (α), находящегося в данный момент на счетчике A , число (β);

$(\varepsilon) = E\gamma$ — выполнить команду (γ), если $(A) = (\alpha) - (\beta) \geq 0$, в противном случае выполнить команду $(\varepsilon + 1)$

Если счетчик A не был предварительно очищен в результате предшествующих команд, то необходима еще какая-

либо команда ($\epsilon-3$), в результате которой стало бы $(A)=0$, т. е. счетчик A очистился бы.

Если в (1) или (2) опускается второй адрес, то мы получаем команду условного перехода, в котором критерием перехода является знак (α) (второе число (β) считается равным 0). Четырехадресная команда

\geq	α		γ	δ
--------	----------	--	----------	----------

означает: перейти к выполнению команды (γ), если $(\alpha) \geq 0$, или к выполнению команды (δ), если $(\alpha) < 0$. Аналогично трехадресная команда

\geq	α		γ
--------	----------	--	----------

означает: перейти к выполнению команды (γ), если $(\alpha) \geq 0$, или продолжать последовательное выполнение команд, если $(\alpha) < 0$.

Пример. Рассмотрим задачу составления программы вычисления функции $F(x)$, определяемую следующими условиями:

$$F(x) = \begin{cases} F_1(x) & \text{если } x > 0 \\ F_2(x) & \text{„ } x = 0 \\ F_3(x) & \text{„ } x < 0 \end{cases}$$

причем, после нахождения $F(x)$ надо перейти к выполнению некоторой команды L .

Так как здесь нас интересует лишь применение команд условных переходов, то мы предположим, что для вычисления каждой из функций $F_1(x)$, $F_2(x)$ и $F_3(x)$ требуется только одна команда соответственно K_1 , K_2 и K_3 .

Проще всего решается стоящая перед нами задача при использовании четырехадресных команд.

Пусть, например, $x = (10)$, $K_1 = (2)$, $K_2 = (6)$, $K_3 = (5)$, $L = (8)$. Легко убедиться в том, что для решения нашей задачи оказывается достаточным всего лишь две четырехадресных команды условного перехода:

$$\begin{aligned}
 (1) &= \begin{array}{|c|c|c|c|c|} \hline & \geq & & 10 & 4 & 2 \\ \hline \end{array} \\
 (2) &= K_1 \\
 (4) &= \begin{array}{|c|c|c|c|c|} \hline & \geq & 10 & & 6 & 5 \\ \hline \end{array} \\
 (5) &= K_3 \\
 (6) &= K_2 \\
 (8) &= L
 \end{aligned}$$

причем, четвертый адрес команд K_1 , K_2 , K_3 есть 8 (адрес команды L).

Действительно, если $x = (10) > 0$, то в силу команды (1) будет выполнена команда (2) = K_1 , осуществляющая вычисление функции $F_1(x)$. Если же $x = (10) \leq 0$, то в силу команды (1) должна быть выполнена команда (4), осуществляющая расщепление неравенства $x \leq 0$ на случай $x = 0$ и $x < 0$.

Если $x = 0$, то будет выполнена команда (6), осуществляющая вычисление $F_2(x)$, так как, в силу команды (4), команда (6) выполняется при условии $(10) = x \geq 0$, а сама команда (4) будет выполняться в силу команды (1) лишь при условии $0 \geq x = (10)$. Если же $x < 0$, то в силу команды (4) будет выполнена команда (5) = K_3 , осуществляющая вычисление функции $F_3(x)$. Во всех случаях после выполнения команд K_1 , K_2 и K_3 перейдем к выполнению команды L .

Приведем аналогичную программу в трехадресном коде

$$\begin{aligned}
 (1) &= \begin{array}{|c|c|c|c|} \hline & \geq & & 10 & 4 \\ \hline \end{array} \\
 (2) &= K_1 \\
 (3) &= \begin{array}{|c|c|c|c|} \hline & \geq & & & 8 \\ \hline \end{array} \\
 (4) &= \begin{array}{|c|c|c|c|} \hline & \geq & 10 & & 7 \\ \hline \end{array} \\
 (5) &= K_3 \\
 (6) &= \begin{array}{|c|c|c|c|} \hline & \geq & & & 8 \\ \hline \end{array} \\
 (7) &= K_2 \\
 (8) &= L
 \end{aligned}$$

§ 21. НЕПОСРЕДСТВЕННОЕ ПРОГРАММИРОВАНИЕ

Мы сейчас будем рассматривать простейшие программы, в которых каждая команда выполняется в процессе вычисления не более одного раза. Примером такой программы является приведенная в § 19 программа деления комплексных чисел. В ней каждая команда выполнялась ровно один раз. Сейчас приведем примеры программ с „разветвлением“ процесса вычислений, реализуемого с помощью команды условного перехода.

Решение квадратного уравнения с действительными коэффициентами. Корни квадратного уравнения

$$ax^2 + bx + c = 0 \quad (1)$$

вычисляются по формуле

$$x_1 = -\frac{b}{2a} + \sqrt{\left(\frac{b}{2a}\right)^2 - \frac{c}{a}}$$
$$x_2 = -\frac{b}{2a} - \sqrt{\left(\frac{b}{2a}\right)^2 - \frac{c}{a}}$$

Поместим константы $-\frac{1}{2}$, a , b , c в ячейках ЗУ C_1 , C_2 , C_3 , C_4 . Содержание и последовательность команд программы вычисления корней x_1 и x_2 на машине задается так же, как и при вычислении корней вручную. Отличие заключается лишь в том, что в программе для каждой команды нужно указать еще третий адрес ячейки, куда послать результат операции. В целях экономии ячеек ЗУ в третьем адресе не помещают для каждой команды новую, в данной программе еще не занятую, ячейку ЗУ, а используется свойство оперативного ЗУ, вследствие чего в третьем адресе можно указывать номер ячейки, вытесняемое содержимое которой не понадобится в дальнейшем.

Корни x_1 и x_2 могут быть комплексно сопряженными. Условимся впредь комплексные числа (в частности и действительные числа) помещать в двух соседних ячейках ЗУ $A + 2j - 1$ и $A + 2j$: в ячейку $A + 2j - 1$ помещаем действительную часть числа, а в ячейку $A + 2j$ помещаем мнимую

часть комплексного числа. Программу вычисления корней x_1 и x_2 составим по этапам.

1) Преобразование уравнения к приведенному виду осуществляется двумя командами деления

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K+1$:	C_3	C_2	$A+1$	В ячейке $A+1$ получим $\frac{b}{a}$
$K+2$:	C_4	C_2	$A+3$	" " $A+3$ " $\frac{c}{a}$

2) Образование дискриминанта $\left(\frac{b}{2a}\right)^2 - \frac{c}{a}$ осуществляется командами

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K+3$	\times	C_1	$A+1$	$A+1$	В ячейке $A+1$ получим $-\frac{b}{2a}$
$K+4$	\times	$A+1$	$A+1$	$A+5$	" " $A+5$ " $\left(\frac{b}{2a}\right)^2$
$K+5$	$-$	$A+5$	$A+3$	$A+3$	" " $A+3$ " $\left(\frac{b}{2a}\right)^2 - \frac{c}{a}$

3) Вычисление $\sqrt{\left|\left(\frac{b}{2a}\right)^2 - \frac{c}{a}\right|}$. Мы не будем составлять

подпрограмму вычисления квадратного корня, а примем операцию извлечения корня из положительного числа за элементарную и будем ее обозначать $\sqrt{\quad}$. Поэтому вычисление осуществляется двумя командами

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K+6$		$A+3$		$A+5$	В ячейке $A+5$ получим $\left \left(\frac{b}{2a}\right)^2 - \frac{c}{a}\right $
$K+7$	$\sqrt{\quad}$	$A+5$		$A+5$	" $A+5$ " $\sqrt{\left \left(\frac{b}{2a}\right)^2 - \frac{c}{a}\right }$

4) Вычисление корней x_1 и x_2 . Здесь процесс вычисления x_1 и x_2 разветвляется в зависимости от знака дискриминанта, а именно:

а) Дискриминант $\left(\frac{b}{2a}\right)^2 - \frac{c}{a} \geq 0$

Корни действительные и находятся двумя командами: сложением и вычитанием

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K + 9$	+	$A + 1$	$A + 5$	$A + 3$	В ячейке $A + 3$ получим корень x_1
$K + 10$	—	$A + 1$	$A + 5$	$A + 1$	„ $A + 1$ „ „ x_2

б) Дискриминант $\left(\frac{b}{2a}\right)^2 - \frac{c}{a} < 0$

Корни комплексно сопряженные и задаются парами чисел. В этом случае отыскание корней x_1 и x_2 не производим сложением чисел. Отыскание корней сводится к составлению двух пар ячеек ЗУ носителей x_1 и x_2 , что осуществляется командами передачи числа с прямым и обратным знаком.

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K + 12$	+	$A + 1$		$A + 3$	В паре $A + 3, A + 4$ получим x_1
$K + 13$	+	$A + 5$		$A + 4$	
$K + 14$	—		$A + 5$	$A + 2$	„ $A + 1, A + 2$ „ „ x_2

Объединение обоих случаев в одну программу производится операцией условного перехода. Команды вычисления корней x_1 и x_2 в обоих случаях остаются неизменными, а выбор случая а) или б) осуществляется командой $(K + 8)$ — сравнением $\left(\frac{b}{2a}\right)^2 - \frac{c}{a}$ с нулем.

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K + 8$	\geq		$A + 3$	$K + 12$	Если $\left(\frac{b}{2a}\right)^2 - \frac{c}{a}$ меньше нуля, то имеем случай б) и переходим к команде $K + 12$.

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K + 9$	+	$A + 1$	$A + 5$	$A + 3$	Случай а) Корни действительные
$K + 10$	—	$A + 1$	$A + 5$	$A + 1$	
$K + 11$	О с т а н о в к а				
$K + 12$	+	$A + 1$		$A + 3$	Случай б) Корни комплексные
$K + 13$	+	$A + 5$		$A + 4$	
$K + 14$	—		$A + 5$	$A + 2$	
$K + 15$	О с т а н о в к а				

Программа вычисления корней x_1 и x_2 в целом имеет вид:

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K + 1$:	C_3	C_2	$A + 1$	Ячейки $A + 2$, $A + 4$ вначале пустые
$K + 2$:	C_4	C_2	$A + 3$	
$K + 3$	\times	C_1	$A + 1$	$A + 1$	
$K + 4$	\times	$A + 1$	$A + 1$	$A + 5$	
$K + 5$	—	$A + 5$	$A + 3$	$A + 3$	
$K + 6$		$A + 3$		$A + 5$	
$K + 7$	$\sqrt{\quad}$	$A + 5$		$A + 5$	
$K + 8$	\geq		$A + 3$	$K + 12$	
$K + 9$	+	$A + 1$	$A + 5$	$A + 3$	
$K + 10$	—	$A + 1$	$A + 5$	$A + 1$	
$K + 11$	О с т а н о в к а				
$K + 12$	+	$A + 1$		$A + 3$	
$K + 13$	+	$A + 5$		$A + 4$	
$K + 14$	—		$A + 5$	$A + 2$	
$K + 15$	О с т а н о в к а				

Решение кубического уравнения с действительными коэффициентами. В качестве следующего примера рассмотрим кубическое уравнение

$$ay^3 + by^2 + cy + d = 0 \quad (2)$$

с действительными коэффициентами a, b, c, d .
Обозначим

$$\frac{p}{3} = -\left(\frac{b}{3a}\right)^2 + \frac{c}{3a}$$

$$\frac{q}{2} = \left(\frac{b}{3a}\right)^3 - \frac{bc}{6a^2} + \frac{d}{2a}$$

Корни x_1, x_2 и x_3 уравнения $x^3 + px + q = 0$, выраженные по формуле Кардана, имеют вид:

1. Если $\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3 \geq 0$, то

$$\left. \begin{aligned} x_1 &= u + v \\ x_2 &= -\frac{1}{2}(u + v) + \frac{i\sqrt{3}}{2}(u - v) \\ x_3 &= -\frac{1}{2}(u + v) - \frac{i\sqrt{3}}{2}(u - v) \end{aligned} \right\} \quad (3)$$

где u — есть действительное значение корня

$$\sqrt[3]{-\frac{q}{2} + \sqrt{\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3}}, \quad v = -\frac{p}{3u}$$

2. Если $\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3 < 0$, то

$$\left. \begin{aligned} x_1 &= 2r \cos \varphi \\ x_2 &= -r (\cos \varphi + \sqrt{3} \sin \varphi) \\ x_3 &= +r (\sqrt{3} \sin \varphi - \cos \varphi) \end{aligned} \right\} \quad (4)$$

где $r = \sqrt[3]{-\frac{p}{3}}$, $\varphi = \frac{1}{3} \arctg \frac{\sqrt{\left|\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3\right|}}{-\frac{q}{2}}$

Как и раньше, комплексное число задается в двух соседних ячейках ЗУ $A + 2j - 1, A + 2j$: действительную часть помещаем в ячейку $A + 2j - 1$, мнимую — в ячейку $A + 2j$.

Составим программу вычисления корней x_1, x_2 и x_3 . В начале процесса вычисления в ЗУ хранятся следующие числа:

№ ячейки	Содержимое ячейки
C_1	a
C_2	b
C_3	c
C_4	d
C_5	$3/2$
C_6	$1/3$
C_7	$1/2$
C_8	$\sqrt{3}$
C_9	$-1/2$
C_{10}	$\sqrt{3/2}$
C_{11}	1
$A+2$	0
$A+4$	0
$A+6$	0

Программу составим по этапам:

1) Вычисление $-\frac{p}{3}$ и $-\frac{q}{2}$. Более экономно по количеству операций располагать вычисления по схеме:

$$-\frac{p}{3} = \frac{1}{3a} \left(\frac{b^2}{3a} - c \right)$$

$$-\frac{q}{2} = \frac{1}{3a} \left[\frac{b}{3a} \left(\frac{3}{2}c - \frac{b^2}{3a} \right) - \frac{3}{2}d \right]$$

Адрес коман- ды	Опера- ция	1 адрес	2 адрес	3 адрес	Примечания
$K+1$:	C_6	C_1	B_1	В ячейке B_1 получим $\frac{1}{3a}$
$K+2$	\times	C_2	B_1	B_2	" " B_2 " $\frac{b}{3a}$
$K+3$	\times	C_2	B_2	B_3	" " B_3 " $\frac{b^2}{3a}$
$K+4$	$-$	B_3	C_3	B_3	" " B_3 " $\frac{b^2}{3a} - c$

Адрес коман- ды	Опера- ция	1 адрес	2 адрес	3 адрес	Примечания
$K+5$	\times	B_3	B_1	B_4	В ячейке B_4 получим $-\frac{p}{3}$
$K+6$	\times	C_3	C_7	B_5	" " B_5 " $\frac{c}{2}$
$K+7$	$-$	B_5	B_3	B_3	" " B_3 " $\frac{3}{2}c - \frac{b^2}{3a}$
$K+8$	\times	B_3	B_2	B_3	" " B_3 " $\frac{b}{3a}\left(\frac{3}{2}c - \frac{b^2}{3a}\right)$
$K+9$	\times	C_4	C_5	B_2	" " B_2 " $\frac{3}{2}d$
$K+10$	$-$	B_3	B_2	B_3	" " B_3 " $\frac{b}{3a}\left(\frac{3}{2}c - \frac{b^2}{3a}\right) - \frac{3}{2}d$
$K+11$	\times	B_3	B_1	B_3	" " B_3 " $-\frac{q}{2}$

2) Вычисление подкоренного выражения $\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3$ осуществляется четырьмя командами:

					Примечания
$K+12$	\times	B_3	B_3	B_2	В ячейке B_2 получим $\left(\frac{q}{2}\right)^2$
$K+13$	\times	B_4	B_4	B_1	" " B_1 " $\left(\frac{p}{3}\right)^2$
$K+14$	\times	B_1	B_4	B_1	" " B_1 " $-\left(\frac{p}{3}\right)^3$
$K+15$	$-$	B_2	B_1	B_1	" " B_1 " $\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3$

3) Вычисление $\sqrt{\left|\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3\right|}$. Как и раньше будем считать извлечение корня из положительного числа элементарной операцией, поэтому вычисление $\sqrt{\left|\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3\right|}$ выполняется двумя командами.

Примечания

$K+16$	<table> <tr> <td> </td> <td> </td> <td>B_1</td> <td></td> <td>B_2</td> </tr> </table>			B_1		B_2	В ячейке B_2 получим $\left \left(\frac{q}{2} \right)^2 + \left(\frac{p}{3} \right)^3 \right $
		B_1		B_2			
$K+17$	<table> <tr> <td>$\sqrt{-}$</td> <td></td> <td>B_2</td> <td></td> <td>B_2</td> </tr> </table>	$\sqrt{-}$		B_2		B_2	" " B_2 " $\sqrt{\left \left(\frac{q}{2} \right)^2 + \left(\frac{p}{3} \right)^3 \right }$
$\sqrt{-}$		B_2		B_2			

4) Вычисление корней x_1 , x_2 и x_3 . Здесь процесс вычисления корней разветвляется в зависимости от знака $\left(\frac{q}{2} \right)^2 + \left(\frac{p}{3} \right)^3$, а именно:

а) $\left(\frac{q}{2} \right)^2 + \left(\frac{p}{3} \right)^3 < 0$. В этом случае все корни действительные.

Условимся считать элементарными следующие операции:

операцию \cos — нахождение $\cos \varphi$ по φ ;

операцию $\operatorname{arctg} (x, y)$ — нахождение ψ по значениям

$$\sin \psi = \frac{x}{\sqrt{x^2 + y^2}} \text{ и } \cos \psi = \frac{y}{\sqrt{x^2 + y^2}}.$$

Соответственно этому программа вычисления корней x_1 , x_2 и x_3 запишется так:

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K+19$	$\sqrt{-}$	B_4		B_1	В ячейке B_1 получим $\sqrt{-\frac{p}{3}}$
$K+20$	arctg	B_2	B_3	B_2	Для получения ψ в соответственной четверти нужно брать ячейки, содержащие x и y
$K+21$	\times	B_2	C_6	B_2	В ячейке B_2 получим $\frac{1}{3} \operatorname{arctg} \frac{\sqrt{\left \left(\frac{q}{2} \right)^2 + \left(\frac{p}{3} \right)^3 \right }}{-\frac{q}{2}} = \varphi$
$K+22$	\cos	B_2		B_2	В ячейке B_2 получим $\cos \varphi$
$K+23$	\times	B_2	B_2	B_3	" B_3 " $\cos^2 \varphi$
$K+24$	$-$	C_{11}	B_3	B_3	" B_3 " $1 - \cos^2 \varphi$
$K+25$	$\sqrt{-}$	B_3		B_3	" B_3 " $\sin \varphi$
$K+26$	\times	B_3	C_8	B_3	" B_3 " $\sqrt{3} \sin \varphi$

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K + 27$	\times	B_1	B_3	B_3	В ячейке B_3 получим $r\sqrt{3}\sin\varphi$
$K + 28$	\times	B_1	B_2	B_2	" B_2 " $r\cos\varphi$
$K + 29$:	B_2	C_7	$A + 1$	Корень $x_1 = 2r\cos\varphi$
$K + 30$	--	B_3	B_2	$A + 3$	" $x_3 = +r(\sqrt{3}\sin\varphi - \cos\varphi)$
$K + 31$	+	B_3	B_2	B_2	В ячейке B_2 получим $r(\sqrt{3}\sin\varphi + \cos\varphi)$
$K + 32$	-		B_2	$A + 5$	Корень $x_2 = -r(\cos\varphi + \sqrt{3}\sin\varphi)$
$K + 33$		Остановка			

б) $\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3 \geq 0$. Корни считаются по формуле (3).

Условимся извлечение кубического корня считать элементарной операцией и обозначать ее $\sqrt[3]{}$. Программа продолжения вычисления корней по формуле (3) после команды ($K + 17$) имеет вид:

Адрес команды	Операция	1 адрес	2 адрес	3 адрес	Примечания
$K + 34$	+	B_2	B_3	B_2	В ячейке B_2 получим $-\frac{q}{2} + \sqrt[3]{\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3}$
$K + 35$	$\sqrt[3]{}$	B_2		B_2	В ячейке B_2 получим u
$K + 36$:	B_4	B_2	B_1	" B_1 " $v = -\frac{p}{3u}$
$K + 37$	+	B_2	B_1	$A + 1$	" $A + 1$ " x_1
$K + 38$	\times	$A + 1$	C_9	$A + 3$	" $A + 3$ " $-\frac{u+v}{2}$
$K + 39$	-	B_2	B_1	$A + 4$	" $A + 4$ " $(u - v)$
$K + 40$	\times	$A + 4$	C_{10}	$A + 4$	В паре $A + 3, A + 4$ получим корень x_2
$K + 41$	+	$A + 3$		$A + 5$	В ячейке $A + 5$ получим $-\frac{u+v}{2}$
$K + 42$	-		$A + 4$	$A + 6$	В паре $A + 5, A + 6$ получим корень x_3
$K + 43$		Остановка			

Объединение случаев а) и б) в одну программу осуществляется командой сравнения ($K+18$); программы случаев а) и б) остаются неизменными. Команда ($K+18$) имеет вид

$$K+18 \quad \left| \begin{array}{c|c|c|c} \geq & B_1 & & K+34 \end{array} \right|$$

что означает: если $\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3 \geq 0$, переходим к команде $K+34$, если же $\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3 < 0$, то переходим к следующей команде.

§ 22. ПРЕОБРАЗОВАНИЕ КОМАНД

Команды обычно кодируются набором цифр в системе, в которой оперирует машина, — двоичной или десятичной. Набор, изображающий команду, можно рассматривать как код некоторого числа. Пусть, например, машина, работающая по десятичной системе с фиксированной запятой, выполняет 9 операций, зашифрованных однозначными десятичными числами (1 — сложение, 2 — умножение, ..., 8 — команда безусловного перехода, 9 — команда условного перехода) и имеет 999 ячеек ЗУ, адреса которых кодируются трехзначными десятичными числами (от 1 до 999). Тогда набор, которым шифруется команда по трехадресному коду, запишется 10-значным десятичным числом, первая цифра которого означает код операции, а 3 группы по 3 цифры означают 3 адреса. Например, команду: выполнить операцию 2 (умножения) над числами, хранящимися в ячейках с адресами 436 и 574, и послать ответ по адресу 478 можно зашифровать в виде числа:

$$K = 2\,436\,574\,478$$

Хранение команд в ЗУ и ввод их в машину ничем не отличается от хранения и ввода чисел. Можно даже делать операции над кодами команд. Если, например, к коду K добавить число 0001001000, то получим набор

$$K_1 = 2\,437\,575\,478$$

который будет кодом команды: совершить операцию 2 (умножение) над числами, хранящимися в ячейках 437 и 575,

и послать ответ по адресу 478. Такая возможность преобразовывать команды, осуществленная в большинстве новых машин, чрезвычайно важна при составлении программ.

Пример. Пусть требуется вычислить сумму $S = \sum_{k=1}^{100} a_k$, где a_k — число, хранящееся в ячейке ЗУ $100 + k$ ($k = 1, 2, \dots, 100$). Обозначим $S_l = \sum_{k=1}^l a_k$ ($S_0 = 0$). Чтобы найти S_{100} , нужно последовательно найти все S_l , $l = 1, 2, \dots, 100$, по формуле $S_l = S_{l-1} + a_l$. Можно построить для этого программу из 100 команд. Обозначаем попрежнему (в десятичной системе) цифрами 1, 2, ..., 9 коды операций (1 есть код сложения), а затем трехзначными числами коды трех адресов (двух адресов слагаемых и адрес суммы); суммы S_l будем хранить в ячейке 300 (в начальный момент в этой ячейке стоит число 0). Получим прямую программу из 100 команд:

Программа А

№ команды	Команда
1	1 300 101 300
2	1 300 102 300
3	1 300 103 300
...
99	1 300 199 300
100	1 300 200 300
	Остановка или переход к другим операциям

Команда № 1: сложить числа, стоящие в ячейках 300 (число $S_0 = 0$) и 101 (число a_1), и полученную сумму $S_1 = S_0 + a_1$ послать в ячейку 300; команда № 2: сложить число из ячейки 300 (т. е. S_1) с числом из ячейки 102 (т. е. a_2) и сумму $S_2 = S_1 + a_2$ послать снова в ячейку 300; после выполнения команды № 3 в ячейке 300 будет стоять число S_3 и т. д., после выполнения команды № 99 — в ячейке 300

будет стоять число S_{99} , а после команды № 100 — искомое число S_{100} .

Заметим сейчас, что все 100 команд прямой программы отличаются лишь вторыми адресами; каждый следующий получается из предыдущего путем прибавления единицы ко второму адресу, т. е. числа (0 000 001 000) к коду предыдущей команды. Пусть это число хранится в ячейке 301, код команды № 1 — в ячейке 1. Напомним, что 8 означает команду безусловного, а 9 — команду условного перехода.

Рассмотрим три последовательных команды:

$$(1) = 1\ 300\ 101\ 300$$

$$(2) = 1\ 001\ 301\ 001$$

$$(3) = 8\ 000\ 000\ 001$$

После выполнения команды (1) (образование суммы S_1 в ячейке 300), команды (2) (преобразование команды (1) в команду (2), программы A) и команды (3) — перехода, мы возвращаемся к выполнению команды (1), которая теперь имеет вид:

$$(1) = 1\ 300\ 102\ 300$$

После ее выполнения образуется сумма S_2 в ячейке 300; далее снова выполним команду (2), которая теперь преобразует команду (1) в команду (3) программы A , и снова вернемся к повторению процесса. Мы будем последовательно образовывать суммы S_k ($k = 1, 2, \dots, 100$) и после 100-го повторения цикла получим искомое число S_{100} в ячейке 300, после чего нужно было бы прекратить процесс. Необходимо несколько изменить и дополнить нашу программу так, чтобы после 100-го повторения цикла работа по образованию сумм S_k прекратилась.

Обеспечить эту заранее заданную стократную повторяемость операций можно, заменив команду безусловного командой условного перехода. Пусть в ячейке β (например, $\beta = 402$) хранится в начале процесса число 0, а в ячейке γ (например, $\gamma = 403$) хранится число $n = 1$. Если после каждого выполнения цикла команд (1) и (2) совершать операцию, выраженную трехадресной командой

$$1\ 402\ 403\ 402$$

(сложить содержимое ячейки 402 с содержимым ячейки 403 и ответ послать в ячейку 402), то в последней ячейке после выполнения 1-го, 2-го, ..., k -го циклов наших операций будет стоять число $1, 2, \dots, k$, т. е. число выполненных циклов.

Нужно прекратить производство этих операций после 100 циклов, т. е. после того как в ячейке 402 будет стоять число 100. Поместим в ячейку 404 число 99 и дополним наш цикл команд командой (4) условного перехода

9 404 403 001

Если $(404) = 99$ больше или равно $(403) = k$, то возвращаемся к выполнению команды (1). Если же $(403) = k = 100$, то нарушается неравенство $(404) \geq (403)$, и мы переходим к выполнению следующей команды, которая будет командой остановки машины или перехода к следующим операциям. Вся программа суммирования состоит из 4 команд:

- (1) = 1 300 101 300
- (2) = 1 001 301 001
- (3) = 1 402 403 402
- (4) = 9 404 403 001
- (5) Остановка или
переход к сле-
дующим операциям

Первая команда вызывает добавление слагаемого a_k (вначале $a_k = a_1$) к сумме предыдущих; вторая — подготовку перехода к следующему циклу; третья — подсчет числа k уже произведенных циклов; четвертая — сравнение этого числа с числом 99 и при $k < 100$ возвращение к повторению цикла; при $k = 100$ машина переходит к команде окончания работы. Ответ S_{100} находится в ячейке 300. Здесь мы встретились с программированием циклических операций, о котором подробнее будет речь в следующем параграфе.

В действительности, подсчет числа уже произведенных сложений при помощи прибавления 1 к содержимому ячейки 402 является по существу излишним, так как число таких сложений можно легко обнаружить по содержанию ячейки 1,

т. е. по команде (1). После k -кратного повторения пары команд (1) и (2), команда (1) будет иметь вид:

$$1\ 300(101 + k) 300$$

После 100-кратного повторения этой пары команд команда (1) будет иметь вид:

$$1\ 300\ 201\ 300$$

Если в какой-нибудь ячейке ЗУ, например 405, хранится число 1 300 200 300, то команду (1) следует повторять до тех пор, пока выполняется условие $(405) \geq (1)$. Программа примет следующий вид:

- | | |
|---------------------|--|
| (1) = 1 300 101 300 | Суммирование |
| (2) = 1 001 301 001 | Изменение команды |
| (3) = 9 405 001 001 | Возвращение к команде (1) при $(405) \geq (001)$ |

Таким образом, в команде условного перехода объектом сравнения бывает разумно делать числа, кодирующие команды.

Для реализации программы, связанной с преобразованием команд, последние должны храниться во внутреннем вытесняющем ЗУ, в то время как при реализации других программ команды могут храниться в постоянном ЗУ.

В машинах с плавающей запятой при добавлении числа α к i -му адресу α_i команды $S \dots \alpha_i \dots$ следует снабдить число α соответствующим порядком. Для того чтобы избежать связанных с этим неудобств, целесообразно предусмотреть специальные операции сложения без учета порядка, т. е. сложения, при котором наборы, изображающие числа, складываются как числа с фиксированной запятой. Такое сложение будем обозначать символом $\dot{+}$. Можно также предусмотреть, как элементарную операцию, прибавление числа к данному адресу.

Иногда приходится заменять в команде $(A) = S \alpha_1 \alpha_2 \dots \alpha_k$ число α_i , стоящее в i -м адресе этой команды, некоторым числом β , хранящимся в некоторой ячейке ЗУ, причем

заранее может быть неизвестна величина $\beta - \alpha_i$ (иногда нам заранее неизвестна и величина α_i). Такое преобразование команды нельзя осуществить непосредственно прибавлением $(\beta - \alpha_i)$ к i -му адресу команды. В этом случае нам нужно предварительно очистить i -й адрес команды $S \dots \alpha_i \dots$ от числа α_i , т. е. заменить все цифры α_i нулями (команде (А) придать вид $S \dots 0 \dots$). Проще всего произвести такую операцию с помощью сличения, если оно имеется в числе элементарных операций машины. Очищение последнего адреса, занимающего l разрядов, можно произвести также двумя последовательными сдвигами вправо и влево на l разрядов. Для того чтобы без операции сличения очистить какой-либо средний адрес команды, нужно выделить его с помощью последовательных сдвигов в обе стороны, а затем вычесть из команды.

Очень часто, после того как некоторая команда (K_0) подвергалась однократному или многократным преобразованиям, приходится возвращать ее к первоначальному виду. Это имеет, например, место, если после выполнения цикла операций, включавшего изменения команды (K_0) и выполнения команды (K_0) в первоначальном и измененном виде, снова приходится возвращаться к выполнению этого цикла (примеры таких циклов приведены в § 30). Такая „реставрация“ команды (K_0) может совершаться различными способами.

Можно выполнить над измененной командой (K_0) операцию, обратную совокупности операций, которым последовательно подвергалась команда (K_0). В примере, разобранным в настоящем параграфе, команда (1) сто раз подвергалась последовательно преобразованию — прибавлению единицы ко второму адресу; для реставрации этой команды нужно произвести вычитание числа 100 из второго адреса измененной команды (100).

Для образования изменяющихся в процессе выполнения команд, а также для их реставрации во многих случаях можно рекомендовать следующий прием: пусть команда (K) программы

$$(K) = S\alpha\beta\gamma$$

должна подвергаться неоднократным изменениям (обычно изменяются ее адреса α , β и γ). В записи программы оставим ячейку K пустой. В то же время в некоторой дополнительной ячейке хранится первоначальная форма команды (K):

$$(I) = S\alpha_0\beta_0\gamma_0$$

Над содержимым ячейки, т. е. этой начальной формой команды (K), совершаются нужные преобразования и результаты посылаются в ячейку K . Программа составляется таким образом, чтобы к моменту очередного выполнения команды (K) в ячейке K находилась нужная форма этой команды. Предыдущий метод преобразования команд можно употреблять только тогда, когда изменение команд совершается по особо простому закону (добавление 1 или другой константы к какому-либо из адресов и т. п.). Этот же метод пригоден при любом законе изменения команд. Реставрация команды является при этом методе излишней. Кроме того, этот метод следует считать более надежным, так как каждое новое формирование той же команды (K) совершается независимо от результатов предыдущих формирований.

Этот способ формирования команд применен в программе § 30 решения системы линейных алгебраических уравнений методом главных элементов и в программе гармонического анализа (стр. 228).

§ 23. ПЕРЕЧЕНЬ КОМАНД И ОБОЗНАЧЕНИЙ

В этом параграфе мы приведем перечень рассмотренных операций, а также их обозначений, употребляемых в дальнейшем. Как и раньше (α) обозначает содержимое ячейки ЗУ с номером α , или, как мы будем для краткости писать, ячейки α .

Программы и команды зависят, очевидно, от того, как хранятся числа в машине, какие команды она выполняет, сколько адресов имеют коды этих команд. Мы все время будем говорить о машинах, работающих по трехадресному коду. Переход к остальным случаям описан в § 19 и труда не представляет.

1. Сложение

+	α	β	γ
---	----------	---------	----------

Сложить (α) и (β), результат отправить в ячейку γ .

2. Вычитание

-	α	β	γ
---	----------	---------	----------

Вычесть (β) из (α), результат отправить в ячейку γ .

3. Умножение

\times	α	β	γ
----------	----------	---------	----------

Умножить (α) на (β), результат отправить в ячейку γ .

4. Деление

:	α	β	γ
---	----------	---------	----------

Разделить (α) на (β), результат отправить в ячейку γ .

5. Сложение без учета порядков¹

+	α	β	γ
---	----------	---------	----------

Сложить отдельно цифровые части (α) и (β) и рядки (α) и (β), полученный цифровой набор отправить в ячейку γ . Аналогичную операцию можно ввести и для вычитания

6. Условный переход

\geq	α	β	γ
--------	----------	---------	----------

Сравнить (α) и (β). Если (α) < (β), перейти к выполнению следующей команды; если (α) \geq (β), перейти к выполнению команды γ .

7. Остановка²

--	--	--	--

Остановить машину.

8. Переход к местному управлению³

М			α
---	--	--	----------

Перейти к местному управлению, выполнить команду α .

9. Переход к центральному управлению³

Ц			
---	--	--	--

Перейти к центральному управлению.

10. Посылка в печать

П	α		
---	----------	--	--

Послать (α) во внешнее устройство для печати.

¹ Если в машине запятая фиксирована, то в этой операции надобности нет, она совпадает с операцией 1.

² Команда остановки осуществляется также и в случаях, когда в коде команды появляется совокупность цифр, не соответствующая ни одному шифру какой-либо другой команды.

³ Эта операция есть только в машинах с отдельными местным и центральным управлениями (см. § 25).

Частным случаем этих операций являются, очевидно, следующие:

11. Пересылка

+	α		β
---	----------	--	---------

 или

+		α	β
---	--	----------	---------

 Послать (α) в ячейку β .

12. Пересылка с обратным знаком

-		α	β
---	--	----------	---------

 Послать $-(\alpha)$ в ячейку β .

13. Гашение содержимого ячейки

+			α
---	--	--	----------

 Послать 0 в ячейку α (очистить ячейку α).

14. Операция безусловного перехода

>			α
---	--	--	----------

 Перейти к выполнению команды (α) .

Эти операции встречаются при решении математических задач очень часто и могут считаться основными. Приведем операции, которые либо встречаются сравнительно редко, либо могут быть легко сведены к выполнению основных операций.

15. Нахождение абсолютной величины числа

		α		β
--	--	----------	--	---------

 Найти $|(\alpha)|$, послать результат в ячейку β .

16. Сравнение по абсолютной величине

>	α	β	γ
---	----------	---------	----------

 Сравнить $|(\alpha)|$ и $|(\beta)|$. Если $|(\alpha)| < |(\beta)|$, перейти к выполнению следующей команды; если $|(\alpha)| \geq |(\beta)|$, перейти к выполнению команды γ .

17. Нахождение целой и дробной частей числа

E	α	β	γ
---	----------	---------	----------

 Найти E $((\alpha))$ и $\{(\alpha)\}$. E $((\alpha))$ отправить в ячейку β , $\{(\alpha)\}$ — в ячейку γ .

18. Логическое умножение

\wedge	α	β	γ
----------	----------	---------	----------

 Над наборами, представляющими (α) и (β) , производится поразрядная операция по правилу: $0 \wedge 0 = 0$, $0 \wedge 1 = 0$, $1 \wedge 0 = 0$, $1 \wedge 1 = 1$. Результат посылается в ячейку γ .

19. Логическая сумма

\vee	α	β	γ
--------	----------	---------	----------

 Над наборами, представляющими (α) и (β) , производится поразрядная операция по правилу: $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$. Результат посылается в ячейку γ .

20. Операция сдвига

→	α	β	γ
---	----------	---------	----------

Набор, представляющий (α) , сдвинуть на $E((\beta))$ вправо (при $(\beta) < 0$ сдвиг фактически производится влево). Освобождающиеся разряды заполняются нулями. Аналогичную операцию можно ввести для сдвига влево.

21. Пересылка порядка в цифровую часть

↓	α		β
---	----------	--	---------

Если (α) представлено в виде $2^p \cdot 0, a_1 a_2 \dots$ (a_1 не обязательно 1), то послать в нормализованном виде p в ячейку β . Аналогично можно определить пересылку с обратным знаком порядка в цифровую часть.

22. Пересылка числа в порядок

↑	α		β
---	----------	--	---------

Послать в нормализованном виде в ячейку β число $2^{E((\alpha))}$; в порядке (β) получим $E((\alpha)) + 1$. Аналогично можно определить пересылку с обратным знаком числа в порядок.

Для действий с удвоенным числом разрядов (см. § 16) необходимы следующие операции:

23. Умножение с выводом полного числа разрядов

\times	α	β	γ
			δ

Умножить (α) на (β) . Первые разряды произведения с получившимся порядком поместить в ячейку γ , последние с тем же порядком — в ячейку δ .

24. Деление с выводом остатка

:	α	β	γ
			δ

Разделить (α) на (β) . Частное поместить в ячейку γ , цифры остатка с порядком частного — в ячейку δ .

При операциях с блокировкой нормализации мы рядом с символом операции будем ставить запятую, например, $+$, $-$, $:$, и т. д.

При записи команды мы считаем, что шифр операции занимает разряды порядка, а адреса — разряды цифровой части.

Набор операций, принятых в машине (так называемых элементарных операций), может быть чрезвычайно разнообразным. В машине Mark I в качестве элементарных операций были операции нахождения \sqrt{x} , $\sin x$, $\cos x$, $\log x$ и др. В большинстве машин вычисление этих функций осуществляется специальным программированием. Однако при программировании задачи, в которой эти функции встречаются, можно операцию нахождения этих функций условно считать элементарной. При окончательном программировании это

соответствует отсылке к местным подпрограммам (см. § 25) или вставке в это место нужного куска.

При составлении программ мы будем пользоваться следующими общими для всех программ обозначениями.

Каждая программа состоит из двух частей: перечня ячеек ЗУ, содержащих исходные, вспомогательные и получаемые величины, и совокупности команд, реализующих программу. Первые ячейки мы будем обозначать обычно $A_1, A_2, \dots, a_1, a_2, \dots, B_1, B_2, \dots, b_1, b_2, \dots$, если последовательность их адресов несущественна, или $A + 1, A + 2, \dots, B + 1, B + 2, \dots$, если она существенна. Адреса ячеек, хранящих команды, будем обозначать $K + 1, K + 2, \dots$ (или в случае надобности $\dots K - 2, K - 1, K$). Подобная систематика в обозначении упрощает чтение программы. Надписи над программой, „адрес команды“, „операция“ и т. д., мы будем опускать. Если записывается содержимое нескольких ячеек, возможно вместо

$$(A_1) = \boxed{\alpha_1} \quad (A_2) = \boxed{\alpha_2}$$

писать $\begin{matrix} A_1 \\ A_2 \end{matrix} \boxed{\begin{matrix} \alpha_1 \\ \alpha_2 \end{matrix}}$. Это, в частности, всюду принято

нами для команд. Примечания (они обычно касаются результата операции) пишутся справа от содержимого соответствующей ячейки. Выполнение программы начинается всегда с первой команды программы.

Таким образом, первая программа § 19 в принятых обозначениях выглядит так:

A_1	a
A_2	b
A_3	c
A_4	d
B_1	что угодно
B_2	" "
B_3	" "
B_4	" "

$K + 1$	\times	A_3	A_3	B_1	$(B_1) = c^2$
$K + 2$	\times	A_4	A_4	B_2	$(B_2) = d^2$
$K + 3$	$+$	B_1	B_2	B_1	$(B_1) = c^2 + d^2$
$K + 4$	\times	A_1	A_3	B_2	$(B_2) = ac$
$K + 5$	\times	A_2	A_4	B_3	$(B_3) = bd$
$K + 6$	$+$	B_2	B_3	B_2	$(B_2) = ac + bd$
$K + 7$	\times	A_2	A_3	B_3	$(B_3) = bc$
$K + 8$	\times	A_1	A_4	B_4	$(B_4) = ad$
$K + 9$	$-$	B_3	B_4	B_3	$(B_3) = bc - ad$
$K + 10$	$:$	B_2	B_1	B_2	$(B_2) = x = \frac{ac + bd}{c^2 + d^2}$
$K + 11$	$:$	B_3	B_1	B_3	$(B_3) = y = \frac{bc - ad}{c^2 + d^2}$
$K + 12$					

§ 24. ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ ПРОЦЕССОВ

Выше неоднократно указывалось, что основным фактором, дающим возможность реализовать в автоматических машинах алгоритмы решения многих задач, является распадение всего процесса решения на циклы повторяющихся операций.

Мы сейчас подробно рассмотрим программирование задач, распадающихся на циклы, иллюстрируя это на конкретных примерах. Простейший пример рассмотрен в § 22.

Пример 1. Пусть нужно найти сумму $n + 1$ членов ряда $1 + \frac{x}{1} + \frac{x^2}{1 \cdot 2} + \frac{x^3}{1 \cdot 2 \cdot 3} + \dots + \frac{x^n}{1 \cdot 2 \dots n}$. Очевидно, общий ход вычислений должен быть таким: получить $a_1 = \frac{x}{1}$ и прибавить его к $s_0 = 1$, получим $s_1 = s_0 + a_1 = 1 + \frac{x}{1}$; вычислить $a_2 = \frac{a_1 x}{2}$ и прибавить к s_1 , получим s_2 и т. д. Для этой задачи можно составить прямую программу.

Поместим в ячейках ЗУ величины:

$$(A) = x, (B) = 1, (C) = 1, (D) = 1, (E) = x$$

В ячейке ЗУ A последовательно будут храниться a_1, a_2, \dots, a_n , в ячейке C накапливается нужная сумма, в ячейке

D — величина p — множитель, на который надо умножить $(p-1)!$ в знаменателе, чтобы получить $p!$.

Программа в трехадресном коде будет иметь вид:

$K+1$	+	A	C	C	A	x
$K+2$	+	B	D	D	B	1
$K+3$	\times	A	E	A	C	1
$K+4$:	A	D	A	D	1
$K+5$	+	A	C	C	E	x
$K+6$	+	B	D	D		
$K+7$	\times	A	E	A		
$K+8$:	A	D	A		
...		
$K+4n-4$:	A	D	A		
$K+4n-3$	+	A	C	C		

Как видно, программа состоит из вычисления s_1 по команде $(K+1)$ и повторения серии четырех команд

+	B	D	D
\times	A	E	A
:	A	D	A
+	A	C	C

Если включить в общую схему и вычисление a_1 , то программа станет совсем однообразной:

$K+1$	+	B	D	D	A	1
$K+2$	\times	A	E	A	B	1
$K+3$:	A	D	A	C	1
$K+4$	+	A	C	C	D	0
$K+5$	+	B	D	D	E	x
$K+6$	\times	A	E	A		
$K+7$:	A	D	A		
$K+8$	+	A	C	C		
$K+9$	+	B	D	D		
...		
$K+4n-1$:	A	D	A		
$K+4n$	+	A	C	C		

Вся программа представляет собой повторение n раз приведенной выше серии четырех команд.

Можно осуществить повторение цикла этих четырех команд следующей программой

$K+1$	+	B	D	D
$K+2$	\times	A	E	A
$K+3$:	A	D	A
$K+4$	+	A	C	C
$K+5$	\geq			$K+1$

Однако процесс вычислений по этой программе продолжался бы, очевидно, бесконечно, а нам нужно закончить вычисления после $n+1$ -го члена. Следовательно, нужно изменить смысл команды $(K+5)$: переход к команде $(K+1)$ должен быть совершен, если число сосчитанных членов меньше $n+1$; вычисления должны прекратиться, если оно равно $n+1$; это может быть осуществлено командой условного перехода. Содержание ячейки D укажет нам номер только что вычисленного члена ряда. Поэтому команда $(K+5)$ условного перехода должна иметь вид:

$K+5$	\geq	F	D	$K+1$
-------	--------	-----	-----	-------

причем в ячейке F хранится число $n-1$.

Вычисления по программе

$K+1$	+	B	D	D	A	1
$K+2$	\times	A	E	A	B	1
$K+3$:	A	D	A	C	1
$K+4$	+	A	C	C	D	0
$K+5$	\geq	F	D	$K+1$	E	x
					F	$n-1$

будут протекать следующим образом: вычисляется a_1 и $s_1 = s_0 + a_1$, сравнивается содержимое ячейки D с $n-1$. Если $n > 1$, то вычисляется a_2 и $s_2 = a_2 + s_1$, изменяется

содержимое ячейки D (оно становится равным 2) и сравнивается содержимое D с n . Так продолжается процесс до вычисления n -го члена. После вычисления a_n и $s_n = a_n + s_{n-1}$ содержимое ячейки D будет n и после сравнения по команде $K+5$ мы закончим эту программу (перейдем к команде $K+6$), а не вернемся к команде $K+1$.

В этой задаче мы встретились с программированием вычислительного процесса, распадающегося на этапы, вычисляемые по одинаковым программам. Разные этапы отличаются лишь исходными данными (содержащимися в ячейках A , C , D), причем в каждый этап вычислений входит изменение этих данных и тем самым подготовка к выполнению следующего этапа.

Замечание. Для того чтобы сделать окончательную программу более короткой, мы несколько усложнили вычисления (ввели „вычисление“ $a_1 = x$). Такое усложнение вычислений для сокращения программы встречается часто (см. следующий пример).

Пример 2. Пусть нужно найти сумму произведений чисел a_1 и b_1 , a_2 и b_2, \dots, a_n и b_n , находящихся в ячейках ЗУ с номерами соответственно $A_1, B_1, A_2, B_2, A_3, B_3, \dots, A_n, B_n$. Как и в предыдущем примере, можно составить прямую программу (программу всех операций). Она будет иметь вид:

$K+1$	\times	A_1	B_1	C'
$K+2$	$+$	C	C'	C
$K+3$	\times	A_2	B_2	C'
$K+4$	$+$	C	C'	C
$K+5$	\times	A_3	B_3	C'
$K+6$	$+$	C	C'	C
\dots	\dots	\dots	\dots	\dots
$K+2n-1$	\times	A_n	B_n	C'
$K+2n$	$+$	C	C'	C

C — номер ячейки ЗУ, в которой должен быть помещен результат.

Вначале $(C) = 0$

(Первые две команды можно заменить, конечно, одной:

\times	A_1	B_1	C
----------	-------	-------	-----

, но мы не хотим нарушать единообразие программы.)

Приведенная программа состоит из повторения одной и той же совокупности двух операций с различными значениями первого и второго адресов в первой из них. Поэтому целесообразно осуществить управление машиной при решении этой задачи следующим образом.

Задать сначала лишь команды $k_1 = \begin{bmatrix} \times & A_1 & B_1 & C \end{bmatrix}$, $k_2 = \begin{bmatrix} + & C & C' & C \end{bmatrix}$, в дальнейшем менять первую команду так, чтобы последовательно получались и осуществлялись команды $k_3 = (K+3), \dots, k_{2n} = (K+2n)$. Для этого нужно, чтобы команды хранились в переменных ячейках ЗУ. Так как над командами могут проводиться операции, как и над числами (см. § 22), то для изменения команды k_1 нужно в программу включить операции, меняющие эту команду.

Поместим числа a_1, \dots, a_n в ячейках $A+1, A+2, \dots, A+n$ и b_1, \dots, b_n в ячейках $B+1, B+2, \dots, B+n$ с номерами, идущими подряд (или вообще с каким-либо постоянным шагом: $A_i = A + is$, $B_i = B + it$, $i = 1, \dots, n$). Тогда цифровой набор k_{2i+1} может быть получен прибавлением (без учета порядков) к k_{2i-1} числа, имеющего запись

$$\begin{array}{c} \begin{bmatrix} & 1 & 1 & \end{bmatrix} \text{ (соответственно } \begin{bmatrix} & s & t & \end{bmatrix}) \\ k_{2i-1} = \begin{bmatrix} \times & A_i & B_i & C' \end{bmatrix} \\ + \\ (u) = \begin{bmatrix} & 1 & 1 & \end{bmatrix} \quad u - \text{ номер ячейки, где хра-} \\ \text{нится число } \begin{bmatrix} & 1 & 1 & \end{bmatrix} \\ \hline k_{2i+1} = \begin{bmatrix} \times & A_{i+1} & B_{i+1} & C' \end{bmatrix} \end{array}$$

k_{2i+1} нужно поместить в ту же ячейку (ячейка $K+1$), где было k_{2i-1} .

Операция изменения k_{2i-1} может быть закодирована следующим образом

$$\begin{bmatrix} + & K+1 & u & K+1 \end{bmatrix}$$

Для того чтобы обеспечить выполнение команд k_{2i-1} , k_{2i} и переход к k_{2i+1} , достаточно задать программу:

$K+1$	\times	$A+1$	$B+1$	C'
$K+2$	$+$	C	C'	C
$K+3$	$\dot{+}$	$K+1$	u	$K+1$
Перейти к $(K+1)$, если число со- считанных произведений меньше n , и окончить вычисления, если оно равно n				

Установить, какой шаг мы делаем, проще всего по команде $(K+1)$; на n -м шагу она должна иметь вид:

$$(K+1) = \begin{array}{|c|c|c|c|} \hline \times & A+n & B+n & C' \\ \hline \end{array}$$

Если мы предположим, что с увеличением адресов команды $\begin{array}{|c|c|c|c|} \hline \times & A_i & B_i & C' \\ \hline \end{array}$ число, изображающее эту команду, увеличивается, то окончательно программе можно придать следующий вид:

$K+1$	\times	$A+1$	$B+1$	C'	v — номер ячейки, где хранится контрольное число $\begin{array}{ c c c c } \hline \times & A+n & B+n & C' \\ \hline \end{array}$ или $\begin{array}{ c c c c } \hline \times & A+n+1 & & \\ \hline \end{array}$
$K+2$	$+$	C	C'	C	
$K+3$	$\dot{+}$	$K+1$	u	$K+1$	
$K+4$	\geq	v	$K+1$	$K+1$	

Процесс вычисления по этой программе состоит из повторения серии операций

$K+1$	\times	$A+m$	$B+m$	C'
$K+2$	$+$	C	C'	C
$K+3$	$\dot{+}$	$K+1$	u	$K+1$

и возвращения к команде $(K+1)$, пока эта команда не примет вид

\times	$A+n$	$B+n$	C'
----------	-------	-------	------

, после чего вычисления по этой программе заканчиваются (переходим к команде $(K+5)$).

Для того чтобы в нужный момент окончить процесс вычислений, можно наряду с только что описанным способом применить также следующий: в какой-либо ячейке ЗУ r хранить число проделанных циклов и сравнивать его с контрольным (ячейка w). Понятно, что изменение этого числа (прибавление к нему 1) должно войти в самый цикл. В этом случае программа будет иметь вид:

$K+1$	\times	$A+1$	$B+1$	C'	$(A+i) = a_i$	$(C) = 0$				
$K+2$	$+$	C	C'	C	$(B+i) = b_i$	$(q) = 1$				
$K+3$	$+$	r	q	r	$(w) = n-1$	$(r) = 0$				
$K+4$	$+$	$K+1$	u	$K+1$	$(u) = $ <table><tr><td></td><td>1</td><td>1</td><td></td></tr></table>			1	1	
	1	1								
$K+5$	\geq	w	r	$K+1$						

Эта программа, как видно, длиннее и требует больше вспомогательных ячеек. Поэтому, если возможно, лучше употреблять первый из указанных способов.

Пример 3. В § 4 уже рассматривалось решение уравнения $x = f(x)$ методом итерации $x_{n+1} = f(x_n)$; там была приведена логическая схема решения этой задачи. Приведем полную программу ее решения. Если мы не будем детализировать вычисление $f(x)$ в программе и обозначим его условно одной командой

f	α		β
-----	----------	--	---------

, что означает: найти значение $f((\alpha))$ и отправить его в β , то программа вычислений может иметь следующий вид:

$K+1$	f	A		C	$(A) = x_0$
$K+2$	$-$	C	A	A	$(B) = \varepsilon$
$K+3$	$ $	A		D	$\left. \begin{matrix} (C) \\ (D) \end{matrix} \right\} \text{ что угодно}$
$K+4$	$+$	C		A	
$K+5$	\geq	D	B	$K+1$	

Если имеется операция сравнения по абсолютной величине, то программа упростится:

$K+1$	f	A		C	$(A) = x_0$
$K+2$	$-$	C	A	D	$(B) = \varepsilon$
$K+3$	$+$	C		A	(C)
$K+4$	$ \geq $	D	B	$K+1$	(D) } что угодно

Все рассмотренные задачи имели одну и ту же логическую схему: *решение их состояло в периодическом повторении основной серии операций.*

Такую серию операций мы будем в дальнейшем называть циклом, а процессы, состоящие в повторении цикла, — циклическими.

Для задач, процесс решения которых циклический, можно указать следующий общий рецепт программирования. Пусть делается последовательность операции (или совокупности операций) A_1, A_2, \dots, A_n и при некотором расположении исходных данных в ЗУ A_{i+1} может быть единообразным способом получено из A_i . Тогда целесообразно составить программу по такому плану:

- I A_1
- II Изменение A_i в A_{i+1} ($A_i \rightarrow A_{i+1}$)
- III Контрольное сравнение

Пример 4. Пусть нужно умножить матрицу на вектор, т. е. найти $y_i = \sum_{j=1}^n a_{ij} x_j$ ($i = 1, 2, \dots, n$). Эта задача естественным образом распадается на циклы — процессы нахождения отдельных величин y_i . Если рассматривать нахождение y_i как цикл A_i , то по указанному рецепту программа будет иметь вид:

- I A_1
- II $A_i \rightarrow A_{i+1}$
- III Контрольное сравнение

Но задача нахождения отдельных y_i уже была решена в примере 2 и для нее уже составлена программа.

$K+1$	\times	$p+s$	q_i+t_i	C'
$K+2$	$+$	C_i	C'	C_i
$K+3$	$\dot{+}$	$K+1$	u_i	$K+1$
$K+4$	\geq	v	$K+1$	$K+1$

$$(p+is) = x_i$$

$$(q_i+jt_i) = a_{ij}$$

$$(u_i) = \begin{array}{|c|c|c|c|} \hline & s & t_i & \\ \hline \end{array}$$

$$(v) = \begin{array}{|c|c|c|c|} \hline \times & (n+1)s & & \\ \hline \end{array}$$

$i, j = 1, 2, \dots, n$

Очевидно, удобно число $t = t_i$ выбрать неизменным. Тогда программа вычисления примет вид:

$K+1$	\times	$p+s$	q_i+t	C'
$K+2$	$+$	C_i	C'	C_i
$K+3$	$\dot{+}$	$K+1$	u	$K+1$
$K+4$	\geq	v	$K+1$	$K+1$

При $i = 1$ это этап I программы.

Для осуществления этапа II удобно, очевидно, считать

$$q_i = q + iz, \quad C_i = C + ir$$

Тогда этап II будет иметь вид:

$K+5$	$\dot{+}$	$K+1$	w_1	$K+1$	w_1		$-ns$	$-nt+z$	
$K+6$	$\dot{+}$	$K+2$	w_2	$K+2$	w_2		r		r

Контрольное сравнение (этап III) можно провести по приказу $(K+1)$ или $(K+2)$.

Возьмем, например, по $(K+2)$. Сравнение будет иметь вид:

$K+7$	\geq	w_3	$K+2$	$K+1$	w_3	$+$	$(n+1)r+C$		
-------	--------	-------	-------	-------	-------	-----	------------	--	--

Окончательно получим программу:

$K+1$	\times	$p+s$	$q+z+t$	C'	$(p+is) = x_i, (q+iz+jt) = a_{ij}$
$K+2$	$+$	$C+r$	C'	$C+r$	$(C+ir) = 0, i, j = 1, 2, \dots, n$
$K+3$	$\dot{+}$	$K+1$	u	$K+1$	
$K+4$	\geq	v	$K+1$	$K+1$	u
$K+5$	$\dot{+}$	$K+1$	w_1	$K+1$	v
$K+6$	$\dot{+}$	$K+2$	w_2	$K+2$	w_1
$K+7$	\geq	w_3	$K+2$	$K+1$	w_2

	\times	s	t	
		$(n+1)s$		
		$-ns$	$z-nt$	
		r		r
	$+$	$C+(n+1)r$		

после окончания вычисления

$$(C+ir) = y_i, \quad (i = 1, \dots, n)$$

Величины p, q, r, s, t, z должны быть, очевидно, такими, чтобы серии $p+is, q+iz+jt, C+ir$ попарно не пересекались. Если нет никаких дополнительных условий, то можно, например, принять $q=0; t=n; z=1; p=n^2; s=1; C=n^2+n; r=1$.

Здесь мы имеем систему циклов: задача была разбита на циклы — процессы вычисления отдельных y_i , каждый из этих циклов, в свою очередь, разбивается на циклы — вычисление отдельных произведений $a_{ij}x_j$ и прибавление $a_{ij}x_j$ к $\sum_{s=1}^{j-1} a_{is}x_s$. Такая система разбиения на циклы, каждый из которых в свою очередь разбивается на циклы, может содержать и более длинную цепочку.

Если, например, нужно вычислить произведение матриц $C=AB$, то это вычисление разбивается на циклы — вычисления отдельных столбцов матрицы C , что приводит к только что рассмотренной задаче.

Замечание. Мы разобрали следующую схему расположения этапов выполнения элементарного цикла:

- I A_1
- II $A_i \rightarrow A_{i+1}$
- III Контрольное сравнение

Можно рассмотреть и другой порядок:

I $A_i \rightarrow A_{i+1}$

II A_0

III Контрольное сравнение

В этом случае в начальный момент значение рабочих команд и вспомогательных ячеек должно быть таким, чтобы при замене A_i на A_{i+1} оно было готово к выполнению первого цикла A_1 . Этап II поэтому и обозначен A_0 .

При составлении программы для задач, разбивающихся на отдельные циклы и этапы, сам процесс программирования соответственно разбивается на составление соответствующих узлов программы. При этом совершенно не нужно составлять всю программу сразу. Достаточно составить независимые (или почти независимые) программы для разных ее этапов и затем смонтировать программу из этих этапов.

Мы видели, что для того, чтобы сделать переход от одного цикла к другому единообразным, понадобилось расположить исходные данные в определенном порядке. Часто удастся (как в примерах 2 и 4) свести задачу к циклам, которые отличаются друг от друга лишь исходными данными. Особенно удобно, если удастся эти данные расположить в ячейках, номера которых идут подряд (или вообще с каким-нибудь постоянным шагом), и для получения команд следующего цикла нужно изменять некоторые фиксированные из команд предыдущего на постоянные (независящие от номера цикла) числа. Именно этот случай и является наиболее часто встречающимся. Поэтому вопрос размещения исходных данных в ЗУ является важным вопросом в каждой задаче.

З а м е ч а н и е. В примерах 1 и 3 циклические процессы осуществлялись с помощью программ, не требовавших преобразования команд; такие программы реализуемы и в условиях хранения команд в постоянном ЗУ, программы же к примерам 2 и 4 включали преобразование команд, поэтому эти программы должны храниться в оперативном (переменном) ЗУ.

§ 25. СОСТАВЛЕНИЕ СЛОЖНЫХ ПРОГРАММ ИЗ БОЛЕЕ ПРОСТЫХ

Программы и подпрограммы. В процессе решения сложных задач приходится решать более элементарные часто встречающиеся стандартные задачи: перевод чисел из одной системы в другую, деление и извлечение квадратного корня в машинах, в которых эти операции не входят в число основных, нахождение значений элементарных функций — тригонометрических, логарифмических, показательной, интерполирование, арифметические операции с удвоенным числом знаков и т. п.; целесообразно иметь набор заранее составленных программ для таких задач. Эти программы будут входить как подпрограммы в программы решений других задач.

Существуют машины с единым блоком управления и машины, в которых имеется, наряду с главным блоком управления, блок (или блоки) местного управления. В машинах первого типа подпрограммы являются частями общей программы.

Рассмотрим сначала случай таких машин при последовательном выполнении команд: пусть после команды (α) следует перейти к подпрограмме. Можно первую команду подпрограммы сделать командой $(\alpha + 1)$ основной программы, тогда, в зависимости от α , изменяются те команды подпрограммы, в которых содержатся адреса команд подпрограммы — команды условного и безусловного перехода и преобразования команды (k -я команда подпрограммы будет иметь в общей программе адрес $\alpha + k$ и это число $\alpha + k$ будет, например, стоять в команде, означающей переход к выполнению этой команды). Такое включение требует предварительного преобразования команд подпрограммы. Этих преобразований можно избежать, если команды данной подпрограммы размещаются всегда в одних и тех же фиксированных ячейках ЗУ. Пусть подпрограмма занимает ячейки $\beta, \beta + 1, \dots, \gamma$. Если после команды (α) основной программы нужно приступить к выполнению подпрограммы, то можно поступить по следующей, например, схеме: $(\alpha + 2)$ будет команда безусловного перехода к выполнению команды (β) , а $(\gamma + 1)$ — команда безусловного перехода к очередной

команде основной программы. Перед командой $(\alpha + 2)$, очевидно, должна быть команда $\alpha + 1$, меняющая $(\gamma + 1)$ нужным образом.

Для машин с свободным выполнением команд включение подпрограммы в программу значительно упрощается. Для включения подпрограммы, содержащейся в ячейках $\beta, \beta + 1, \dots, \gamma$, в основную программу после команды (α) достаточно указать в четвертом адресе команды (α) ячейку β , а в четвертом адресе команды γ ячейку, содержащую ту команду основной программы, к которой следует перейти после выполнения подпрограммы. Если подпрограмма размещена в стандартных ячейках ЗУ, то для такого включения требуется лишь одна команда в основной программе: команда, которая обеспечивает как необходимое изменение 4-го адреса команды γ , так и переход к подпрограмме.

В машинах с блоком местного управления этот блок управляет выполнением включенных в него подпрограмм. Основная программа осуществляется центральным блоком управления. Особая команда реализует *передачу управления с центрального управления на местное* для выполнения подпрограммы.

Очевидно, эта команда должна заключать или номер (шифр) подпрограммы или номер команды, с которого начинается выполнение этой подпрограммы. Подпрограмма должна заканчиваться особой командой *возвращения к центральному управлению*. Выполнение основной программы начинается после этого с прерванного места.

Для машин с свободным выполнением команд при переходе к местному управлению должны быть заданы два адреса: первый адрес нужной подпрограммы и адрес команды основной программы, к которой следует перейти, выполнив подпрограмму.

Очевидно, все команды подпрограммы, подвергавшиеся изменению в процессе ее выполнения, должны быть реставрированы перед вторичным использованием этой подпрограммы (см. § 21). Подпрограмма либо заканчивается, либо начинается такой реставрацией.

При местном управлении наличие подпрограммы по существу расширяет набор элементарных операций. Отсылка к местной подпрограмме эквивалентна команде о выполнении операций, реализуемых этой подпрограммой. При достаточно широком наборе подпрограмм задача программирования в значительной мере сводится к монтажу этих подпрограмм.

Каждая из подпрограмм предусматривает использование при вычислениях некоторых ячеек ЗУ, которые могут оказаться занятыми результатами вычислений по основной программе. Это следует иметь в виду при включении подпрограммы в программу. В некоторых случаях может оказаться целесообразным включение в подпрограмму операций очистки тех ячеек ЗУ, которые потребуются при вычислениях.

Схема составления программ. Обычно большая вычислительная задача разбивается на ряд этапов, каждый из которых представляет собой небольшую самостоятельную задачу.

Если мы умеем составлять программы для каждого из этих этапов, то задача составления всей программы сводится к соединению программ для этих этапов.

Рассмотрим простейший пример. Пусть $\sin x$ вычисляется следующим образом: с помощью формул приведения приходим к вычислению $\sin x$, $0 \leq x \leq \frac{\pi}{2}$; при $0 \leq x \leq \frac{\pi}{4}$ вычисляем

$\sin x = x - \frac{x^3}{3!} + \dots$; при $\frac{\pi}{4} < x \leq \frac{\pi}{2}$ вычисляем

$$\cos\left(\frac{\pi}{2} - x\right) = 1 - \frac{\left(\frac{\pi}{2} - x\right)^2}{2!} + \dots$$

Эту задачу можно разбить на следующие этапы:

1. Вычислить $x - E\left(\frac{x}{2\pi}\right)2\pi$ и заменить x на эту величину.
2. Сравнить $x > \pi$ или $x \leq \pi$.
3. Запомнить результат сравнения в особой ячейке ζ посылкой в нее -1 , если $y > \pi$, и посылкой в нее $+1$, если $y \leq \pi$; если известно вначале, что $(\zeta) = 1$, то второй вариант этапа 3 не нужен. Это мы и будем предполагать.

4. При $x > \pi$ вычислить $x - \pi$ и заменить x на эту величину.

5. Сравнить $x > \frac{\pi}{2}$ или $x \leq \frac{\pi}{2}$.

6. При $x > \frac{\pi}{2}$ вычислить $\pi - x$ и заменить x на эту величину.

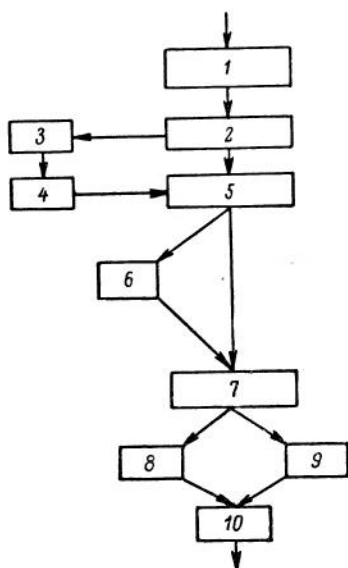
7. Сравнить $x > \frac{\pi}{4}$ или $x \leq \frac{\pi}{4}$.

8. При $x > \frac{\pi}{4}$ вычислить $\cos\left(x - \frac{\pi}{4}\right)$.

9. При $x \leq \frac{\pi}{4}$ вычислить $\sin x$.

10. Величину, полученную в результате этапов 8 или 9, умножить на содержимое ячейки ζ . Это и есть ответ.

Если нарисовать схему, то получится следующая картина:

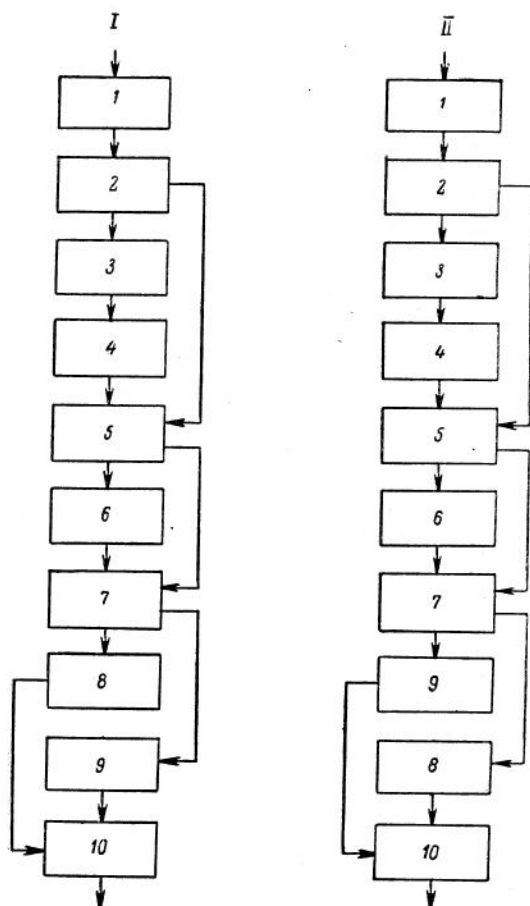


Такое объединение этапов в схему с указанием связи между этапами (порядок перехода от одного этапа к другому) мы будем называть блок-схемой для данной задачи. Составление блок-схемы упрощает процесс программирования; нужно составить программы для отдельных этапов (блоков схемы) и соединить их согласно блок-схеме. Это соединение сводится к взаимному расположению блоков в общей программе и расстановке между ними в случае надобности команд условного или безусловного переходов. Принципиально располагать блоки в программе можно

в любом порядке; удобнее, конечно, число дополнительных, связывающих команд иметь возможно меньшим. Если какой-либо блок кончается командой условного перехода, удобно

один из блоков, к которому может перейти процесс, поместить под этим блоком; если по окончании какого-либо блока необходимо безусловно перейти к другому, удобно второй поместить непосредственно под первым. Для машины со свободным порядком выполнения команд соединение блоков производится, очевидно, совсем просто, так как в этом случае порядок расположения не играет никакой роли.

В нашем примере можно расположить блоки следующим образом:



В первом случае после 8-го этапа, а во втором — после 9-го необходимо, все-таки, добавить команду безусловного перехода. Других связующих команд нигде не требуется. В главе V читатель найдет примеры блок-схем.

В случае, если один из этапов является стандартной задачей, для которого имеется стандартная подпрограмма, присоединение этого этапа происходит так, как это указано в начале параграфа.

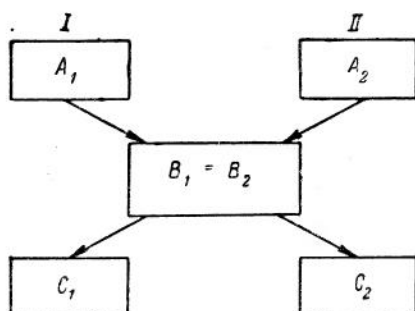
Преобразование этапов программы. Пусть два этапа вычислений A и B совершаются последовательно. Проще всего составить программу для выполнения A и программу для выполнения B , вторую программу взять продолжением первой. Однако, если этапы A и B имеют много общего, то целесообразно часто программу последовательности A, B построить так: программа A , программа, переводящая программу A в программу B .

Так, решение дифференциального уравнения методом Адамса — Штермера состоит из двух больших этапов: нахождение решения в нескольких начальных (исходных) точках, продолжение процесса решения — нахождение решения в последующих точках. Даже без детального составления программ видно, что эти этапы близки друг к другу. Программа § 32 составлена как раз таким образом: программа вычисления решения в первых точках, небольшое добавление, преобразующее эту программу к виду, при котором по ней вычисляется решение в последующих точках. При этом первая половина состоит из 73 команд, вторая — всего из 26.

Объединение программ или этапов. Когда приходится решать несколько задач, имеющих общие этапы или легко преобразуемые один из другого, можно составить объединенную более экономную программу для всех этих задач.

Рассмотрим простейший случай такого объединения: мы имеем программы I и II для решения двух задач; программа I состоит из последовательно выполняемых этапов A_1, B_1, C_1 ; программа II — из этапов A_2, B_2, C_2 , причем

$B_1 = B_2$. Можно составить объединенную программу по следующей схеме:



Переход от этапов A_1 или A_2 к этапу $B_1 = B_2$ совершается в общем так, как было указано выше. В машинах последовательного выполнения команд, если, например, в программе этап $B_1 = B_2$ не стоит непосредственно за A_1 то, этап A_1 должен заканчиваться командой безусловного перехода (пересылающей к первой команде этапа B_1). Реализация разветвления после окончания этапа $B_1 = B_2$ может быть осуществлена, например, следующим образом: программам I и II мы относим шифры — числа, соответственно, α_1 и $\alpha_2 \neq \alpha_1$, посылаемые в начале процесса вычисления во вспомогательную ячейку γ . Этап $B_1 = B_2$ заканчивается командой условного перехода, который в зависимости от содержимого ячейки γ заставляет машину переходить к выполнению этапа C_1 или C_2 .

Схема соединения программ может усложняться, в зависимости от числа объединяемых программ, числа общих их этапов и т. д.

Если разные этапы одной и той же программы имеют общие части, то можно также их объединять; например, в приведенной выше схеме вычисления значений синуса могут быть объединены этапы (8) и (9) — вычисление степенных рядов для синуса и косинуса. В § 28 приведена программа вычисления значений синуса, где объединены эти этапы.

Заключительные замечания. В последних трех параграфах мы указали на приемы, которыми мы заставляем машину решать математические задачи, требующие большого числа вычислений, с помощью не слишком длинных программ.

Во многих программах лишь меньшая часть команд относится к выполнению арифметических и иных операций над числами, в то время как большинство команд выполняют функции управления машиной — обеспечивают циклическое повторение некоторых серий операций, последовательное выполнение этапов вычислений и т. п.

В упоминавшейся программе интегрирования баллистического уравнения из 99 команд только 26 команд относятся к выполнению операций над числами, остальные управляют процессом вычислений. В программе решения систем линейных алгебраических уравнений из 63 команд лишь 11 команд относятся к вычислению над числами; 12 команд относятся к выбору „главного элемента“ (наибольшего элемента матрицы), 4 — к упорядочению строк и столбцов, остальные управляют общим ходом процесса вычислений.

§ 26. ПРОГРАММИРОВАНИЕ ДЛЯ СЛУЧАЯ УМЕНЬШЕННЫХ ВОЗМОЖНОСТЕЙ МАШИНЫ

Универсальная вычислительная машина, предназначенная для решения самых различных задач, должна, естественно, быть значительно более сложной, чем специализированная машина, способная решать какой-либо определенный узкий класс задач.

Стремление упростить универсальную машину за счет уменьшения диапазона чисел, с которыми оперирует машина, уменьшения объема ЗУ или же за счет отказа от некоторых операций, в частности операций над кодами команд, приводит к созданию машин, хотя и способных решать разнообразные задачи, но все же не являющихся вполне универсальными. Задача программирования для такой машины с ограниченными возможностями становится значительно более трудной. При этом не всегда сразу удается опреде-

лечь границы применимости машины и ответить на вопрос о разрешимости данной задачи на такой машине.

Приводимая ниже программа численного интегрирования уравнений внешней баллистики рассчитана на машину, у которой имеется лишь 20 оперативных ячеек ЗУ и 30 ячеек ЗУ, в которых можно хранить постоянные числа. Кроме того, машина не способна преобразовывать команды и число команд в программе, включая команду остановки, не должно превышать 63. Машина может оперировать только с числами в диапазоне $(-1,1)$; число двоичных разрядов равно 16. В машине предусмотрено печатание полученных результатов, которое осуществляется командой отправки результата в специальную ячейку ЗУ. Объем ЗУ машины, ограничение числа команд числом 63 и отсутствие операций преобразования команд не позволили решить задачу интегрирования движения центра массы снаряда полностью.

Как известно, правые части системы дифференциальных уравнений движения центра массы снаряда содержат три эмпирические функции: $\tau(y)$ — относительное изменение температуры с высотой, $H_\tau(y)$ — относительное изменение плотности воздуха с высотой и $G(v)$ — функция сопротивления воздуха; $\tau(y)$ является линейной функцией высоты

$$\tau(y) = 1 - Ay$$

и ее вычисление не представляет трудностей. Для вычисления функции $H_\tau(y)$ до высоты в 9300 м здесь принята формула

$$H_\tau(y) = \frac{Ay^2 + B(Ay) + C}{D(Ay) - C}$$

Вычисление $H_\tau(y)$ для больших высот потребовало бы дополнительных команд в программе и число команд программы вышло бы за установленный предел. Для вычислений функции сопротивления воздуха, последняя была аппроксимирована на различных интервалах изменения аргумента полиномами третьей степени. Для того чтобы коэффициенты

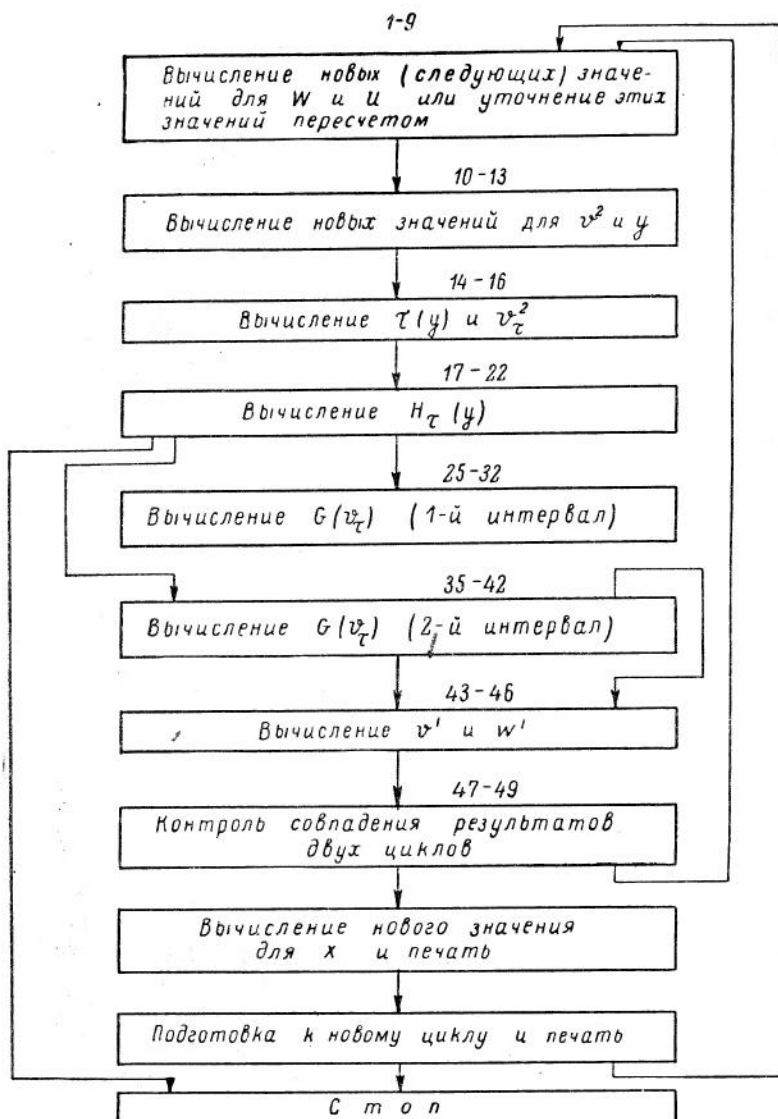
полиномов не превосходили единицы, фактически приходится иметь дело с полиномами третьей степени относительно линейной функции квадрата скорости. Эта линейная функция оказывается различной на разных интервалах аппроксимации функции сопротивления. Ограниченные возможности машины не позволили ввести в программу вычисление функции сопротивления сразу для всех значений скорости. Данная программа предусматривает вычисление функции сопротивления для двух соседних интервалов аппроксимации.

Таким образом, предлагаемая программа рассчитана на вычисление траекторий, высота которых не превосходит 9300 м, а приведенная скорость (точнее, квадрат скорости) движения снаряда должна меняться в пределах, определяемых содержанием 211 и 219 ячеек ЗУ. В том случае, когда скорость выходит за установленные пределы, программа обеспечивает автоматическое прекращение вычислений. Для продолжения вычислений в этом случае требуется изменить содержание ячеек постоянного ЗУ 211 — 225. Интегрирование ведется методом Эйлера с пересчетом по формуле трапеции. Для того чтобы обеспечить необходимую точность расчетов и не выходить за пределы установленного диапазона чисел, для искомых функций введены различные масштабные коэффициенты, равные степеням двух.

При интегрировании дифференциальных уравнений методом Эйлера, естественно, возникает необходимость в уменьшении шага интегрирования, что ведет к увеличению числа операций, необходимых для проведения всего процесса. Здесь возникает опасность быстрого роста ошибок за счет ошибок округления и ошибок метода. Благодаря устойчивому характеру решения системы дифференциальных уравнений баллистики такого роста в действительности не наблюдается. Экспериментальные расчеты показывают, что потери точности за счет округлений минимальны и для получения четырех верных десятичных знаков достаточно вести вычисления с одним запасным десятичным знаком.

Эти результаты и приведенная ниже программа интегрирования уравнений баллистики показывают, что машина, предназначенная специально для интегрирования системы

Блок-схема программы



дифференциальных уравнений, может обладать значительно более скромным оборудованием, чем универсальная вычислительная машина.

В приведенной ниже программе ячейки ЗУ, у которых первая цифра номера есть 2, представляют собой ячейки постоянного ЗУ. Остальные ячейки ЗУ (первая цифра номера 1) — оперативные ячейки, из которых можно брать числа и в которые можно отправлять результаты промежуточных вычислений. Команды имеют свою нумерацию от 1 до 63.

Программа для численного решения уравнений баллистики.

Содержание ячеек ЗУ

Оперативные ячейки ЗУ			Ячейки постоянного ЗУ		
101	w_0	201	ε	(211) — (225) — константы для вычисления функции со- противления воздуха	
102	u_0	202	h		
103	$y_0 = 0$	203	$\frac{h}{2}$		
104	$x_0 = 0$	204	$2^{-3}h$		
105	$\frac{1}{2} w'_0$	205	g	коэффициенты для вычисления $H_\tau(y)$	
106	$\frac{1}{2} w'_0$	206	$1-2^{-16}$		
107	$\frac{1}{2} u'_0$	207	A		
108	$\frac{1}{2} u'_0$	208	B		
110	$1-2^{-16}$	209	D		
		210	C		

Команды

1	+	105	106	113	9	×	109	109	106
2	+	107	108	117	10	×	114	114	108
3	×	113	202	109	11	+	106	108	113
4	×	117	202	111	12	×	204	112	112
5	×	113	203	112	13	+	103	112	112
6	+	101	112	112	14	×	112	207	118
7	+	101	109	109	15	—	206	118	116
8	+	102	111	114	16	:	113	116	116

17	—	118	208	108	41	×	116	106	106
18	×	108	118	108	42	+	106	224	106
19	+	108	210	108	43	×	108	106	108
20	×	209	118	106	44	×	109	108	106
21	—	106	210	106	45	×	114	108	108
22	:	108	106	108	46	—	106	205	106
23	≥	116	211	63	47	—	110	111	116
24	≥	212	116	34	48	+	111		110
25	—	116	218	116	49	≥	116	201	1
26	×	116	213	116	50	П	113		
27	×	116	214	106	51	×	117	203	110
28	+	106	215	106	52	+	102	110	110
29	×	106	116	106	53	×	110	204	111
30	+	106	216	106	54	+	111	104	104
31	×	106	116	106	55	П	104		
32	+	106	217	106	56	+	109		101
33	≥			43	57	+	114		102
34	≥	219	116	63	58	+	112		103
35	—	116	225	116	59	+	106		105
36	×	116	220	116	60	+	108		107
37	×	116	221	106	61	П	103		
38	+	106	222	106	62	≥	103		1
39	×	116	106	106	63				
40	+	106	223	106					

§ 27. ПРОГРАММЫ К ЗАДАЧАМ, РАССМОТРЕННЫМ В ГЛАВЕ III

В главе III рассмотрены стандартные задачи и стандартные приемы решения задач. Ниже приведены программы для некоторых рассмотренных там задач: вычисление многочленов, образование последовательных сумм и разностей, решение уравнений методом проб, преобразование чисел из двоичной системы в десятичную, пример программы вычисления с контролем.

Вычисление многочленов. О схемах вычисления многочленов смотри § 16.

1. В качестве 1-го примера приведем программу вычисления $\ln(1+x)$ на интервале $(1,2)$ с точностью до 10^{-6} по формуле ¹

$$\ln(1+x) = 0,9999925x - 0,4996101x^2 + 0,3287936x^3 - 0,2267714x^4 + 0,1350726x^5 - 0,0548851x^6 + 0,0105550x^7$$

Программу составим по схеме Горнера.

$$\ln(1+x) = ((((((0,0105550x - 0,0548851)x + 0,1350726)x - 0,2267714)x + 0,3287936)x - 0,4996101)x + 0,9999925)x$$

A + 1	+ 0,9999925
A + 2	- 0,4996101
A + 3	+ 0,3287936
A + 4	- 0,2267714
A + 5	+ 0,1350726
A + 6	- 0,0548851
A + 7	+ 0,0105550

K + 1	+	C ₃	A + 7	C ₃	C ₁	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> $+ \left C_3 \right \frac{x}{A+1} \left C_3 \right$ </div>
K + 2	×	C ₁	C ₃	C ₃	C ₂	
K + 3	÷	K + 1	C ₂	K + 1	K ₁	
K + 4	>	K + 1	K ₁	K + 1	C ₃	
K + 5						

З а м е ч а н и е. Если значение $\ln(1+x)$ нужно находить в ряде точек, то для восстановления первоначального вида программы нужно предварительно выполнить две команды

K - 1	+			C ₃	K ₂ =	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> $+ \left C_3 \right \frac{x}{A+1} \left C_3 \right$ </div>
K	+	K ₂		K + 1		

¹ Эта формула получена разложением функции $\ln(1+x)$ по полиномам Чебышева. См. С. Lanczos. Trigonometrical interpolation journal of mathematics and physics 1938 г.

2. В качестве второго примера рассмотрим интерполяцию по формуле Эверетта.

Эта формула имеет вид

$$f(x) = \sum_{i=0}^{k-1} \left\{ (1-y)^{[2i+2]-1} \frac{\delta^{2i} f(x_0)}{(2i+1)!} + y^{[2i+2]-1} \frac{\delta^{2i} f(x_0+h)}{(2i+1)!} \right\} + R(x) \quad (1)$$

где h — расстояние между соседними узлами интерполяции x_0, x_1, \dots, x_n ,

$$y = \frac{x-x_0}{h}; \quad y^{[2i+2]-1} = y(y^2-1)(y^2-2^2) \dots (y^2-i^2)$$

$\delta^{2i} f(x_0)$ — центральная разность порядка $2i$

$$\delta^{2i} f(x_0) = \Delta^{2i} f(x_0 - ih); \quad \Delta^S f(x_0) = \Delta^{S-1} f(x_0+h) - \Delta^{S-1} f(x_0)$$

$$R(x) = (y-k)y(y^2-1)(y^2-2^2) \dots [y^2-(k-1)^2] \frac{h^{2k}}{2k!} f^{(2k)}(\xi)$$

ξ — промежуточная точка интервала $(x_0 - kh, x_0 + kh)$.

Ограничимся случаем $k=4$. Для $k>4$ все выкладки проводятся аналогично.

После простых преобразований формулу Эверетта для $k=4$ можно записать в виде:

$$\begin{aligned} f(x) = & \{ (1-y)[f(x_0) + (y^2-2y)A(x_0)] + y[f(x_0+h) + \\ & + (y^2-1)A(x_0+h)] \} + \{ (1-y)(y^2-2y)^2 B(x_0) + \\ & + y(y^2-1)^2 B(x_0+h) \} + \left\{ (1-y)(y^2-2y)^3 \frac{\delta^6 f(x_0)}{7!} + \right. \\ & \left. + y(y^2-1)^3 \frac{\delta^6 f(x_0+h)}{7!} \right\} + R(x) \end{aligned} \quad (2)$$

здесь

$$A(x_0) = \frac{\delta^2 f(x_0)}{3!} - \frac{3}{5!} \delta^4 f(x_0) + \frac{24}{7!} \delta^6 f(x_0)$$

$$B(x_0) = \frac{\delta^4 f(x_0)}{5!} - \frac{11}{7!} \delta^6 f(x_0)$$

$R(x)$ — остаточный член, который на интервале $0 \leq y \leq 1$ не превосходит $2^{-9} h^8 M_8$, где M_8 — максимальное по абсолютной величине значение производной восьмого порядка $f^{(8)}(x)$ на интервале $(x_0 - 4h, x_0 + 4h)$. Пренебрегая величиной, стоящей

в последней фигурной скобке формулы (2), получим $f(x)$ с точностью не ниже, чем

$$2^{-9} h^6 \left(\frac{M_6}{8} + \frac{h}{16} M_7 + h^2 M_8 \right)$$

где M_6, M_7, M_8 — максимальные по абсолютной величине значения производных $f^{(6)}(x), f^{(7)}(x), f^{(8)}(x)$ на интервале $(x_0 - 4h, x_0 + 4h)$.

Если же последнюю фигурную скобку формулы (2) заменить многочленом более низкой степени, полагая при $-1 \leq y \leq 1$

$$\begin{aligned} (y^2 - 2y)^3 &\approx -0,75(y^2 - 2y)^2 \\ (y^2 - 1)^3 &\approx -0,75(y^2 - 1)^2 \end{aligned} \quad (3)$$

то получим $f(x)$ с точностью не ниже

$$2^{-9} h^6 \left(\frac{M_6}{1024} + \frac{h M_7}{64} + h^2 M_8 \right)$$

Программу вычисления $f(x)$ на интервале (a, b) по формуле (2) с заменой последней фигурной скобки по формуле (3) будем вести по схеме Горнера, для чего запишем „искаженную“ формулу Эверетта в виде:

$$\begin{aligned} f(x) = & y \{ f(x_0 + h) + (y^2 - 1) [A(x_0 + h) + \\ & + (y^2 - 1) B^*(x_0 + h)] \} + (1 - y) \{ f(x_0) + \\ & + (y^2 - 2y) [A(x_0) + (y^2 - 2y) B^*(x_0)] \} + R(x) \end{aligned} \quad (4)$$

где x_0 — узел, ближайший к данному x .

$$B^*(x_0) = \frac{\delta^4 f(x_0)}{5!} - \frac{11,75}{7!} \delta^6 f(x_0)$$

Пусть $\frac{b-a}{h} = n$, и известные нам числа $f(a), A(a), B^*(a), f(a+h), A(a+h), B^*(a+h), \dots; f(a+nh), A(a+nh), B^*(a+nh)$ расположены последовательно в $3n+3$ ячейках ЗУ с номерами $A+1, A+2, A+3, \dots, A+1+3n, A+2+3n, A+3+3n$. Узел x_0 , ближайший слева к данному x , отыскиваем выделением целой части числа $\frac{x-a}{h}$. Номер

ячейки ЗУ, содержащий $f(x_0)$, равен $A+1+3E \frac{x-a}{h}$.

Вычисление $f(x)$ по формуле (4) распадается на два цикла, считая циклом вычисление фигурной скобки, причем при переходе ко 2-му циклу заменяем y на $1-y$. Вычисление фигурных скобок ведется по схеме Горнера.

Программа записывается следующим образом

$K+1$	—	C_5	C_3	C_5
$K+2$:	C_5	C_4	C_5
$K+3$	E	C_5	C_5	C_6
$K+4$	\times	C_5	C_7	C_5
$K+5$	+	C_2	C_5	C_5
$K+6$	\wedge	C_5	C_8	C_5
$K+7$	$\dot{+}$	K_2	C_5	$K+13$
$K+8$	+			C_{12}
$K+9$	+			C_{13}
$K+10$	\times	C_6	C_6	C_5
$K+11$	—	C_5	C_0	C_5
$K+12$	\times	C_{12}	C_5	C_{12}
$K+13$				
$K+14$	\div	$K+13$	C_9	$K+13$
$K+15$	+	C_{13}	C_0	C_{13}
$K+16$	\geq	C_1	C_{13}	$K+12$
$K+17$	\times	C_{12}	C_6	C_{14}
$K+18$	—	C_0	C_6	C_6
$K+19$	$\dot{+}$	$K+17$	C_{10}	$K+17$
$K+20$	\geq	K_1	$K+17$	$K+8$
$K+21$	+	C_{14}	$C_{14}+1$	$C_{14}+1$
$K+22$	\div	$K+17$	C_{11}	$K+17$
$K+23$				

В ячейке C_5 получим x_0

В ячейке C_6 получим y

Команды ($K+5$) и ($K+6$) служат для перевода $3E\left(\frac{x-a}{h}\right)$ в единицы 2-го адреса. В ячейке C_5 получим

		$3E\left(\frac{x-a}{h}\right)$	
--	--	--------------------------------	--

В ячейке $K+13$ получим

+	C_{12}	$A+3+3E\left(\frac{x-a}{h}\right)$	C_{12}
---	----------	------------------------------------	----------

В ячейке C_5 получим $y^2 - 1$

В ячейке C_6 получим $1 - y$. Команды ($K+18$) и ($K+19$) служат для изменения содержания ячеек C_6 и $K+17$ при переходе ко 2-му циклу

Команда ($K+22$) нужна для восстановления первоначального вида команды $K+17$ при переходе к следующей точке

$A+1$		$f(a)$		
$A+2$		$A(a)$		
$A+3$		$B^*(a)$		
.				
.				
.				
.				
$A+1+3n$		$f(a+nh)$		
$A+2+3n$		$A(a+nh)$		
$A+3+3n$		$B^*(a+nh)$		
C_0		1		
C_1		2		
C_2		2^{2s}		
C_3		a		
C_4		h		
C_5		x		
C_6		Что угодно		
C_7		3		
C_8		1...1		
C_9		1		
C_{10}				1
C_{11}				2
K_1	\times	C_{12}	C_6	$C_{14}+1$
K_2	$+$	C_{12}	$A+3$	C_{12}

s — число разрядов, отводимых под один адрес ячейки

3. В качестве следующего примера интерполирования по таблице заданных значений $f(a_0), f(a_1), \dots, f(a_n)$ составим программу для интерполяционной формулы Лагранжа по узлам a_0, a_1, \dots, a_n , расположенным как угодно. Эта формула записывается в виде:

$$f(x) = \left(\sum_{i=0}^n \frac{f(a_i) A(x)}{(x-a_i) A'(x)_{x=a_i}} \right) + R_n(x) \quad (1)$$

где $A(x) = (x - a_0)(x - a_1) \dots (x - a_n)$; $R_n(x) = A(x) \frac{f^{n+1}(\xi)}{(n+1)!}$

ξ — промежуточная точка интервала, содержащего a_0, a_1, \dots, a_n .

Для программирования представим формулу Лагранжа в виде суммы произведений

$$f(x) = \sum_{i=0}^n \prod_{j \neq i} \left(\frac{x - a_j}{a_i - a_j} \right) \cdot f(a_i) + R_n(x)$$

и будем отыскивать каждый сомножитель $\frac{x - a_j}{a_i - a_j}$. Вычисление таким способом увеличивает количество умножений примерно в 2 раза по сравнению с программированием по формуле (1), если $A(x)$ вынести за знак суммирования, но зато точность ответа значительно выше.

Программа записывается следующим образом:

K + 1	+	A + n + 1		C ₇
K + 2	—	C ₁	A	C ₅
K + 3	—	A	A	C ₆
K + 4	≥		C ₆	K + 7
K + 5	:	C ₅	C ₆	C ₆
K + 6	×	C ₆	C ₇	C ₇
K + 7	÷	K + 2	C ₂	K + 2
K + 8	÷	K + 3	C ₂	K + 3
K + 9	≥	K ₁	K + 2	K + 2
K + 10	+	C ₇	C ₈	C ₈
K + 11	÷	K ₂		K + 2
K + 12	÷	K + 3	C ₄	K + 3
K + 13	÷	K + 3	C ₃	K + 3
K + 14	÷	K + 1	C ₃	K + 1
K + 15	≥	K ₃	K + 3	K + 1
K + 16				

Выполнением команд (K + 3) и (K + 4) множитель $\frac{x - a_i}{a_i - a_j}$ при $j = i$ опускается

C₈ вначале пусто. В ячейке C₈ накапливаем ответ

A			a_0	
$A + 1$			a_1	
$A + 2$			a_2	
\cdot			\cdot	
\cdot			\cdot	
\cdot			\cdot	
$A + n$			a_n	
$A + n + 1$			$f(a_0)$	
$A + n + 2$			$f(a_1)$	
\cdot			\cdot	
\cdot			\cdot	
\cdot			\cdot	
\cdot			\cdot	
$A + 2n + 1$			$f(a_n)$	
C_1			x	
C_2				
C_3				
C_4				
K_1	—	C_1	$A + n$	C_5
K_2	—	C_1	A	C_5
K_3	—	$A + n$	A	C_6

Вычисление последовательных сумм $S_n^1 = \sum_{i=1}^n a_i$, $S_n^2 = \sum_{i=1}^n S_i^1, \dots, S_n^N = \sum_{i=1}^n S_i^{N-1}$. Пусть константы a_1, a_2, \dots, a_n

расположены в ячейках $A + 1, A + 2, \dots, A + n$. При вычислении последовательных сумм i -м циклом является процесс образования промежуточных сумм $S_1^i, S_2^i, \dots, S_n^i$. В целях экономии ЗУ промежуточные суммы i -го цикла помещаем в тех же ячейках ЗУ $A + 1, A + 2, \dots, A + n$. Чтобы при последующих циклах не терять найденные в ячейке $A + n$ значения S_n^i , они пересылаются в ячейки $B + i$.

В нижеследующей программе первые 3 команды служат для образования цикла, а команды $(K+4)$ и $(K+5)$ — для передачи S_n^i в ячейку $B+i$.

$K+1$	$\dot{+}$	K_3		$K+2$
$K+2$				
$K+3$	$\dot{+}$	$K+2$	C_2	$K+2$
$K+4$	\geq	K_1	$K+2$	$K+2$
$K+5$	$+$	$A+n$		$B+1$
$K+6$	$\dot{+}$	$K+5$	C_1	$K+5$
$K+7$	\geq	K_2	$K+5$	$K+1$
$K+8$				

C_1			1
C_2		1	1
K_1	$+$	$A+n-1$	$A+n$
K_2	$+$	$A+n$	$B+n$
K_3	$+$	$A+1$	$A+2$

Часто, например, при вычислении кратного интеграла в точках $x_0, x_0+h, x_0+2h, \dots, x_0+nh$ нужно вычислить суммы $S_1^N=a_1, S_2^N, \dots, S_n^N$. Как легко видеть, эти суммы получаются в ячейках $A+1, A+2, \dots, A+n$.

Если N мало по сравнению с n , что обычно имеет место, и, кроме того, числа a_1, a_2, \dots, a_n образуются по ходу вычислений, то вместо предварительного образования a_1, a_2, \dots, a_n и последующего суммирования, что сильно загружает ЗУ, можно вести вычисление последовательных сумм $S_n^1, S_n^2, \dots, S_n^N$ по ходу образования чисел a_1, a_2, \dots, a_n . Вычисление сумм ведем „по столбцам“, т. е. в последовательности:

$S_1^1, S_1^2, \dots, S_1^N$ — один цикл,

затем $S_2^1, S_2^2, \dots, S_2^N$ — второй цикл и т. д., ...,

наконец, $S_n^1, S_n^2, \dots, S_n^N$ — n -й цикл.

Пусть M — программа, образующая в ячейке B последовательно a_1, a_2, \dots, a_n . Тогда программа вычисления $S_n^1, S_n^2, \dots, S_n^N$ по ходу имеет вид:

получать всевозможные разности путем последовательного вычитания $(A+i) - (A+i-1)$ и передачи образованной разности в ячейку $A+i+m$ для $i = 1, 2, \dots, km$.

$K+1$	$-$	$A+1$	A	$A+m+1$
$K+2$	$+$	$K+1$	C_1	$K+1$
$K+3$	\geq	K_1	$K+1$	$K+1$
$K+4$				

C_1		1	1	1
K_1	$-$	$A+mk$	$A+mk-1$	$A+m(k+1)$

При этом, очевидно, разности $\Delta_n^0, \Delta_n^1, \dots, \Delta_n^k$ для точки $x_0 + nh$ находятся в ячейках $A+n, A+n+m, A+n+2m, \dots, A+n+km$. При большом k такой способ невыгоден, так как мы производим лишних $1+2+3+\dots+k = \frac{k(k+1)}{2}$ операций и удлиняем диапазон номеров ячеек, содержащих разности на $\frac{k(k+1)}{2}$ ячеек за счет образования нерабочих разностей, получаемых вычитанием разностей разных порядков $(A+m+1) - (A+m)$; $(A+2m+1) - (A+2m)$ и т. д. Поэтому при большом k рекомендуется вводить команду сравнения, чтобы избежать образования этих разностей.

Аналогично строится программа образования всевозможных разделенных разностей.

Перейдем теперь к образованию разностей и разделенных разностей в точке x_0 .

В качестве примера приведем программу вычисления обратных разностей $\rho_1(x_1, x_0), \rho_2(x_2, x_1, x_0), \dots, \rho_n(x_n, x_{n-1}, \dots, x_0)$,

где $\rho_i(x_i, x_{i-1}, \dots, x_0) = \frac{x_i - x_{i-1}}{\rho_{i-1}(x_i, \dots, x_1) - \rho_{i-1}(x_{i-1}, x_{i-2}, \dots, x_0)}$.

Обратными разностями мы пользуемся при составлении рационального приближения

$$R(x) = \frac{a_0 + a_1x + a_2x^2 + \dots}{b_0 + b_1x + b_2x^2 + \dots}$$

по известным значениям функции $y = f(x)$ в точках x_0, x_1, \dots, x_n в виде непрерывной дроби (см. § 17)¹

$$R(x) = y_0 + \frac{x - x_0}{\rho_1(x_1, x_0) + \frac{x - x_1}{\rho_2(x_2, x_1, x_0) + \frac{x - x_2}{\rho_3(x_3, x_2, x_1, x_0) + \dots + \frac{x - x_n}{\rho_n(x_n, x_{n-1}, \dots, x_0)}}}$$

Пусть y_0, y_1, \dots, y_n содержатся в ячейках $B, B+1, \dots, B+n$, а $x_0, x_1, x_2, \dots, x_{n-1}, x_n$ — в ячейках $A, A+1, A+2, \dots, A+n$.

Образует все первые разделенные разности

$\rho_1(x_n, x_{n-1}) = \frac{x_n - x_{n-1}}{y_n - y_{n-1}}$, $\rho_1(x_{n-1}, x_{n-2}), \dots, \rho_1(x_1, x_0)$ и поместим их в ячейках $B+n, B+n-1, \dots, B+1$. Затем образуем вторые разделенные разности

$$\rho_2(x_n, x_{n-1}, x_{n-2}), \dots, \rho_2(x_2, x_1, x_0)$$

и поместим их в ячейках $B+n, B+n-1, \dots, B+2$ и т. д. В результате, после n циклов получим $y_0, \rho_1(x_1, x_0), \rho_2(x_2, x_1, x_0), \dots, \rho_n(x_n, x_{n-1}, \dots, x_0)$ в ячейках $B, B+1, B+2, \dots, B+n$.

Программа имеет вид:

K+1	—	B+n	B+n-1	B+n	A	x ₀			
K+2	—	A+n	A+n-1	C ₄	A+1	x ₁			
K+3	:	C ₄	B+n	B+n	.	.			
K+4	÷	K+1	C ₁	K+1	.	.			
K+5	÷	K+2	C ₃	K+2	A+n	x _n			
K+6	÷	K+3	C ₂	K+3	B	y ₀			
K+7	≥	K+1	K ₂	K+1	B+1	y ₁			
K+8	÷	K ₄		K+1	.	.			
K+9	÷	K ₁		K+2	B+n	y _n			
K+10	÷	K ₃		K+3	C ₁	1	1	1	1
K+11	÷	K ₂	C ₁	K ₂	C ₂	1	1		
K+12	≥	K ₄	K ₂	K+1	C ₃	A+n	A+n-1	C ₄	
K+13					K ₁	B+1	B	B+1	
					K ₂	C ₄	B+n	B+n	
					K ₃	B+n	B+n-1	B+n	
					K ₄	B+n	B+n-1	B+n	

¹ См. Милн. Численный анализ, И. Л., 1951 г., стр. 170.

Рассмотрим, наконец, программу образования разностей $\Delta_n^0, \Delta_n^1, \dots, \Delta_n^k$ последовательно в точках $x_n = x_0 + nh$. Такая задача возникает, например, при осуществлении контроля по k -м разностям (см. § 18): последовательно образуя k -е разности в точках $x_k, x_k + h, \dots$, сравнивают их с заданным числом ε ; если $|\Delta_n^k| < \varepsilon$, где $n \geq k$, то ошибки в вычислении $f(x_0 + nh)$ нет, если же $|\Delta_n^k| > \varepsilon$, то допущена ошибка и машина останавливается.

В качестве примера приведем программу контроля. Сначала по программе P в ячейке y находим $\Delta_0^0 = f(x_0)$, затем по той же программе P находим $\Delta_1^0 = f(x_1)$, после чего составляем разность $\Delta_1^1 = \Delta_1^0 - \Delta_0^0$, которую помещаем в ячейку $A+1$; возвращаемся обратно к программе P и находим $\Delta_2^0 = f(x_2)$, после чего находим $\Delta_2^1 = \Delta_2^0 - \Delta_1^0$; $\Delta_2^2 = \Delta_2^1 - \Delta_1^1$ и помещаем эти разности в ячейках $A+1, A+2$ и т. д. После k шагов мы получим $\Delta_k^0, \Delta_k^1, \dots, \Delta_k^k$ в ячейках $A, A+1, \dots, A+k-1, C_8$. Сравнением $\Delta_k^k = (C_8)$ и ε по абсолютной величине проверяем правильность вычисления $f(x_k)$. При переходе к следующим точкам x_{k+1}, x_{k+2} и т. д., мы каждый раз получаем в ячейках $A, A+1, \dots, A+k-1, C_8$ величины $\Delta_n^0, \Delta_n^1, \dots, \Delta_n^k$

P				
$K+1$	+	C_4		C_7
$K+2$	—	y	A	C_8
$K+3$	—	C_7	C_2	C_7
$K+4$	+	y		A
$K+5$	+	C_8		y
$K+6$	+	$K+2$	C_5	$K+2$
$K+7$	+	$K+4$	C_6	$K+4$
$K+8$	\geq	C_7	C_2	$K+2$
$K+9$	+	K_2		$K+2$
$K+10$	+	K_1		$K+4$
$K+11$	+	C_4	C_2	C_4
$K+12$	\geq	C_3	C_4	P
$K+13$	+	K_3		$K+11$
$K+14$	+			$K+12$
$K+15$	\geq			$K+11$

Ячейки $A, \dots, A+k-1$ вначале пустые
В ячейке C_8 образуем Δ_n^k

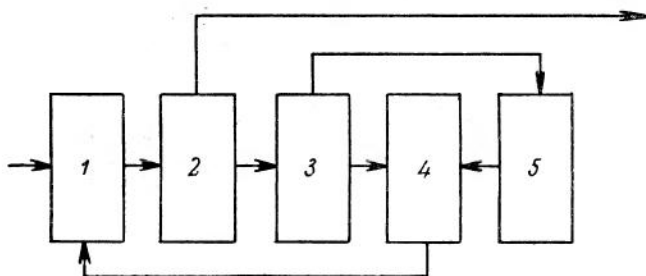
В ячейке C_4 вырабатываем для $n < k$ признак, когда переходить к отысканию следующей точки Δ_{n+1}^0 по программе P . Для $n \geq k$ ячейку C_4 оставляем неизменной, и поэтому в ячейку $K+11$ подаем команду о сравнении Δ_n^k с ε

C_1			ε	
C_2			1	
C_3			$k-1$	
C_4			1	
C_5			1	
C_6				1
K_1	+	y		A
K_2	-	y	A	C_8
K_3	$ \geq $	C_1	C_8	P

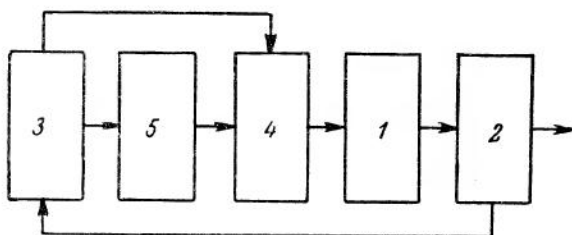
Метод сравнений. Приведем программу решения уравнения $f(x)=0$ методом сравнений. Пусть $f(a)f(b)<0$, вычисляем $f\left(\frac{a+b}{2}\right)$; если $f(a)f\left(\frac{a+b}{2}\right)<0$, то перейдем к рассмотрению интервала $\left(a, \frac{a+b}{2}\right)$; если $f(a)f\left(\frac{a+b}{2}\right)>0$ — к интервалу $\left(\frac{a+b}{2}, b\right)$. Вычисления будем вести до тех пор, пока при вычислении функции мы не получим достаточно малой величины. Можно разбить задачу на этапы:

1. Вычислить $\xi = \frac{a+b}{2}$, $f(\xi)$.
2. Сравнить по абсолютной величине $f(\xi)$ и заранее заданное ε . При $|f(\xi)|<\varepsilon$ прекратить решение этой задачи.
3. При $|f(\xi)|\geq\varepsilon$ образовать $f(\xi)f(a)$ и сравнить: $f(\xi)f(a)>0$ или $f(\xi)f(a)<0$.
4. При $f(\xi)f(a)\leq 0$ возьмем $\frac{a+b}{2}$ вместо b , $f\left(\frac{a+b}{2}\right)$ вместо $f(b)$, далее перейдем к этапу 1.
5. При $f(\xi)f(a)>0$ заменим a на b и $f(a)$ на $f(b)$, далее перейдем к этапу 4.

Такому разбиению на этапы соответствует следующая блок-схема:



В программе их естественно расположить в следующем порядке:



Для того, чтобы начать с этапа 3, достаточно в ячейки ЗУ, где будут находиться $\frac{a+b}{2}$ и $f\left(\frac{a+b}{2}\right)$, поместить соответственно b и $f(b)$.

Программа для вычислений имеет вид:

$K+1$	\times	A'_1	A'_2	A_4
$K+2$	\geq		A_4	$K+5$
$K+3$	$+$	A'_3		A'_2
$K+4$	$+$	A_3		A_2
$K+5$	$+$	A_1		A_3
$K+6$	$+$	A'_1		A'_3
$K+7$	$+$	A_2	A_3	A_1
$K+8$	$:$	A_1	A_5	A_1
$K+9$	f	A_1		A'_1
$K+10$	$ \geq $	A'_1	A_6	$K+1$
$K+11$				

$(A_1) = b$ $(A'_1) = f(b)$
 $(A_2) = a$ $(A'_2) = f(a)$
 $(A_3) = b$ $(A'_3) = f(b)$
 (A_4) — что угодно $(A_5) = 2$
 $(A_6) = \varepsilon$ — заранее заданная точность
 f — символ нахождения $f(x)$:

f	A		B
-----	-----	--	-----

 — найти $f((A))$ и отправить в ячейку B

По окончании вычислений $x = (A_1)$.

Здесь команды $(K+1)$, $(K+2)$ соответствуют этапу 3; $(K+5)$, $(K+6)$ — этапу 4; $(K+3)$, $(K+4)$ — этапу 5; $(K+7)$ — $(K+9)$ — этапу 1; $(K+10)$ — этапу 2.

Предварительно требуется знать $f(a)$ и $f(b)$ (точнее величины того же знака, что $f(a)$ и $f(b)$).

Если эти величины не вычислены, то надо добавить команды:

$K-2$	f	A_1		A'_1
$K-1$	$+$	A'_1		A'_3
K	f	A_2		A'_2

Эта программа, как видно, не требует арифметических операций над командами.

Преобразование чисел из одной системы в другую. Полные программы, осуществляющие такие преобразования, зависят от особенностей машин. В самом деле, при преобразовании, например, двоичной записи числа в десятичную нужно получить не только набор десятичных цифр, представляющих данное число, но тем или иным способом осуществить самую запись этого числа (хотя бы в двоично-десятичной системе) и выдачу полученной записи. Эти последние этапы зависят от особенностей машины.

Приведем часть программы преобразования двоичной записи в десятичную в машинах с плавающей запятой, именно: преобразования числа $x = 2^k \cdot 0, \alpha_1 \alpha_2 \alpha_3 \dots \alpha_n$ ($\alpha_1 = 1$, $\alpha_i = 0$ или 1) к виду $x = 10^s \cdot 0, \beta_1 \beta_2 \beta_3 \dots \beta_t$ ($\beta_1 \neq 0$) (здесь s — десятичный порядок числа x , β_i — десятичные цифры). Мы ограничимся программой определения десятичного порядка s и набора десятичных цифр $\beta_1, \beta_2, \dots, \beta_t$.

Программа, естественно, распадается на два этапа: первый этап — определение десятичного порядка s и десятичной цифровой части $y = x \cdot 10^{-s}$; второй этап — последовательное определение десятичных цифр $\beta_1, \beta_2, \dots, \beta_t$ по заданному y .

Приведем два из возможных вариантов программы, отличающихся своими первыми этапами.

Первый вариант требует большего числа действий для определения десятичного порядка, но меньшей нагрузки ЗУ.

Пусть $-t$ — наименьшая целая степень 10, изображаемая машиной: $10^{-t} > 2^{-n} > 10^{-t-1}$. Поместим в ячейки ЗУ следующие числа $(A_1) = 0$, $(A_1 + 1) = -t$; $(A_2) = 1$; в последующем $(A_2) = 10^{-t}$. Сравниваем числа x и $(A_2) = 1$. При x , большем единицы, циклически повторяется следующая серия операций: умножаем (A_2) на 10 с посылкой ответов в A_2 , добавляем к (A_1) единицу, так что все время $(A_2) = 10^{(A_1)}$, сравниваем (A_2) с x . Эта серия операций продолжается пока не появится неравенство $(A_2) > x$. В момент появления этого неравенства, после s -кратного повторения, имеем: $(A_1) = s$, $(A_2) = 10^s > x \geq 10^{s-1}$. Очевидно s есть искомый десятичный порядок числа x .

Далее

$$y = x : (A_2) = x \cdot 10^{-s}$$

есть искомая десятичная цифровая часть числа x .

Второй вариант основан на том, что искомый десятичный порядок s связан с двоичным порядком k числа x одним из двух возможных равенств:

$$s = E(k \lg_{10} 2) \text{ или } s = E(k \lg_{10} 2) + 1$$

Пусть в ячейках $A + p$, $p = -t, -t + 1, \dots, 0, 1, \dots, t$ хранятся числа 10^p . Образует число $s_1 = E(k \lg_{10} 2)$, для этого нужно двоичный порядок k числа x превратить в нормализованное число, помножить его на $\lg_{10} 2$, взять целую часть произведения и послать в ячейку $B : (B) = s_1$.

Сравним содержимое ячейки $A + (B) = A + s$ с числом x . При $(A + s_1) = 10^{s_1} > x$, число s_1 есть десятичный порядок x : $s_1 = s$, а число $y = x : (A + s_1)$ — цифровая десятичная часть числа x . При $(A + s_1) \leq x$, $s = s_1 + 1$ и $y = x : (A + s_1 + 1)$.

Замечание. Можно было бы заменить нахождение числа s , для которого удовлетворяется неравенство $10^s > x \geq 10^{s-1}$, поисками в таблице степеней 10 по методу проб.

Второй этап — определения цифр $\beta_1, \beta_2, \dots, \beta_t$ ведется последовательно по формулам: $\beta_i = E(10 \beta_{i-1})$, $\beta_{i-1} = \{\beta_{i-1}\}$, $\beta_0 = y$; эти операции повторяются циклически t раз.

Первый вариант

$K+1$	$>$	A_2	A_3	$K+l+11$	A_1	0
$K+2$	\times	A_2	A_4	A_2	A_2	1
$K+3$	$+$	A_5	A_1	A_1	A_3	x
$K+4$	\geq	A_3	A_2	$K+2$	A_4	10
$K+5$	$:$	A_3	A_1	A_2	A_5	1
$K+6$	$+$	A_5	A_6	A_5	A_6	0
$K+7$	\times	A_4	A_2	A_2	A_7	$t-1$
$K+8$	E	A_2	B	A_2	A_8	10^{-t}
$K+9$	A_9	$-t$
.						
.						
$K+l+8$		
$K+l+9$	\geq	A_9	A_6	$(K+6)$		
$K+l+10$		
$K+l+11$		
$K+l+12$	$-$	A_1	A_7	A_1		
$K+l+13$	$+$	A_8		A_2		
$K+l+14$	$+$	A_9		A_1		

A_4, A_5, A_7, A_8 — ячейки постоянного ЗУ.

Команды $(K+2) - (K+4)$ служат для определения десятичного порядка s , который будет в конце процесса храниться в ячейке A_1 . При $x < 1$ команды $(K+l+12) - (K+l+14)$ предварительно меняют содержимое ячеек A_1 и A_2 и возвращают к выполнению основной программы.

Команда $(K+5)$ образует $y = x \cdot 10^{-s}$. Команды $(K+7)$, $(K+8)$ образуют десятичные цифры $\beta_1, \beta_2, \dots, \beta_i$, которые у нас (условно) поступают в ячейку B. Далее должны следовать l опускаемых команд, связанных с образованием десятичного числа $0, \beta_1, \beta_2, \dots, \beta_l$. Команда $(K+6)$ подсчитывает число образованных десятичных цифр; когда это число становится равным t , команда $(K+l+11)$ прекращает процесс образования этих цифр и вывода их. Мы оставили незаполненными команды $(K+9) - (K+l+9)$ и одну команду $(K+l+10)$, связанные с выводом десятичной записи числа.

Процесс вывода полученного десятичного числа может включать, например, предварительную двоично-десятичную

запись числа y ; полученная после выполнения команды $(K+8)$ двоичная запись (B) десятичной цифры β_i ставится после двоично-десятичной записи числа $0, \beta_1 \beta_2 \dots \beta_{i-1}$. Пусть эта запись хранится в ячейке C (в начальный момент $(C) = 0$).

В этом случае пропущенная часть программы будет иметь вид: $(t = 3)$

$K+9$	\rightarrow	B	D_2	B	B	0
$K+10$	\leftarrow	C	D_1	C	C	0
$K+11$	$\dot{+}$	B	C	C	D_1	4
					D_2	$4(t-1)$

(D_1 и D_2 — ячейки постоянного ЗУ).

Эти три команды превратят двоично-десятичное число $(C) = 0, \underbrace{00\dots 0\beta_1\beta_2\dots\beta_{i-1}}_i$ в число $0, \underbrace{00\dots 00\beta_1\beta_2\dots\beta_{i-1}}_{t-1}\beta_i$. После t циклов будем иметь: $(C) = 0, \beta_1\beta_2 \dots \beta_t$. Пропущенная команда $(K+l+10) = K+13$ должна быть командой вывода десятичного числа (C) .

При этом варианте требуется произвести до t умножений на 10 для определения десятичного порядка. Приведем другой вариант программы, где эти лишние умножения отсутствуют, но зато загружены лишних $2t+1$ ячеек постоянного ЗУ.

Второй вариант

$K+1$	\downarrow	A_3		A_1	A_1	
$K+2$	\times	A_1	A_8	A_2	A_2	
$K+3$	E	A_2	A_1		A_3	x
$K+4$	\rightarrow	A_1	A_9	A_2	A_4	10
$K+5$	$\dot{+}$	A_2	$K+6$	$K+6$	A_5	1
$K+6$:	A_3	A	A_2	A_6	0
$K+7$	\geq	A_5	A_2	$K+l+14$	A_7	$t-1$
$K+8$	\times	A_2	A_4	A_2	A_8	$\lg_{10} 2$
$K+9$	$\dot{+}$	A_6	A_5	A_6	A_9	p
$K+10$	E	A_2	B	A_2	$A+i$	10^i
$K+11$		
.						
.						
.						

($i = -t, -t+1, \dots, 0, 1, \dots, t$)

$K + l + 10$
$K + l + 11$	\geq	A_7	A_6	$K + 9$
$K + l + 12$
$K + l + 13$
$K + l + 14$	$+$	A_5	A_1	A_1
$K + l + 15$	\geq			$K + 9$

Ячейки $A_4, A_5, A_7, A_8, A + i$ — ячейки постоянного ЗУ (p — означает число разрядов, на которое надо сдвинуть натуральное число, чтобы поместить его во второй адрес). Команды $(K + 1) - (K + 3)$ служат для образования числа $s_1 = E(k \lg_{10} 2) = (A_1)$. В результате команд $(K + 4)$ и $(K + 5)$ во втором адресе $(K + 6)$ будет стоять число $A + s_1 + 1$; $(A + s_1 + 1) = 10^{s_1}$. Команда $(K + 6)$ образует число $(A_2) = x : (A + s_1) = x 10^{-s_1}$. Искомый десятичный порядок s равен $(A_1) = s_1$ или $(A_1) + 1 = s_1 + 1$, в зависимости от того, будет ли $(A_2) = x \cdot 10^{-s_1} < 1$ или $(A_2) \geq 1$. Команда условного перехода $(K + 7)$ соответственно разветвляет процесс вычисления. Команды $(K + l + 14), (K + l + 15)$ включают в общий процесс случай $s = s_1 + 1$, т. е. $(A_2) \geq 1$. Команды $(K + 8) - (K + 10)$ образуют, как в предыдущем варианте, десятичные цифры $\beta_1, \beta_2, \dots, \beta_t$ и подсчитывают число уже образованных цифр. Дальше программа идет, как в предыдущем варианте, в частности, тем же целям служат пропущенные команды $(K + 11) - (K + l + 10)$ и $(K + l + 12)$.

Операции с удвоенным числом разрядов. Мы приведем программу операции деления с удвоенным числом разрядов. Как рассматривалось в § 16,

$$z_1 + 2^{-n} z_2 = \frac{x_1 + 2^{-n} x_2}{y_1 + 2^{-n} y_2} = \left(\frac{x_1}{y_1} \right)_1 + 2^{-n} \frac{0 \left(\frac{x_1}{y_1} \right) + x_2 - \left(\frac{x_1}{y_1} \right)_1 y_2}{y_1}$$

где $\left(\frac{x_1}{y_1} \right)_1$ — n -значное частное $\frac{x_1}{y_1}$, $0 \left(\frac{x_1}{y_1} \right)$ — остаток от деления $\frac{x_1}{y_1}$. При реализации вычислений действия должны были

бы производиться с блокировкой нормализации (см. § 11), но для того чтобы выделить из второго слагаемого целую

часть, мы проведем его вычисление с учетом нормализации, а затем приведем к нужному порядку. Порядок действий при составлении программы здесь очевиден; программа является примером „непосредственного программирования“.

Программа имеет следующий вид:

$K + 1$	\div	A_1	A_2	A_3	$(A_1) = x_1$
$K + 2$				A'_3	$(A'_1) = x_2$
$K + 3$	\times	A_3	A'_2	A'_2	$(A_2) = y_1$
$K + 4$	$-$	A'_1	A'_2	A'_2	$(A'_2) = y_2$
$K + 5$	$+$	A'_3	A'_2	A'_3	$(C) = 2^{-n}$
$K + 6$	$:$	A'_3	A_2	A'_3	По оконча- нии вычис- лений
$K + 7$	E	A'_3	A_1	A'_3	
$K + 8$	\times	A_1	C	A_1	$(A_3) = z_1$
$K + 9$	$+$	A_1	A_3	A_3	$(A'_3) = z_2$

Программа очень проста и пояснений не требует.

Как указывалось в § 16, перед делением целесообразно „сдвинуть“ числитель и знаменатель влево. Этот сдвиг есть частный случай операции деления (деление с блокировкой нормализации на $0,0 \dots 010 \dots 0$) и может быть проведен по программе для деления, которое в этом случае получается точным. Необходимо только выделить в ненормализованном виде 2^{-s} , где s — число нулей после запятой в y_1 . Это можно реализовать программой

$K' + 1$	$ \geq $	A_2	F	$K' + 5$	$(A_2) = y_1$				
$K' + 2$	$:$	A_2	F	D	(D) — что угодно				
$K' + 3$	\wedge	D	E	D	E <table> <tr> <td>$1...1$</td> <td>$100...$</td> <td></td> <td></td> </tr> </table>	$1...1$	$100... $		
$1...1$	$100... $								
$K' + 4$	$+$	D		D	$(F) = \frac{1}{2}$				

Аналогичные программы можно дать для умножения (и, следовательно, сдвига вправо), сложения, вычитания.

Операции над векторами. При решении многих математических задач встречаются операции над векторными величинами. Конечно, каждая из таких операций эквивалентна совокупности операций над компонентами заданных векторов и поэтому нет необходимости вводить операции над векторами в число основных операций, осуществляемых машиной. Однако при программировании решения математических задач использование векторной символики может принести существенную пользу. При составлении программы каждую операцию над векторами можно записывать формально в виде одной команды с тем, чтобы каждую из таких команд заменить стандартной подпрограммой, осуществляющей данную операцию над векторами. Эти стандартные подпрограммы можно составить следующим образом.

1. Программа сложения и вычитания векторов

$K+1$	\pm	δ	ν	$K+2$	δ μ ν	\pm	α	β	γ
$K+2$	\pm	α	β	γ					
$K+3$	\dagger	$K+2$	μ	$K+2$			1	1	1
$K+4$	\geq	δ	$K+2$	$K+2$			$n-1$	$n-1$	$n-1$
$K+5$	\geq			$K+m$					

В ячейке ЗУ δ помещаются код операции и адреса последних компонент векторов-слагаемых и вектора-суммы. Содержание команды ($K+2$) перед выполнением подпрограммы в сущности не играет никакой роли, так как команда ($K+2$) образуется при исполнении команды ($K+1$). Содержимое ячейки ЗУ μ служит для изменения всех трех адресов команды ($K+2$) на одну единицу. Содержимое ячейки ЗУ ν зависит от размерности n рассматриваемых векторов.

2. Программа умножения вектора на скаляр

$K+1$	\cdot	δ	ν_1	$K+2$	δ μ_1 ν_1	\times	α	β	γ
$K+2$	\times	α	β	γ					
$K+3$	\dagger	$K+2$	μ_1	$K+2$			1		1
$K+4$	\geq	δ	$K+2$	$K+2$			$n-1$		$n-1$
$K+5$	\geq			$K+m$					

В ячейке ЗУ β помещается скаляр. Компоненты вектора размещены в ячейках ЗУ $\alpha - n + 1, \dots, \alpha$.

3. Программа умножения матрицы на вектор

$K+1$	$\dot{-}$	δ	v_2	$K+3$
$K+2$	$\dot{-}$	δ_1	v_1	$K+4$
$K+3$	\times	α	β	ρ
$K+4$	$+$	γ	ρ	γ
$K+5$	$\dot{+}$	$K+3$	μ_2	$K+3$
$K+6$	\geq	δ	$K+3$	$K+3$
$K+7$	$\dot{-}$	$K+3$	v_3	$K+3$
$K+8$	$\dot{+}$	$K+4$	μ_1	$K+4$
$K+9$	\geq	δ_1	$K+4$	$K+3$
$K+10$	\geq			$K+m$

δ	\times	α	β	ρ
δ_1	$+$	γ	ρ	γ
μ_1		1		1
μ_2		1	1	
v_1		$n-1$		$n-1$
v_2		$n-1$	n^2-1	
v_3		n		

Здесь предполагается, что компоненты вектора x , на который умножается матрица $A = (a_{ij})$, размещены в ячейках ЗУ $\alpha - n + 1, \dots, \alpha$, а элемент a_{ij} матрицы находится в ячейке ЗУ $\beta - n^2 + n(i-1) + j$. Компоненты вектора-результата оказываются в ячейках ЗУ $\gamma - n + 1, \dots, \gamma$.

Приведенная здесь программа по существу совпадает с программой из § 24. Отсутствующие там операции, осуществляемые здесь командами $(K+1)$ и $(K+2)$, и последняя команда $K+10$ нужны здесь лишь для превращения этой программы в стандартную подпрограмму.

Пользоваться этими стандартными подпрограммами при программировании можно двумя различными способами. Один из возможных способов состоит в том, что операция над векторами, записанная в программе формально в виде одной команды, записывается в виде команд $K+1, K+2, K+3, K+4, \dots$ и номерам ячеек $\alpha, \beta, \gamma, \dots$ придается конкретное значение. Команда $(K+5)$ ($(K+10)$ для умножения матрицы на вектор) оказывается в этом случае излишней. Второй способ предусматривает наличие в самой машине рассматриваемой стандартной подпрограммы, и ее включение в основную программу осуществляется при помощи следующих команд: команды отправки в ячейки ЗУ δ, δ_1 кон-

кретных адресов, команды посылки в последний адрес команды $K + 5$ ($K + 10$ — при умножении) номера команды основной программы, к которому следует перейти после выполнения операции над векторами, и, наконец, команды перехода к команде ($K + 1$). Таким образом, при включении стандартной подпрограммы требуется лишь три (соответственно, четыре) команды вместо четырех (соответственно, девяти) команд для осуществления векторной операции по первому способу.

Вычисление коэффициентов Фурье. Рассмотрим вопрос о численном определении коэффициентов Фурье

$c_r = \int_0^1 f(x) \sin r\pi x dx$ по заданным значениям $f_i = f\left(\frac{i}{n}\right)$ при $i = 1, 2, \dots, n - 1$.

Для приближенного вычисления c_r , $r = 1, 2, \dots, n - 1$ применяются формулы

$$c_r \approx \frac{1}{n} \sum_{i=1}^{n-1} f_i \sin \frac{ir\pi}{n}$$

Составим программу вычисления коэффициентов c_r .

Рассмотрим случай, когда в ЗУ хранятся значения $\sin \frac{ir\pi}{n}$, $0 \leq i < \frac{n}{2}$ (из первой четверти).

Программа предполагает приведение $\sin \frac{ir\pi}{n}$ к первому полупериоду (команды $(K + 6) - (K + 10)$), т. е. нахождение целого числа j_{ir} , $0 \leq j_{ir} < n$, сравнимого с ir по модулю n , $j_{ir} = ir - mn$ (m — целое). Тогда $\sin \frac{ir\pi}{n} = (-1)^m \frac{\sin j_{ir}\pi}{n}$, где $m = E\left(\frac{ir}{n}\right)$. После того как найдено $j_{i-1, r}$, для нахождения j_{ir} мы находим сначала $\tilde{j}_{ir} = j_{i-1, r} + r$ (команда $(K + 6)$). Если полученное число \tilde{j}_{ir} меньше n , то $j_{ir} = \tilde{j}_{ir}$. Если же оно больше или равно n , то, вычтя из него n , найдем

$j_{ir} = \tilde{j}_{ir} - n$ (команды $(K+7)$, $(K+8)$). При этом мы вступили в следующий полупериод, т. е. должны изменить знак синуса (команда $(K+9)$).

Для приведения к первой четверти вставим две команды $(K+11)$ и $(K+12)$. Мы получим после их выполнения $(D_4) = j'_{ir} 2^{-s_2}$, где $0 \leq j'_{ir} \leq \frac{n}{2}$ и $\sin \frac{j'_{ir} \pi}{n} = \sin \frac{j_{ir} \pi}{n}$. Прибавление этого числа к (C_8) означает прибавление j'_{ir} ко второму адресу (C_8) . Команды $(K+13)$ и $(K+14)$ добавляют соответственно числа i и j'_{ir} к первому и второму адресу (C_8) и формируют команду $(K+15)$, которая примет вид:

$$(K+15) = \left[\times \mid A + i \mid B + j'_{ir} \mid D_8 \right]$$

Выполнение команды $(K+15)$ образует число $f_i \sin \frac{j'_{ir} \pi}{n} = \left| f_i \sin \frac{ir\pi}{n} \right|$, а команда $(K+16)$ придает этому числу нужный знак. $(K+17)$ образует $\sum f_i \sin \frac{ir\pi}{n}$, а $(K+18)$ сравнивает фактически i и $n-1$. При $i < (n-1)$ мы повторяем цикл для следующего значения $i+1$. При $i = n-1$ мы заканчиваем вычисление данного коэффициента Фурье

$$c_r \approx \frac{1}{n} \sum_{i=1}^{n-1} f_i \sin \frac{ir\pi}{n}. \text{ Вычислительная команда } (K+20) \text{ обра-}$$

зует число c_r и посылает его в ячейку $l+r$. Предварительно $(K+19)$ образует нужный третий адрес команды $(K+20)$.

Следующие команды $(K+21)$, $(K+22)$ осуществляют переход от r к $r+1$ и при $r < n-1$ реализуют возвращение к выполнению большого цикла вычисления c_r . Первые команды $(K+1) - (K+5)$ возвращают содержимое ячеек ЗУ к первоначальному виду.

Программа имеет такой вид: в начале процесса вычисления в ЗУ хранятся следующие числа:

$A + 1$	f_1	
.	.	
.	.	
$A + i$	f_i	
$B + 1$	$\sin \frac{\pi}{n}$	
.	.	
.	.	
$B + j$	$\sin \frac{j\pi}{n}$	
C_1	n	Единица второго адреса
C_2	-1	
C_3	2^{-s_2}	
C_4	$(n-1) \cdot 2^{-s_1}$	Единица первого адреса
C_5	2^{-s_1}	
C_6	1	
C_7	2^{-s_3}	Единица третьего адреса
C_8	$\times A B D_8$	
C_9	$n \cdot 2^{-s_2}$	
C_{10}	$\frac{n}{2} \cdot 2^{-s_2}$	
C_{11}	$: D_2 C_1 l$	
D_6	1	

$K + 1$	+	C_6		D_1	Передача 1 в D_1
$K + 2$	+			D_2	Очистка ячейки D_2
$K + 3$	+			D_5	Очистка ячейки D_5
$K + 4$	+			D_7	Очистка ячейки D_7
$K + 5$	+	D_7	C_5	D_7	Посылка 1 первого адреса ячейки D_7
$K + 6$	+	D_5	D_6	D_5	В ячейке D_5 образуется \tilde{j}_{ir}
$K + 7$	\geq	C_1	D_5	$K + 9$	Число n сравнивается с числом \tilde{j}_{ir}
$K + 8$	-	D_5	C_1	D_5	Из числа \tilde{j}_{ir} вычитается число n

$K + 9$	\times	D_1	C_2	D_1
$K + 10$	\times	D_5	C_3	D_4
$K + 11$	\geq	C_{10}	D_4	$K + 13$
$K + 12$	$-$	C_9	D_4	D_4
$K + 13$	$\dot{+}$	C_8	D_4	$K + 15$
$K + 14$	$\dot{+}$	$K + 15$	D_7	$K + 15$
$K + 15$				
$K + 16$	\times	D_1	D_3	D_3
$K + 17$	$+$	D_2	D_3	D_2
$K + 18$	\geq	C_4	D_7	$K + 5$
$K + 19$	$\dot{+}$	$K + 20$	C_7	$K + 20$
$K + 20$				
$K + 21$	$+$	D_6	C_6	D_6
$K + 22$	\geq	C_1	D_6	$K + 1$
$K + 23$				

В D_1 образуется $\text{sign} \sin \frac{ir\pi}{n}$

Величина j_{ir} засылается во 2-й адрес

Число $\frac{n}{2} \cdot 2^{-s_2}$ сравнивается с числом $2^{-s_2} \cdot j_{ir}$

Из $n \cdot 2^{-s_2}$ вычитается $2^{-s_2} \cdot j_{ir}$

Формирование команды ($K + 15$)

Формирование команды ($K + 15$)

В ячейке D_3 образуется $\left| f_i \sin \frac{ir\pi}{n} \right|$

В ячейке D_3 образуется $f_i \sin \frac{ir\pi}{n}$

В ячейке D_2 образуется

$$\sum f_i \cdot \sin \frac{ir\pi}{n}$$

Сравнение с $(n-1) \cdot 2^{-s_1}$

Формирование 3-го адреса команды ($K + 20$)

В ячейке $(l + r)$ образуется коэффициент c_r

Замена r на $r + 1$

Глава V

ПРОГРАММЫ ДЛЯ РЕШЕНИЯ НЕКОТОРЫХ МАТЕМАТИЧЕСКИХ ЗАДАЧ

Каждая программа должна начинаться с этапа, связанного с вводом в машину всех команд и числовых данных.

Мы этот этап в программах пропускаем, так как он существенно зависит от индивидуальных особенностей машины. По той же причине мы опускаем те части программы, которые связаны с выводом полученных ответов, ограничившись указанием ячеек, в которых ответы образуются, или указанием команды послылки в печать.

Для машин, работающих по двоичной системе, ввод и вывод числовых данных связан обычно с преобразованием чисел из одной системы в другую. Схемы вычислений для перевода приведены в § 16. Программа, реализующая эти схемы, зависит также от индивидуальных особенностей машины (см. § 27).

Оформление программ и соответствующие обозначения такие же, как и в главе IV.

Наряду с перечнем команд приводится содержимое ячеек ЗУ к началу процесса; (α) означает содержимое ячейки с номером α . Для единообразия и удобства чтения программ применяется следующая система обозначений: ячейки, хранящие рабочие команды, обозначаются $K + 1, \dots, K + s$, запасные команды K_1, K_2, \dots, K_s ; ячейки, хранящие вспомогательные величины $A_1, A_2, \dots, B_1, \dots, a_1, \dots$, или $A + 1, A + 2, \dots$, если их порядок существует.

§ 28. ВЫЧИСЛЕНИЕ ФУНКЦИЙ

В § 17 указаны различные методы задания функций в машине. Здесь мы приводим для иллюстрации программы вычисления некоторых функций.

Нахождение $\log_2 x$ при помощи интерполяционной формулы. Приведем программу вычисления $\log_2 x$ при произвольном x с точностью до 2^{-35} . Пусть x нормализовано

$$x = 2^m \cdot 0,1 \alpha_1 \alpha_2 \alpha_3 \alpha_4 \dots \quad (1)$$

где $\alpha_i = 0$ или 1.

Обозначим через y число, которое получится, если в двоичном представлении (1) числа x положить $m = 0$ и $\alpha_i = 0$ для $i \geq 4$, т. е.

$$y = 0,1 \alpha_1 \alpha_2 \alpha_3 000 \dots \quad (2)$$

y пробегает следующие возможные значения

$$\frac{8}{16} = 0,1000 \dots; \frac{9}{16} = 0,100100 \dots; \dots; \frac{15}{16} = 0,1111000 \dots$$

Положим

$$x : 2^m y = z \quad (3)$$

Очевидно, z меняется в пределах от 1 до $9/8$. Таким образом, $\log_2 x$ представляется как сумма

$$\log_2 x = m + \log_2 y + \log_2 z \quad (4)$$

Составим сначала программу отыскания $\log_2 z$. Вычисление будем вести при помощи интерполяционного многочлена, составленного по узлам $z = 1, \frac{49}{48}, \frac{50}{48}, \frac{51}{48}, \frac{52}{48}, \frac{53}{48}, \frac{9}{8}$.

Расположим интерполяционный многочлен $P(z)$ по убывающим степеням центральной разности $z - \frac{51}{48} = z - \frac{17}{16}$

$$P(z) = a_6 \left(z - \frac{17}{16}\right)^6 + a_5 \left(z - \frac{17}{16}\right)^5 + \dots + a_1 \left(z - \frac{17}{16}\right) + a_0 \quad (5)$$

$$\text{где } a_0 = P\left(\frac{17}{16}\right) = \log_2 \frac{17}{16}; a_1 = \frac{d}{dz} [P(z)]_{z=\frac{17}{16}};$$

$$a_2 = \frac{1}{2!} \frac{d^2}{dz^2} [P(z)]_{z=\frac{17}{16}}; \dots, a_6 = \frac{1}{6!} \frac{d^6}{dz^6} [P(z)]_{z=\frac{17}{16}}$$

$\log_2 u$ и $\log_2 z$. Отыскание числа u осуществляется поразрядным умножением числа x и ячейки C_5 . Поместим числа $\log_2 u$ для $u = \frac{8}{16}, \frac{9}{16}, \frac{10}{16}, \dots, \frac{15}{16}$ в ячейках $A+7, A+8, \dots, A+14$.

Тогда отыскание $\log_2 u$ по заданному u сводится к выборке этого числа из соответствующей ячейки ЗУ. Номер искомой ячейки, очевидно, равен $A - 1 + u \cdot 2^4$. Вычисление $\log_2 z$ ведется по вышеизложенной программе. Число m находится выполнением команды \downarrow передачи порядка в число (см. § 23). В целях экономии операций в ячейках $A+7, A+8, \dots, A+14$ поместим числа $\log_2 \frac{8}{16} + \log_2 \frac{17}{16} = \log_2 \frac{8 \cdot 17}{256}, \dots, \log_2 \frac{15}{16} + \log_2 \frac{17}{16} = \log_2 \frac{15 \cdot 17}{256}$.

В соответствии с этим, программа вычисления $\log_2 x$ в общем случае имеет вид:

$K+1$	\downarrow	C_2		C_8
$K+2$	\wedge	C_2	C_5	C_9
$K+3$	\wedge	C_2	C_6	C_2
$K+4$	$:$	C_2	C_9	C_2
$K+5$	\rightarrow	C_9	C_7	C_9
$K+6$	$\dot{+}$	K_3	C_9	$K+14$
$K+7$	$+$			C_4
$K+8$	$\dot{+}$	K_1		$K+10$
$K+9$	$-$	C_2	C_1	C_2
$K+10$				
$K+11$	\times	C_4	C_2	C_4
$K+12$	$\dot{+}$	$K+10$	C_3	$K+10$
$K+13$	\geq	K_2	$K+10$	$K+10$
$K+14$				
$K+15$	$+$	C_4	C_8	C_4
$K+16$				

В ячейке C_9 получим
1 $\alpha_1 \alpha_2 \alpha_3$ 0000...

В ячейке C_2 получим
1 $\alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5$...

В ячейке C_9 получим

		...	1	α_1	α_2	α_3	
--	--	-----	---	------------	------------	------------	--

Команда $(K+6)$ образует номер ячейки, содержащей $\log_2 u$, и пересылает его в команду $(K+14)$

В ячейке C_4 получим $\log_2 x$

$A + 1$	a_6
$A + 2$	a_5
$A + 3$	a_4
$A + 4$	a_3
$A + 5$	a_2
$A + 6$	a_1
$A + 7$	$\log_2 \frac{8 \cdot 17}{256}$
$A + 8$	$\log_2 \frac{9 \cdot 17}{256}$
$A + 9$	$\log_2 \frac{10 \cdot 17}{256}$
$A + 10$	$\log_2 \frac{11 \cdot 17}{256}$
$A + 11$	$\log_2 \frac{12 \cdot 17}{256}$
$A + 12$	$\log_2 \frac{13 \cdot 17}{256}$
$A + 13$	$\log_2 \frac{14 \cdot 17}{256}$
$A + 14$	$\log_2 \frac{15 \cdot 17}{256}$
C_1	$17/16$
C_2	x
C_7	$2s - 4$
C_3	
C_5	11110...0
C_6	1 ... 1
K_1	+
K_2	+
K_3	+
	C_4
	$A + 1$
	$A + 6$
	$A - 1$
	C_4

s — число разрядов, отводимых под один адрес

Вычисление $\sin x$. Приведем программу вычисления $\sin x$ с точностью до δ . Составим сначала программу вычисления $\sin x$ для $0 \leq |x| \leq \frac{\pi}{2}$. Для этого воспользуемся рядами

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots \quad (6)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots \quad (7)$$

При $|x| \leq \frac{\pi}{4}$ будем пользоваться аргументом x и рядом (6), а для $|x| \geq \frac{\pi}{4}$ — аргументом $\left(\frac{\pi}{2} - |x|\right)$ и рядом (7). Последовательное вычисление членов u_n или u'_n ведется при помощи рекуррентного соотношения

$$u_{n+1} = - \frac{x^2}{(2n+3)(2n+2)} u_n$$

или

$$u'_{n+1} = - \frac{x^2}{(2n+2)(2n+1)} u'_n$$

Привлечение двух рядов вместо одного при таком способе вычисления не приведет к резкому удлинению программы, так как программы вычисления рядов (6) и (7) мало чем отличаются друг от друга. Полиномы $P(n) = -(2n+3)(2n+2)$ и $P'(n) = (2n+2)(2n+1)$ вычисляются последовательно с шагом 1 по разностям, а именно

$$\begin{aligned} P(n) &= P(n-1) + \Delta P(n-1) + \Delta^2 P \\ P'(n) &= P'(n-1) + \Delta P'(n-1) + \Delta^2 P' \end{aligned}$$

в качестве исходного берем $n=0$; при этом

$$\begin{aligned} P(-1) &= 0; \Delta P(-1) = +2; \Delta^2 P = -8 \\ P'(-1) &= 0; \Delta P'(-1) = +6; \Delta^2 P' = -8 \end{aligned}$$

являются начальными элементами, которые загружаются в ЗУ. Как видно, программы вычисления $\sin x$ и $\cos x$ отличаются величиной u_0 (для $\sin x$ $u_0 = x$, для $\cos x$ $u_0 = 1 \cdot \text{sign } x$) и величиной $\Delta P(-1)$ (для $\sin x$ $\Delta P(-1) = 2$, а для $\cos x$ $\Delta P'(-1) = 6$). Таким образом, в случае, когда $|x| \geq \frac{\pi}{4}$ и нужно считать по ряду (7), добавляются всего 4 команды: команда $(K+9)$ — условного перехода к команде $(K+13)$,

если $|x| \leq \frac{\pi}{4}$, команда $(K+10)$ — передачи 1 со знаком x в ячейку, содержащую u_0 , команда $(K+11)$ — о передаче в ячейку, содержащую $\Delta P(-1)$ числа 4, и команда $(K+12)$ — о переходе к аргументу $\frac{\pi}{2} - |x|$.

Остаточный член $R_n(x)$ оценивается, как обычно, для рядов со знакопередающимися и монотонно убывающими по абсолютной величине членами, величиной первого отбрасываемого члена u_n , т. е. сравнением по абсолютной величине u_n с заданной точностью δ .

Программа имеет следующий вид:

$K+8$		C_8		C_9	
$K+9$	\geq	C_5	C_9	$K+13$	
$K+10$:	C_8	C_9	C_8	В ячейке C_8 получим единицу со знаком числа x
$K+11$	+	C_4		C_{10}	Вначале C_{10} пусто
$K+12$	-	C_6	C_9	C_9	
$K+13$	+	C_3	C_{10}	C_{10}	
$K+14$	\times	C_9	C_9	C_9	
$K+15$	+	C_8	C_{12}	C_{12}	В ячейке C_{12} накапливаем ответ
$K+16$	+	C_2	C_{10}	C_{10}	
$K+17$	+	C_{10}	C_{11}	C_{11}	В ячейке C_{11} образуем $P_1(n)$, или $P'_1(n)$
$K+18$:	C_9	C_{11}	C_{13}	
$K+19$	\times	C_8	C_{13}	C_8	В ячейке C_8 образуем u_n
$K+20$	$ \geq $	C_8	C_7	$K+15$	
$K+21$					

C_2	-8
C_3	2
C_4	4
C_5	$\frac{\pi}{4}$
C_6	$\frac{\pi}{2}$
C_7	δ
C_8	x

Для произвольного x нужно предварительно выполнить приведение x к углу $\psi(x)$, для которого $|\psi(x)| < \frac{\pi}{2}$ и $\sin x = \sin [\psi(x)]$, т. е. найти $\psi(x)$ по формуле (см. § 17):

$$\psi(x) = \left\lfloor \left\{ \frac{x}{2\pi} - \frac{1}{4} \right\} 2\pi - \pi \right\rfloor - \frac{\pi}{2}$$

Программа отыскания $\psi(x)$ весьма проста.

$K+1$:	C_8	C_{15}	C_8	C_8	x
$K+2$	—	C_8	C_{16}	C_8	C_6	$\frac{\pi}{2}$
$K+3$	E	C_8	C_9	C_8	C_{14}	π
$K+4$	\times	C_8	C_{15}	C_8	C_{15}^*	2π
$K+5$	—	C_8	C_{14}	C_8	C_{16}	$\frac{1}{4}$
$K+6$		C_8		C_8		
$K+7$	—	C_8	C_6	C_8		

Вычисление $\Gamma(1+z)$ (при $R(z) > 0$ и $|z| < 1$) с точностью до δ .

Для вычисления $\Gamma(1+z)$ воспользуемся разложением

$$\text{в ряд } \Gamma(1+z) = \sum_{m=0}^{\infty} c_m z^m,$$

$$\text{где } c_0 = 1, \dots, c_{m+1} = \frac{\sum_{k=0}^m (-1)^{k+1} s_{k+1} c_{m-k}}{m+1} \quad (8)$$

$$s_n = \sum_{k=1}^{\infty} \frac{1}{k^n}, \quad n \geq 2$$

$s_1 = C$ — эйлерова постоянная.

Простоты ради, будем считать числа s_k для $k = 1, 2, \dots$ заданными в готовом виде в ЗУ (числа s_k можно найти в таблицах; программа для их вычислений весьма проста).

Коэффициенты c_m знакопереваются и убывают монотонно по абсолютной величине, поэтому остаточный член для

* См. Рыжик. Таблицы интегралов, сумм, рядов и произведений, стр. 290.

действительного положительного z меньше первого отбрасываемого члена. В общем случае $|R_n| \leq \frac{|c_n z^n|}{1 - |z|}$. Нужно

количество членов машина находит путем сравнения R_n с заданной точностью δ . Как только наступит такое n , что $|R_n| < \delta$, процесс прекращается. Коэффициенты c_1, c_2, \dots машина отыскивает по ходу вычислений и направляет их

в ячейки $D - 1, D - 2, \dots$. Сумма $\sum_{i=0}^n c_i z^i$ образуется по обыч-

ной схеме вычисления многочленов почленно.

Программа для вычисления $\Gamma(1 + z)$ имеет следующий вид:

$K + 16$		B_4		B_{15}	Команды $(K + 16)$ и $(K + 17)$ образуют $1 - z $ в ячейке B_{15}
$K + 17$	—	B_1	B_{15}	B_{15}	
$K + 18$	+			B_{10}	Команда гашения B_{10} нужна при переходе к образованию следующего коэффициента
$K + 19$	\times	$A + 1$	D	B_{11}	Команды $(K + 19)$ и $(K + 20)$ служат для образования слагаемого $(-1)^{k+1} s_{k+1} c_{m-k}$
$K + 20$	+	B_{11}	$D - 1$	$D - 1$	
$K + 21$	$\dot{+}$	$K + 19$	B_7	$K + 19$	Преобразуем команду $(K + 19)$ для перехода к следующему слагаемому $(-1)^{k+2} s_{k+2} c_{m-k-1}$
$K + 22$	$\dot{+}$	B_7	B_{10}	B_{10}	
$K + 23$	\geq	B_{12}	B_{10}	$K + 19$	Вначале B_{12} пуста, в ячейках B_{10} и B_{12} вырабатывается признак, когда кончить суммирование $\sum_{k=0}^m (-1)^{k+1} s_{k+1} c_{m-k}$
$K + 24$	+	B_1	B_{13}	B_{13}	Вначале B_{13} пуста, в ячейке B_{13} образуется $m + 1$
$K + 25$:	$D - 1$	B_{13}	$D - 1$	В ячейках $D, D - 1, \dots$ получаются коэффициенты c_0, c_1, c_2, \dots
$K + 26$	$\dot{+}$	B_7	B_{12}	B_{12}	Команды $(K + 26) - (K + 30)$ преобразуют команды $(K + 19), (K + 20), (K + 25)$ для перехода к образованию следующего коэффициента c_{m+2}

$K + 27$	\div	$K + 19$	B_6	$K + 19$
$K + 28$	\div	$K + 19$	B_{10}	$K + 19$
$K + 29$	\div	$K + 20$	B_8	$K + 20$
$K + 30$	\div	$K + 25$	B_9	$K + 25$
$K + 31$	\times	B_4	B_2	B_2
$K + 32$	\times	B_2	$D - 1$	B_{14}
$K + 33$	$+$	B_{14}	B_3	B_3
$K + 34$	\div	$K + 32$	B_6	$K + 32$
$K + 35$	$:$	B_{14}	B_{15}	B_{10}
$K + 36$	\geq	B_{10}	B_5	$K + 18$
$K + 37$				

В ячейке $K + 25$ получаются

$$\begin{array}{|c|c|c|c|} \hline : & D - 2 & B_{13} & D - 2 \\ \hline \end{array},$$

$$\begin{array}{|c|c|c|c|} \hline : & D - 3 & B_{13} & D - 3 \\ \hline \end{array} \text{ и т. д.}$$

В ячейке B_2 образуются последовательно z, z^2, \dots, z^n

В ячейке B_{14} образуются $c_{n+1}z^{n+1}$

В ячейке B_3 накапливается $\sum c_m z^m$

Команды $(K + 35)$ и $(K + 36)$ образуют и оценивают остаточный член: если он больше или равен δ , то переходим к образованию следующего члена

$$A + 1$$

$$A + 2$$

$$A + 3$$

$$A + 4$$

$$\vdots$$

$$\vdots$$

$$A + n$$

$$D$$

$$B_1$$

$$B_2$$

$$B_3$$

$$B_4$$

$$B_5$$

$$B_6$$

$$B_7$$

$$B_8$$

$$B_9$$

$$-s_1 = -C$$

$$s_2$$

$$-s_3$$

$$+s_4$$

$$(-1)^n \cdot s_n$$

$$1$$

$$1$$

$$1$$

$$1$$

$$z$$

$$\delta$$

		1	
1		1	
		1	1
1			1

Замечание 1. Коэффициенты c_m медленно убывают, так как $\frac{s_m}{m} < |c_m| < \frac{2s_m}{m}$, откуда следует медленная сходимость ряда для $|z|$, близких к 1. Поэтому для чисел z таких, что $|z| > \frac{1}{2}$ следует пользоваться формулой

$$\Gamma(z) = \frac{\pi}{\sin \pi z \Gamma(1-z)}$$

Замечание 2. Если $\Gamma(1+z)$ считается для серии значений $|z| \geq |z+h|, \dots, |z+lh|$, то начиная со второго значения коэффициенты c_m уже не вычисляются и приступают сразу к команде $(K+31)$. Соответственно этому в третий адрес команды $(K+36)$ следует поместить число $K+31$. Ответы $\Gamma(1+z)$, $\Gamma(1+h+z)$, ... помещаем в ячейках $E+1$, $E+2$, ..., что достигается командами $(K+37)$ и $(K+38)$. Для восстановления первоначального вида ячеек $K+32$, B_2 , B_3 и перехода к следующей точке служат команды $(K+39) - (K+45)$. Исправление команды $(K+36)$ нужно выполнить только один раз. Поэтому, после исправления команда $(K+45)$ переделывается командой $(K+46)$ в условную команду. Кроме того, нужно вновь образовать команду остановки после получения $\Gamma(1+lh+z)$. Это достигается командой $(K+47)$ — командой гашения ячейки $K+46$.

$K+37$	+	B_3		$E+1$
$K+38$	$\dot{+}$	$K+37$	B_{17}	$K+37$
$K+39$	$\dot{+}$	K_1		$K+32$
$K+40$	+	B_4	B_{16}	B_4
$K+41$		B_4		B_{15}
$K+42$	—	B_1	B_{15}	B_{15}
$K+43$	+	B_1		B_2
$K+44$	+	B_1		B_3
$K+45$	$\dot{+}$	K_2		$K+36$
$K+46$	$\dot{+}$	K_4		$K+45$
$K+47$	+			$K+46$
$K+48$	>			$K+31$

В ячейках $E+1$, $E+2$, ... получим значения $\Gamma(1+z)$, $\Gamma(1+h+z)$, ...

B_{16}		h		
B_{17}				1
K_1	\times	B_2	$D-1$	B_{14}
K_2	$ \geq $	B_{10}	B_5	$K+31$
K_3	$+$	B_3		$E+l$
K_4	\geq	K_3	$K+37$	$K+31$

Замечание 3. Мы предполагали числа $s_m = (-1)^m \sum_{k=1}^{\infty} \frac{1}{k^m}$ заданными в готовом виде в ЗУ. Эти числа можно вычислить предварительно на машине и направить их в ячейки ЗУ с теми же номерами, что и раньше. В связи с медленной сходимостью ряда $\sum_{k=1}^{\infty} \frac{1}{k^m}$ при малом m , составим программы вычисления s_m для $m \geq 5$. Поскольку ряд вычисляется для z с модулем $|z| \leq \frac{1}{2}$, числа s_m нужно определить только для $m < \log_2 \frac{1}{\delta}$. Потребуем, чтобы погрешность в определении $\Gamma(1+z)$ от неточности коэффициентов s_m для $m \geq 5$ не превосходила $\frac{\delta}{2}$ и соответственно этому будем вести вычисления в ряде (8) до такого i , что $\frac{|c_i z^i|}{1-|z|} < \frac{7\delta}{16}$. Очевидно, наше требование будет выполнено, если s_j для $j \geq 5$ считать с точностью до 8δ .

Приведенная ниже программа вычисления s_m для $m \geq 5$ с точностью до 8δ составлена с расчетом небольшой загрузки ЗУ.

Числа s_m для $m = 5, 6, \dots, E \log_2 \frac{1}{\delta}$ вычисляются одновременно путем последовательного накопления сумм обратных степеней

$$-\frac{1}{k^5}, +\frac{1}{k^6}, \dots, \left(-\frac{1}{k}\right)^{E \log_2 \frac{1}{\delta}}$$

в ячейках $A+5, A+6, \dots, A+E \log_2 \frac{1}{\delta}$ для $k = 1, 2, \dots$ (команды $(K+1) - (K+6)$).

Оценку n -го остаточного члена R_n^m суммы s_m производим по ходу вычислений по формуле, соответствующей интегральному признаку сходимости Коши:

$$R_n^m \leq \frac{1}{(m-1)n^{m-1}}$$

Сначала ведем оценку остаточного члена для $m = E \log_2 \frac{1}{\delta}$. Как только наступит такое n , что $R_n^m < 8\delta$, в ячейке $A + m$ больше не накапливаем; дальнейшее накопление сумм производится только в ячейках $A + 5, A + 6, \dots, A + m - 1$, оценка остаточного члена производится уже для s_{m-1} и т. д.

$K + 1$	+	K_7		$K + 8$	
$K + 2$	+	B_{18}	B_{19}	B_{19}	
$K + 3$:	B_1	B_{19}	B_{15}	Команды $(K + 3)$ и $(K + 4)$ образуют в ячейках B_{15} и B_{13} величину
$K + 4$	+	B_{15}		B_{13}	$-\frac{1}{k}$
$K + 5$	\times	B_{15}	B_{13}	B_{13}	Команда $(K + 5)$ образует в ячейке B_{13} последовательно:
					$\frac{1}{k^2}, -\frac{1}{k^3}, +\frac{1}{k^4}, \dots$
$K + 6$	+	$K + 8$	B_8	$K + 8$	Команда $(K + 6)$ вырабатывает номер ячейки, куда прибавить
					$-\frac{1}{k^5}, +\frac{1}{k^6}, \dots$
$K + 7$	\geq	K_5	$K + 8$	$K + 5$	Команда сравнения $(K + 7)$ позволяет избежать прибавления
					$\frac{1}{k^2}, -\frac{1}{k^3}, +\frac{1}{k^4}, \dots$
$K + 8$	+	B_{13}	$A + 1$	$A + 1$	в ячейках $A + 1, A + 2, A + 3, A + 4$; В ячейках $A + 5, A + 6, \dots$ накапливаем s_5, s_6, \dots
$K + 9$	\geq	K_6	$K + 8$	$K + 5$	$(K + 9)$ обрывает процесс образования $\frac{(-1)^i}{k^i}$ на m -м шаге
$K + 10$	\times	B_{19}	B_{13}	B_{13}	
$K + 11$:	B_{13}	B_{20}	B_{13}	Команды $(K + 10) - (K + 12)$ образуют и оценивают остаточный член для s_m, s_{m-1}, \dots
$K + 12$	\geq	B_{13}	B_{12}	$K + 1$	

$K + 13$	$-$	K_6	B_8	K_6
$K + 14$	$+$	B_{20}	B_{18}	B_{20}
$K + 15$	\geq	B_{19}	B_{22}	$K + 1$
$K + 16$				

Команда $(K + 13)$ уменьшает на 1 номер, когда обрывать процесс обр-зования $\frac{(-1)^i}{k^i}$

Команды $(K + 14)$ и $(K + 15)$ вырабаты-вают признак, когда кончить процесс вычисления s_i

B_{18}

B_{19}

B_{20}

B_{21}

B_{22}

B_1

B_8

K_5

K_6

K_7

		-1	
		0	
		$E \log_2 \frac{1}{\delta} - 1$	
		8δ	
		5	
		1	
		1	1
$+$	B_{13}	$A + 4$	$A + 4$
$+$	B_{13}	$A + E \log_2 \frac{1}{\delta}$	$A + E \log_2 \frac{1}{\delta}$
$+$	B_{13}	$A + 1$	$A + 1$

Вычисление функций разложением в непрерывные дроб-би. В § 17 приведены разложения функций e^x , $\text{th } x$, $\text{tg } x$, $\text{cos } x$ в непрерывные дроби. Для всех этих дробей подхо-дящие дроби $\frac{P_n}{Q_n}$ подсчитываются машиной наиболее просто по схеме Горнера для многочленов, если в последней заме-нить умножение делением. В качестве первого примера при-ведем программу вычисления $\text{tg } x$ для произвольного x с точностью до 10^{-10} при помощи непрерывной дроби

$$\text{tg } x = \frac{x}{y}, \text{ где } y = 1 - \frac{x^2}{3 - \frac{x^2}{5 - \frac{x^2}{7 - \dots}}}$$

Составим сначала программу для $|x| \leq \frac{\pi}{4}$. В этих пределах изменения x для получения $\operatorname{tg} x$ с точностью 10^{-10} достаточно непрерывную дробь y заменить подходящей дробью седьмого порядка

$$\frac{P_7}{Q_7} = y' = 1 - \frac{x^2}{3 - \frac{x^2}{5 - \frac{x^2}{7 - \frac{x^2}{9 - \frac{x^2}{11 - \frac{x^2}{13 - \frac{x^2}{15}}}}}}$$

По схеме Горнера y' запишется в виде:

$$y' = 1 - x^2 : (3 - x^2 : (5 - x^2 : (7 - x^2 : (9 - x^2 : (11 - x^2 : (13 - x^2 : 15))))))$$

Программа вычисления $\operatorname{tg} x$ по схеме Горнера записывается следующим образом:

$K+1$	\times	C_1	C_1	C_5	В ячейке C_5 получим x^2
$K+2$	$+$	C_2		C_6	" " C_6 накапливаем $\frac{P_7}{Q_7}$
$K+3$	$:$	C_5	C_6	C_6	
$K+4$	$-$	C_2	C_3	C_2	
$K+5$	$-$	C_2	C_6	C_6	
$K+6$	$>$	C_2	C_4	$K+3$	
$K+7$	$:$	C_1	C_6	C_6	В ячейке C_6 получим $\operatorname{tg} x$

C_1
 C_2
 C_3
 C_4

x
15
2
3

Перейдем к случаю произвольного x . Положим

$$x = n \frac{\pi}{4} + \varphi, \text{ где } |\varphi| < \frac{\pi}{4}$$

Обозначим через $\psi(x)$ разрывную функцию

$$\begin{aligned}\psi(x) &= \varphi, & \text{если } n &= 4k \\ \psi(x) &= \left(\frac{\pi}{4} - |\varphi|\right) \operatorname{sign} x, & \text{„ } n &= 4k + 1 \\ \psi(x) &= -\varphi, & \text{„ } n &= 4k + 2 \\ \psi(x) &= \left(|\varphi| - \frac{\pi}{4}\right) \operatorname{sign} x, & \text{„ } n &= 4k + 3\end{aligned}$$

$$\begin{aligned}\text{Очевидно } \operatorname{tg} x &= \operatorname{tg} \psi(x), & \text{„ } n &= 4k, n = 4k + 3 \\ \operatorname{tg} x &= \frac{1}{\operatorname{tg} \psi(x)}, & \text{„ } n &= 4k + 1, n = 4k + 2\end{aligned}$$

Используя операции $|x|$, $E(x)$ и $\{x\}$ так же, как и для $\sin x$, можно функцию $\psi(x)$ найти по формуле

$$\psi(x) = \left(\left\{ \frac{x}{\pi} - \frac{1}{4} \right\} - \frac{1}{2} \left| -\frac{1}{4} \right| \right) \pi$$

Далее легко видеть, что для $n = 4k$ или $n = 4k + 3$

$$\text{имеем } \left\{ \frac{x}{\pi} - \frac{1}{4} \right\} \geq \frac{1}{2}, \quad \text{а для } n = 4k + 1 \text{ или } n = 4k + 2$$

$$\text{имеем } \left\{ \frac{x}{\pi} - \frac{1}{4} \right\} \leq \frac{1}{2}.$$

Соответственно этому:

$$\begin{aligned}\text{если } \left\{ \frac{x}{\pi} - \frac{1}{4} \right\} &\geq \frac{1}{2}, \text{ то } & \operatorname{tg} x &= \operatorname{tg} \psi(x) = \frac{\psi(x)}{y'} \\ \text{„ } \left\{ \frac{x}{\pi} - \frac{1}{4} \right\} &\leq \frac{1}{2}, \text{ „ } & \operatorname{tg} x &= \frac{1}{\operatorname{tg} \psi(x)} = \frac{y'}{\psi(x)}\end{aligned}$$

Таким образом, в случае произвольного x последовательность вычислений такова:

а) находим $\psi(x)$ — команды $(K + 1) - (K + 7)$

б) „ $y' = \frac{P_7}{Q_7}$ — команды $(K + 8) - (K + 14)$

с) сравниваем $\left\{ \frac{x}{\pi} - \frac{1}{4} \right\}$ с числом $\frac{1}{2}$ и в зависимости от знака разности берем $\frac{\psi(x)}{y'}$ или $\frac{y'}{\psi(x)}$ — команды $(K + 15) - (K + 18)$.

Программа вычисления $\operatorname{tg} x$ в общем случае имеет вид:

$K+1$:	C_1	C_8	C_1
$K+2$	—	C_1	C_9	C_1
$K+3$	E	C_1	C_1	C_7
$K+4$	—	C_7	C_{10}	C_1
$K+5$		C_1		C_1
$K+6$	—	C_1	C_9	C_1
$K+7$	\times	C_1	C_8	C_1
$K+8$	\times	C_1	C_1	C_5
$K+9$	+	C_2		C_6
$K+10$:	C_5	C_6	C_6
$K+11$	—	C_2	C_3	C_2
$K+12$	—	C_2	C_6	C_6
$K+13$	\geq	C_2	C_4	$K+10$
$K+14$	\geq	C_7	C_{10}	$K+17$
$K+15$:	C_1	C_6	C_6
$K+16$				
$K+17$:	C_6	C_1	C_6
$K+18$				

В ячейке C_6 накапливаем $\frac{P_7}{Q_7} = y'$

В ячейке C_6 получим $\operatorname{tg} \psi(x)$

" " C_6 " $\frac{1}{\operatorname{tg} \psi(x)}$

C_1	x
C_2	15
C_3	2
C_4	3
C_8	π
C_9	$1/4$
C_{10}	$1/2$

В качестве 2-го примера приведем программу вычисления e^x с точностью 10^{-11} по ее разложению в непрерывную дробь (см. § 17).

Составим сначала программу вычисления e^x для $0 \leq |x| \leq 1$.

$$\text{Имеем: } e^x = \frac{y+x}{y-x}, \text{ где } y = 2 + \frac{x^2}{6 + \frac{x^2}{10 + \frac{x^2}{14 + \frac{x^2}{18 + \dots}}}}$$

Для получения e^x с заданной точностью, достаточно, как это указано в § 17, заменить y подходящей дробью 6-го

порядка $\frac{P_6}{Q_6} = 2 + \frac{x^2}{6 + \frac{x^2}{10 + \frac{x^2}{14 + \frac{x^2}{18 + \frac{x^2}{22}}}}$

Программа вычисления $\frac{P_6}{Q_6}$ для $0 \leq |x| \leq 1$ по схеме

Горнера записывается следующим образом:

$K+5$	\times	C_1	C_1	C_5	В ячейке C_5 получим x^2
$K+6$	$+$	C_{11}		C_6	
$K+7$	$+$	C_{11}		C_8	
$K+8$	$:$	C_5	C_6	C_6	
$K+9$	$-$	C_8	C_{12}	C_8	В ячейке C_6 накапливаем $\frac{P_6}{Q_6}$
$K+10$	$+$	C_8	C_6	C_6	
$K+11$	\geq	C_8	C_{13}	$K+8$	В ячейке C_5 получим $y+x$ " " C_6 " $y-x$ " " C_6 " e^x
$K+12$	$+$	C_1	C_6	C_5	
$K+13$	$-$	C_6	C_1	C_6	
$K+14$	$:$	C_5	C_6	C_6	

C_1	x
C_{11}	22
C_{12}	4
C_{13}	6

В случае произвольного x для вычисления e^x выгодно переходить от функции e^x к функции 2^y , полагая $e^x = 2^y$. Функцию 2^y достаточно уметь вычислять только для $|y| \leq 1$, поскольку числа в быстродействующей цифровой машине представлены в двоичной системе, и, следовательно, образование 2^n при целом n сводится к прибавлению n к порядку числа 1.

Положим $e^x = 2^y$, откуда $y = (\log_2 e)x$. Отыскание 2^y сводится к вычислению $2^{\{y\}}$ и умножению найденного числа на $2^{E(y)}$; вычисление $2^{\{y\}}$ осуществляется по предыдущей программе вычисления функции e^z для аргумента $z = \frac{\{y\}}{\log_2 e}$.

Таким образом, для вычисления e^x в общем случае надо выполнить предварительно команды $(K+1) - (K+4)$, образующие y , $\{y\}$, $E(y)$, $\frac{\{y\}}{\log_2 e}$ и $2^{E(y)}$

$K+1$	\times	C_1	C_{14}	C_7	
$K+2$	E	C_7	C_7	C_1	В ячейке C_7 получим $E(y)$
$K+3$	$:$	C_1	C_{14}	C_1	В ячейке C_1 получим $\{y\}$
$K+4$	\uparrow	C_7		C_7	В ячейке C_1 получим $\frac{\{y\}}{\log_2 e}$

где $(C_{14}) = \log_2 e$

затем перейти к программе вычисления $\frac{P_6}{Q_6}$, а в конце выполнить команду $(K+14)$ — команду умножения (C_7) на число (C_6)

$K+14$	\times	C_6	C_7	C_6
--------	----------	-------	-------	-------

§ 29. ПРЕОБРАЗОВАНИЕ АЛГЕБРАИЧЕСКИХ ВЫРАЖЕНИЙ

Пусть дано алгебраическое выражение $u(x_1, x_2, x_3, \dots, x_l) = \sum_{i=1}^n a_i x_1^{a_{1i}} x_2^{a_{2i}} \dots x_l^{a_{li}}$, в котором x_1, x_2, \dots, x_l рассматриваются как буквенные аргументы, а $a_i, a_{1i}, a_{2i}, \dots, a_{li}$ — как числа. Под преобразованием алгебраических выражений

мы понимаем операции тождественного преобразования выражений, связанные с изменением числовых параметров этих выражений, т. е. коэффициентов a_i и показателей $\alpha_{1i}, \alpha_{2i}, \dots, \alpha_{li}$. Остановимся сначала подробно на алгебраическом сложении. Простоты ради, ограничимся случаем двух переменных x_1, x_2 . Алгебраическое выражение $u(x_1, x_2)$, в котором $a_i, \alpha_{1i}, \alpha_{2i}$ рассматриваются как числа, а x_1, x_2 как буквенные аргументы, задается в ЗУ как совокупность одночленов $u_i(x_1, x_2) = a_i x_1^{\alpha_{1i}} x_2^{\alpha_{2i}}$. Одночлен u_i задается в ЗУ числами $a_i, \alpha_{1i}, \alpha_{2i}$. Аргументы x_1, x_2 и программа вычисления $u(x_1, x_2)$ в ЗУ не задаются. Таким образом, для каждого $u_i(x_1, x_2)$ отводится в ЗУ, вообще говоря, три ячейки; но если α_{1i}, α_{2i} суть натуральные числа, код одночлена u_i можно разместить в двух ячейках ЗУ: в одной поместить код коэффициента a_i , а в другой — коды показателей α_{1i}, α_{2i} в том или ином порядке в соответствующих группах t_1, t_2 разрядов этой ячейки.

Пусть, например, коды показателей

	α_{1i}	α_{2i}
t_1 разр.	t_2 разр.	

размещены последовательно в порядке α_{1i}, α_{2i} . Число $R_i = 2^{t_1} \alpha_{1i} + \alpha_{2i}$ назовем кодом показателей α_{1i}, α_{2i} . Здесь мы рассмотрим только случай, когда α_{1i}, α_{2i} суть натуральные числа. Операция сложения для алгебраической суммы

$$u(x_1, x_2) = \sum_{i=1}^n u_i \text{ сводится к приведению подобных членов.}$$

В результате приведения сумма $\sum_{i=1}^n u_i(x_1, x_2)$ перейдет в сум-

му $\sum_{j=1}^m v_j(x_1, x_2)$, где $v_j = b_j x_1^{\alpha_{1j}} x_2^{\alpha_{2j}}$, b_j — сумма всех коэф-

фициентов a_i , подобных между собой членов u_i с кодом показателей $T_j = 2^{t_1} \alpha_{1j} + \alpha_{2j}$. Для приведенного многочлена

$$v(x_1, x_2) = \sum_{j=1}^m v_j(x_1, x_2) \text{ символ } \sum \text{ не вносит в ЗУ никаких изменений.}$$

Наиболее просто приведение подобных производится в случае, когда числа t_1, t_2 достаточно малы, так что диапазон чисел R_i меньше диапазона свободных ячеек ЗУ. Допустим, что численный коэффициент a_i размещен в ячейке $A_{2i} = A + 2i$, а код показателей R_i размещен в разрядах последнего адреса ячейки $A_{2i-1} = A + 2i - 1$. Операция приведения сводится к тому, что для каждого члена $u_i, i = 1, 2, \dots, n$ код показателей R_i пересылаем из ячейки ЗУ A_{2i-1} в ячейку ЗУ с номером $B + 1 + 2R_i$, а коэффициент a_i , находящийся в ячейке A_{2i} , прибавляем к содержимому ячейки $B + 2 + 2R_i$. Константа B подбирается с таким расчетом, чтобы ячейки ЗУ $B + 1 + 2R_1, B + 2 + 2R_1, \dots, B + 1 + 2R_n, B + 2 + 2R_n$ были свободны. После n шагов для $i = 1, 2, \dots, n$ многочлен $u(x_1, x_2)$ будет приведен, причем в ячейках ЗУ $B + 1 + 2T_1, B + 1 + 2T_2, \dots, B + 1 + 2T_m$ будут расположены коды показателей T_1, T_2, \dots, T_m , а в ячейках $B + 2 + 2T_1, B + 2 + 2T_2, \dots, B + 2 + 2T_m$ будут находиться коды коэффициентов b_1, b_2, \dots, b_m . Заметим, что при таком способе приведения мы получим $v(x_1, x_2)$, упорядоченным по возрастанию кода показателей T_j , т. е. с возрастанием кода показателей T_j растут номера ячеек, носителей кода одночлена v_j .

Описанный способ приведения задается следующей программой (в примечаниях указаны результаты операций после однократного выполнения команд)

$K + 1 \rightarrow$	$A + 1$	C_1	C_5	$(C_5) =$				$2R_1$
$K + 2 \vdash$	K_1	C_5	$K + 6$	$(K + 6) =$	$+$	$A + 1$		$B + 1 + 2R_1$
$K + 3 \vdash$	$K + 6$	C_4	$K + 7$	$(K + 7) =$	$+$	$A + 2$	$B + 2$	$B + 2 + 2R_1$
$K + 4 \rightarrow$	C_5	C_2	C_5	C_5			$2R_1$	
$K + 5 \vdash$	$K + 7$	C_5	$K + 7$	$(K + 7) =$	$+$	$A + 2$	$B + 2 + 2R_1$	$B + 2 + 2R_1$
$K + 6$								
$K + 7$								
$K + 8 \vdash$	$K + 1$	C_3	$K + 1$	$(K + 1) =$	\rightarrow	$A + 3$	C_1	C_5
$K + 9 \vdash$	K_1	C_3	K_1	K_1	$+$	$A + 3$		$B + 1$
$K + 10 \geq$	K_2	K_1	$K + 1$					
$K + 11$								

C_1	- 1			
C_2	- s			
C_3		2		
C_4		1	$B + 2$	1
K_1	+	$A + 1$		$B + 1$
K_2	+	$A + 2n$		$B + 1$

s — число разрядов, отводимых для одного адреса ячейки
В ячейке C_3 находится 1 на $2s + 2$ разряде

Изложенный способ приведения обладает следующим недостатком: в случае, когда коды показателей получаются в процессе работы машины, нам известны только номера ячеек ЗУ, их содержащие, но не сами коды, и после приведения мы не будем знать номера ячеек, содержащих коды членов приведенного многочлена. Чтобы устранить этот недостаток, следует переслать приведенный многочлен обратно в пары $(A + 1, A + 2) \dots (A + 2m - 1, A + 2m)$. Пересылка, если известны границы R_{\min} и R_{\max} изменения кода показателей R_1, R_2, \dots, R_n , осуществляется последовательно сравнением по абсолютной величине чисел $(B + 2 + 2R_{\min})$, $(B + 4 + 2R_{\min}), \dots, (B + 2 + 2R_{\max})$ с нулем. Если содержимое ячейки ЗУ $B + 2 + 2R_{\min}$ отлично от нуля, то пересылаем его в ячейку A_2 , а содержимое ячейки $B + 1 + 2R_{\min}$ пересылаем в ячейку A_1 ; если же $(B + 2 + 2R_{\min}) = 0$, то переходим к сравнению по абсолютной величине содержимого следующей ячейки $(B + 4 + 2R_{\min})$ с нулем и т. д., пока не приходим к ячейке $B + 2 + 2R_{\max}$. Пересылая таким способом, мы, кроме того, оставляем после приведения только члены $v_j(x_1, x_2)$ с отличными от 0 коэффициентами b_j .

Программа пересылки записывается следующим образом:

$K+11$	$ > $		$B+2+2R_{\min}$	$K+18$
$K+12$	$\dot{+}$	K_3	C_7	K_3
$K+13$	$\dot{+}$	$K+11$	K_3	$K+16$

В ячейке $K+16$ получим

$\dot{+}$	$B+2+2R_{\min}$	$A+2$
-----------	-----------------	-------

$K+14$	$\dot{-}$	$K+16$	C_6	$K+17$	В ячейке $K+17$ получим
$K+15$	$+$	C_5	C_8	C_8	В ячейке C_8 получим число N членов с различными от 0 коэффициентами
$K+16$					
$K+17$					
$K+18$	$+$	$K+11$	C_3	$K+11$	
$K+19$	\geq	K_4	$K+11$	$K+11$	
$K+20$					

$+$	$B+1+2R_{\min}$	$A+1$
-----	-----------------	-------

C_5		1	
C_3		2	
C_6		1	1
C_7			2
K_3	$(+)-(\geq)$		$A-18-K$
K_4	\geq	$B+3+R_{\max}$	

В разрядах кода операций ячейки K_3 записано число, которое в сумме с кодом операции \geq дает в разрядах кода операций ячейки $(K+16)$ код операции сложения.

Замечание. Об одном способе упорядочения совокупности целых чисел. Выше отмечалось, что после приведения многочлен располагается по возрастающим степеням кода показателей, т. е. в лексикографическом порядке. Это обстоятельство можно использовать для упорядочения по величине совокупности целых чисел a_1, a_2, \dots, a_n , расположенных в ячейках ЗУ $M_1 = M + 1, M_2 = M + 2, \dots, M_n = M + n$. Для этого нужно число a_i для $i = 1, 2, \dots, n$ переслать из ячейки $M_i = M + i$ в ячейку ЗУ с номером $B + a_i$, как это делалось при приведении подобных, после чего уже упорядоченные по величине числа a_1, a_2, \dots, a_n послать обратно в ячейки M_1, M_2, \dots, M_n как в программе пересылки. Число B подбирается так, чтобы $B + a_i$ было положительным при любом i и соответствовало номерам свободных ячеек ЗУ. Числа $a_1, a_2, \dots, a_n, B + a_1, \dots$

$B + a_n$ предполагаются записанными в нормализованном виде $a_i = 2^{m_i} \cdot 0,1 \beta_1 \beta_2, \dots$, $B + a_i = 2^{l_i} \cdot 0,1 j_1 j_2 j_3 \dots$. Для передачи числа $B + a_i$ в 3-й адрес нужно перевести его в последние разряды ячейки C_6 . (Число разрядов, на которое нужно сдвинуть вправо число $B + a_i$, равно $3S - l_i$.) Поэтому здесь добавляются еще 3 команды $(K+1) - (K+3)$. Упорядочение таким простым способом возможно, когда диапазон чисел для a_1, a_2, \dots, a_n невелик; это имеет место, например, для совокупности показателей многочленов от многих переменных или при решении системы линейных уравнений (см. § 30). Упорядочение совокупности целых чисел записывается следующей программой:

$K+1$	+	C_1	$M+1$	C_6	C_0	2^{2S}			
$K+2$	+	C_6	C_0	C_6	C_1	B			
$K+3$	\wedge	C_6	C_5	C_6	C_2				
$K+4$	+	K_4	C_6	$K+5$	C_3				1
$K+5$					C_4	1			
$K+6$	+	K_4	C_3	K_4	C_5		1		
$K+7$	+	$K+1$	C_4	$K+1$	K_1				1...1
$K+8$	\geq	K_1	$K+1$	$K+1$	K_2	+	C_1	$M+n$	C_6
$K+9$	\geq		$B+a_{\min}$	$K+13$	K_3	$(+)-(\geq)$			$M-(k+13)$
$K+10$	+	K_2	C_2	K_2	K_4	\geq		$B+a_{\max}$	
$K+11$	+	$K+9$	K_2	$K+12$		+	$M+1$		
$K+12$									
$K+13$	+	$K+9$	C_3	$K+9$					
$K+14$	\geq	K_3	$K+9$	$K+9$					
$K+15$									

Контроль приведения подобных. Контроль приведения можно осуществить разными путями. Наиболее просто проконтролировать выполнение равенства $\sum_{i=1}^n a_i = \sum_{l=1}^k d_l$, где d_1, d_2, \dots, d_k — содержимое ячеек $M+2, M+4, \dots, M+2k$ после пересылки. Однако это равенство

не контролирует самого процесса приведения. Приведение подобных можно проконтролировать проверкой равенства

$$\sum_{i=1}^n R_i = \sum_{j=1}^m m_j T_j, \text{ где } m_j - \text{число подобных между собой член}$$

нов u_i с кодом показателей T_j , однако при этом не контролируется правильность вычисления коэффициентов b_j .

Поэтому, целесообразно объединить эти 2 контроля. Полный, хотя и громоздкий, контроль можно также осуществить проверкой равенства

$$\sum_{i=1}^n a_i R_i = \sum_{l=1}^k d_l \cdot S_l, \text{ где } S_1, S_2, \dots,$$

S_l — содержимое ячеек $M+1, M+3, \dots, M+2n-1$ после

обратной пересылки. При этом $\sum_{i=1}^n a_i R_i$ считается по ходу,

$$\text{а } \sum_{l=1}^k a_l S_l - \text{после приведения. Для машины с плавающей}$$

запятой эффективный контроль можно довольно просто получить проверкой равенства $u(x_1, x_2) = v(x_1, x_2)$ в точке $(x_1, x_2) = (2, 2)$.

Приведение подобных в общем случае.

Составим теперь программу приведения в общем случае,

не накладывая ограничений на величину t_1, t_2 . Как и раньше

будем считать, что коэффициенты a_i расположены в ячейках

ЗУ $A_2 = A + 2, A_4 = A + 4, \dots, A_{2n} = A + 2n$, а коды пока-

зателей R_i расположены в ячейках ЗУ $A_1 = A + 1, A_3 =$

$= A + 3, \dots, A_{2n-1} = A + 2n - 1$. Пусть сумма первых l

членов $\sum_{i=1}^l u_i$ уже приведена и размещена в парах ячеек ЗУ

$(A_1, A_2), (A_3, A_4), \dots, (A_{2p-1}, A_{2p})$. Приведение подобных

в сумме $\sum_{i=1}^l u_i + u_{l+1}$ осуществляется сравнением кода пока-

зателей R_{l+1} последовательно с содержимым ячеек ЗУ $A_1,$

A_3, \dots . Если найдется такое число $i \leq p$, что $A_{2i-1} = R_{l+1}$,

то прибавляем число a_{l+1} к содержимому ячейки ЗУ A_{2i}

и переходим к следующему члену u_{l+2} . Если же такого

$i \leq p$ не нашлось, то число a_{l+1} посылаем в ячейку ЗУ

с номером A_{2p+2} , а код показателей R_{l+1} — в ячейку ЗУ с номером A_{2p+1} и переходим к следующему члену u_{l+2} и т. д.

Переход к следующему члену u_{l+2} сводится к замене чисел A_{2l+1} , A_{2l+2} в некоторых из адресов соответствующих команд числами A_{2l+3} , A_{2l+4} . Кроме того, при переходе к следующему члену u_{l+2} нужно начать сравнения R_{l+2} с кодами показателей приведенных членов снова с ячейки с номером A_1 . Поэтому следует соответствующие адреса команд сравнения привести в начальное положение.

В нижеследующей программе приведения подобных команды $(K+1) - (K+4)$ служат для отыскания номеров пары A_{2l-1} , A_{2l} , с которой u_{l+1} подобен; команды $(K+5) - (K+13)$ служат для образования из команды $(K+3)$ кода команд $(K+14)$, $(K+15)$ — команд сложения коэффициентов и передачи кода показателей. Для экономии ЗУ мы направляем приведенные члены обратно в ячейки $A_1 = A + 1$, $A_2 = A + 2, \dots$. Если в сумме $\sum_{i=1}^l u_i$ подобных u_{l+1} нет,

то вместо сложения коэффициентов выполняем команду передачи коэффициента, что достигается командой сравнения $(K+11)$. Команды $(K+16) - (K+19)$ служат для перестройки команды $(K+3)$ и константы C_8 при переходе к следующему члену u_{l+2} .

Программа приведения в общем случае:

$K+1$	$\dot{+}$	$K+3$	C_6	$K+3$
$K+2$	$>$	$K+3$	K_1	$K+5$
$K+3$	$-$	$A+3$	$A-1$	C_{10}
$K+4$	$ \geq $	C_{10}	C_2	$K+1$
$K+5$	\wedge	$K+3$	C_3	C_{11}

Команда $(K+2)$ означает: если среди приведенных членов подобного u_{l+1} не оказалось, переходим к команде $(K+5)$; в противном случае переходим к команде $(K+3)$.

Команды $(K+3) - (K+4)$ сравнивают коды показателей R_{l+1} с содержимым ячейки $A + 2i - 1$.

В ячейке C_{11} получим

	$A + 2l + 1$		
--	--------------	--	--

$K+6$	\wedge	$K+3$	C_7	C_{12}
$K+7$	\rightarrow	C_{12}	C_1	$K+14$
$K+8$	$\dot{+}$	C_{11}	$K+14$	$K+14$
$K+9$	$\dot{+}$	C_5	$K+14$	$K+14$
$K+10$	$\dot{+}$	$K+14$	C_4	$K+15$
$K+11$	\geq	$K+3$	K_1	$K+14$
$K+12$	$\dot{+}$	$K+15$	C_9	$K+15$
$K+13$	$\dot{+}$	$K+15$	C_{12}	$K+15$
$K+14$				
$K+15$				
$K+16$	\geq	K_1	$K+3$	$K+18$
$K+17$	$\dot{+}$	$K+3$	C_9	K_1
$K+18$	$\dot{+}$	K_1	C_8	K_1
$K+19$	$\dot{+}$	C_{11}	K_3	$K+3$
$K+20$	\geq	K_2	$K+3$	$K+1$
$K+21$				

В ячейке C_{12} получим

		$A + 2i - 1$	
--	--	--------------	--

Команды $(K+7) - (K+9)$ служат для преобразования ячейки C_{12} в команду

$+$	$A + 2l + 1$	$A + 2i - 1$
-----	--------------	--------------

Команды $(K+10) - (K+13)$ служат для образования команды $(K+15)$

Команда $(K+15)$ имеет вид

$+$	$A + 2l + 2$	$A + 2i$	$A + 2i$
-----	--------------	----------	----------

если $i \leq p$

Команда $(K+15)$ имеет вид

$+$	$A + 2l + 2$	$A + 2i$
-----	--------------	----------

если $i > p$

Команды $(K+16) - (K+18)$ служат для образования K_1 при переходе к следующему члену u_{l+2} . Если u_{l+1} подобен какому-либо из членов приведенной суммы, то K_1 имеет вид

$-$	$A + 2l + 3$	$A + 2p + 1$	C_{10}
-----	--------------	--------------	----------

если же подобных u_{l+1} нет, то ячейка K_1 имеет вид

$-$	$A + 2l + 3$	$A + 2p + 3$	C_{10}
-----	--------------	--------------	----------

C_1	$ \begin{array}{c} s \\ 2^{-\delta} \\ 1 \dots 1 \\ 1 \\ 2 \\ 1 \dots 1 \end{array} $			
C_2				
C_3				
C_4				
C_5				
C_6				
C_7				

$2^{-\delta}$ — наименьшее положительное число, имеющее в данной машине отличное от нуля представление

C_8		2		
C_9			1	
K_1	—	$A + 3$	$A + 2$	C_{10}
K_2	—	$A + 2n - 1$	$A - 1$	C_{10}
K_3	—	2	$A - 1$	C_{10}

Сравнивая оба способа приведения подобных, мы видим, что в программе общего случая число операций значительно больше. Поэтому в случае, когда код показателей R_i уместается в пределах одного адреса, что, обычно, имеет место для многочлена одного или двух переменных, приведение подобных предпочтительнее делать первым способом. Приведением подобных решается ряд задач статистической обработки данных: „сортировка данных“, выборка из массива по группе признаков, определение взвешенного среднеарифметического, определение моды и т. д.

Так например, задача сортировки данных, или так называемая задача подсчета признаков, заключающаяся в том, чтобы найти частоты b_1, b_2, \dots, b_l наблюдений A_1, A_2, \dots, A_n , равных соответственно одному из возможных значений m_1, m_2, \dots, m_l , сводится к приведению подобных, если числа A_1, A_2, \dots, A_n рассматривать как коды показателей, а вес наблюдений, равный 1, как коэффициенты: $a_1 = a_2 = \dots = a_n = 1$.

Рассмотрим теперь кратко умножение алгебраических многочленов. Как и раньше, ограничимся случаем двух буквенных аргументов x_1, x_2 . Умножение двух многочленов

$$P(x_1, x_2) = \sum_{i=1}^n a_i x_1^{\alpha_{1i}} x_2^{\alpha_{2i}} \text{ и } Q(x_1, x_2) = \sum_{j=1}^m b_j x_1^{\beta_{1j}} x_2^{\beta_{2j}}$$

где x_1, x_2 рассматриваются как буквы, а $a_i, b_j, \alpha_{1i}, \alpha_{2i}, \beta_{1j}, \beta_{2j}$ — как числа, сводится к образованию многочлена

$$v(x_1, x_2) = \sum_{i=1, j=1}^{n, m} c_{ij} x_1^{\alpha_{1i} + \beta_{1j}} x_2^{\alpha_{2i} + \beta_{2j}} \text{ и приведению подобных}$$

членов внутри $v(x_1, x_2)$. Приведение подобных экономнее вести по ходу образования членов, отсылая их в ячейки

$$B + (\alpha_{1i} + \beta_{1j}) 2^i + (\alpha_{2i} + \beta_{2j}) \text{ и } B + 1 + (\alpha_{1i} + \beta_{1j}) 2^i + (\alpha_{2i} + \beta_{2j})$$

Последовательность умножений ведется по обычным правилам алгебры. Умножение l алгебраических многочленов

$\prod_{k=1}^l P_k(x_1, x_2)$ ведется по два: сначала находим $P_1 \cdot P_2$, затем $(P_1 \cdot P_2) P_3$ и т. д.

Умножение буквенных выражений может понадобиться, например, при преобразовании произведений сумм в ряды в общем виде.

Комбинируя операции алгебраического сложения и умножения, легко ввести в машину также и другие алгебраические действия над буквенными выражениями.

Операции дифференцирования и интегрирования над многочленами. Легко осуществимо на машине взятие точной производной и интеграла от алгебраических выражений. Пусть, например, требуется определить точную производную от многочлена $a_1 x^{m_1} + a_2 x^{m_2} + \dots + a_n x^{m_n}$. Как уже отмечалось выше, алгебраический многочлен задается в ЗУ совокупностью пар ячеек $(A+1, A+2)$, $(A+3, A+4)$, \dots , $(A+2n-1, A+2n)$.

Взять производную — значит, начиная с пары $(A+1, A+2)$, первую компоненту пары умножить на вторую и затем вычесть единицу из второй компоненты. Это записывается следующей программой.

$K+1$	\times	$A+1$	$A+2$	$A+1$
$K+2$	$-$	$A+2$	C_1	$A+2$
$K+3$	$+$	$K+1$	C_2	$K+1$
$K+4$	$+$	$K+2$	C_3	$K+2$
$K+5$	\geq	C_4	$K+1$	$K+1$
$K+6$				

C_1	1		
C_2	2	2	2
C_3	2		2
C_4	\times	$A+2n-1$	$A+2n-1$

Также определяется на машине частная производная от многочлена $u(x_1, x_2, \dots, x_j)$. Переменное, по которому производится дифференцирование, передается машине указанием номера ячейки, содержащей код показателя этого переменного, или указанием той или иной группы разрядов $t_1, t_1 + t_2, \dots$, если коды показателей размещены в одной ячейке ЗУ. Операция интегрирования многочленов в общем виде вводится аналогично, если $m_k \neq -1$.

С дифференцированием многочленов мы встречаемся, например, при составлении таблиц, когда нужно находить разложение функции $f(x)$ в ряд Тейлора последовательно в окрестности точек $a_0 + h, a_0 + 2h, \dots, a_0 + nh$.

Пусть $f(x)$ задана разложением в окрестности точки a_0

$$f(x) = f(a_0) + (x - a_0) \frac{f'(a_0)}{1} + \\ + \dots + \frac{(x - a_0)^m}{m!} f^m(a_0) \quad (1)$$

причем m настолько велико, что в пределах нужной точности можно отыскивать производные от $f(x)$ дифференцированием заданного многочлена (1). Разложить $f(x)$ в окрестности точки $a_0 + h$ значит найти первые m коэффициентов этого разложения

$$f(a_0 + h), \frac{f'(a_0 + h)}{1!}, \dots, \frac{f^m(a_0 + h)}{m!} = \frac{f^m(a_0)}{m!}$$

Приведем программу разложения в ряд Тейлора в окрестности точки $a_0 + h$. В этой программе команды $(K+1) - (K+3)$ служат для последовательного образования степеней h, h^2, \dots, h^m , команды $(K+4) - (K+7)$ — для последовательного вычисления $f(a_0 + h), \frac{f'(a_0 + h)}{1!}, \dots, \frac{f^m(a_0 + h)}{m!}$, команды $(K+8) - (K+11)$ — для выполнения операции дифференцирования, команды $(K+12) - (K+20)$ — для преоб-

разования соответствующих команд при переходе к вычислению следующего коэффициента.

$K + 1$	\times	C_1	D	$D + 1$	В ячейках $D, D + 1, \dots, D + m$ получим $1, h, h^2, \dots, h^m$				
$K + 2$	$\dot{+}$	$K + 1$	C_5	$K + 1$					
$K + 3$	\geq	K_2	$K + 1$	$K + 1$					
$K + 4$	\times	$A + 1$	$D + 1$	C_7					
$K + 5$	$+$	C_7	A	A	В ячейке A накапливаем сумму $f(a_0 + h)$				
$K + 6$	$\dot{+}$	$K + 4$	C_4	$K + 4$	В ячейке $A + 1$ накапливаем $f'(a_0 + h)$ и т. д.				
$K + 7$	\geq	C_3	$K + 4$	$K + 4$					
$K + 8$	$+$	D	C_8	C_8					
$K + 9$	\times	C_8	$A + 1$	$A + 1$					
$K + 10$	$\dot{+}$	$K + 9$	C_5	$K + 9$					
$K + 11$	\geq	K_1	$K + 9$	$K + 8$					
$K + 12$	$+$	C_0	C_8	C_8	Команды $(K + 12) - (K + 15)$ служат для перевода числа C_8 в адрес				
$K + 13$	\wedge	C_8	C_6	C_8	В ячейке C_8 получим				
					<table border="1"> <tr> <td></td> <td></td> <td>m</td> <td></td> </tr> </table>			m	
		m							
$K + 14$	\rightarrow	C_8	C_2	C_9	В ячейке C_9 получим				
					<table border="1"> <tr> <td></td> <td></td> <td></td> <td>m</td> </tr> </table>				m
			m						
$K + 15$	$\dot{+}$	C_9	C_8	C_9					
$K + 16$	\div	K_3	C_9	$K + 9$					
$K + 17$	\div	K_4	C_8	$K + 4$					
$K + 18$	$\dot{+}$	$K + 5$	C_5	$K + 5$					
$K + 19$	$+$			C_8					
$K + 20$	\geq	C_9	C_5	$K + 4$					
$K + 21$									

C_0			2^s	
C_1			h	
C_2			s	
C_3			2^{2s}	
D			1	
C_4		1	1	
C_5			1	1
C_6			1...1	
K_1	×	C_8	$A + m$	$A + m$
K_2	×	C_1	$D + m - 1$	$D + m$
K_3	×	C_8	$A + 2 + m$	$A + 2 + m$
K_4	×	$D + 1$	$A + 2 + m$	C_7
A			$f(a_0)$	
$A + 1$			$f'(a_0)$	
$A + 2$			$\frac{f''(a_0)}{2!}$	
.				
.				
.				
$A + m$			$\frac{f^{(m)}(a_0)}{m!}$	

s — число разрядов
одного адреса

§ 30. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ ПРЯМЫМИ МЕТОДАМИ

А. Метод главных элементов

Описание видоизмененного метода главных элементов

Ввиду того что метод главных элементов хорошо известен, ограничимся здесь описанием лишь измененного метода главных элементов и указанием отличий этого метода от первоначального метода.

Пусть имеем систему m совместных линейных алгебраических уравнений с m неизвестными:

$$\sum_{j=1}^m a_{ij} x_j = b_i \quad (i = 1, 2, \dots, m) \quad (1)$$

или в матричной форме:

$$A \mathbf{x} = \mathbf{b}$$

где $A = [a_{ij}]$ — квадратная матрица порядка m ;

\mathbf{x} — столбцовая матрица неизвестных;

\mathbf{b} — столбцовая матрица правых частей системы (1).

Так же, как и в обычном методе главных элементов, мы находим прежде всего в матрице A наибольший по модулю элемент a_{i_0, j_0} , называемый главным элементом матрицы A .

Далее из матриц A и \mathbf{b} образуем матрицу

$$C = [c_{ij}]_{(m, m+1)} = [a_{ij}, b_i] \quad (2)$$

и производим вычисление матрицы

$$C^{(1)} = [c_{ij}^{(1)}]_{(m, m+1)}$$

вычисляя сначала элементы i_0 -й строки (главной строки) по формуле:

$$c_{i_0 j}^{(1)} = \frac{c_{i_0 j}}{c_{i_0 j_0}}, \text{ где } j \neq j_0 \text{ и } c_{i_0 j_0} = a_{i_0 j_0} \quad (3)$$

а затем, вычисляя остальные элементы этой матрицы для всех ее столбцов, за исключением j_0 -го столбца (главного столбца), по формуле:

$$c_{ij}^{(1)} = c_{ij} - c_{ij_0} \cdot c_{i_0 j}^{(1)} \quad (4)$$

где $i \neq i_0, j \neq j_0$; j_0 -й столбец этой матрицы остается тем же, что и в исходной матрице C .

Матрицу, полученную из матрицы A путем описанного преобразования, обозначим символом $A^{(1)}$. Среди всех элементов этой матрицы, за исключением элементов, находящихся в i_0 -й строке и j_0 -м столбце, находим максимальный по модулю элемент $a_{i_k j_k}^{(1)}$, который называем главным элементом матрицы $A^{(1)}$.

Вообще элементы матрицы $C^{(k+1)}$ вычисляются из элементов матрицы $C^{(k)}$ посредством следующего процесса. Сначала находим максимальный по модулю элемент $a_{i_k j_k}^{(k)}$

матрицы $A^{(k)}$, не считая элементов главных строк i_0, i_1, \dots, i_{k-1} и главных столбцов j_0, j_1, \dots, j_{k-1} предыдущих A -матриц.

Затем вычисляем все элементы i_k -й строки, за исключением элементов столбцов j_0, j_1, \dots, j_k матрицы $C^{(k+1)}$ по формуле:

$$c_{i_k j}^{(k+1)} = \frac{c_{i_k j}^{(k)}}{c_{i_k j_k}} \quad (3')$$

Остальные элементы $c_{ij}^{(k+1)}$ матрицы $C^{(k+1)}$, за исключением элементов главных столбцов этой матрицы, вычисляем по формуле:

$$c_{ij}^{(k+1)} = c_{ij}^{(k)} - c_{ij_k}^{(k)} \cdot c_{i_k j}^{(k+1)} \quad (4')$$

где $i \neq i_k, j \neq j_0, j \neq j_1, \dots, j \neq j_{k-1}$; элементы столбцов j_0, j_1, \dots, j_k матрицы $C^{(k+1)}$ остаются теми же, что и в матрице $C^{(k)}$.

Повторяя такой процесс m раз, получаем матрицу, последний столбец которой, т. е. $(m+1)$ -й столбец, содержит значения неизвестных x_1, \dots, x_m нашей системы. Элементы всех остальных столбцов матрицы $C^{(m)}$, т. е. $A^{(m)}$ -матрицы, остаются теми же, что и в матрице $C^{(m-1)}$.

Итак, окончательная матрица, т. е. матрица $C^{(m)}$, содержащая решение системы уравнений, имеет следующее строение: последний, т. е. $(m+1)$ -й столбец содержит значения неизвестных x_1, \dots, x_m , расположенных в какой-то последовательности.

Субматрица $A^{(m)}$ этой матрицы состоит из главных столбцов всех предыдущих A -матриц: j_k -й столбец матрицы $A^{(m)}$ совпадает с j_k -м столбцом матрицы $A^{(k-1)}$, где $k=0, 1, \dots, m-1$.

Если мы решали бы систему уравнений (1) обычным методом главных элементов, то получили бы матрицу, последний столбец которой содержал бы также решение этой системы, но матрица $A^{(m)}$ имела бы при этом другой вид. А именно, на тех местах, где в предыдущих A -матрицах:

$$A, A^{(1)}, \dots, A^{(m-1)}$$

находились главные элементы, в матрице $A^{(m)}$ стояли бы единицы, а все остальные ее элементы были бы нулями, т. е. в результате применения обычного метода главных элементов мы привели бы систему (1) к следующему виду:

$$x_{j_i} = c_{i, m+1}^{(m)} \quad (5)$$

где $i = 1, 2, \dots, m$;

j_i — некоторая взаимнооднозначная функция от номера строки i , определяемая заданием „координат“ $(i_0 j_0), (i_1 j_1), \dots, (i_k j_k), \dots, (i_{m-1} j_{m-1})$ главных элементов всех предыдущих A -матриц.

В силу взаимной однозначности соответствия j_i , система (5) эквивалентна следующей системе:

$$x_j = c_{i_j, m+1}^{(m)} \quad (6)$$

где $j = 1, 2, \dots, m$;

i_j — взаимнооднозначная функция от номера столбца j , т. е. функция, обратная функции j_i .

Таким образом, при решении системы (1) обычным методом главных элементов в последнем столбце матрицы $C^{(m)}$, т. е. в $(m+1)$ -м ее столбце, значения неизвестных расположены в последовательности x_{j_i} , т. е. в той же последовательности, в какой будут расположены номера главных столбцов j_k , если соответствующие им номера главных строк i_k расположены в натуральной последовательности. Как видно из формулы (6), элементы $c_{i, m+1}^{(m)}$ последнего столбца матрицы $C^{(m)}$ необходимо переупорядочить в последовательности $c_{i_j, m+1}^{(m)}$, т. е. расположить в $(m+1)$ -м столбце сверху вниз в такой последовательности, в какой будут расположены номера главных строк i_k , если соответствующие им номера главных столбцов j_k расположены в натуральной последовательности.

Сказанное справедливо и для видоизмененного метода главных элементов, так как и для него справедливы, очевидно, равенства (5) и (6); отличие видоизмененного ме-

тогда от обычного метода главных элементов заключается лишь в том, что при этом методе главные столбцы не вычисляются, а остаются без всякого изменения. Это, кстати сказать, почти в два раза уменьшает число необходимых для решения системы уравнений вычислений, если не считать операций, необходимых для нахождения главных элементов. В отношении этих операций описанный метод не отличается от обычного метода.

Описание программы решения систем совместных линейных алгебраических уравнений

Программа составлена для одновременного решения k систем уравнений $Ax = b^{(r)}$, $r = 1, 2, \dots, k$, различающихся лишь правыми частями $b^{(r)} = (b_1^{(r)}, \dots, b_m^{(r)})$. Исходной матрицей будет матрица $[c_{ij}]_{(m,n)}$ с m строками и n столбцами, где $c_{ij} = a_{ij}$ при $j \leq m$, $c_{i,m+r} = b_i^{(r)}$.

Предполагается, что главный элемент $a_{i_0 j_0}$ исходной A -матрицы известен и известны также его „координаты“ i_0 и j_0 .

Ячейки ЗУ, в которых в начальный момент находятся элементы c_{ij} исходной матрицы $[c_{ij}]_{(mn)}$, где m — число строк, а n — число столбцов этой матрицы, мы будем обозначать символами z_{ij} , т. е. в начальный момент будем иметь равенства: $(z_{ij}) = c_{ij}$. Матрицу $[(z_{ij})]_{(mn)}$ будем считать „развертываемой“ по столбцам, т. е. будем считать, что

$$z_{ij} = z_{1,1} + i - 1 + (j - 1)m \quad (7)$$

Как видно из этой формулы, для того чтобы перейти от ячейки ЗУ z_{ij} к ячейке ЗУ $z_{i+1,j}$, т. е. опуститься в j -м столбце на одну строку, надо число z_{ij} увеличить на единицу, т. е.

$$z_{i+1,j} = z_{ij} + 1$$

Если же требуется передвинуться на один элемент по одной и той же i -й строке, т. е. перейти от номера z_{ij} к номеру $z_{i,j+1}$, то нужно, как видно из равенства (7), прибавить к z_{ij} не единицу, а m единиц, т. е.

$$z_{i,j+1} = z_{ij} + m$$

Содержание ячеек в начальный момент:

$$\begin{aligned}
 (x) &= (x+1) = \dots = (x+m) = m \\
 (y) &= (y+1) = \dots = (y+n) = n \\
 (a_0) &= \varepsilon; & (a_1) &= 1; & (a_2) &= m-1; & (a_3) &= n-1 \\
 (a_4) &= 0; & (a_5) &= 0; & (a_6) &= 0; & (a_7) &= z_{i_0 j_0} \\
 (a_8) &= i_0 - 1; & (a_9) &= (a_8 + 1) = m; & (a_{10}) &= j_0 - 1; & (a_{11}) &= 2^s \\
 (a_{12}) &= 2s; & (a_{13}) &= m \cdot 2^{2s}
 \end{aligned}$$

$$(a_{14}) = \boxed{\geq \mid a_5 \mid y \mid K + 55}, \quad (a_{15}) = \boxed{\geq \mid a_4 \mid a_8 \mid K + 41}$$

$$(a_{16}) = \boxed{\dot{+} \mid a_8 \mid \mid x}, \quad (a_{17}) = \boxed{\dot{+} \mid a_{16} \mid \mid y}$$

$$(a_{18}) = \boxed{: \mid z_{11} \mid a_7 \mid b}, \quad (a_{19}) = \boxed{\dot{+} \mid b \mid \mid z_{11}}$$

$$(a_{20}) = \boxed{\times \mid z_{11} \mid b \mid c}, \quad (a_{21}) = \boxed{\dot{+} \mid z_{11} \mid \mid e}$$

$$(a_{22}) = \boxed{\dot{+} \mid d \mid \mid z_{11}}, \quad (a_{23}) = \boxed{\geq \mid a_4 \mid x \mid K + 52}$$

$$(a_{24}) = 0,$$

$$(a_{27}) = \boxed{\dot{+} \mid a_{10} \mid \mid u}$$

Содержание остальных ячеек ЗУ в начальный момент безразлично.

Приведенная программа содержит 63 команды, из которых 16 служат для отыскания главного элемента $c_{i_k j_k}^{(k)}$, его „координат“ i_k и j_k и для упорядочения найденных номеров главных строк и столбцов; 9 команд используются для осуществления контроля получаемых результатов и две команды — для упорядочения индексов найденных неизвестных.

Блок-схема. В дальнейшем через k мы обозначим переменный индекс вычисляемой матрицы $C^{(k)}$, через i, j — переменные индексы элемента $c_{ij}^{(k)}$ этой матрицы; $i_k j_k$ — „координаты“ главного элемента матрицы $C^{(k)}$. Для формирования меняющихся команд здесь применяется метод, описанный на стр. 177. Соответственно, ряд этапов программы формирует меняющиеся команды, зависящие от различных индексов.

Блок-схема состоит из 6 больших этапов, соединенных отдельными командами условного перехода. Команды $(K+1) — (K+13)$ этапа I упорядочивают номера главных столбцов и строк и формируют команды, зависящие от их номеров.

Команда $(K+14)$ служит для пропуска главного j_{k-1} столбца матрицы $C^{(k-1)}$, который не меняется при переходе к матрице $C^{(k)}$.

Команды II этапа $(K+15) — (K+27)$ осуществляют формирование команд, зависящих от текущего номера j столбца, и вычисление элементов $c_{i_{k-1}j}^{(k)}$ главной строки.

После этого команда $(K+28)$ осуществляет пропуск этой уже вычисленной строки. Далее следуют команды $(K+29) — (K+36)$ этапа III, формирующие команды, зависящие от текущего индекса строки и осуществляющие вычисление всех остальных элементов $c_{ij}^{(k)}$ матрицы $C^{(k)}$.

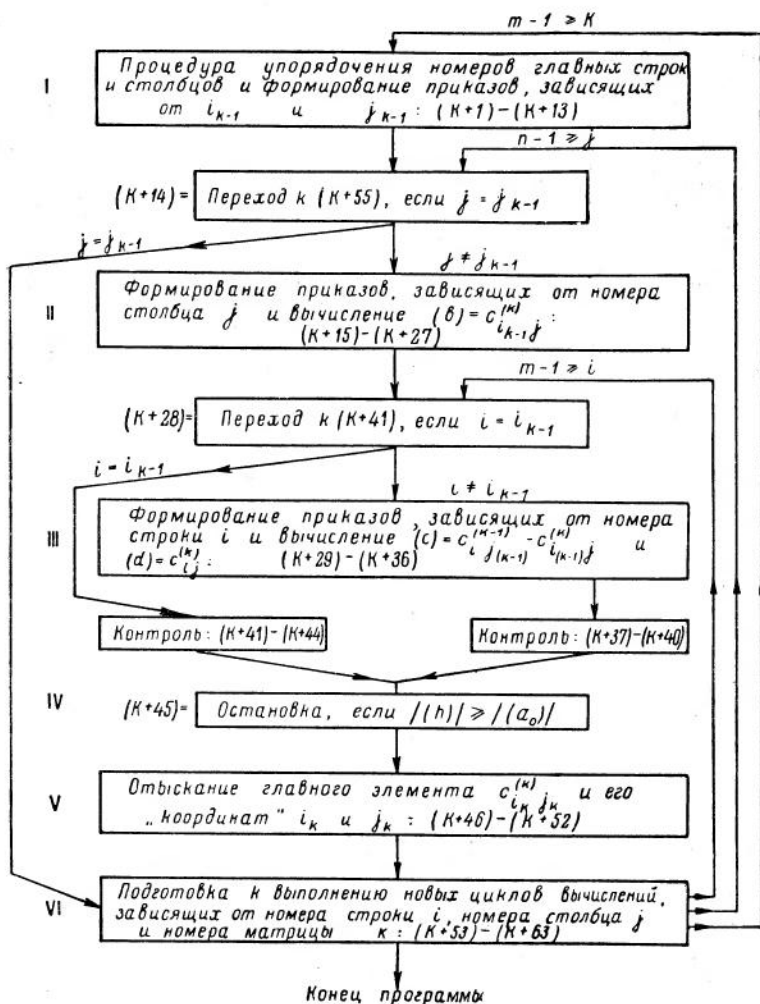
IV этап — команды $(K+37) — (K+45)$ осуществляют контроль при вычислении каждого элемента. Если обнаруживается ошибка, превышающая установленный допуск ϵ , машина останавливается.

V этап — команды $(K+46) — (K+52)$ осуществляют отыскание главного элемента $c_{i_k j_k}^{(k)}$ матрицы $C^{(k)}$ и его координат $i_k j_k$.

Последний этап — команды $(K+53) — (K+63)$ служат для подготовки программы к вычислению элементов следующей строки, следующего столбца и следующей матрицы.

В программе имеются 11 пустых вначале мест (ячейки $K+2, K+4, K+6, K+14, K+19, K+20, K+28, K+31, K+33, K+36$ и $K+47$). Начальные значения — „зародыши“ этих команд хранятся в ячейках $a_{14}, a_{15}, \dots, a_{23}, a_{27}$.

Блок-схема программы решения системы совместных алгебраических уравнений методом главных элементов



Образование вычислительных команд. Для вычисления элемента матрицы

$$c_{ij}^{(k)} = c_{ij}^{(k-1)} - c_{ij_{k-1}}^{(k-1)} \cdot \left(\frac{c_{i_{k-1}j}^{(k-1)}}{a_{i_{k-1}j_{k-1}}^{(k-1)}} \right)$$

где $i \neq i_k$, требуется выполнить три операции: деление, умножение и вычитание. В программе образуются три команды, осуществляющие указанные арифметические операции:

$$(K + 19) = \left[\begin{array}{|c|c|c|c|} \hline : & z_{i_{k-1}j} & a_7 & b \\ \hline \end{array} \right], \quad \text{где } (a_7) = z_{i_{k-1}j_{k-1}}$$

$$(K + 31) = \left[\begin{array}{|c|c|c|c|} \hline \times & z_{ij_{k-1}} & b & c \\ \hline \end{array} \right]$$

$$(K + 34) = \left[\begin{array}{|c|c|c|c|} \hline - & e & c & d \\ \hline \end{array} \right], \quad \text{где } (e) = c_{ij}^{(k-1)}$$

В результате выполнения этих команд получим равенства:

$$(b) = \frac{c_{i_{k-1}j}^{(k-1)}}{c_{i_{k-1}j_{k-1}}^{(k-1)}} = c_{i_{k-1}j}^{(k)}, \quad (c) = c_{ij_{k-1}}^{(k-1)} \cdot c_{i_{k-1}j}^{(k)}, \quad (d) = c_{ij}^{(k)},$$

где $i \neq i_{k-1}$.

Вычисленные элементы $c_{i_{k-1}j}^{(k)}$, $c_{ij}^{(k)}$ отправляются в ячейки $z_{i_{k-1}j}$, z_{ij} , соответственно, что осуществляется командами:

$$(K + 20) = \left[\begin{array}{|c|c|c|c|} \hline + & b & & z_{i_{k-1}j} \\ \hline \end{array} \right]$$

$$(K + 36) = \left[\begin{array}{|c|c|c|c|} \hline + & d & & z_{ij} \\ \hline \end{array} \right]$$

Сохранение элементов $c_{i_{k-1}j}^{(k)}$, $c_{ij}^{(k)}$ в постоянных ячейках b и d позволяет осуществить подпрограмму контроля из команд с постоянными адресами.

Образование изменяющихся команд. Из приведенных выше трех вычислительных команд первые две $(K + 19)$ и $(K + 31)$ содержат адреса, зависящие от номера строки i , номера столбца j и „координат“ i_{k-1} , j_{k-1} главного элемента матрицы $C^{(k-1)}$.

Эти команды должны быть сформированы к моменту их выполнения из их „зародышей“.

Команда $(K + 19)$ формируется следующим образом. В результате выполнения команды

$$(K + 8) = \left[\begin{array}{|c|c|c|c|} \hline \leftarrow & a_8 & a_{12} & K + 2 \\ \hline \end{array} \right]$$

число $(a_8) = i_{k-1} - 1$ сдвигается на $2s$ двоичных разряда влево и полученное таким образом число $2^{2s} (i_{k-1} - 1)$ помещается в ячейку ЗУ $K + 2^*$. Следующая команда

$$(K + 9) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & a_{18} & 2 & a_{29} \\ \hline \end{array} \right]$$

из „зародыша“ команды $(K + 19)$

$$(a_{18}) = \left[\begin{array}{|c|c|c|c|} \hline : & z_{11} & a_7 & b \\ \hline \end{array} \right]$$

и числа $(2) = (i_{k-1} - 1) 2^{2s}$, образует „заготовку“

$$(a_{29}) = \left[\begin{array}{|c|c|c|c|} \hline : & z_{i_{k-1}1} & a_7 & b \\ \hline \end{array} \right]$$

окончательной „рабочей“ команды $(K + 19)$.

Для получения этой команды необходимо преобразовать первый адрес $z_{i_{k-1}1}$ полученной „заготовки“ в адрес $z_{i_{k-1}j}$; это осуществляется посредством команд $(K + 15)$ и $(K + 18)$:

$$(K + 15) = \left[\begin{array}{|c|c|c|c|} \hline \times & a_5 & a_{13} & K + 2 \\ \hline \end{array} \right]$$

$$(K + 18) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & a_{29} & K + 2 & K + 19 \\ \hline \end{array} \right]$$

Команда $(K + 15)$ образует число $(K + 2) = (j - 1) m 2^{2s}$. Команда $(K + 18)$, прибавляя это число к „заготовке“ (a_{29}) команды $(K + 19)$, придает последней ее окончательный вид:

$$(K + 19) = \left[\begin{array}{|c|c|c|c|} \hline : & z_{i_{k-1}j} & a_7 & b \\ \hline \end{array} \right]$$

Команда $(K + 20)$, отправляющая элемент $c_{i_{k-1}j}^{(k)}$ из ячейки b в $z_{i_{k-1}j}$, формируется из ее „зародыша“

$$(a_{19}) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & b & & z_{11} \\ \hline \end{array} \right]$$

посредством следующих трех команд:

$$(K + 7) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & a_{19} & a_8 & a_{30} \\ \hline \end{array} \right]$$

* Команда $(K + 2)$ к этому моменту уже выполнена.

$$(K+16) = \begin{array}{|c|c|c|c|} \hline \rightarrow & K+2 & a_{12} & K+4 \\ \hline \end{array}$$

$$(K+17) = \begin{array}{|c|c|c|c|} \hline \dot{+} & a_{30} & K+4 & K+20 \\ \hline \end{array}$$

Команда $(K+7)$ превращает этот зародыш в „полуфабрикат“ команды $(K+20)$:

$$(a_{30}) = \begin{array}{|c|c|c|c|} \hline + & b & & z_{i_{k-1}1} \\ \hline \end{array}$$

отличающейся от окончательной команды лишь тем, что в ее третьем адресе указан номер $z_{i_{k-1}1}$ вместо требуемого номера $z_{i_{k-1}j}$.

Команда $(K+16)$, сдвигая число $(j-1)m2^{2s}$ на $2s$ разрядов вправо, образует число $(j-1)m$, помещает его в ячейку ЗУ $K+4$, вместо уже выполненной к этому времени команды $(K+4)$.

Команда $(K+17)$, прибавляя это число к „полуфабрикату“ (a_{30}) команды $(K+20)$, придает последней ее окончательный вид:

$$(K+20) = \begin{array}{|c|c|c|c|} \hline \dot{+} & b & & z_{i_{k-1}j} \\ \hline \end{array}$$

Аналогично, вторая вычислительная команда $(K+31)$ формируется из ее „зародыша“.

$$(a_{20}) = \begin{array}{|c|c|c|c|} \hline \times & z_{11} & b & c \\ \hline \end{array}$$

посредством следующих четырех команд:

$$(K+10) = \begin{array}{|c|c|c|c|} \hline \times & a_{10} & a_{13} & K+4 \\ \hline \end{array}$$

$$(K+11) = \begin{array}{|c|c|c|c|} \hline \dot{+} & K+4 & a_{20} & a_{31} \\ \hline \end{array}$$

$$(K+29) = \begin{array}{|c|c|c|c|} \hline \leftarrow & a_4 & a_{12} & K+2 \\ \hline \end{array}$$

$$(K+30) = \begin{array}{|c|c|c|c|} \hline \dot{+} & a_{31} & K+2 & K+31 \\ \hline \end{array}$$

В результате выполнения команды $(K+10)$ в ячейке $K+4$ оказывается произведение чисел $(a_{10}) = (j_{k-1} - 1)$ и $(a_{13}) = m2^{2s}$. Команда $(K+11)$ прибавляет число $(K+4) = (j_{k-1} - 1) m2^{2s}$ к „зародышу“ (a_{20}) команды $(K+31)$ и, тем самым, образует „полуфабрикат“ команды $(K+31)$:

$$(a_{31}) = \left[\begin{array}{|c|c|c|} \hline \times & z_{1,j_{k-1}} & b \quad c \\ \hline \end{array} \right]$$

Сдвигая число $(a_4) = i - 1$ на $2s$ двоичных разряда влево, команда $(K+29)$ образует число $(i - 1) 2^{2s}$ и помещает это число в ячейке $3U (K+2)$.

Складывая это число с „полуфабрикатом“ (a_{31}) команды $(K+31)$, команда $(K+30)$ преобразует этот „полуфабрикат“ в „рабочую“ команду:

$$(K+31) = \left[\begin{array}{|c|c|c|} \hline \times & z_{ij_{k-1}} & b \quad c \\ \hline \end{array} \right]$$

Полученное в результате выполнения команды $(K+31)$ произведение $c_{ij_{k-1}}^{(k-1)} \cdot c_{i_{k-1}j}^{(k)} = (c)$ вычитается затем из числа $(e) = c_{ij}^{(k-1)}$. Команда $(K+34)$ помещает разность $(e) - (c) = c_{ij}^{(k)}$ в промежуточную ячейку $3U d$. Но для осуществления этой команды надо до ее выполнения поместить в ячейку $3U e$ элемент $c_{ij}^{(k-1)}$. Это осуществляется посредством команды $(K+33)$.

Формирование команды $(K+33)$ из ее „зародыша“

$$(a_{21}) = \left[\begin{array}{|c|c|c|} \hline + & z_{11} & e \\ \hline \end{array} \right]$$

осуществляется посредством следующих команд:

$$(K+24) = \left[\begin{array}{|c|c|c|} \hline + & a_{21} & K+2 \quad a_{33} \\ \hline \end{array} \right]$$

$$(K+32) = \left[\begin{array}{|c|c|c|} \hline + & a_{33} & K+2 \quad K+33 \\ \hline \end{array} \right]$$

Команда $(K+24)$, прибавляя число $(K+2) = (j - 1) m 2^{2s}$, образованное командой $(K+15)$, к „зародышу“ (a_{21}) команды $(K+33)$, образует „полуфабрикат“

$$(a_{33}) = \left[\begin{array}{|c|c|c|} \hline + & z_{1j} & e \\ \hline \end{array} \right]$$

„рабочей“ команды $(K+33)$.

Команда $(K+32)$, прибавляя число $(K+2) = (i-1)2^{2s}$, образованное командой $(K+29)$, к этому полуфабрикату, формирует „рабочую“ команду

$$(K+33) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & z_{ij} & & e \\ \hline \end{array} \right]$$

Полученный в результате выполнения команд $(K+33)$ и $(K+34)$ элемент $(d) = c_{ij}^{(k)}$ должен быть отослан в ячейку $3U z_{ij}$, что осуществляется посредством команды $(K+36)$.

Формирование команды $(K+36)$ из ее „зародыша“

$$(a_{22}) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & d & & z_{11} \\ \hline \end{array} \right]$$

осуществляется посредством команд

$$(K+22) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & a_{22} & K+4 & a_{36} \\ \hline \end{array} \right], (K+35) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & a_{36} & a_4 & K+36 \\ \hline \end{array} \right]$$

Прибавляя к „зародышу“ (a_{22}) команды $(K+36)$ число $(K+4) = (j-1)m$, команда $(K+22)$ образует „полуфабрикат“

$$(a_{36}) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & d & & z_{1j} \\ \hline \end{array} \right]$$

команды $(K+36)$. Из этого „полуфабриката“ команда $(K+35)$ образует путем сложения его с числом $(a_4) = i-1$ окончательную или „рабочую“ команду

$$(K+36) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & a & & z_{ij} \\ \hline \end{array} \right]$$

Процедура упорядочения номеров главных строк и столбцов. Каждый раз, когда номер столбца j совпадает с одним из номеров главных столбцов всех ранее вычисленных A -матриц, должна пропускаться та часть нашей программы, которая осуществляет вычисление элемента $c_{ij}^{(k)}$, его контроль и его использование для нахождения нового главного элемента, т. е. все команды, начиная с $(K+15)$ и кончая $(K+54)$.

Пропуск этой части программы осуществляется посредством команды $(K+14)$ следующим образом. Перед началом выполнения части $(K+14) - (K+57)$ нашей программы, зависящей от номера столбца j , выполняются подготовительные команды $(K+12)$ и $(K+13)$.

В результате выполнения команды

$$(K+12) = \left[\begin{array}{|c|c|c|c|} \hline + & & & a_5 \\ \hline \end{array} \right]$$

ячейка ЗУ a_5 , выполняющая функции фиксации числа j уже вычисленных столбцов матрицы $C^{(k)}$, „гасится“, т. е. приводится в начальное состояние $(a_5) = 0$.

В результате выполнения команды $(K+13)$ в ячейку $(K+14)$ программы помещается команда

$$(a_{14}) = \left[\begin{array}{|c|c|c|c|} \hline \geq & a_5 & y & K+55 \\ \hline \end{array} \right]$$

После выполнения команды $(K+13)$ команда $(K+14)$ принимает свой начальный вид:

$$(K+14) = \left[\begin{array}{|c|c|c|c|} \hline \geq & a_5 & y & K+55 \\ \hline \end{array} \right]$$

Команда $(K+14)$ в этом виде является командой условного перехода к команде

$$(K+55) = \left[\begin{array}{|c|c|c|c|} \hline + & K+14 & a_{11} & K+14 \\ \hline \end{array} \right]$$

в случае, если $(a_5) \geq (y)$. В противном случае, т. е. если $(a_5) < (y)$, будет выполнена непосредственно следующая за командой $(K+14)$ команда $(K+15)$.

Переход к выполнению команды $(K+55)$ может иметь место при выполнении команды $(K+14)$ в первый раз (т. е. сразу после выполнения команды $(K+13)$) лишь в том случае, если в числе главных столбцов ранее вычисленных матриц имелся столбец с номером $j=1$. В этом, и только в этом случае возможно осуществление условия

$$(a_5) \geq (y)$$

Действительно, если среди главных столбцов ранее вычисленных матриц имелся столбец с номером $j=1$, то в ячейке ЗУ y будет находиться уже не начальное значение n , а значение 0. Как осуществляется эта замена, будет объяснено ниже.

С другой стороны, $(a_5)=0$ (в результате выполнения команды $(K+12)$). Поэтому выполняется неравенство

$(a_5)=0 \geq 0=(y)$, являющееся условием перехода к выполнению команды $(K+55)$.

В противном случае будет сохраняться первоначальное равенство $(y)=n$; будет справедливо противоположное неравенство $(a_5)=0 < n=(y)$, и, стало быть, будет производиться вычисление элементов первого столбца матрицы $C^{(k)}$.

После того как все вычисления, относящиеся к 1-й строке, будут закончены, выполняется команда $(K+55)$, осуществляющая преобразование первоначального вида команды $(K+14)$ в команду

$$(K+14) = \left[\begin{array}{c|c|c|c} \geq & a_5 & y+1 & K+55 \end{array} \right]$$

Такое преобразование совершается каждый раз при переходе от вычисления элементов $(j-1)$ -го столбца к вычислению элементов j -го столбца и независимо от того, производились ли действительно эти вычисления, или они пропускались. Условием пропуска j -го столбца является выполнение неравенства

$$(a_5) \geq (y+j-1)$$

Это условие выполняется в случае, если $(y+j-1)=j-1$, а это равенство обеспечивается подпрограммой упорядочения номеров главных столбцов только в том случае, если в числе главных столбцов ранее вычисленных матриц есть столбец с номером j .

В противном случае будет сохраняться первоначальное равенство $(y+j-1)=n$, и приведенное условие перехода к команде $(K+55)$ не может быть выполнено, так как $(a_5)=j-1 < n$.

Преобразование команды $(K+14)$, осуществляемое посредством команды $(K+55)$, обеспечивает пропуск тех и только тех столбцов матрицы $C^{(k)}$, номера которых совпадают с номерами главных столбцов ранее вычисленных матриц.

Содержимое ячейки a_5 увеличивается на единицу каждый раз, когда происходит переход от j -го столбца к $(j-1)$ -му. Это осуществляется посредством команды

$$(K+56) = \left[\begin{array}{c|c|c|c} + & a_1 & a_5 & a_5 \end{array} \right]$$

Всякий раз после выполнения этой команды выполняется команда

$$(K+57) = \left[\begin{array}{|c|c|c|c|} \hline \geq & a_3 & a_5 & K+14 \\ \hline \end{array} \right]$$

которая осуществляет возвращение процесса к началу подпрограммы, зависящей от номера столбца j , т. е. к команде, $(K+14)$, в случае, когда

$$(a_3) = n - 1 \geq (a_5) = j$$

Когда j достигает максимального своего значения $j = n$, это неравенство, вызывающее переход к выполнению команды $(K+14)$, уже не удовлетворяется, и машина переходит к выполнению заключительных команд $(K+58) - (K+63)$.

Упорядочение номеров главных столбцов, в порядке их возрастания осуществляется посредством первых двух команд программы.

Первая команда

$$(K+1) = \left[\begin{array}{|c|c|c|c|} \hline + & a_{10} & a_{17} & K+2 \\ \hline \end{array} \right]$$

служит для формирования команды

$$(K+2) = \left[\begin{array}{|c|c|c|c|} \hline + & a_{10} & & y + j_{k-1} - 1 \\ \hline \end{array} \right]$$

которая собственно и осуществляет упорядочение номеров главных столбцов. В результате выполнения команды $(K+2)$ получается равенство

$$(y + j_{k-1} - 1) = j_{k-1} - 1$$

число $j_{k-1} - 1$, хранящееся при вычислении матрицы $C^{(k)}$ в ячейке ЗУ a_{10} , передается в ячейку ЗУ $j_{k-1} - 1$.

Содержимое ячеек $y, y+1, \dots, y+m$ не гасится и сохраняется неизменным в процессе вычислений. Поэтому по выполнении первых двух команд в k -й раз (т. е. при вычислении матрицы $C^{(k)}$) будут иметь место равенства:

$(y + j_0 - 1) = j_0 - 1, (y + j_1 - 1) = j_1 - 1, \dots, (y + j_{k-1} - 1) = j_{k-1} - 1$
а при $j \neq j_0, j_1, \dots, j_{k-1}$ сохраняется равенство

$$(y + j) = n$$

Таким образом, все номера (или, точнее, числа на единицу меньшие этих номеров) главных столбцов ранее вычисленных матриц оказываются упорядоченными в порядке их возрастания. При этом они находятся в ячейках, номера которых отличаются от номеров главных столбцов на одинаковое положительное число $y-1$. Во всех y -х ячейках ЗУ, не занятых номерами главных элементов, сохраняется их начальное содержимое — число n^* .

Следующие две команды программы, команды $(K+3)$ и $(K+4)$, аналогичным образом осуществляют упорядочение номеров главных строк всех уже вычисленных матриц.

Пропуск строк в подпрограмме для нахождения главного элемента осуществляется аналогично пропуску столбцов.

Команда

$$(K+23) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & a_{23} & & K+47 \\ \hline \end{array} \right]$$

аналогична команде $(K+13)$. В результате выполнения команды $(K+23)$ в ячейку $K+47$ помещается команда

$$(a_{23}) = \left[\begin{array}{|c|c|c|c|} \hline \geq & a_4 & x & K+52 \\ \hline \end{array} \right]$$

и команда $(K+47)$ принимает свой начальный вид

$$(K+47) = \left[\begin{array}{|c|c|c|c|} \hline \geq & a_4 & x & K+52 \\ \hline \end{array} \right]$$

Нахождение главного элемента. С выполнения команды $(K+47)$ начинается выполнение подпрограммы нахождения главного элемента матрицы $C^{(k)}$, содержащей кроме этой команды еще 5 команд: $(K+48) - (K+52)$.

Последняя команда этой подпрограммы

$$(K+52) = \left[\begin{array}{|c|c|c|c|} \hline \dot{+} & K+47 & a_{11} & K+47 \\ \hline \end{array} \right]$$

преобразует команду $(K+47)$ каждый раз при переходе от строки i к строке $i+1$ аналогично тому, как команда $(K+55)$ преобразует команду $(K+14)$ при переходе от столбца j к столбцу $j+1$.

* Можно было бы число n заменить любым большим числом.

Собственно подпрограмма отыскания главного элемента состоит из следующих четырех команд:

$$\begin{aligned}
 (K+48) &= \begin{array}{|c|c|c|c|} \hline \geq & a_{24} & d & K+52 \\ \hline \end{array} \\
 (K+49) &= \begin{array}{|c|c|c|c|} \hline + & d & & a_{24} \\ \hline \end{array} \\
 (K+50) &= \begin{array}{|c|c|c|c|} \hline + & a_4 & & a_{25} \\ \hline \end{array} \\
 (K+51) &= \begin{array}{|c|c|c|c|} \hline + & a_5 & & a_{26} \\ \hline \end{array}
 \end{aligned}$$

Первая из этих команд осуществляет сравнение абсолютных величин чисел, содержащихся в ячейках ЗУ a_{24} и d . Если $|(a_{24})| \geq |(d)| = |c_{ij}^{(k)}|$, то команды $(K+49)$, $(K+50)$ и $(K+51)$ пропускаются, и сразу выполняется команда $(K+52)$, преобразующая описанную таким образом команду $(K+47)$. Если же $|(a_{24})| < |(d)|$, то после команды $(K+48)$ будет выполнена команда $(K+49)$, в результате чего в ячейку a_{24} будет передано число $(d) = c_{ij}^{(k)}$. Оно будет сохраняться в ячейке до тех пор, пока в ячейке a_{24} не окажется число, большее по абсолютной величине этого числа. Тогда, в результате выполнения команд $(K+48)$ и $(K+49)$ сохранявшийся до сих пор в ячейке ЗУ a_{24} элемент $c_{ij}^{(k)}$ будет заменен большим элементом той же матрицы $C^{(k)}$.

Так как „развертка“ матрицы производится по столбцам, то находящийся в ячейке элемент $c_{ij}^{(k)}$ заменяется ближайшим к нему и большим по абсолютной величине элементом последовательности

$$c_{11}^{(k)}; c_{21}^{(k)}; \dots; c_{m1}^{(k)}; c_{12}^{(k)}; c_{22}^{(k)}; \dots; c_{ij}^{(k)}; c_{i+1j}^{(k)}; \dots; c_{i_{k-1}j}^{(k)}; \dots$$

В результате выполнения команды $(K+50)$ в ячейку a_{25} передается из ячейки a_4 номер строки i , в которой находится элемент $c_{ij}^{(k)}$, поступивший в ячейку a_{24} в результате выполнения команды $(K+49)$. Аналогично, команда $(K+51)$ служит для передачи в ячейку a_{26} из ячейки a_5 номера столбца j , в которой находится элемент $c_{ij}^{(k)}$, только что поступивший в ячейку a_{24} .

Таким образом команды $(K + 50)$ и $(K + 51)$ фиксируют в ячейках a_{25} и a_{26} „координаты“ i, j элемента $c_{ij}^{(k)}$, поступившего в ячейку a_{24} в результате выполнения команды $(K + 49)$.

После выполнения этой подпрограммы, т. е. после вычисления всех элементов матрицы $C^{(k)}$ в ячейках a_{24}, a_{25} и a_{26} , будут зафиксированы, соответственно, главный элемент $c_{i_k j_k}^{(k)}$ и его „координаты“ i_k и j_k .

Команда $(K + 52)$ подготавливает команду $(K + 47)$ для выполнения ее в следующий раз с новым значением ее второго адреса, а команда

$$(K + 53) = \begin{array}{|c|c|c|c|} \hline \dot{+} & a_1 & a_4 & a_4 \\ \hline \end{array}$$

увеличивает на единицу находившееся ранее в ячейке a_4 число $i - 1$, т. е. после выполнения команды $(K + 53)$ имеет место равенство $(a_4) = i$.

Следующая далее команда

$$(K + 54) = \begin{array}{|c|c|c|c|} \hline \geq & a_2 & a_4 & K + 28 \\ \hline \end{array}$$

является командой условного перехода к выполнению команды $(K + 28)$, если $(a_2) \geq (a_4)$, т. е. если $i \leq m$. Если же $(a_2) < (a_4)$, то после команды $(K + 54)$ выполняется следующая команда $(K + 55)$, функции которой мы уже рассмотрели.

Команда $(K + 54)$ осуществляет возвращение процесса к началу подпрограммы $(K + 28) - (K + 54)$, зависящей от индекса строки i , до тех пор, пока не будут вычислены и проверены все элементы данного столбца и не будет найден в нем максимальный элемент.

Подпрограмма $(K + 28) - (K + 54)$ является основной частью всей программы, так как содержит подпрограмму вычисления всех элементов $c_{ij}^{(k)}$ матрицы $C^{(k)}$ (за исключением элементов главной строки), подпрограмму контроля этих элементов и подпрограмму отыскания главного элемента вычисляемой матрицы.

В ячейке ЗУ a_{15} постоянно хранится зародыш команды $(K + 28)$. Посредством команды

$$(K + 21) = \begin{array}{|c|c|c|c|} \hline \dot{+} & a_{15} & & K + 28 \\ \hline \end{array}$$

этот зародыш передается в ячейку ЗУ $K + 28$. Таким образом, перед началом вычисления элементов каждого столбца команда $(K + 21)$ восстанавливает первоначальную форму команды $(K + 28)$:

$$(K + 28) = \begin{array}{|c|c|c|c|} \hline \geq & a_4 & a_8 & K + 41 \\ \hline \end{array}$$

Команда $(K + 28)$ является командой условного перехода к выполнению команды

$$(K + 41) = \begin{array}{|c|c|c|c|} \hline \dot{+} & K + 28 & a_{11} & K + 41 \\ \hline \end{array}$$

в случае, когда

$$(a_4) = i - 1 \geq i_k - 1 = (a_8)$$

Команда $(K + 41)$ осуществляет сложение команды $(K + 28)$ указанного вида с числом $(a_{11}) = 2^s$, в результате чего происходит прибавление 1 ко второму адресу команды $(K + 28)$:

$$a_8 + 1 = a_9$$

и команда $(K + 28)$ принимает вид:

$$(K + 28) = \begin{array}{|c|c|c|c|} \hline \geq & a_4 & a_9 & K + 41 \\ \hline \end{array}$$

Так как в ячейке a_9 постоянно хранится число заведомо большее всякого значения, которое может находиться в ячейке ЗУ a_4 , служащей для хранения текущего номера строки $((a_4) = i - 1)$, то, стало быть, никогда не может осуществиться условие $(a_4) \geq (a_9)$ и, следовательно, из всех строк вычисляемой матрицы будет пропущена только та строка, номер которой i совпадает с номером i_{k-1} главной строки матрицы $C^{(k-1)}$, над элементами которой производятся в данное время вычисления.

Необходимость пропуска i_{k-1} строки при выполнении подпрограммы $(K + 28) - (K + 54)$ следует из того, что элементы этой строки уже вычислены ранее.

Контроль. В описываемой здесь подпрограмме использован один из вариантов контроля посредством „суммирования по столбцам“.

Так как элемент $c_{ij}^{(k)}$ матрицы $C^{(k)}$ вычисляется посредством формул (4) и (3), то, если все вычисления совершаются точно, выполняется следующее равенство:

$$\sum_{\alpha=1}^i c_{\alpha j}^{(k)} = \sum_{\alpha=1}^i c_{\alpha j}^{(k-1)} - \sum_{\alpha=1}^i c_{\alpha j_{k-1}}^{(k-1)} \cdot c_{i_{k-1}j}^{(k)} + c_{i_{k-1}j}^{(k)} \cdot 1(i) \quad (8)$$

где:

$$1(i) = \begin{cases} 0, & \text{если } i < i_{k-1} \\ 1, & \text{если } i \geq i_{k-1} \end{cases}$$

а штрихи над знаками сумм в правой части означают, что в суммах отсутствуют слагаемые с $\alpha = i_{k-1}$.

В действительности это равенство выполняется лишь приближенно. Если абсолютная величина разности его правой и левой части не превышает некоторой заданной величины ε (допуск), то можно считать, что все вычисления проделаны удовлетворительно и машина должна производить вычисления следующего элемента матрицы $C^{(k)}$. В противном случае машина должна остановиться.

В случае, когда $i < i_k$, выполняются следующие приказы подпрограммы контроля: $(K+37)$, $(K+38)$, $(K+39)$, $(K+40)$, $(K+41)$, $(K+49)$.

В результате выполнения команды

$$(K+37) = \begin{array}{|c|c|c|c|} \hline + & e & f & f \\ \hline \end{array}$$

к содержимому ячейки f прибавляется число $(e) = c_{ij}^{(k-1)}$. Перед началом выполнения подпрограммы $(K+23) - (K+54)$ ячейка f очищается, т. е. ее содержимое гасится, посредством команды

$$(K+26) = \begin{array}{|c|c|c|c|} \hline + & & & f \\ \hline \end{array}$$

В результате выполнения команды

$$(K+38) = \begin{array}{|c|c|c|c|} \hline - & f & c & f \\ \hline \end{array}$$

из содержимого ячейки f вычитается $(c) = c_{\alpha j_{k-1}}^{(k-1)} \cdot c_{i_{k-1}j}^{(k)}$ и разность $(f) - (c)$ отправляется в ячейку f . Таким образом, после i -кратного выполнения команд $(K+37)$, $(K+38)$

в ячейке ЗУ f будет находиться число

$$\sum_{\alpha=1}^i c_{\alpha j}^{(k-1)} - \sum_{\alpha=1}^i c_{\alpha j_{k-1}}^{(k-1)} \cdot c_{i_{k-1} j}^{(k)}$$

Команда $(K+39) =$

+	d	g	g
---	-----	-----	-----

прибавляет только что вычисленный элемент $(d) = c_{ij}^{(k)}$ к ранее накопленной в ячейке g сумме элементов j -го столбца вычисляемой матрицы $C^{(k)}$. В результате i -кратного выполнения этой команды получается равенство $(g) = \sum_{\alpha=1}^i c_{\alpha j}^{(k)}$ в случае, если перед началом выполнения подпрограммы $(K+28) - (K+54)$ ячейка g была пуста.

Гашение ячейки g осуществляется каждый раз перед началом выполнения этой подпрограммы посредством команды

$$(K+27) =$$

+			g
---	--	--	-----

Далее команда

$$(K+40) =$$

\geq			$K+44$
--------	--	--	--------

безусловного перехода к команде $(K+44)$ осуществляет обход трех команд: $(K+41)$, $(K+42)$ и $(K+43)$, выполнение которых необходимо лишь при $i = i_{k-1}$.

В результате выполнения команды

$$(K+44) =$$

-	g	f	h
---	-----	-----	-----

в ячейке ЗУ h получаем разность $(g) - (f)$, т. е. разность правой и левой частей приближенного равенства (8). Эта разность далее сравнивается по модулю в процессе выполнения приказа

$$(K+45) =$$

$ \geq $	h	a_0	ξ
----------	-----	-------	-------

с числом ϵ , хранящимся в ячейке ЗУ a_0 .

Если модуль этой разности не меньше модуля числа ϵ , то машина останавливается, так как в третьем адресе команды $(K+45)$ указана ячейка, в которой находится

команда остановки машины. Если во внутреннем ЗУ машины нет ячейки с номером 0 и если машина построена так, что отсылка к несуществующему в ней адресу влечет остановку машины, то машина будет останавливаться и в том случае, если третий адрес команды ($K + 45$) будет пуст, т. е. если команда ($K + 45$) будет иметь вид:

$$\left| \begin{array}{|c|c|c|c|} \hline | \gg | & h & a_0 & \\ \hline \end{array} \right|$$

Если же модуль этой разности не достигает значения $|\varepsilon|$, выполняется команда

$$(K + 46) = \left| \begin{array}{|c|c|c|c|} \hline > & a_5 & a_9 & K + 53 \\ \hline \end{array} \right|$$

Эта команда осуществляет тем самым пропуск или выключение программы отыскания максимального $c_{ij}^{(k)}$ в случаях, когда $j \geq m + 1$.

В случае, когда $i = i_{k-1}$, т. е. когда в процессе вычисления элементов данного столбца j машина дошла до элемента главной строки, будет выполнено условие $(a_4) \geq (a_8)$ и команда

$$(K + 28) = \left| \begin{array}{|c|c|c|c|} \hline > & a_4 & a_8 & K + 41 \\ \hline \end{array} \right|$$

передает сразу управление уже описанной выше команде ($K + 41$).

После того, как команда ($K + 41$) будет выполнена, будут выполнены последовательно команды:

$$(K + 42) = \left| \begin{array}{|c|c|c|c|} \hline + & b & g & g \\ \hline \end{array} \right|$$

$$(K + 43) = \left| \begin{array}{|c|c|c|c|} \hline + & b & f & f \\ \hline \end{array} \right|$$

которые раньше, при $i < i_{k-1}$ пропускались.

Команды ($K + 42$) и ($K + 43$) добавляют элемент $c_{i_{k-1}j}^{(k)} = (b)$ к ранее находившимся в ячейках f и g суммам. В результате выполнения этих команд ячейки ЗУ f и g будут содержать следующие числа:

$$(f) = \sum_{\alpha=1}^i c_{\alpha j}^{(k-1)} - \sum_{\alpha=1}^i c_{\alpha j_{k-1}}^{(k-1)} \cdot c_{i_{k-1}j}^{(k)} + c_{i_{k-1}j}^{(k)}$$

где

$$i < i_{k-1}$$

$$(g) = \sum_{\alpha=1}^{i_{k-1}} c_{\alpha j}^{(k)}$$

После выполнения этих двух команд, выполняется описанная уже часть подпрограммы контроля (команды $(K+44)$ и $(K+45)$), общая для обоих случаев.

При $i > i_{k-1}$ будет выполняться та же часть подпрограммы контроля, что и при $i < i_{k-1}$. В самом деле, при $i = i_{k-1}$ команда $(K+41)$ преобразовала команду $(K+28)$ к виду

$$(K+28) = \left[\begin{array}{c|c|c|c} \geq & a_4 & a_9 & K+41 \end{array} \right]$$

При этом, как было выяснено выше, условие $(a_4) \geq (a_9)$ перехода к команде $(K+41)$ не может быть выполнено ни при каком значении $i \leq m$.

При $m \geq i > i_{k-1}$ ячейки ЗУ f и g будут, очевидно, иметь следующее содержание

$$(f) = \sum_{\alpha=1}^i c_{\alpha j}^{(k-1)} - \sum_{\alpha=1}^i c_{\alpha j_{k-1}}^{(k-1)} \cdot c_{i_{k-1}j}^{(k)} + c_{i_{k-1}j}^{(k)}$$

$$(g) = \sum_{\alpha=1}^i c_{\alpha j}^{(k)}$$

где штрихи при знаках сумм имеют прежнее значение.

Как только i превысит m , неравенство $(a_2) \geq (a_4)$, являющееся в команде $(K+54)$ условием возвращения к команде $(K+28)$, не будет уже выполняться, и выполнение цикла $(K+28) - (K+54)$ прекратится. Машина перейдет к выполнению команд $(K+55)$ и $(K+56)$, подготовительных к вычислению элементов следующего $(j+1)$ -го столбца.

Подготовка возвращения к началу цикла. После того, как будет вычислен и проверен последний элемент $c_{mn}^{(k)}$ матрицы $C^{(k)}$, неравенство $(a_3) \geq (a_5)$, являющееся в команде $(K+57)$ условием перехода к команде $(K+14)$, будет нарушено: в этот момент $(a_5) = n$, а в ячейке a_3 по-прежнему хранится число $n-1$.

Поэтому будут выполнены непосредственно следующие за командой $(K + 57)$ команды:

$$(K + 58) = \begin{array}{|c|c|c|} \hline \dot{+} & a_{24} & \\ \hline \hline & & a_7 \\ \hline \end{array}$$

$$(K + 59) = \begin{array}{|c|c|c|} \hline \dot{+} & a_{25} & \\ \hline \hline & & a_8 \\ \hline \end{array}$$

$$(K + 60) = \begin{array}{|c|c|c|} \hline \dot{+} & a_{26} & \\ \hline \hline & & a_{10} \\ \hline \end{array}$$

$$(K + 61) = \begin{array}{|c|c|c|} \hline \dot{+} & a_1 & a_6 \\ \hline \hline & & a_6 \\ \hline \end{array}$$

$$(K + 62) = \begin{array}{|c|c|c|} \hline \dot{+} & & \\ \hline \hline & & a_{24} \\ \hline \end{array}$$

$$(K + 63) = \begin{array}{|c|c|c|} \hline \geq & a_2 & a_6 \\ \hline \hline & & K + 1 \\ \hline \end{array}$$

осуществляющие подготовку программы для вычисления следующей матрицы, т. е. матрицы $C^{(k+1)}$.

В результате выполнения команд $(K + 58)$, $(K + 59)$ и $(K + 60)$ получаем равенства: $(a_7) = c_{i_k j_k}^{(k)}$; $(a_8) = i_k - 1$, $(a_{10}) = j_k - 1$. Эти команды помещают в ячейки a_7 , a_8 , a_{10} значение главного элемента матрицы $C^{(k)}$ и его „координаты“ $i_k - 1$ и $j_k - 1$.

Команда $(K + 61)$ увеличивает на 1 значение текущего номера вычисляемой матрицы и в результате выполнения этой команды будет иметь место равенство $(a_6) = k$. Команда $(K + 62)$ гасит ячейку a_{24} , подготавливая ее тем самым к вычислению новой матрицы $C^{(k+1)}$, а команда $(K + 63)$ осуществляет переход к вычислению этой матрицы, если только выполняется условие перехода к команде $(K + 1)$:

$$(a_2) = n - 1 \geq k = (a_6)$$

Как только содержимое ячейки ЗУ a_6 достигнет значения $k = n$, это условие уже не будет выполняться и вычисление по нашей программе закончено.

Хранение ответов. Неизвестные x_j , как уже говорилось, расположены в некоторой последовательности в $(m + 1)$ -м столбце матрицы $C^{(m)}$. Можно или упорядочить эти значения неизвестных в порядке их индексов, или, как мы делаем, выдавать значение неизвестного $x_{j_i} = c_{i, m+1}^{(m)}$ вместе с его индексом j_i . Для этого введем команду $(K + 6)$. Для ее формирования служит команда $(K + 5)$. В результате выполнения всей программы в ячейках $z_{i, m+1}$ будут храниться найденные значения неизвестных, а их индексы — в соответствующих ячейках u , $u + 1, \dots, u + m - 1$.

Программа решения системы линейных алгебраических уравнений
методом главных элементов.

$K+1$	$\dot{+}$	a_{10}	a_{17}	$K+2$
$K+2$				
$K+3$	$\dot{+}$	a_8	a_{16}	$K+4$
$K+4$				
$K+5$	$\dot{+}$	a_8	a_{27}	$K+6$
$K+6$				
$K+7$	$\dot{+}$	a_{19}	a_8	a_{30}
$K+8$	\leftarrow	a_8	a_{12}	$K+2$
$K+9$	$\dot{+}$	a_{18}	$K+2$	a_{29}
$K+10$	\times	a_{10}	a_{13}	$K+4$
$K+11$	$\dot{+}$	$K+4$	a_{20}	a_{31}
$K+12$	$\dot{+}$			a_5
$K+13$	$\dot{+}$	a_{14}		$K+14$
$K+14$				
$K+15$	\times	a_5	a_{13}	$K+2$
$K+16$	\rightarrow	$K+2$	a_{12}	$K+4$
$K+17$	$\dot{+}$	a_{30}	$K+4$	$K+20$
$K+18$	$\dot{+}$	a_{29}	$K+2$	$K+19$
$K+19$				
$K+20$				
$K+21$	$\dot{+}$	a_{15}		$K+28$
$K+22$	$\dot{+}$	a_{22}	$K+4$	a_{36}
$K+23$	$\dot{+}$	a_{23}		$K+47$
$K+24$	$\dot{+}$	a_{21}	$K+2$	a_{33}
$K+25$	$\dot{+}$			a_4
$K+26$	$\dot{+}$			f
$K+27$	$\dot{+}$			g
$K+28$				
$K+29$	\leftarrow	a_4	a_{12}	$K+2$
$K+30$	$\dot{+}$	a_{31}	$K+2$	$K+31$
$K+31$				
$K+32$	$\dot{+}$	a_{33}	$K+2$	$K+33$

$K+33$				
$K+34$	$-$	e	c	d
$K+35$	$\dot{+}$	a_{36}	a_4	$K+36$
$K+36$				
$K+37$	$+$	e	f	f
$K+38$	$-$	f	c	f
$K+39$	$+$	d	g	g
$K+40$	\geq			$K+44$
$K+41$	$\dot{+}$	$K+28$	a_{11}	$K+28$
$K+42$	$+$	b	g	g
$K+43$	$+$	b	f	f
$K+44$	$-$	g	f	h
$K+45$	$ \geq $	h	a_0	ξ
$K+46$	\geq	a_5	a_9	$K+53$
$K+47$				
$K+48$	$ \geq $	a_{24}	d	$K+52$
$K+49$	$\dot{+}$	d		a_{24}
$K+50$	$\dot{+}$	a_4		a_{25}
$K+51$	$\dot{+}$	a_5		a_{26}
$K+52$	$\dot{+}$	$K+47$	a_{11}	$K+47$
$K+53$	$\dot{+}$	a_1	a_4	a_4
$K+54$	\geq	a_2	a_4	$K+28$
$K+55$	$\dot{+}$	$K+14$	a_{11}	$K+14$
$K+56$	$\dot{+}$	a_1	a_5	a_5
$K+57$	\geq	a_3	a_5	$K+14$
$K+58$	$\dot{+}$	a_{24}		a_7
$K+59$	$\dot{+}$	a_{25}		a_8
$K+60$	$\dot{+}$	a_{26}		a_{10}
$K+61$	$\dot{+}$	a_1	a_6	a_6
$K+62$	$\dot{+}$			a_{24}
$K+63$	\geq	a_2	a_6	$K+1$
$K+64$				

Б. Метод ортогонализации

Предложенный Шрейдером (см. ДАН, т. LXXVI, № 5) метод ортогонализации решения системы уравнений:

$$a_{11}x_1 + \dots + a_{1n}x_n = a_{10} \quad (9)$$

$$\dots \dots \dots$$

$$a_{n1}x_1 + \dots + a_{nn}x_n = a_{n0}$$

или

$$Ax = a \quad (9')$$

состоит в следующем¹.

Вычтем из каждого уравнения, начиная со второго, кратное первому так, чтобы полученные строки коэффициентов при неизвестных (мы будем их обозначать a_i , $a_i = \{a_{i1}, \dots, a_{in}\}$) были ортогональны первой строке:

$$a'_i = a_i - \lambda_{i1} a_1, \quad \lambda_{i1} = \frac{\sum_{k=1}^n a_{ik} a_{1k}}{\sum_{k=1}^n a_{1k}^2}, \quad a'_{i0} = a_{i0} - \lambda_{i1} a_{10}$$

С полученными $n-1$ уравнениями (второе, ..., n -е) проделаем то же самое и т. д.

После всех таких преобразований придем к системе уравнений

$$b_{11}x_1 + \dots + b_{1n}x_n = b_{10}$$

$$\dots \dots \dots$$

$$b_{n1}x_1 + \dots + b_{nn}x_n = b_{n0}$$

или

$$Bx = b, \quad (10')$$

¹ Мы несколько изменяем порядок действий.

в которой строки коэффициентов при неизвестных попарно ортогональны: $\sum_{s=1}^n b_{is} b_{js} = 0$ при $i \neq j$.

Система $Bx = b$ такова, что $B = LU$, где U — ортогональная, а L — диагональная матрицы.

$$L = \left\| \begin{array}{cccc} \sqrt{(b_1, b_1)} & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \sqrt{(b_n, b_n)} \end{array} \right\|$$

Из (10') получаем $x = B^{-1} b = B^* L^{-2} b$, $b_{ij}^* = b_{ji}$.

Вычисление $B^* L^{-2} b$, очевидно, проводится следующим образом: надо разделить b_{i0} на $\sum_{s=1}^n b_{is}^2$ и затем вычислить суммы произведений $\sum_{t=1}^n b_{ti} \frac{b_{t0}}{\sum_{s=1}^n b_{ts}^2}$, это и есть нужные ответы.

Таким образом, вычисление разбивается на 3 этапа: I — ортогонализация строк (получение уравнения $Bx = b$), II — нахождение $y = L^{-2} b$, III — нахождение $x = B^* y$. Этапы II и III просты (в § 24 нами уже рассматривались близкие к этому примеры 2 и 4).

Этап I представляет собой систему циклов: полная ортогонализация разбивается на циклы — ортогонализация к отдельной j -й строке всех последующих; эта операция, очевидно, разбивается в свою очередь на циклы — ортогонализация одной строки к другой. Элементарный прием — ортогонализация i -й строки к j -й состоит, как мы видели, из четырех этапов (коэффициенты обозначаем b_{ij}):

a) нахождение $\sum_{s=1}^n b_{js}^2$

b) нахождение $\sum_{s=1}^n b_{is} b_{js}$

$$c) \text{ нахождение } \lambda_{ij} = \frac{\sum_{s=1}^n b_{is} b_{js}}{\sum_{s=1}^n b_{js}^2}$$

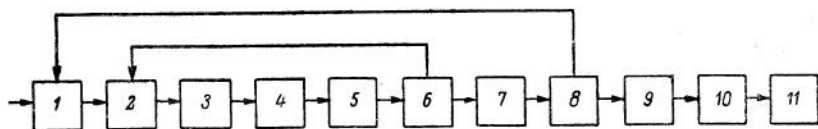
d) нахождение $b_i - \lambda_{ij} b_j$, $b_{i0} - \lambda_{ij} b_{j0}$

Каждый из этих этапов легко программируется. Этап a) общий для всех циклов — циклов ортогонализации к строке $\{b_{i1}, \dots, b_{in}\}$.

Посмотрим, на какие этапы окончательно разбилось решение задачи. Вместо „поменять в программе параметры, соответствующие $\alpha = \alpha_0$ на $\alpha = \alpha_0 + 1$ “, мы пишем „поменять α_0 на $\alpha_0 + 1$ “. Вычисления рассматриваем в момент ортогонализации i -го уравнения к j -му.

1. Найти (b_j, b_j) .
2. Найти (b_i, b_j) .
3. Найти $\lambda_{ij} = \frac{(b_i, b_j)}{(b_j, b_j)}$.
4. Найти $b_i - \lambda_{ij} b_j$, $b_{i0} - \lambda_{ij} b_{j0}$.
5. Поменять i на $i + 1$.
6. Сравнить i с n : при $i \leq n$ перейти к 2, при $i > n$ — к 7.
7. Поменять j на $j + 1$, принять $i = j + 1$ (теперь $j + 2$).
8. Сравнить j с $n - 1$: при $j \leq n - 1$ перейти к 1, при $j > n - 1$ — к 9.
9. Найти (b_n, b_n) .
10. Найти $\frac{b_{s0}}{(b_s, b_s)}$, $s = 1, \dots, n$.
11. Найти $\sum_{s=1}^n b_{st} \frac{b_{s0}}{(b_s, b_s)}$, $t = 1, \dots, n$.

Эти этапы соединяются в следующую блок-схему:



Очевидно, этот же порядок расположения в программе является самым естественным.

Окончательно по лучим следующую программу:

$K+1$	\times	$n+1$	$n+1$	A_1
$K+2$	$+$	n^2+n+1	A_1	n^2+n+1
$K+3$	$\dot{+}$	$K+1$	A_2	$K+1$
$K+4$	\geq	A_3	$K+1$	$K+1$
$K+5$	\times	$n+1$	$n+2$	A_1
$K+6$	$+$	A_1	A_4	A_4
$K+7$	$\dot{+}$	$K+5$	A_2	$K+5$
$K+8$	\geq	A_3	$K+5$	$K+5$
$K+9$	$:$	A_4	n^2+n+1	A_4
$K+10$	\times	1	A_4	A_1
$K+11$	$-$	2	A_1	2
$K+12$	$\dot{+}$	$K+10$	A_5	$K+10$
$K+13$	$\dot{+}$	$K+11$	A_6	$K+11$
$K+14$	\geq	A_3	$K+10$	$K+10$
$K+15$	$\dot{+}$	$K+5$	A_7	$K+5$
$K+16$	$\dot{+}$	$K+10$	A_8	$K+10$
$K+17$	$\dot{+}$	$K+11$	A_9	$K+11$
$K+18$	$+$			A_4
$K+19$	\geq	A_{10}	$K+11$	$K+5$
$K+20$	$\dot{+}$	$K+5$	A_{11}	$K+5$
$K+21$	$\dot{+}$	$K+11$	A_{12}	$K+11$
$K+22$	$\dot{+}$	A_{11}	A_{13}	A_{11}
$K+23$	$\dot{+}$	A_{12}	A_{14}	A_{12}
$K+24$	$\dot{+}$	$K+1$	A_{15}	$K+1$
$K+25$	$\dot{+}$	$K+2$	A_{14}	$K+2$
$K+26$	$\dot{+}$	$K+9$	A_{12}	$K+9$
$K+27$	$\dot{+}$	$K+10$	A_{16}	$K+10$
$K+28$	\geq	A_{17}	$K+1$	$K+1$
$K+29$	\times	$2n$	$2n$	A_1
$K+30$	$+$	A_1	n^2+2n	n^2+2n
$K+31$	$\dot{+}$	$K+29$	A_2	$K+29$
$K+32$	\geq	A_3	$K+29$	$K+29$
$K+33$	$:$	1	n^2+n+1	1
$K+34$	$\dot{+}$	$K+33$	A_{18}	$K+33$
$K+35$	\geq	A_{19}	$K+33$	$K+33$
$K+36$	$+$			n^2+n+1

$K + 37$	\times	1	$n + 1$	A_1
$K + 38$	$+$	$n^2 + n + 1$	A_1	$n^2 + n + 1$
$K + 39$	\div	$K + 37$	A_{20}	$K + 37$
$K + 40$	$>$	A_{21}	$K + 37$	$K + 37$
$K + 41$	\div	$K + 36$	A_{22}	$K + 36$
$K + 42$	\div	$K + 37$	A_5	$K + 37$
$K + 43$	\div	$K + 38$	A_{14}	$K + 38$
$K + 44$	\geq	A_{23}	$K + 38$	$K + 36$
$K + 45$				

Перед вычислениями

$$(i + nj) = a_{ij}, (n^2 + n + i) = 0, i = 1, 2, \dots, n, j = 0, 1, 2, \dots, n$$

A_1	что угодно			
A_2		n	n	
A_3	\times	$n^2 + n$		
A_4		n		
A_5		n		n
A_6		$-n^2$	$1 - n^2$	
A_7		$-n^2 - n$		
A_8		$1 - n - n^2$		$1 - n - n^2$
A_9	$-$	$n + 1$		
A_{10}		1	$2 - n$	
A_{11}		$1 - n$		$1 - n$
A_{12}			1	
A_{13}		1		1
A_{14}		$1 - n^2$	$1 - n^2$	
A_{15}		1		
A_{16}	\times	$2n$		
A_{17}		1	1	1
A_{18}	$:$	$n + 1$		
A_{19}		1	1	
A_{20}	\times	$n + 1$		
A_{21}				1
A_{22}				
A_{23}	$+$	$n^2 + n + 1$		

После вычислений $(n^2 + n + i) = x_i, i = 1, 2, \dots, n.$

Здесь команды $(K+1) - (K+4)$ соответствуют этапу 1, $(K+5) - (K+8) - 2$, $(K+9) - 3$, $(K+10) - (K+14) - 4$, $(K+15) - (K+18) - 5$, $(K+19) - 6$, $(K+20) - (K+27) - 7$, $(K+28) - 8$, $(K+29) - (K+32) - 9$, $(K+33) - (K+35) - 10$, $(K+36) - (K+44) - 11$.

Мы не рассматривали здесь контроля операций; он производится обычным образом и требует лишь незначительных добавлений к программе.

Этапы 1, 2 и 9 очень похожи и могут быть заменены одним с добавлением соответствующих операций сравнения. Такое изменение, сократив программу, сделало бы ее несколько более сложной.

§ 31. ИНТЕГРИРОВАНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ МЕТОДОМ РУНГЕ-КУТТА

Рассмотрим следующий процесс численного интегрирования обыкновенного дифференциального уравнения

$$y' = f(x, y)$$

с начальными условиями $y = y_0$ при $x = x_0$.

Пусть известно значение y_n искомой функции при $x = x_n$. Тогда приращение Δy_n при изменении аргумента от x_n до $x_{n+1} = x_n + h$ вычисляется по следующим формулам:

$$\Delta y_n = \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)$$

где

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = hf(x_n + h, y_n + k_3)$$

значения x_n и y_n . Следующее значение y_{n+1} искомой функции получится в результате последовательного прибавления к содержимому ячейки ЗУ η'' , в которой в начале каждого шага содержится y_n , величин $\sigma_i \frac{k_i}{2}$. Если программа предусматривает операции, в результате которых в ячейках ЗУ σ, μ вместо значений σ_i, μ_i появляются σ_{i+1}, μ_{i+1} , то весь цикл вычислений на одном шаге интегрирования можно осуществить как четырехкратное повторение одной и той же последовательности операций. Так как $\sigma_i = \frac{1}{3} \mu_{i+1}$ (считая, что $\mu_4 = \frac{1}{2}$), то вместо двух параметров σ_i и μ_i можно использовать один параметр μ_i следующим образом. Вслед за получением частичного приращения k_i , это частичное приращение, также как шаг интегрирования h , умножается на μ_i и вычисляются значения аргументов $x_n + \mu_i \frac{h}{2}$ и $y_n + \mu_i \frac{k_i}{2}$ для получения следующего частичного приращения. Затем производится операция получения следующего значения μ_{i+1} параметра, и частичное приращение k_i умножается на μ_{i+1} , получившееся произведение умножается на $\frac{1}{3}$ и прибавляется к содержимому ячейки ЗУ, в которой накапливается сумма

$$y_{n+1} = y_n + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)$$

Далее повторяется процесс вычисления следующего частичного приращения.

После того, как будут вычислены все четыре частичные приращения и новое значение функции, т. е. произойдет четырехкратное повторение описанного цикла операций, следует, если это требуется, отправка нового значения функции в печать и подготовка к новому шагу интегрирования.

Операции, превращающие μ_i в μ_{i+1} , можно осуществить различным способом. Одна из возможных программ для

осуществления такого преобразования параметра следующая:

$K_s + 1$	—	α_5	α_6	α_6
$K_s + 2$	+	μ	α_6	μ
$K_s + 3$	\geq	α_7	μ	$K_s + 6$
$K_s + 4$	—	μ	α_3	μ

где $(\alpha_5) = 1$, $(\alpha_6) = 1$ (исходное значение), $(\alpha_7) = 2$ и $(\mu) = \mu_i$. Команду $(K_s + 5)$, следующую за последней из команд для преобразования параметра μ , удобно использовать для выработки признака, по которому можно было бы выбрать путь дальнейших вычислений после того, как очередное частичное приращение k_i , умноженное на $\frac{1}{3} \mu_{i+1}$, будет прибавлено к сумме $y_n + \frac{1}{3} \mu_2 k_1 + \dots + \frac{1}{3} \mu_i k_{i-1}$. Вся программа выглядит следующим образом

$K + 1$	f	γ_1	β_1	β_2
$K + 2$	+	γ_2	α_1	γ_1
$K + 3$	\times	β_2	α_1	β_3
$K + 4$	+	β_5	β_3	β_1
$K + 5$	—	α_5	α_6	α_6
$K + 6$	+	μ	α_6	μ
$K + 7$	\geq	α_7	μ	$K + 10$
$K + 8$	—	μ	α_7	μ
$K + 9$	+	γ_1		γ_3
$K + 10$	\times	α_2	μ	α_1
$K + 11$	\times	α_1	α_3	α_4
$K + 12$	\times	β_2	α_4	β_3
$K + 13$	+	β_3	β_4	β_4
$K + 14$	\geq	γ_2	γ_3	$K + 1$
$K + 15$	Π	β_4		
$K + 16$	+	β_4		β_5
$K + 17$	+	β_4		β_1
$K + 18$	+	γ_1		γ_2
$K + 19$	$>$	γ_4	γ_2	$K + 1$
$K + 20$				

Начальное содержание ячеек ЗУ

α_1	$h/2$	Конец промежутка интегрирования
α_2	$h/2$	
α_3	$1/3$	
α_4	любое	
α_5	1	
α_6	1	
α_7	2	
β_1	y_0	
β_2	любое	
β_3	любое	
β_4	y_0	
β_5	y_0	
γ_1	x_0	
γ_2	x_0	
γ_3	x_0	
γ_4	b	

Команда $(K+1)$ выписанной программы представляет собой по сути целую подпрограмму вычислений правой части дифференциального уравнения по аргументам, заданным в ячейках ЗУ γ_1 и β_1 с выдачей результата в ячейку ЗУ β_2 .

Хотя приведенная программа является программой численного интегрирования одного дифференциального уравнения, ее можно стандартным приемом превратить в программу интегрирования системы дифференциальных уравнений. Для этого достаточно команды, содержащие хотя бы один из адресов β_k , понимать как подпрограммы действий над векторами. При этом команды $(K+4)$, $(K+13)$, $(K+16)$, $(K+17)$ оказываются подпрограммами сложения двух векторов, а команды $(K+3)$ и $(K+12)$ — подпрограммами умножения вектора на скаляр. Команду $(K+1)$ следует в случае системы дифференциальных уравнений понимать как подпрограмму вычисления вектор-функции.

§ 32. ИНТЕГРИРОВАНИЕ УРАВНЕНИЙ ВНЕШНЕЙ БАЛЛИСТИКИ

В настоящем параграфе мы приведем в качестве примера сложной программы программу численного решения системы дифференциальных уравнений внешней баллистики методом Адамса.

Система дифференциальных уравнений движения центра массы снаряда при аргументе t имеет вид:

$$\left. \begin{aligned} \frac{dx}{dt} &= u \\ \frac{dy}{dt} &= w \\ \frac{du}{dt} &= -Eu \\ \frac{dw}{dt} &= -Ew - g \end{aligned} \right\} \quad (1)$$

при начальных условиях $x|_{t=0} = y|_{t=0} = 0, u|_{t=0} = u_0, w|_{t=0} = w_0$.
Здесь

$$E = c G \left(\frac{v}{V^\tau} \right) H_\tau(y) \quad (2)$$

где $v = \sqrt{u^2 + w^2}$, $\tau = \tau(y)$, c — константа, носящая название „баллистический коэффициент“, G, H и τ — эмпирические функции, g — ускорение силы тяжести.

Физический смысл величин, входящих в уравнение (1): x — горизонтальная, а y — вертикальная составляющие пути, пройденного снарядом; u — горизонтальная, а w — вертикальная составляющие скорости снаряда; v — скорость снаряда.

Мы будем применять для численного решения системы (1) метод, который удобнее пояснить на примере одного уравнения

$$\frac{dy}{dx} = f(x, y), \quad y|_{x=x_0} = y_0 \quad (3)$$

Перенос метода на случай системы не представляет труда.

В качестве начального процесса для подсчета первых четырех значений y_0, y_1, y_2, y_3 мы примем следующий прием. Положим в качестве нулевого приближения

$$y_0^{(0)} = y_1^{(0)} = y_2^{(0)} = y_3^{(0)} = y_0 \quad (4)$$

и затем начнем процесс уточнения приближений по формулам:

$$\left. \begin{aligned} f(x_1, y_1^{(k)}) &= y_1'^{(k)} \\ f(x_2, y_2^{(k)}) &= y_2'^{(k)} \\ f(x_3, y_3^{(k)}) &= y_3'^{(k)} \end{aligned} \right\} \quad (5)$$

и

$$\left. \begin{aligned} y_1^{(k+1)} &= y_0 + h \left(\frac{3}{8} y_0' + \frac{19}{24} y_1'^{(k)} - \frac{5}{24} y_2'^{(k)} + \frac{1}{24} y_3'^{(k)} \right) \\ y_2^{(k+1)} &= y_0 + h \left(\frac{1}{3} y_0' + \frac{4}{3} y_1'^{(k)} + \frac{1}{3} y_2'^{(k)} \right) \\ y_3^{(k+1)} &= y_0 + h \left(\frac{3}{8} y_0' + \frac{9}{8} y_1'^{(k)} + \frac{9}{8} y_2'^{(k)} + \frac{3}{8} y_3'^{(k)} \right) \end{aligned} \right\} \quad (5')$$

Легко показать, что при наличии ограниченной производной $\frac{\partial f}{\partial y}$ и достаточно малом h итерационный процесс (5) — (5') сходитс

к некоторым значениям y_1, y_2, y_3 , которые мы и примем за приближения к значениям решения уравнения (3) в точках x_1, x_2 и x_3 .

В дальнейшем мы будем продолжать процесс обычным методом Адамса, по формулам:

$$y_{n+1} = y_n + h \left(\frac{55}{24} y_n' - \frac{59}{24} y_{n-1}' + \frac{37}{24} y_{n-2}' - \frac{3}{8} y_{n-3}' \right) \quad (6)$$

$$y_{n+1} = y_n + h \left(\frac{3}{8} y'_{n+1} + \frac{19}{24} y'_n - \frac{5}{24} y'_{n-1} + \frac{1}{24} y'_{n-2} \right) \quad (7)$$

Этот метод, обобщенный на систему уравнений, мы и применим к системе (1).

Относительно эмпирических функций мы будем предполагать следующее. Функция $\tau(y)$ на трех разных интервалах вычисляется по разным формулам:

$$\left. \begin{aligned} 1) \text{ при } y_1 - y \geq 0, \tau(y) &= \tau_1 + \alpha(y_1 - y) \\ 2) \text{ при } 0 \geq y_1 - y \geq y_2, \tau(y) &= \tau_1 + \alpha(y_1 - y) + \beta(y_1 - y)^2 \\ 3) \text{ при } y_1 - y \leq y_2, \tau(y) &= \tau_0 \end{aligned} \right\} \quad (8)$$

Здесь $y_1 = 9300$; $y_2 = -2700$.

Мы будем рассматривать G , как функцию от $\frac{v^2}{\tau}$, предполагая, что она с достаточной точностью представима полиномами

$$G\left(\frac{v^2}{\tau}\right) = A_k \left(\frac{v^2}{\tau}\right)^3 + B_k \left(\frac{v^2}{\tau}\right)^2 + C_k \frac{v^2}{\tau} + D_k \quad (9)$$

соответственно, на интервалах

$$v_{k-1}^2 \geq \frac{v^2}{\tau} \geq v_k^2$$

Аналогично будем считать функцию $H_\tau(y)$ составленной из полиномиальных кусков.

Программа, реализующая только что указанный процесс, состоит из 99 команд (включая команды, предназначенные для возвращения всех изменившихся за время работы команд и ячеек ЗУ в первоначальное положение). Все команды и ячейки занимают 312 мест в ЗУ (из них более половины составляют ячейки ЗУ, значение которых не изменяется). Программа выглядит следующим образом:

$K+1$	$\dot{+}$	K_1		$K+5$	$K+36$	$\dot{+}$	S_7		S_9
$K+2$	$\dot{+}$	K_2		$K+10$	$K+37$	\times	p_2	p_1	p_1
$K+3$	$\dot{+}$	K_3		K_4	$K+38$	$+$	$d-3$	p_1	p_1
$K+4$	$\dot{+}$			p_2	$K+39$	$+$	$K+38$	S_7	$K+38$
$K+5$	\times	$a+6$	$b+1$	p_1	$K+40$	$\dot{+}$	S_7	S_9	S_9
$K+6$	$+$	p_1	p_2	p_2	$K+41$	\geq	S_8	S_9	$K+37$
$K+7$	$\dot{+}$	$K+5$	S_1	$K+5$	$K+42$	\geq	$K+37$	K_7	$K+48$
$K+8$	\geq	K_5	$K+5$	$K+5$	$K+43$	$\dot{+}$	p_1		p_4
$K+9$	\times	p_2	m_6	p_2	$K+44$	$\dot{+}$	K_8		$K+34$
$K+10$	$+$	$a+4$	p_2	$a+10$	$K+45$	$\dot{+}$	K_7		$K+37$
$K+11$	$\dot{+}$	$K+5$	S_2	$K+5$	$K+46$	$\dot{+}$	K_9		$K+38$
$K+12$	$\dot{+}$	$K+10$	S_3	$K+10$	$K+47$	\geq			$K+32$
$K+13$	\geq	K_4	$K+5$	$K+4$	$K+48$	\times	p_1	p_4	p_1
$K+14$	\geq		S_3	$K+19$	$K+49$	\times	p_1	m_7	p_1
$K+15$	$\dot{+}$	$K+5$	S_4	$K+5$	$K+50$	\times	$a+9$	p_1	$a+11$
$K+16$	$\dot{+}$	$K+10$	S_5	$K+10$	$K+51$	\times	$a+10$	p_1	$a+12$
$K+17$	$\dot{+}$	K_4	S_6	K_4	$K+52$	$-$	$a+12$	m_9	$a+12$
$K+18$	\geq	K_4	K_6	$K+4$	$K+53$	$\dot{+}$	K_{10}		$K+34$
$K+19$	\times	$a+9$	$a+9$	p_1	$K+54$	$\dot{+}$	$K+37$	S_6	$K+37$
$K+20$	\times	$a+10$	$a+10$	p_2	$K+55$	$\dot{+}$	K_{11}		$K+38$
$K+21$	$+$	p_1	p_2	p_2	$K+56$	$\dot{+}$	K_{12}		$K+10$
$K+22$	$-$	m_1	$a+8$	p_3	$K+57$	\geq		S_3	$K+73$
$K+23$	$\dot{+}$	m_3		p_1	$K+58$	$\dot{+}$	$K+19$	S_{10}	$K+19$
$K+24$	\geq	p_3		$K+27$	$K+59$	$\dot{+}$	$K+20$	S_{10}	$K+20$
$K+25$	\times	p_3	m_4	p_4	$K+60$	$\dot{+}$	$K+22$	S_{11}	$K+22$
$K+26$	$+$	p_4	p_1	p_1	$K+61$	$\dot{+}$	$K+50$	S_{12}	$K+50$
$K+27$	\times	p_1	p_3	p_1	$K+62$	$\dot{+}$	$K+51$	S_{12}	$K+51$
$K+28$	$+$	p_1	m_{11}	p_1	$K+63$	$\dot{+}$	$K+52$	S_{12}	$K+52$
$K+29$	\geq	p_3	m_2	$K+31$	$K+64$	\geq	K_{13}	$K+19$	$K+19$
$K+30$	$\dot{+}$	m_5		p_1	$K+65$	$\dot{+}$	p_5	m_{12}	p_5
$K+31$	$:$	p_2	p_1	p_2	$K+66$	\geq	m_{10}	p_5	$K+87$
$K+32$	$\dot{+}$	$K+34$	S_7	$K+34$	$K+67$	$\dot{+}$	S_{13}		S_2
$K+33$	$\dot{+}$	$K+38$	S_8	$K+38$	$K+68$	$\dot{+}$	S_{14}		S_3
$K+34$	\geq	c	p_2	$K+32$	$K+69$	$\dot{+}$	K_{21}		$K+13$
$K+35$	$\dot{+}$			p_1	$K+70$	$\dot{+}$	K_{20}		K_5

$K+71$	$\dot{+}$	K_{15}		$K+5$	$K+85$	$\dot{+}$	K_{19}		$K+5$
$K+72$	\geq			$K+4$	$K+86$	\geq	$a+14$		$K+4$
$K+73$	$-$	$a+27$	p_5	p_7	$K+87$	$\dot{+}$	$K+19$	S_{15}	$K+19$
$K+74$	$-$	$a+28$	p_6	p_8	$K+88$	$\dot{+}$	$K+20$	S_{15}	$K+20$
$K+75$	$\dot{+}$	$a+27$		p_5	$K+89$	$\dot{+}$	$K+22$	S_{19}	$K+22$
$K+76$	$\dot{+}$	$a+28$		p_6	$K+90$	$\dot{+}$	$K+50$	S_{16}	$K+50$
$K+77$	$ \geq $	p_7	m_8	$K+84$	$K+91$	$\dot{+}$	$K+51$	S_{16}	$K+51$
$K+78$	$ \geq $	p_8	m_8	$K+84$	$K+92$	$\dot{+}$	$K+52$	S_{16}	$K+52$
$K+79$	$\dot{+}$	$a+7$		$a+1$	$K+93$	\geq	S_3		$K+1$
$K+80$	$\dot{+}$	$K+79$	S_{20}	$K+79$	$K+94$	$\dot{+}$	K_{20}		K_5
$K+81$	\geq	K_{16}	$K+79$	$K+79$	$K+95$	$\dot{+}$	S_{17}		S_2
$K+82$	$\dot{+}$	K_{17}		$K+79$	$K+96$	$\dot{+}$	S_{18}		S_3
$K+83$	\geq			$K+70$	$K+97$	$\dot{+}$			p_5
$K+84$	$\dot{+}$	K_{18}		K_5	$K+98$	$\dot{+}$	K_{14}		$K+13$
					$K+99$				

Ячейки, содержащие числа

$a+1$	x_0	$a+11$	$u'_0(u'_1)$	$a+21$	$u_0(u_3)$
$a+2$	y_0	$a+12$	$w'_0(w'_1)$	$a+22$	$w_0(w_3)$
$a+3$	u_0	$a+13$	$x_0(x_2)$	$a+23$	$u'_0(u'_3)$
$a+4$	w_0	$a+14$	$y_0(y_2)$	$a+24$	$w'_0(w'_3)$
$a+5$	u'_0	$a+15$	$u_0(u_2)$	$a+25$	$0(x_4)$
$a+6$	w'_0	$a+16$	$w_0(w_2)$	$a+26$	$0(y_4)$
$a+7$	$x_0(x_1)$	$a+17$	$u'_0(u'_2)$	$a+27$	$0(u_4)$
$a+8$	$y_0(y_1)$	$a+18$	$w'_0(w'_2)$	$a+28$	$0(w_4)$
$a+9$	$u_0(u_1)$	$a+19$	$x_0(x_3)$	$a+29$	$0(u'_4)$
$a+10$	$w_0(w_1)$	$a+20$	$y_0(y_3)$	$a+30$	$0(w'_4)$

$b + 1$	$\frac{3}{8}$	$c + 1$	v_1^2	m_1	y_1
$b + 2$	$\frac{19}{24}$	$c + 2$	v_2^2	m_2	y_2
$b + 3$	$-\frac{5}{24}$	\dots	\dots	m_3	α
$b + 4$	$\frac{1}{24}$	$c + 10$	v_{10}^2	m_4	β
$b + 5$	$\frac{1}{24}$	$d + 1$	A_1	m_5	τ_0
$b + 6$	$\frac{3}{24}$	$d + 2$	B_1	m_6	h
$b + 7$	$\frac{4}{3}$	$d + 3$	C_1	m_7	$-c$
$b + 8$	$\frac{1}{3}$	$d + 4$	D_1	m_8	δ
$b + 9$	0	\dots	\dots	m_9	g
$b + 10$	$\frac{3}{8}$	$d + 37$	A_{10}	m_{10}	$N-1$
$b + 11$	$\frac{9}{8}$	$d + 38$	B_{10}	m_{11}	τ_1
$b + 12$	$\frac{9}{8}$	$d + 39$	C_{10}	m_{12}	1
$b + 13$	$\frac{3}{8}$	$d + 40$	D_{10}	p_1	0
$b + 14$	$-\frac{3}{8}$	$e + 1$	y_1	p_2	0
$b + 15$	$\frac{37}{24}$	\dots	\dots	p_3	0
$b + 16$	$-\frac{59}{24}$	$e + 10$	y_{10}	p_4	0
$b + 17$	$\frac{55}{24}$	$f + 1$	α_1	p_5	0
$b + 18$	$\frac{1}{24}$	$f + 2$	β_1	p_6	0
$b + 19$	$-\frac{5}{24}$	$f + 3$	γ_1	p_7	0
$b + 20$	$\frac{19}{24}$	$f + 4$	δ_1	p_8	0
	$\frac{3}{8}$	\dots	\dots		
	$\frac{3}{8}$	$f + 40$	δ_{10}		

Ячейки, содержащие коды некоторых команд

K_1	\times	$a + 6$	$b + 1$	p_1	K_{12}	$+$	$a + 22$	p_2	$a + 28$
K_2	$+$	$a + 4$	p_2	$a + 10$	K_{13}	\times	$a + 27$	$a + 27$	
K_3	\times	$a + 6$	$b + 13$		K_{14}	\geq	K_4	$K + 5$	$K + 4$
K_4	\times	$a + 6$	$b + 13$		K_{15}	\times	$a + 6$	$b + 13$	p_1
K_5	\times	$a + 27$			K_{16}	$+$	$a + 31$		
K_6	\times	$a + 3$			K_{17}	$+$	$a + 7$		$a + 1$
K_7	\times	p_3	p_1	p_1	K_{18}	\times	$a + 33$		
K_8	\geq	e	p_3	$K + 32$	K_{19}	\times	$a + 12$	$b + 17$	p_1
K_9	$+$	$f - 3$	p_1	p_1	K_{20}	\times	$a + 27$		
K_{10}	$>$	c	p_2	$K + 32$	K_{21}	\geq	$K + 10$	K_{22}	$K + 4$
K_{11}	$+$	$d - 3$	p_1	p_1	K_{22}	$+$	$a + 19$		

Ячейки, содержащие числа, прибавляемые к командам

S_1	6	1		S_{11}		6	
S_2	-24			S_{12}	6		6
S_3				S_{13}	-25	-4	
S_4	-1	-12	6	S_{14}	-1		-1
S_5	-1			S_{15}	-18	-18	
S_6	-1		-19	S_{16}	-18		-18
S_7	1			S_{17}	-24		
S_8	4			S_{18}			6
S_9				S_{19}		-18	
S_{10}	6	6		S_{20}	1		1

Пояснения к программе начнем с ячеек ЗУ. Ячейки $a + 1 \div a + 30$ в начале процесса содержат значения $x_0, y_0, u_0, w_0, u'_0, w'_0$. Рядом с начальным содержимым ячеек в скобках указаны те величины, значения которых будут храниться в этих ячейках после вычисления первых четырех точек траектории. Содержимое ячеек $b + 1 \div b + 20$ не меняются в процессе работы. Они содержат коэффициенты формул (5'),

(6), (7). Ячейки $c + 1 \div c + 10$, $d + 1 \div d + 40$ содержат параметры, упомянутые в формулах (9), за исключением v_0^2 . Аналогично, ячейки $e + 1 \div e + 10$, $f + 1 \div f + 40$ задают значения параметров в аппроксимационных формулах:

$$H_k(y) = \alpha_k(y_1 - y)^3 + \beta_k(y_1 - y)^2 + \gamma_k(y_1 - y) + \delta_k \quad (10)$$

соответственно, на интервалах

$$\bar{y}_{k-1} \geq y_1 - y \geq \bar{y}_k \quad (k = 1, 2, \dots, 10)$$

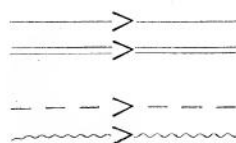
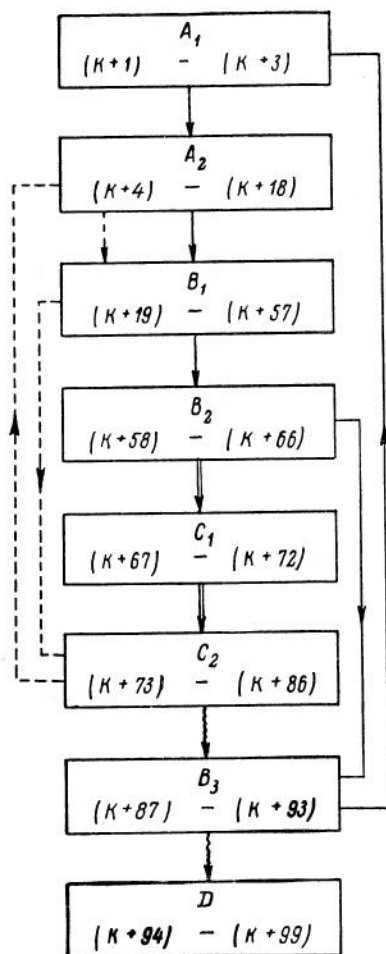
Все ячейки $c + i$, $d + i$, $e + i$, $f + i$ также остаются в течение всего времени неизменными. Остальные числовые параметры расположены в ячейках $m_1 \div m_{12}$. Это $\alpha_1, \beta_1, \gamma_1, \delta_1, \tau_1$ (см. формулы (8)), а также баллистический коэффициент c , заданный с обратным знаком, шаг интегрирования h , ускорение силы тяжести g , величина δ , характеризующая точность процесса, число $N - 1$, где N — количество итераций, достаточное для получения первоначальных значений, и, наконец, единица.

Ячейки $p_1 - p_8$ являются рабочими. В начале процесса все они пусты (впрочем, некоторые из них могут также содержать произвольные числа). Ячейки $S_1 - S_{20}$ содержат числа, предназначенные для операций над командами, снабженные знаками и разбитые на группы, соответственно разрядам, отводимым в командах под адреса ячеек ЗУ. Ячейки $K_1 - K_{22}$ содержат коды команд. О роли этих ячеек мы подробнее скажем ниже.

Перейдем к разбору самого процесса. Он делится на четыре основные группы, показанные на блок-схеме:

(А) команды $(K + 1) - (K + 18)$ — подсчет по формулам типа (5'); (Б) команды $(K + 19) - (K + 66)$ и $(K + 87) - (K + 93)$ — подсчет u' и w' по формулам (1) — (2); (С) команды $(K + 67) - (K + 86)$ — подсчет по формулам (6) — (7); (Д) команды $(K + 94) - (K + 99)$ — возвращение программы в первоначальное состояние. Группы (А), (Б) и (С) — (Д) рассматриваются, соответственно, в пунктах I, II и III.

Блок-схема



I. Первые 4 команды, как легко видеть, в начале процесса ничего не меняют. Команда $(K+5)$ составляет произведение $\frac{3}{8} w'_0$ и отправляет его в ячейку p_1 . Команда $(K+6)$ переправляет это произведение в ячейку p_2 . Команда $(K+7)$ изменяет $(K+5)$ так, что теперь команда $(K+5)$ имеет вид:

$$K+5 \quad \left[\begin{array}{c|c|c|c} \times & a+12 & b+2 & p_1 \end{array} \right]$$

Так как $(K+5) < (K_5)$, то в соответствии с командой $(K+8)$ возвращаемся вновь к команде $(K+5)$ и отправляем в ячейку p_1 произведение $\frac{19}{24} w_1^{(0)'}$, а команда $(K+6)$ образует в ячейке p_2 сумму $\frac{3}{8} w'_0 + \frac{19}{24} w_1^{(0)'}$. Этот процесс повторяется до тех пор, пока в ячейке p_2 не образуется величина

$$\sigma = \frac{3}{8} w'_0 + \frac{19}{24} w_1^{(0)' } - \frac{5}{24} w_2^{(0)' } + \frac{1}{24} w_3^{(0)' }$$

Тогда команда $(K+5)$ будет иметь вид:

$$K+5 \quad \left[\begin{array}{c|c|c|c} \times & a+30 & b+5 & p_1 \end{array} \right]$$

и в соответствии с командой $(K+8)$, мы перейдем к команде $(K+9)$, т. е. умножим σ на h и командой $(K+10)$ образуем w'_1 в соответствии с формулами (5'). Команды $(K+11)$ и $(K+12)$ придадут $(K+5)$ и $(K+10)$ следующий вид:

$$\begin{array}{l} K+5 \quad \left[\begin{array}{c|c|c|c} \times & a+6 & b+5 & p_1 \end{array} \right] \\ K+10 \quad \left[\begin{array}{c|c|c|c} + & a+4 & p_2 & a+16 \end{array} \right] \end{array}$$

Возвратившись в соответствии с командой $(K+13)$ к команде $(K+4)$, мы очистим ячейку (p_2) и начнем процесс, совершенно аналогичный только что проведенному и дающий $w_2^{(1)}$ в соответствии с формулами (5'). После того, как таким же

образом будет подсчитано $w_3^{(1)}$, команды $(K+5)$ и $(K+10)$ примут вид:

$K+5$	\times	$a+6$	$b+13$	p_1
$K+10$	$+$	$a+4$	p_2	$a+28$

Таким образом, в соответствии с командами $(K+13)$ и $(K+14)$ (роль последней выяснится позднее), мы перейдем к командам $(K+15) - (K+17)$, после которых команды $(K+5)$, $(K+10)$ и (K_4) примут вид:

$K+5$	\times	$a+5$	$b+1$	p_1
$K+10$	$+$	$a+3$	p_2	$a+6$
K_4	\times	$a+5$	$b+13$	

Вернувшись теперь к команде $(K+4)$, мы начнем подсчет $u_1^{(1)}$, $u_2^{(1)}$ и $u_3^{(1)}$ через их производные по формулам (5'). В частности, только что написанные значения команд $(K+5)$ и $(K+10)$ дадут возможность вычислить $u_1^{(1)}$ и отправить его в ячейку $a+9$. Так как $y' = w$ и $x' = u$, то следующие два цикла вычислят $x_k^{(1)}$ и $y_k^{(1)}$ ($k = 1, 2, 3$) по формулам типа

$$x_1^{(1)} = x_0 + h \left(\frac{3}{8} u_0 + \frac{19}{24} u_1^{(1)} - \frac{5}{24} u_2^{(1)} + \frac{1}{24} u_3^{(1)} \right)$$

Циклы окончатся лишь после вычисления $x_3^{(1)}$, когда ячейка ЗУ K_4 будет содержать

K_4	\times	$a+2$	$b+13$	
-------	----------	-------	--------	--

и в соответствии с командой $(K+18)$ мы перейдем к команде $(K+19)$, с которой начинается вычисление правых частей двух последних уравнений системы (1).

Таким образом, благодаря высокой цикличности процесса 100 сложений и умножений уместились в 15 командах.

II. Команды $(K+19) - (K+21)$ образуют сумму $u_1^2 + w_1^2$, где под u_1 и w_1 понимаем их приближенные значения,

находящиеся в ячейках $3Y \ a + 9$ и $a + 10$. Команды $(K+22) - (K+30)$ вычисляют $\tau(y_1^{(1)})$ по формулам (8). В самом деле, команда $(K+22)$ образует $y_1 - y_1^{(1)}$. Затем команда $(K+24)$ раздвигает процесс так, что множимое команды $(K+27)$ равно α при $y_1 - y_1^{(1)} \geq 0$ и $\alpha + \beta(y_1 - y_1^{(1)})$ при $y_1 - y_1^{(1)} < 0$. В первом случае команда $(K+28)$ образует $\tau_1 + \alpha(y_1 - y_1^{(1)})$, во втором $\tau_1 + \alpha(y_1 - y_1^{(1)}) + \beta(y_1 - y_1^{(1)})^2$ в соответствии с формулами (8). Наконец, при $y_1 - y_1^{(1)} \leq y_2$, команда $(K+30)$ посылает τ_0 в ячейку p_1 , вытесняя накопленное в p_1 выражение. Команда $(K+31)$, единственная команда деления в программе, образует $\frac{u_1^2 + w_1^2}{\tau(y_1^{(1)})} = v^2$.

Группа команд с $(K+32)$ по $(K+41)$ вычисляет $G\left(\frac{u_1^2 + w_1^2}{\tau(y_1^{(1)})}\right)$. В результате выполнения команд $(K+32)$ и $(K+33)$ команды $(K+34)$ и $(K+38)$ приобретают вид:

$$\begin{array}{l} K+34 \\ K+38 \end{array} \quad \begin{array}{|c|c|c|c|} \hline \geq & c+1 & p_2 & K+32 \\ \hline + & d+1 & p_1 & p_1 \\ \hline \end{array}$$

Таким образом, при $\frac{u_1^2 + w_1^2}{\tau(y_1^{(1)})} \geq v_1^2$ (т. е. если $\frac{u_1^2 + w_1^2}{\tau(y_1^{(1)})}$ входит в первый интервал $v_0^2 \geq \frac{v^2}{\tau} \geq v_1^2$) мы переходим к команде $(K+37)$, оставляющей (в соответствии с командой $(K+35)$) нуль в ячейке p_1 . Команда $(K+38)$ посылает A_1 в ячейку p_1 , а $(K+39)$ изменяет команду $(K+38)$, придавая ей вид:

$$K+38 \quad \begin{array}{|c|c|c|c|} \hline + & d+2 & p_1 & p_1 \\ \hline \end{array}$$

и после команды $(K+41)$ мы вновь возвратимся к команде $(K+37)$. Легко видеть, что последующие циклы дадут в ячейке p_1 последовательно

$$\begin{aligned} A_1 v^2 + B_1 \\ A_1 v^4 + B_1 v^2 + C_1 \\ A_1 v^6 + B_1 v^4 + C_1 v^2 + D_1 \end{aligned}$$

В случае же, если v^2 входит в r -й интервал, первый адрес команды $(K + 38)$ сдвинется на 4 $(r - 1)$, так как сравнение в команде $(K + 34)$ повторится r раз до того момента, когда его первый адрес будет содержать число, меньшее v^2 . Команды $(K + 42) - (K + 47)$ предназначены для повторного использования команд $(K + 32) - (K + 41)$ для вычисления $H_\tau(y_1^{(1)})$. Так как $(K + 37) < (K_\tau)^*$, то мы переходим к команде $(K + 43)$ и отправляем $G(v^2)$ в ячейку p_4 . ($H_\tau(y_1^{(1)})$ будет подсчитываться в p_1 .) Затем изменяем команды $(K + 34)$, $(K + 37)$ и $(K + 38)$ в соответствии с их новым назначением, так что эти команды примут вид:

$K + 34$	\geq	e	p_3	$K + 32$
$K + 37$	\times	p_3	p_1	p_1
$K + 38$	$+$	$f - 3$	p_1	p_1

Затем, после возврата к команде $(K + 32)$, та же группа команд вычислит $H_\tau(y_1^{(1)})$ в соответствии с формулами (10), и теперь уже команда $(K + 37)$ даст нам переход к команде $(K + 48)$, вычисляющей произведение $G(v^2) H_\tau(y_1^{(1)})$.

Примененная нами схема требует 19 команд (с $(K + 32)$ по $(K + 47)$ и 3 команды $(K + 53) - (K + 55)$), в то время как схема, отличающаяся от данной тем, что $G(v^2)$ и $H_\tau(y_1^{(1)})$ подсчитывались бы отдельно, потребовала бы 24 команды (10 команд $(K + 32) - (K + 41)$, 2 команды на исправление команд $(K + 34)$ и $(K + 38)$ и аналогичная программа для $H_\tau(y_1^{(1)})$). Таким образом здесь мы сэкономили 5 команд.

Команда $(K + 49)$ образует $-E$. Команда $(K + 50)$ образует $u_1^{(1)}$ в соответствии с уравнением (1) и формулами (5), а $(K + 51)$ и $(K + 52)$ образуют $w^{(1)}$. Команды $(K + 53) - (K + 55)$ возвращают команды $(K + 34)$, $(K + 37)$, $(K + 38)$ в их первоначальное положение. Команда $(K + 56)$ посылает в ячейку ЗУ $K + 10$ команду, которая в дальнейшем

* Полагаем $p_3 = p_2 + 1$.

сотрется и поэтому в данный момент нас не интересует. Переходя к команде $(K + 58)$, мы исправляем команды $(K + 19)$, $(K + 20)$, $(K + 22)$, $(K + 50)$, $(K + 51)$ и $(K + 52)$ с тем, чтобы они, исходя из $u_2^{(1)}$, $w_2^{(1)}$ и $y_2^{(1)}$, подсчитали бы $u_2^{(1)}$ и $w_2^{(1)}$. Так как команды с $(K + 19)$ по $(K + 63)$ не изменяются, за исключением уже исправленных $(K + 34)$, $(K + 37)$, $(K + 38)$, то можно прямо переходить к команде $(K + 19)$. После подсчета $u_2^{(1)}$, $w_2^{(1)}$, $u_3^{(1)}$ и $w_3^{(1)}$, т. е. еще после двух циклов, мы переходим к команде $(K + 87)$: команды с $(K + 87)$ по $(K + 93)$ возвращают команды $(K + 19)$, $(K + 20)$, $(K + 22)$, $(K + 50)$, $(K + 51)$ и $(K + 52)$ в первоначальное состояние.

Теперь все готово для начала следующего итерационного цикла, т. е. вычисления $x_i^{(2)}$, $y_i^{(2)}$, $u_i^{(2)}$, $w_i^{(2)}$, $u_i^{'(2)}$ и $w_i^{'(2)}$ ($i = 1, 2, 3$). Первые три команды исправляют ячейки $K + 5$, $K + 10$ и K_4 , возвращая их в первоначальное состояние, и процесс повторяется без всяких изменений. Он повторяется N раз, т. е. до того момента, пока (p_5) не превысит (m_{10}) . Через N циклов переходом к команде $(K + 67)$ начинается основной процесс — вычисление по формулам (6) — (7). В целях сокращения программы мы используем для этого нового процесса те же команды $(K + 4) — (K + 56)$, что и для подсчета первоначальных значений.

III. Команды $(K + 67) — (K + 71)$ изменяют предшествующую им программу так, что она, как мы далее увидим, вычислит первое приближение для x_4 , y_4 , u_4 , w_4 , u_4' , w_4' по формуле (6). После этих команд мы имеем:

$K + 5$	\times	$a + 6$	$b + 13$	p_1
$K + 10$	$+$	$a + 22$	p_2	$a + 28$
$K + 13$	\geq	$K + 10$	K_{22}	$K + 4$
S_2		$- 25$	$- 4$	
S_3		$- 1$		$- 1$
K_5	\times	$a + 27$		

Команда $(K + 10)$ будет иметь указанный вид, благодаря команде $(K + 56)$ (она не сотрется).

Перейдем к команде $(K + 4)$. После первого цикла в ячейке p_2 образуется сумма

$$-\frac{3}{8}w'_0 + \frac{37}{24}w'_1 - \frac{59}{24}w'_2 + \frac{55}{24}w'_3$$

Затем, в соответствии с командой $(K + 10)$, получим первое приближение для w_4 :

$$w_4 = w_3 + h \left(-\frac{3}{8}w'_0 + \frac{37}{24}w'_1 - \frac{59}{24}w'_2 + \frac{55}{24}w'_3 \right)$$

При этом после подсчета последнего члена $\frac{55}{24}w'_3$ этого выражения команда $(K + 5)$ примет вид:

$$K + 5 \quad \left| \begin{array}{|c|c|c|c|} \hline \times & a + 30 & b + 17 & p_1 \\ \hline \end{array} \right|$$

После команд $(K + 11)$ и $(K + 12)$ команды $(K + 5)$ и $(K + 10)$ будут таковы:

$$\begin{array}{l} K + 5 \quad \left| \begin{array}{|c|c|c|c|} \hline \times & a + 5 & b + 13 & p_1 \\ \hline \end{array} \right| \\ K + 10 \quad \left| \begin{array}{|c|c|c|c|} \hline + & a + 21 & p_2 & a + 27 \\ \hline \end{array} \right| \end{array}$$

В соответствии с командой $(K + 13)$ возвратимся к команде $(K + 4)$ и начнем вычисление u_4 . Вычислив таким же образом y_4 и x_4 , мы перейдем непосредственно к команде $(K + 19)$ (так как $(S_3) < 0$). Команда $(K + 19)$ будет иметь вид:

$$K + 19 \quad \left| \begin{array}{|c|c|c|c|} \hline \times & a + 27 & a + 27 & p_1 \\ \hline \end{array} \right|$$

Это объясняется тем, что при последнем переходе через команду $(K + 66)$ мы не перешли к командам $(K + 87) - (K + 93)$, оставив $(K + 19)$ и другие команды в своем конечном состоянии. Однако именно эти значения команд $(K + 19)$, $(K + 20)$, $(K + 22)$, $(K + 50)$, $(K + 51)$ и $(K + 52)$ нам сейчас и нужны. Поэтому, мы приступаем непосредственно к подсчету u'_4 и w'_4 (команды $(K + 19) - (K + 55)$). Командой $(K + 56)$ возвращаем команду $(K + 10)$ в ее новое первоначальное положение, а благодаря команде $(K + 57)$, мы обходим серию команд, изменяющих $(K + 19)$ и другие команды.

Команды $(K + 73) - (K + 78)$ составляют группу команд контроля. Контроль производится по u и w , причем, если u_{n+1} (в данном случае u_4) и w_4 обе изменились достаточно мало, то считаем процесс на данном шаге законченным и переходим к следующему шагу. Так как разности, полученные в ячейках p_5 и p_6 после первой экстраполяции, достаточно велики, то мы переходим к команде $(K + 84)$, и, подставив значения

$K + 5$	\times	$a + 12$	$b + 17$	p_1
K_5	\times	$a + 33$		

возвращаемся к команде $(K + 4)$, начиная первый интерполяционный цикл. Этот цикл мало чем отличается от экстраполяционного, так что мы на нем подробно не останавливаемся.

Остановимся на другой возможности — когда обе разности, содержащиеся в ячейках p_5 и p_6 , достаточно малы, и мы переходим к команде $(K + 79)$. Команды с $(K + 79)$ по $(K + 82)$ сдвигают все значения ячеек с $(a + 7)$ до $(a + 30)$ на шесть мест назад, после чего мы возвращаемся к команде $(K + 70)$, и, установив соответствующие значения ячеек $K + 5$ и K_5 , начинаем вновь экстраполяционный процесс, высчитывая следующую точку траектории. Когда $(a + 14) < 0$ и, таким образом, точно просчитана по крайней мере одна точка с $y < 0$, процесс прекращается (команда $(K + 86)$) и мы переходим к командам восстановления $(K + 87) - (K + 99)$, не требующим особых пояснений. Отметим лишь, что, благодаря своему расположению, команды $(K + 87) - (K + 92)$ автоматически используются еще раз в конце программы.

В заключение сделаем несколько замечаний относительно других вариантов данной программы.

1) Программа без всякого труда видоизменяется на случай, когда берется за основу более точная (с большим количеством узлов) или менее точная (с меньшим количеством узлов) формула в методе Адамса. Точнее, в ней изме-

няются только некоторые параметры и увеличиваются (или уменьшаются) группы ячеек

$$a + 1 \div a + 30; b + 1 \div b + 20$$

2) Разностная форма записи метода Адамса нецелесообразна, так как более сложна и требует более длинной программы, а также (при фиксированном времени сложения и вычитания) и большего времени.

3) В целях сокращения программы мы привели самый простой метод контроля необходимого количества итераций при подсчете первоначальных значений. Методы, контролирующие разность n -го и $n + 1$ -го приближений и применяемые нами для контроля в остальных точках, более надежны, так как обеспечивают контроль всего процесса, включая точность, величину шага, отсутствие грубых ошибок и т. д. Следует отметить, что для применяемого нами метода итерации можно доказать, что наибольшие ошибки при подсчете первоначальных значений падают на значения в третьем узле, а поэтому достаточно вести контроль лишь по u_3 и w_3 .

4) Изменение программы на случай, когда за аргумент берется не t , а, например, x , не представляет особых трудностей. Почти вся программа осталась бы при этом неизменной.

5) Программа предназначена для расчета одной траектории. Тем не менее, так как она возвращает все команды в первоначальное положение, можно ей предпослать программу изменения u_0 и w_0 (или других параметров) от траектории к траектории. Такая программа могла бы автоматически вычислять целые пучки траекторий.

6) Можно также после программы добавить программу интерполяции и сглаживания окончательных результатов.

7) Аппроксимацию эмпирических функций G и H полиномами можно было бы заменить заданием их значений в отдельных точках, с тем чтобы программа подсчитывала эти функции при помощи интерполяции. Однако для такого задания функций потребовался бы больший объем ЗУ.

8) Вместо указанного в программе способа отсчета номера интервала, на котором следует считать $G\left(\frac{v^2}{\tau}\right)$ (отсчет номера с возвращением к первоначальным значениям), можно применить другой метод (сохранение предыдущего номера с изменением его на ± 1 при каждом переходе через конец интервала). Этот метод потребовал бы меньше времени, но удлинил бы программу, и, кроме того, следовало бы разделить программы вычисления $H_\tau(y)$ и $G\left(\frac{v^2}{\tau}\right)$.

9) Если нас интересует не только дальность полета снаряда, но также и его траектория, а объем ЗУ достаточно обширен, то можно поместить в него вычисленные точки траектории, вставив между командами $(K+82)$ и $(K+83)$ следующие 4 команды (с соответствующим последующим изменением номеров других ячеек $(K+i)$ при $i \geq 83$).

$K' + 1$	+	$a + 1$		$l + 1$
$K' + 2$	+	$K' + 1$	S_{20}	$K' + 1$
$K' + 3$	\geq	K'_1	$K' + 1$	$K' + 1$
$K' + 4$	+	$K' + 1$	S'_1	$K' + 1$

Здесь (K'_1) и (S'_1) равны:

K'_1	+	$a + 5$		
S'_1		-4		

Как легко видеть в этом случае, в $l+1$ и следующих ячейках будут размещены значения x , y , u и w для всех точек траектории, кроме нулевой.

Отметим еще, что при передаче значений в другие ячейки можно легко видоизменить программу, введя в нее удвоение промежутков интегрирования. Такой метод при небольшом увеличении программы привел бы к значительному сокращению времени работы машины.

10) Наконец, отметим, что почти во всех циклах программы используется одна и та же схема сравнения

с заранее заготовленной командой для того, чтобы цикл повторялся нужное число раз. Эта схема выгоднее, чем схема прибавления единицы в некоторую ячейку и сравнения накопленного числа с заранее заготовленным числом циклов. Тем не менее, в некоторых случаях (например, в команде $(K+13)$), ввиду того, что команду, с которой мы сравниваем, приходится часто менять, было бы, пожалуй, выгоднее принять второй вариант и в конечном счете несколько сократить как программу, так и количество ячеек K_i .

§ 33. ВЫЧИСЛЕНИЕ ТАБЛИЦЫ БЕССЕЛЕВЫХ ФУНКЦИЙ

Настоящая программа иллюстрирует одно из важных приложений универсальных цифровых машин — составление математических таблиц.

Приведенная ниже программа вычисления таблицы значений функции Бесселя $J_\nu(x)$ основана на представлении

$$J_\nu(x) = \sum_{m=0}^{\infty} u_m$$

где

$$u_m = \left(\frac{x}{2}\right)^{\nu+2m} \frac{(-1)^m}{m! \Gamma(m+1+\nu)}$$

и асимптотическом равенстве

$$J_\nu(x) \sim \sqrt{\frac{2}{\pi x}} \left[\cos\left(x - \frac{\pi\nu}{2} - \frac{\pi}{4}\right) \sum_{k=0}^{\infty} \nu_{2k} + \right. \\ \left. + \sin\left(x - \frac{\pi\nu}{2} - \frac{\pi}{4}\right) \sum_{k=0}^{\infty} \nu_{2k+1} \right]$$

где

$$\nu_k = (-1)^{\frac{k(k+1)}{2}} \frac{(\nu, k)}{(2x)^k}$$

$$(\nu, k) = \frac{(4\nu^2 - 1)(4\nu^2 - 3) \dots [4\nu^2 - (2k - 1)^2]}{2^{2k} k!}, \quad (\nu, 0) = 1$$

Для определенности мы принимаем $\nu = \frac{5}{6}$, однако, программа пригодна для вычислений с любым ν в пределах $0 \leq \nu \leq 1$.

Вычисления начинаются от нулевого значения аргумента x . После вычисления очередного значения функции это значение отсылается в печать (команда $(K + 16)$).

Под командами $(K + 8)$, $(K + 38)$, $(K + 39)$ и $(K + 40)$ следует понимать отсылку к соответствующим подпрограммам вычисления функции, указанной в графе операции. Если в машине осуществлено в виде специальной операции извлечение корня, то команда $(K + 40)$ оказывается обычной командой. Возведение x в дробную степень (команда $(K + 8)$) можно осуществить, вычисляя функцию $2^{\nu \lg_2 x}$.

При вычислении членов u_m ряда Тейлора используются рекуррентные соотношения $u_{m+1} = \frac{u_m x^2}{4\alpha_{m+1}}$, где $\alpha_m = -m^2 - m\nu$, и $\alpha_m = \alpha_{m-1} + \rho_m$, где $\rho_m = 1 - \nu - 2m$. Величины ρ_m последовательно вычисляются вычитанием двойки из величины $1 - \nu$, а величины α_m — последовательным прибавлением ρ_m к α_{m-1} .

Переход от вычислений ряда Тейлора к вычислениям по асимптотической формуле обеспечивается командой условного перехода $(K + 21)$.

При вычислении членов ν_k асимптотического разложения используется рекуррентное соотношение $\nu_{k+1} = (-1)^{k+1} \frac{\beta_{k+1} \nu_k}{(k+1)x}$, где $\beta_k = \frac{\nu^2}{2} - \frac{(2k-1)^2}{8}$, причем для вычисления β_k используется соотношение $\beta_k = \beta_{k-1} + \sigma_k$, $\sigma_k = 1 - k$. Величины σ_k получаем последовательным вычитанием единицы, а величины β_k — последовательным прибавлением σ_k к β_{k-1} . Так как множитель $\frac{\beta_k}{kx} = \frac{4\nu^2 - (2k-1)^2}{8kx}$ при $\nu < \frac{1}{2}$ возрастает по абсолютной величине с ростом k начиная с $k = 1$, а при $\nu > \frac{1}{2}$ убывает по абсолютной величине с ростом k от 1 до $\left[\nu + \frac{1}{2}\right]$ и возрастает по абсолютной величине с ростом k начиная с $k = \left[\nu + \frac{1}{2}\right] + 1$, то достаточно суммы $\Sigma \nu_{2k}$ и $\Sigma \nu_{2k+1}$ считать до тех пор, пока в одной из сумм не встретится такой член ν_k , что $|\nu_k| < \delta$, где δ — допустимая погрешность формул,

если только в каждой из сумм есть член, удовлетворяющий этому неравенству. Величину $X=4$ (содержимое $A+2$ ячейки ЗУ), начиная с которой счет происходит по асимптотической формуле, мы выбрали здесь так, чтобы

$$\max_k \left(\min_k |v_{2k}|, \min_k |v_{2k+1}| \right) < \delta \text{ при } \frac{x}{2} > X \text{ и}$$

$$\max_k |u_k| < 128 \text{ при } \frac{x}{2} < X$$

Последнее неравенство необходимо удовлетворить для того, чтобы вычисляемые члены умещались на тридцатидвухзначном двоичном счетчике (в предположении, что счет ведется с 25 двоичными знаками после запятой). Величину X надо менять в зависимости от ν .

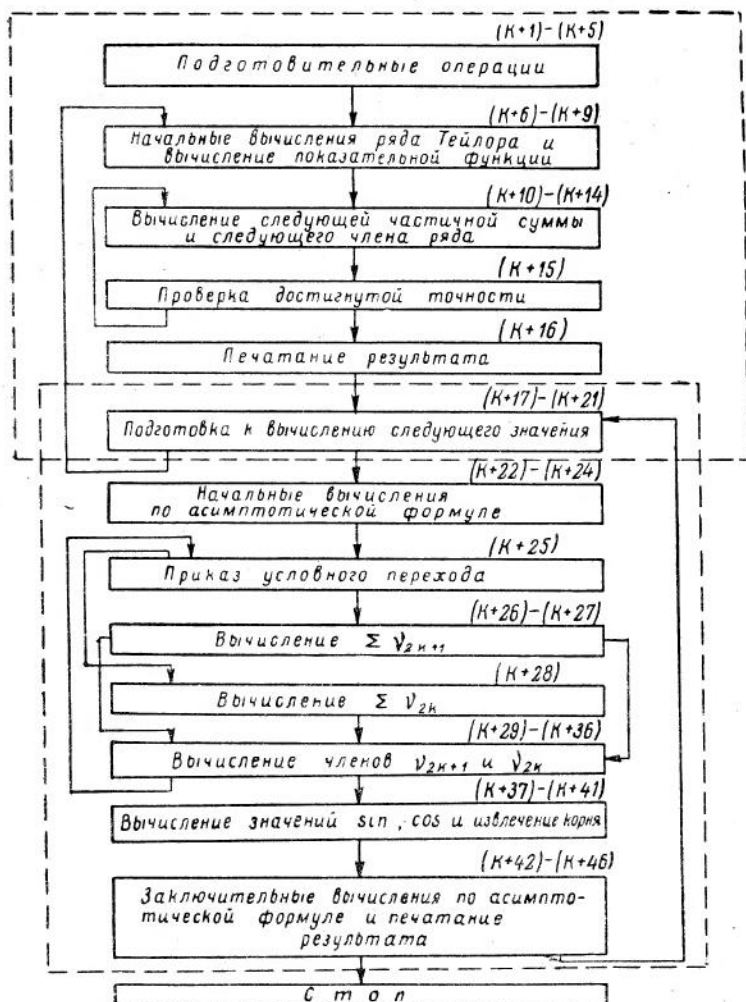
Команды, попавшие в верхний пунктирный прямоугольник, обеспечивают вычисления по ряду Тейлора. Команды, попавшие в нижний пунктирный прямоугольник, предназначены для вычислений по асимптотической формуле.

Содержание некоторых ячеек ЗУ

$A + 1$	$h/2$
$A + 2$	4
$A + 3$	δ
$A + 4$	2
$A + 5$	ν
$A + 6$	$\Gamma(1 + \nu)$
$A + 7$	1
$A + 8$	-1
$A + 9$	$1/8$
$A + 10$	$\pi/2$
$A + 11$	$\sqrt{2\pi}$
$A + 12$	X_ω
$A + 13$	$1/2$
$B + 1$	0

Начальное содержание остальных ячеек ЗУ ($B+2$, $B+3$, ... и т. д.) роли не играет.

Блок-схема программы



$K+1$	+	$A+13$	$A+5$	$B+20$
$K+2$	\times	$A+10$	$B+20$	$B+20$
$K+3$	\times	$A+5$	$A+5$	$B+21$
$K+4$	\times	$B+21$	$A+13$	$B+21$
$K+5$	—	$B+21$	$A+9$	$B+21$
$K+6$	—	$A+7$	$A+5$	$B+7$
$K+7$	\times	$B+1$	$B+1$	$B+2$
$K+8$	$()^y$	$B+1$		$B+4$
$K+9$:	$B+4$	$A+6$	$B+4$
$K+10$	+	$B+4$	$B+5$	$B+5$
$K+11$	—	$B+7$	$A+4$	$B+7$
$K+12$	+	$B+6$	$B+7$	$B+6$
$K+13$	\times	$B+2$	$B+4$	$B+4$
$K+14$:	$B+4$	$B+6$	$B+4$
$K+15$	$ \geq $	$B+4$	$A+3$	$K+10$
$K+16$	Π	$B+5$		
$K+17$	+	$B+1$	$A+1$	$B+1$
$K+18$	+			$B+5$
$K+19$	+			$B+6$
$K+20$	+			$B+7$
$K+21$	\geq	$B+1$	$A+2$	$K+6$
$K+22$	+	$B+1$	$B+1$	$B+2$
$K+23$	+	$A+7$		$B+12$
$K+24$	+	$A+7$		$B+10$
$K+25$	\geq	$B+10$	$A+7$	$K+28$
$K+26$	+	$B+12$	$B+6$	$B+6$
$K+27$	\geq			$K+29$
$K+28$	+	$B+12$	$B+5$	$B+5$
$K+29$	\times	$B+10$	$A+8$	$B+10$
$K+30$	—	$B+21$	$B+7$	$B+21$
$K+31$	+	$B+7$	$A+7$	$B+7$
$K+32$	\times	$B+21$	$B+10$	$B+11$
$K+33$:	$B+11$	$B+7$	$B+19$
$K+34$:	$B+19$	$B+2$	$B+3$
$K+35$	\times	$B+3$	$B+12$	$B+12$
$K+36$	$ \geq $	$B+12$	$A+3$	$K+25$
$K+37$	—	$B+2$	$B+20$	$B+50$

$K + 38$	\sin	$B + 50$		$B + 51$
$K + 39$	\cos	$B + 50$		$B + 52$
$K + 40$	$\sqrt{\quad}$	$B + 1$		$B + 2$
$K + 41$	$:$	$A + 11$	$B + 2$	$B + 2$
$K + 42$	\times	$B + 51$	$B + 6$	$B + 6$
$K + 43$	\times	$B + 52$	$B + 5$	$B + 5$
$K + 44$	$+$	$B + 5$	$B + 6$	$B + 5$
$K + 45$	\times	$B + 2$	$B + 5$	$B + 5$
$K + 46$	\geq	$A + 12$	$B + 1$	$K + 16$
$K + 47$				

ЛИТЕРАТУРА

1. Айкен Х. Х. и Хоппе Г. М. Автоматически управляемая вычислительная машина. Успехи математических наук, 1948, вып. 4, стр. 119—142.
 2. Быховский М. Л. Основы электронных математических машин дискретного счета. Успехи математических наук, 1949, вып. 3, стр. 69—124.
 3. Кудрявцев Л. Д. О принципах производства логических действий на вычислительных машинах. Успехи математических наук, 1950, вып. 3, стр. 104—127.
 4. Лебедев С. А. Быстродействующая электронная счетная машина АН СССР. Часть I. Общее описание машин, 1952.
 5. Муррей. Теория математических машин. Издательство, 1949.
 6. Хартрей. Современное развитие счетных машин. Переведено Г. Н. Б.
 7. Hamming. Error detecting and error correcting codes. T. Bell Tel. s. technical Journal, 26, 1950.
 8. Rutinchauer, Speiser, Stiefel. Programmgesteuerte digitale Rechengerate. Zeitschr. ang. Math. u. Phys., I. 1950. (Приведена обширная библиография.)
 9. Tompkins, Wakelin. High-speed computing Devices, 1950.
-

ОГЛАВЛЕНИЕ

Предисловие	3
Глава I. Цифровые машины и автоматизация вычислений . .	5
§ 1. Системы счисления	5
Цифровые машины. Двоичная и троичная системы. Двоично-десятичная и двоично-пятеричная системы.	
§ 2. Арифметические действия и их автоматизация	8
§ 3. Счетно-аналитические машины	12
§ 4. Автоматические вычислительные машины	16
§ 5. Быстродействующие автоматические машины	20
§ 6. Запоминающее устройство	22
Глава II. Операции над числами в цифровых машинах	27
§ 7. Представление чисел	27
Изображение знаков. Изображение чисел в машинах с фиксированной запятой. Представление чисел в машинах с плавающей запятой.	
§ 8. Поразрядные операции	36
Поразрядное дополнение. Поразрядное сложение. Пораз- рядное логическое сложение. Поразрядное (логическое) умножение. Перемена знака числа. Образование абсолют- ной величины числа. Изменение знака числа x в зави- симости от знака числа y . Передача числа из одних ча- стей машины в другие.	
§ 9. Операции сдвига и нормализации	41
Сдвиг. Операция нормализации числа.	
§ 10. Сложение и вычитание	44
Сложение положительных чисел в машинах с фиксиро- ванной запятой. Сложение чисел произвольного знака в машинах с фиксированной запятой. Указатель запрета. Вычитание. Сложение и вычитание чисел в машинах с плавающей запятой.	
§ 11. Умножение и деление чисел, заданных прямым кодом, и извлечение квадратного корня	55

Умножение при фиксированной запятой. Умножение чисел в машинах с плавающей запятой. Деление в машинах с фиксированной запятой. Деление в машинах с плавающей запятой. Извлечение квадратного корня. Операции с блоковой нормализацией.	
§ 12. Умножение и деление чисел, заданных дополнительным и обратным кодом	71
Умножение чисел, заданных дополнительным кодом. Умножение чисел, заданных обратным кодом. Деление чисел, заданных дополнительным и обратным кодом. Другие варианты умножения и деления чисел, заданных обратным кодом.	
§ 13. Применение двоичной системы с цифрами 1, -1 для умножения, деления и извлечения квадратного корня . .	90
Двоичная система (1,1). Умножение. Деление. Извлечение квадратного корня.	
§ 14. Другие операции	106
Взятие целой части числа. Операции с порядками. Операция сравнения.	
§ 15. Точность производства операций и округление результата	111
Операция сдвига. Изменение порядка и нормализация. Сложение и вычитание. Умножение. Деление.	
Глава III. Некоторые стандартные процессы при автоматических вычислениях	121
§ 16. Стандартные задачи	121
Нахождение сумм произведений. Перевод чисел из одной системы в другую. Операции с удвоенным числом знаков. Вычисление многочленов. Вычисление значений элементарных функций. Нахождение последовательных сумм и разностей. Решение уравнений.	
§ 17. Задание функций в машине	129
Табличное задание функции. Ввод функций, заданных аналитическими процессами. Итерационные методы. Аппроксимация функций.	
§ 18. Контрольные вычисления	138
Контроль, предусмотренный программой. Некоторые автоматические формы контроля: а) Контроль арифметических операций, б) Контроль передач.	
Глава IV. Программирование решений математических задач	145
§ 19. Программы и команды	145
Команды. Адреса. k -адресные коды. Последовательность выполнения команд. Элементарные операции. Пример программы.	

§ 20. Команды безусловного и условного перехода	156
§ 21. Непосредственное программирование	162
Решение квадратного уравнения с действительными коэффициентами. Решение кубического уравнения с действительными коэффициентами.	
§ 22. Преобразование команд	171
§ 23. Перечень команд и обозначений	177
§ 24. Программирование циклических процессов	182
§ 25. Составление более сложных программ из более простых программ и подпрограмм. Схема составления программ. Преобразование этапов программы. Объединение программ или этапов. Заключительные замечания.	193
§ 26. Программирование для случая уменьшенных возможностей машины	200
§ 27. Программы к задачам, рассмотренным в главе III	205
Вычисление многочленов. Вычисление последовательных сумм. Вычисление последовательных разностей. Метод сравнений. Преобразование чисел из одной системы в другую. Операции с удвоенным числом разрядов. Операции над векторами. Вычисление коэффициентов Фурье.	
Глава V. Программы для решения некоторых математических задач	
	232
§ 28. Вычисление функций	233
Нахождение $\log_2 x$ с помощью интерполяционной формулы. Вычисление $\sin x$. Вычисление $\Gamma(1+z)$. Вычисление функций разложением в непрерывные дроби.	
§ 29. Преобразование алгебраических выражений	250
§ 30. Решение систем линейных алгебраических уравнений прямыми методами	263
А. Метод главных элементов. Б. Метод ортогонализации.	
§ 31. Интегрирование обыкновенных дифференциальных уравнений методом Рунге-Кутты	294
§ 32. Интегрирование уравнений внешней баллистики	299
§ 33. Вычисление таблицы бесселевых функций	317

