50X1-HUM

INFORMATION REPORT INFORMATION REPORT

CENTRAL INTELLIGENCE AGENCY

S-E-C-R-E-T
NOFORN

50X1-HUM

COUNTRY USSR

REPORT

SUBJECT Photocopy of Book Entitled Sistema Standartnykh Podprogramm (on the Soviet M-2 computer)

DATE DISTR. 16 December 1959

NO. PAGES 1

REFERENCES RD

50X1-HUM

DATE OF INFO.

PLACE & DATE ACQ.

50X1-HUM

SOURCE EVALUATIONS ARE DEFINITIVE. APPRAISAL OF CONTENT IS TENTATIVE.

1. Russian-language book entitled Sistema Standartnykh Podprogramm (Standard Subprogramming System) by Ye. A. Zhogolev, G.S. Roslyakov, N.P. Trifonov, and M.R. Shura-Bura

50X1-HUM

2. This 230-page book was edited by M.R. Shura-Bura and was published by the State Publishing House of Physical-Mathematical Literature in Moscow, 1958. It describes some aspects of the construction, operation, and programming of the Soviet M-2 computer.

4. This book when detached from this report is UNCLASSIFIED.

50X1-HUM

S-E-C-R-E-T
NOFORN

STATE	x	ARMY	x	NAVY	x	AIR	x	NSA		FBI		NIO	1		

(Note: Washington distribution indicated by "X"; Field distribution by "#".)

INFORMATION REPORT INFORMATION REPORT

50X1-HUM

БИБЛИОТЕКА ПРИКЛАДНОГО АНАЛИЗА
И ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Е. А. ЖОГОЛЕВ, Г. С. РОСЛЯКОВ,
Н. П. ТРИФОНОВ, М. Р. ШУРА-БУРА

СИСТЕМА СТАНДАРТНЫХ ПОДПРОГРАММ

Под редакцией проф. М. Р. ШУРА-БУРА

ГОСУДАРСТВЕННОЕ ИЗДАТЕЛЬСТВО
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА 1958

Цена 6 р. 10 к.

ГОСУДАРСТВЕННОЕ
ИЗДАТЕЛЬСТВО
ФИЗИКО-МАТЕМАТИЧЕСКОЙ
ЛИТЕРАТУРЫ

«ФИЗМАТГИЗ»

БИБЛИОТЕКА ПРИКЛАДНОГО АНАЛИЗА
И ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Г. С. САЛЕХОВ. Вычисление рядов.

В. С. РЯБЕНЬКИЙ и А. Ф. ФИЛИППОВ. Об устойчивости разностных уравнений.

А. Н. ХОВАНСКИЙ. Приложение цепных дробей и их обобщений к вопросам приближенного анализа.

Г. С. ХОВАНСКИЙ. Номограммы с ориентированными транспарантами.

С. М. НИКОЛЬСКИЙ. Квадратурные формулы.

М. А. КАРЦЕВ. Арифметические устройства электронных цифровых машин.

Ю. В. ВОРОБЬЕВ. Метод моментов в прикладной математике.

Е. А. ЖОГОЛЕВ, Г. С. РОСЛЯКОВ, Н. П. ТРИФОНОВ, М. Р. ШУРА-БУРА. Система стандартных подпрограмм. Под ред. проф. М. Р. ШУРА-БУРА.

СИСТЕМА СТАНДАРТНЫХ ПОДПРОГРАММ

ПОД РЕДАКЦИЕЙ
ПРОФ. М. Р. ШУРА-БУРА

11-5-4

Библиотека выпускается под общим руководством кафедры вычислительной математики Московского государственного университета. Заведующий кафедрой акад. С. Л. Соболев.

АННОТАЦИЯ

Одним из путей автоматизации программирования является использование библиотеки стандартных подпрограмм. В настоящей книге приводится описание системы и библиотеки стандартных подпрограмм, разработанных в Вычислительном центре Московского государственного университета.

Книга рассчитана на лиц, работающих в области вычислительной математики, и учащихся, специализирующихся в этой области, а также на широкий круг читателей, интересующихся данными вопросами.

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Автоматизация работы по составлению программ для решения математических задач на быстродействующих автоматических цифровых вычислительных машинах является одной из основных проблем вычислительной математики.

Одним из путей автоматизации программирования является путь использования библиотеки стандартных подпрограмм. Сущность этого метода заключается в том, что для наиболее часто встречающихся алгоритмов раз и навсегда тщательно составляются подпрограммы, которые затем могут быть использованы в качестве готовых частей программы при решении любой конкретной задачи, где требуется реализация соответствующих алгоритмов. Это дает существенную экономию труда программиста.

Для большей эффективности этого метода требуется разработка определенной системы, обеспечивающей наиболее полную автоматизацию процесса подключения стандартных подпрограмм к общей программе и максимальные удобства их использования в различных задачах, а также наличие достаточно обширной библиотеки, подпрограммы которой удовлетворяют некоторым стандартным требованиям выбранной системы.

Предлагаемая книга в основном представляет собой описание системы стандартных подпрограмм, применявшейся в Вычислительном центре Московского университета в 1955—1956 годах.

Книга состоит из введения и двух частей.

Во введении изложены общие принципы устройства и работы быстродействующих цифровых вычислительных машин, а также основные понятия и приемы программирования для этих машин. Следующие главы помогут читателю на примере конкретной машины усвоить эти основные понятия и принципы.

В первой части дано краткое описание вычислительной машины М-2 Лаборатории управляющих машин и систем АН СССР, разработанной и построенной под руководством чл.-корр. АН СССР И. С. Брука, а также изложены особенности программирования и выбор системы стандартных подпрограмм для данной машины.

Во второй части (гл. IV—VII) приводятся некоторые подпрограммы из библиотеки, составленной применительно к выбранной системе.

Подпрограммы, приводимые в библиотеке, рассчитаны на конкретную машину, однако система в целом, так же как и алгоритмы, используемые в программах, могут быть применены и уже с успехом применяются на различных автоматических цифровых вычислительных машинах.

Книга рассчитана на читателей, работающих в области вычислительной математики, и учащихся, специализирующихся в этой области, а также на широкий круг читателей, интересующихся проведением расчетных работ на быстродействующих цифровых вычислительных машинах.

Подпрограммы, включенные в настоящую книгу, а также содержание книги обсуждались на заседаниях семинара, в работе которого, наряду с сотрудниками вычислительного центра, принимали участие академик С. Л. Соболев, профессор К. А. Семендяев и доцент И. С. Березин. На этих заседаниях был сделан ряд ценных замечаний, за которые мы глубоко благодарим всех участников семинара.

В написании главы VII кроме авторов принимали участие сотрудники вычислительного центра МГУ В. М. Васильев и Н. М. Ершова, которые составили программы, включенные в эту главу.

Авторы выражают также глубокую признательность Ю. М. Безбородову за большой труд по редактированию книги.

Е. А. Жоголев, Г. С. Росляков, Н. П. Трифонов, М. Р. Шура-Бура

ВВЕДЕНИЕ

§ 1. Состав и работа математической машины

Современные цифровые вычислительные машины используются главным образом для численного решения математических задач. Численное решение является частным случаем задачи переработки дискретной информации по заданным правилам. То обстоятельство, что значительную, если не подавляющую, часть перерабатываемой информации в этом частном случае составляют числа, накладывает, конечно, определенный отпечаток на характер переработки и, следовательно, на конструкцию современной цифровой машины. Однако основные черты такой машины определяются общей задачей автоматической переработки обширной дискретной информации по заданным правилам, т. е. задачей автоматической реализации алгоритмов.

Известно, что любой алгоритм может быть реализован как серия элементарных актов, каждый из которых принадлежит к не зависящему от данного алгоритма, заранее выбранному конечному набору актов переработки информации. Это обстоятельство позволяет получить универсальную машину, создав устройство, в котором может реализоваться любой акт переработки информации из такого конечного набора и в котором эти акты могут быть реализованы в любом наперед заданном порядке.

Автоматической такая универсальная машина может стать только в случае, если обеспечена возможность хранения в машине как всей исходной информации, так и информации, появляющейся в результате актов переработки.

Вся эта информация часто бывает весьма обширной. В составе исходной информации должны находиться, в частности, полные сведения об алгоритме, т. е. о перечне

элементарных актов переработки, выполнение которых приведет к решению задачи.

Информация в цифровых машинах задается при помощи упорядоченных цифровых последовательностей, так называемых *кодов*. Здесь широко распространено применение двоичных цифр, каждая из которых принимает одно из двух возможных значений. Конкретный смысл коды приобретают лишь в силу тех или иных соглашений. Ниже мы остановимся несколько подробнее на различных способах изображения числовой информации при помощи двоичных кодов.

Для ввода в машину всей исходной информации и вывода полученных результатов в составе современных машин предусмотрены специальные устройства ввода и вывода, позволяющие в значительной степени автоматизировать и ускорить эту часть работы.

В современных универсальных цифровых машинах функции переработки и хранения информации разделены между двумя весьма различными по конструкции частями машины. Переработка информации происходит в так называемом арифметическом устройстве, а хранение подавляющей части информации осуществляется в памяти машины. Само собой разумеется, что предусмотрены пересылки информации из памяти в арифметическое устройство и обратно.

Следует отметить, что значительный объем участвующей при решении многих задач информации, с одной стороны, и сравнительная сложность и невысокая надежность быстродействующей памяти, с другой стороны, приводит к тому, что в современных машинах, как правило, предусмотрено наличие памяти нескольких видов, отличающихся друг от друга быстродействием. Более быстрая память, непосредственно связанная с арифметическим устройством, называется внутренней или оперативной, а остальная часть — внешней. Разумные соотношения между объемами и скоростями работы устройств внешней и внутренней памяти позволяют добиться того, что меньшая по сравнению со скоростью работы внутренней памяти скорость работы внешней памяти, по существу, не снижает производительности машины.

Реализованные в машине пересылки естественным образом разбивают информацию на отдельные наборы цифр, т. е. коды, каждый из которых состоит из одного и того же фиксированного для данной машины числа цифр. Это число

называется *разрядностью* машины, а под машинным кодом или просто кодом обычно понимается код, число цифр которого равно разрядности машины.

Память машины удобно представлять себе в виде перенумерованных отдельных ячеек, в каждой из которых может быть записан произвольный код. Номер ячейки называется ее *адресом*. Акт выборки из какой-либо ячейки памяти кода и пересылка его в другое устройство машины, так же как и пересылка кода в память машины с записью его в ту или иную ячейку памяти, называется *обращением* к памяти по соответствующему адресу.

Упомянутые выше соглашения о смысле тех или иных наборов цифр касаются в первую очередь смысла машинных кодов. Часть этих соглашений, безусловно, находит свое отражение в конструкции машины. Код изображает как отдельное число, так и информацию о той или иной операции переработки, которую должна выполнить машина, т. е. *команду*.

Набор элементарных операций переработки информации, производимых машиной, в значительной степени определяется конструкцией арифметического устройства и наличием тех или иных связей между последним и памятью машины. Переработка информации в арифметическом устройстве заключается в выполнении операции над кодами, т. е. в выработке по заданным кодам новых. При этом объем как исходной, так и полученной в результате операции совокупности кодов ограничен.

Любой код и, следовательно, любая совокупность кодов, может рассматриваться как набор значений логических переменных, каждое из которых принимает одно из двух значений 0 или 1. Поэтому выполнение операций можно интерпретировать как определение значения некоторых логических функций от конечного числа логических переменных. Как известно, каждую из таких функций можно легко выразить через операции логического сложения, логического умножения и операцию отрицания, причем последняя может быть заменена сложением по модулю 2. Следовательно, принципиально возможно ограничиться набором из трех и даже двух операций над одноразрядными числами, так как, например, логическое сложение без труда выражается через логическое умножение и операцию отрицания. Однако практически

применение машины с таким скудным набором операций было бы весьма затруднительным. Дело в том, что в действительности весьма существенным оказывается число элементарных актов, на которые данный алгоритм распадается при реализации на машине и которые следует указать при описании алгоритма. Естественно, что наличие в наборе операций лишь элементарных логических операций приводит к тому, что описание даже сравнительно простых алгоритмов оказывается весьма громоздким. Поэтому в наборе элементарных операций машины предусмотрено наличие и более сложных операций. Такие более сложные операции переработки информации в арифметическом устройстве реализованы как совокупность элементарных логических операций. (В ряде случаев оказывается удобным пренебрегать этим расчленением и рассматривать операцию вместе со связанными с ней пересылками информации, как элементарную операцию.) Однако арифметическое устройство способно воспринимать и хранить одновременно лишь весьма ограниченный объем информации, поэтому в каждой операции участвует лишь несколько кодов. Это обстоятельство приводит к тому, что в наборе операций, выполняемых арифметическим устройством машины, не могут присутствовать слишком сложные операции. Иногда под термином элементарной операции мы будем понимать саму операцию переработки, выполняемую в арифметическом устройстве машины, игнорируя связанные с этой операцией пересылки информации.

Понятие элементарной операции тесно связано со способом задания команд. Рассматривая ниже понятие команды, мы уточним и понятие элементарной операции.

Как было указано выше, уже очень небольшой набор элементарных операций обеспечивает возможность реализации любого алгоритма. Однако в машине, предназначенной в основном для решения математических задач, такой набор должен обеспечивать в первую очередь достаточно простое выполнение арифметических операций над числами.

При решении сколько-нибудь сложных математических задач в связи с необходимостью переработки также и неарифметической информации, в первую очередь информации об алгоритме решения, возникает потребность в достаточно простой реализации некоторых вспомогательных операций, не являющихся арифметическими. Эта потребность в той или иной

степени удовлетворяется набором элементарных операций большинства современных цифровых машин.

Следует иметь в виду, что термин элементарная операция определяется исключительно конструкцией машины. Например, операция деления двух чисел на машине М-2 является элементарной операцией, а на машине «Стрела» деление осуществляется умножением делимого на обратную величину делителя; обратная же величина находится при помощи имеющегося в машине набора операций, в котором отсутствует операция деления. В некоторых машинах среди элементарных операций присутствует операция извлечения квадратного корня, в большинстве же машин извлечение корня приходится проводить при помощи серии элементарных операций.

Каждой элементарной операции данной машины присваивается некоторый номер или код операции.

Выполнение машиной конкретной элементарной операции обусловливается тем, что в устройство управления поступает код, изображающий соответствующую команду. Формально команда есть произвольный код, поступивший в устройство управления. Команды поступают в устройство управления из памяти. Автоматическая смена команд обеспечивается тем обстоятельством, что при выполнении любой элементарной операции наряду с прочим производится информация о месте памяти, из которого должен поступить код, изображающий следующую команду, т. е. адрес следующей команды, и в каждую элементарную операцию входит акт выборки из памяти соответствующего кода в качестве команды.

Элементарная операция есть совокупность всех актов, возникающих в машине по заданной команде. Время выполнения элементарной операции называется *тактом*. В понятие элементарной операции включается и обращение к памяти, т. е. выборка или запись кодов по тем или иным адресам. В большинстве случаев эти адреса указываются в команде. Однако некоторые из адресов обращения могут быть не указаны в коде команды. К таким адресам относится в большинстве машин в первую очередь адрес выборки следующей команды. Имеются, в частности на машине М-2, и другие случаи обращения к ячейкам памяти, адреса которых явно не указаны в коде команды. Кроме указаний адресов обращения, в команде помещается код операции, определяющий

характер работы машины при выполнении данной команды. Код операции в сущности задает ту или иную элементарную операцию, остальная же часть команды представляет собой как бы дополнительные сведения, конкретизирующие работу по заданной команде. Как для кода операции, так и для адресов обращения в коде команды отводятся раз навсегда фиксированные группы разрядов. Число групп, отведенных в команде для адресов обращения, называется *адресностью* машины. Эти группы разрядов перенумерованы и называются, соответственно, первым адресом, вторым адресом и т. д.

Группа разрядов, отведенных для номера операции, называется *кодом операции*. Следует обратить внимание на то, что адресом называется и номер ячейки запоминающего устройства, и часть кода команды, а кодом операции называется и номер операции, и часть кода команды, в которой фиксируется этот номер. Эта двойственность терминологии обычно не приводит к недоразумению, так как из существа дела, как правило, ясно, о чем идет речь. Однако этот двойной смысл терминов все же следует иметь в виду. Отметим кстати, что в некоторых командах в том или ином адресе может помещаться информация, отличная от адресов обращения, тем не менее мы будем говорить о ней, как об адресе команды.

Особого рассмотрения заслуживают способы указания адреса выборки следующей команды. В большинстве машин для экономии информации, записываемой в память машины, выборка команды производится по адресу, находящемуся в специальном счетчике, к которому при выполнении каждой элементарной операции прибавляется единица, в результате чего, кроме случаев выполнения особых операций изменения содержимого этого счетчика, следующая команда выбирается из ячейки памяти, имеющей адрес на единицу больше адреса ячейки, из которой была выбрана выполненная команда. Выборка команды из ячейки, отличной от следующей, может происходить при этом способе лишь в результате выполнения операций, изменяющих нужным образом содержимое этого счетчика, так называемых команд перехода или передачи управления. При помощи таких команд можно задать любой порядок выборки команд из памяти машины.

В ряде случаев оказывается целесообразным перейти к выполнению той или иной команды в зависимости от значений тех или иных кодов, т. е. осуществить передачу управления в зависимости от выполнения некоторых условий. Для более простого осуществления такой передачи в составе команд машины обычно предусмотрены специальные операции, изменяющие счетчик выборки команд в зависимости от выполнения определенных условий. Наличие таких элементарных операций условного перехода (условных передач управления) упрощает программирование и в большинстве случаев существенно экономит число элементарных операций, реализующих на машине какой-либо алгоритм. В отличие от условных передач управления операцию передачи управления, выполняемую независимо от каких-либо условий, мы будем называть безусловным переходом или безусловной передачей управления.

В команде перехода должна содержаться информация о необходимом изменении содержания счетчика выборки команд. Для команды безусловного перехода достаточно в одном из адресов указать адрес выборки следующей команды, т. е. новое желаемое состояние счетчика.

Без существенных осложнений программирования можно ограничиться операциями условного перехода, результатом выполнения которых является одно из двух возможных состояний счетчика выборки команд. При этом можно условиться, что одно из этих двух состояний есть номер, полученный путем стандартного увеличения содержимого счетчика выборки команд на единицу и лишь второе из возможных состояний может быть задано произвольным. Результатом такого соглашения является экономия информации в командах условного перехода, так как в заметном большинстве случаев одной из двух желаемых команд является следующая по номеру команда.

Второе из возможных состояний счетчика выборки команд в явном виде задается в команде условного перехода, занимая в ней один адрес. В многоадресной машине с такими операциями передач управления часть адресов оказывается свободной, что позволяет совместить операции изменения состояния счетчика с другими операциями, например с пересылками информации из одного места памяти в другое или

с операцией выработки признака, определяющего тот или иной характер изменения счетчика выборки команд в операции условного перехода.

§ 2. Основные понятия и приемы программирования

Для обеспечения автоматической работы машины необходимо приготовить в соответствующей форме всю исходную информацию. Выше уже отмечалось, что в составе последней должна находиться полная информация об алгоритме решения. Информация об алгоритме называется *программой*. Программа должна быть точным и совершенно полным описанием алгоритма решения данной задачи в терминах операций машины. Однако простое перечисление в программе всей последовательности элементарных операций, которую должна выполнить машина для решения той или иной задачи, т. е. задание в явной форме отдельной команды для каждой элементарной операции, входящей в состав алгоритма, свело бы на нет эффективность применения быстродействующих вычислительных машин для решения математических задач.

Дело в том, что для решения всякой задачи необходимо ввести в память машины исходный материал, в том числе и программу. Для быстродействующих машин время ввода одной команды во много раз больше времени выполнения этой команды машиной, поэтому если каждая команда программы в процессе решения задачи будет выполняться только один раз, то быстродействие машины будет в значительной степени потеряно. Кроме того, такой способ составления программ вряд ли позволил бы сократить ручной труд человека, подготовляющего задачу для решения на машине, по сравнению с трудом вычислителя, за исключением, может быть, того случая, когда по составленной программе производится серийное решение однотипных задач. Поэтому описание алгоритма решения задачи должно быть более компактным.

С другой стороны, для решения задач на машинах необходимо выбирать такие алгоритмы, которые можно описать компактно с помощью имеющихся элементарных операций машины.

Вообще говоря, не для всякой задачи удается составить такую программу, число команд в которой было бы меньше

числа потребных для решения этой задачи операций. Однако это, как правило, очень простые задачи, требующие для своего решения небольшого числа операций. Для решения же многих реальных задач можно выбрать такой алгоритм, в котором бы циклически повторялись какие-либо последовательности операций. Это дает возможность составить программу, число команд в которой во много раз меньше числа операций, необходимых для решения данной задачи. Особенности алгоритмов решения приводят к ряду понятий и приемов программирования, на которых мы коротко остановимся.

Наличие в большинстве математических алгоритмов циклически повторяющихся последовательностей операций приводит к тому, что основным приемом сокращения числа команд в программах является составление программ из циклов. Под *циклом* мы понимаем часть программы, которая не только обеспечивает счет по какой-либо формуле, но и заставляет машину повторить этот процесс столько раз, сколько нам нужно.

При осуществлении того или иного цикла мы не имеем целью многократно получать один и тот же результат — нас интересует применение данного процесса ко многим различным числам, или вообще кодам, для получения все новых и новых результатов, причем каждый из них нам часто приходится запоминать на более или менее продолжительный период времени в разных ячейках оперативной памяти. Это достигается в различных циклах по-разному. Некоторые циклы в качестве исходного числа (или группы чисел) для очередной серии операций используют результат (или группу результатов) предыдущей серии тех же операций. В программе, реализующей такой цикл, достаточно результат каждой серии помещать в те же ячейки, в которых находились исходные данные, и обеспечить нужное число повторений. Такие циклы называются *итерационными*.

В других задачах приходится последовательно применять какую-либо серию операций к группе различных чисел. Для этого необходимо производить предварительную переработку некоторых команд перед их очередным выполнением, например изменить адрес, по которому нужно запомнить полученный результат. Команды, обеспечивающие такие изменения, также включаются в цикл. Команды, которые перерабатываются в процессе вычислений, называются *переменными*,

а операция изменения переменных команд называется *переадресацией*. Цикл с переменными командами, меняющимися при помощи команд цикла, можно назвать циклом с переадресацией. Перед обращением к такому циклу все переменные команды должны быть приведены в первоначальное состояние, т. е. должны быть *восстановлены*. Иногда вид этих команд зависит каким-либо образом и от других частей программы; тогда перед обращением к такому циклу переменные команды должны быть *сформированы* в соответствующем виде. Восстановление переменных команд по ряду причин удобно производить даже в том случае, если обращение к данному циклу в процессе решения задачи производится всего лишь один раз.

Успешное решение всех этих вопросов во многом зависит от системы элементарных операций машины. С точки зрения компактности программы выгодно, чтобы машина имела возможно больший набор элементарных операций, как арифметических, так и вспомогательных. Но добавление каждой новой элементарной операции усложняет конструкцию машины, принципиально не увеличивая ее возможности, так как и при небольшом наборе элементарных операций машина может быть универсальной. Конечно, задача составления программы усложняется при уменьшении набора элементарных операций, так как алгоритм, вообще говоря, будет сводиться к бо́льшему числу операций.

Нельзя думать, что компактность программы является единственным критерием того, насколько удачно она составлена. Существуют алгоритмы, которые можно описать сжато в элементарных операциях машины. Однако они не всегда находят применение при решении задач на машинах. Большое значение при составлении программы имеет и то обстоятельство, насколько быстро будут получены интересующие нас результаты по выбранному алгоритму. Для небольших задач, время решения которых сравнимо со временем ввода необходимой информации, это еще не играет большой роли, но уже для более крупных задач часто приходится выбирать заве́домо более громоздкий алгоритм, лишь бы он обеспечил бо́льшую скорость вычислительного процесса. Кроме того, выбранный нами алгоритм должен обеспечить требуемую точность вычислений. Как правило, повышение точности ведет либо к увеличению количества команд программы,

либо к замедлению вычислительного процесса, а иногда и к тому, и к другому.

Все эти вопросы настолько тесно связаны между собой, что правильные ответы на них можно дать лишь в каждом конкретном случае с учетом особенностей данной машины и характера решаемой задачи. Однако даже для конкретной задачи правильное решение этих вопросов и составление удачной программы является сложным делом, требующим затраты большого труда. Поэтому очень большое значение приобретают приемы, позволяющие экономить усилия при составлении программы, т. е. приемы, облегчающие программирование. Одним из таких приемов является прием разбиения сложной задачи на отдельные более простые части и программирование этих частей более или менее независимо друг от друга.

При независимом составлении частей программы большое значение приобретает правильное распределение между ними объема памяти машины. Разделение памяти между частями программы без их взаимного пересечения является самым простым решением вопроса. Однако такое разделение, как правило, оказывается невыполнимым из-за ограниченности объема памяти. Поэтому необходимо проводить более экономное распределение памяти. Если мы назовем *оперативной памятью программы* (или ее части) совокупность всех ячеек памяти, в которых записывается как программная, так и числовая информация, необходимая для работы программы, включая все промежуточные и окончательные результаты, то задачу рационального распределения памяти можно сформулировать как задачу правильного совмещения оперативной памяти различных частей программы.

В состав оперативной памяти программы всегда входят ячейки для хранения промежуточных результатов вычислений. Совмещение этой части оперативной памяти для различных кусков программы не вызывает никаких затруднений и широко применяется. Ячейки памяти, принадлежащие к этой части оперативной памяти программы, называются *рабочими ячейками*. Так как рабочие ячейки нужны при решении любой задачи, то целесообразно постоянно отводить для них одни и те же ячейки памяти, занимая другие ячейки только в том случае, когда требуется большое количество рабочих ячеек.

439-О

Прием программирования сложных задач путем разбиения их на более простые, по возможности независимые, части часто имеет и то преимущество, что программы для этих отдельных частей можно использовать при решении различных задач. С целью такого использования программ целесообразно делать эти программы более универсальными. В том случае, когда в распоряжении программиста имеется достаточно широкий набор таких программ, составление новой программы может в значительной мере свестись к простому их соединению. Развитием этого приема является метод программирования с помощью стандартных подпрограмм.

Применение стандартных подпрограмм позволяет как бы расширить набор элементарных операций машины без усложнения ее конструкции. Это обстоятельство делает метод особенно желательным в применении к машине со сравнительно бедным набором элементарных операций. Однако и для машин с широким набором операций его значение трудно переоценить, так как даваемое им расширение набора операций остается очень существенным и в этом случае. Аппаратный метод расширения набора элементарных операций вряд ли может конкурировать с методом расширения этого набора программным путем. Следует иметь в виду, что метод стандартных подпрограмм является также одним из возможных путей автоматизации программирования.

§ 3. Двоичная система изображения чисел

Любое положительное целое число N при фиксированном n однозначно представимо в виде суммы

$$N=\sum_{j=0}^{n-1}\varepsilon_j 2^j,$$

где $\varepsilon_j=0;1$ и n настолько велико, что $N<2^n$.

В силу этого представления между целыми числами в интервале

$$0\leqslant N<2^n$$

и упорядоченными последовательностями из n нулей и единиц, т. е. двоичным кодом $\varepsilon_{n-1}\varepsilon_{n-2}\ldots\varepsilon_1\varepsilon_0$ устанавливается взаимно однозначное соответствие.

Величины ε_j, $j=0,\ldots,n-1$, называются цифрами n-значного двоичного представления числа N. Если $2^{n-1}\leqslant N<2^n$, то $\varepsilon_{n-1}=1$. В случае же, если $2^{k-1}\leqslant N<2^k<2^n$, то $\varepsilon_{n-1}=\varepsilon_{n-2}=\ldots=\varepsilon_k=0$, но $\varepsilon_{k-1}=1$; тогда ε_{k-1} называется старшей значущей цифрой, а k-значным двоичным представлением такого числа N будет последовательность $1\varepsilon_{k-2}\varepsilon_{k-3}\ldots\varepsilon_1\varepsilon_0$. Число N называется k-значным, и любое его двоичное представление имеет вид

$$00\ldots01\,\varepsilon_{k-2}\varepsilon_{k-3}\ldots\varepsilon_1\varepsilon_0.$$

Во многих случаях удобно игнорировать нули, предшествующие старшей значущей цифре, отождествляя все представления числа N с самим числом.

Количество цифр двоичного представления примерно в 3,3 раза больше количества обычных десятичных цифр. Так для изображения чисел в интервале от нуля до тысячи требуются десятизначные двоичные или трехзначные десятичные числа.

К двоичному изображению чисел применимы обычные правила арифметики, с той разницей, что очень сильно сокращаются необходимые таблицы сложения и умножения. При сложении двух чисел достаточно пользоваться правилами $0+0+0=0$, $1+0+0=1$, $1+1+0=10$ и $1+1+1=11$. Здесь третья цифра представляет собой цифру переноса из младшего разряда в случае получения там двухзначного ответа. Так, например, для сложения $47+51$ в двоичной системе воспользуемся двоичными разложениями слагаемых: $47=101111$ и $51=110011$.

$$\begin{array}{r} 111111 \\ 101111 \\ +\quad 110011 \\ \hline 1100010=2^6+2^5+2^1=98. \end{array}$$

Над цифрами первого слагаемого отмечены единицы переносов. Таблица умножения в двоичной системе сводится к правилам

$$0\times0=0,\qquad 0\times1=0,\qquad 1\times1=1.$$

Заметим, что степени двойки изображаются единицей с последующими нулями: $\underbrace{100\ldots\ldots0}_{k\text{ нулей}}=2^k$. Поэтому умножение

на степень двух сводится к приписыванию справа соответствующего числа нулей. Отсюда следует, что при умножении многозначных двоичных чисел частичные произведения множимого на цифры множителя либо равны нулю, либо получаются из множимого приписыванием соответствующего числа нулей правее младшей цифры множимого. Например, умножение $5 \times 6 = 30$ выглядит в двоичной системе следующим образом:

$$\begin{array}{r} 101 \\ \times\ 110 \\ \hline 000 \\ 1010 \\ 10100 \\ \hline 11110 \end{array} = 2^4 + 2^3 + 2^2 + 2^1 = 30.$$

Обычный способ деления сильно упрощается в двоичной системе благодаря тому, что выбор очередной цифры частного следует провести всего лишь из двух возможных цифр 0 и 1.

Цифры двоичного разложения числа N можно получить при помощи следующего алгоритма последовательного деления на два. Пусть

$$\begin{aligned} N = N_0 &= 2N_1 + \varepsilon_0 \\ N_1 &= 2N_2 + \varepsilon_1 \\ N_2 &= 2N_3 + \varepsilon_2 \\ &\cdots\cdots\cdots\cdots \\ N_{k-3} &= 2N_{k-2} + \varepsilon_{k-3} \\ N_{k-2} &= 2N_{k-1} + \varepsilon_{k-2} \\ N_{k-1} &= 2 \cdot 0 + \varepsilon_{k-1} \neq 0, \end{aligned} \qquad (*)$$

где $N_1, N_2, \ldots, N_{k-1}$ — положительные целые числа, а ε_j — равные нулю или единице остатки от деления на два. Так как для $N_j > 1$ имеем $N_j > N_{j+1} \geqslant 1$, то для любого $N = N_0$ существуют последовательности $N_1 N_2 \ldots N_{k-1}$ и $\varepsilon_{k-1}\varepsilon_{k-2} \ldots \varepsilon_0$ (где $N_{k-1} = \varepsilon_{k-1} = 1$), удовлетворяющие соотношениям (*). Последовательность $\varepsilon_{k-1}\varepsilon_{k-2} \ldots \varepsilon_0$ оказывается двоичным представлением числа N, так как индукцией по k получаем:

$$N = \varepsilon_{k-1}2^{k-1} + \varepsilon_{k-2}2^{k-2} + \ldots + 2\varepsilon_1 + \varepsilon_0.$$

Задача получения двоичного разложения числа N, заданного обычными десятичными цифрами, сводится к вычислению значения полинома в том случае, когда мы желаем воспользоваться средствами двоичной арифметики и заранее знаем двоичные разложения десятичных цифр. Действительно, если $\delta_{s-1}\delta_{s-2} \ldots \delta_1\delta_0$ — десятичные цифры числа N, то

$$N = \delta_{s-1} \cdot 10^{s-1} + \delta_{s-2} \cdot 10^{s-2} + \ldots + \delta_1 \cdot 10 + \delta_0 =$$
$$= (\ldots((\delta_{s-1} \cdot 10 + \delta_{s-2}) \cdot 10 + \delta_{s-3})10 + \ldots + \delta_1) \cdot 10 + \delta_0.$$

Для получения двоичного разложения последнего выражения достаточно воспользоваться следующей таблицей двоичных разложений:

	Двоичное разложение		Двоичное разложение
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001
		10	1010

(**)

и провести все требуемые по формуле арифметические операции в двоичной системе счисления.

Задача получения цифр десятичного изображения числа N, заданного своим двоичным разложением, может быть решена путем подсчета суммы

$$\varepsilon_k 2^k + \varepsilon_{k-1}2^{k-1} + \ldots + 2\varepsilon_1 + \varepsilon_0 =$$
$$= (\ldots(2\varepsilon_k + \varepsilon_{k-1})2 + \ldots + \varepsilon_1)2 + \varepsilon_0,$$

если мы проведем эти вычисления в десятичной системе счисления. В случае, если мы желаем воспользоваться средствами двоичной арифметики, десятичные цифры числа N, точнее говоря, двоичные разложения этих цифр, могут быть

получены как остатки при выполнении последовательных делений на десять. Действительно, пусть

$$\begin{aligned} N = N_0 &= (1010)\,N_1 + \delta_0 \\ N_1 &= (1010)\,N_2 + \delta_1 \\ &\cdots\cdots\cdots\cdots \\ N_{s-2} &= (1010)\,N_{s-1} + \delta_{s-2} \\ N_{s-1} &= (1010)\,0 + \delta_{s-1} \neq 0, \end{aligned}$$

где $N_1, N_2, \ldots, N_{s-1}$ — положительные целые числа, а δ_j ($j = 0, 1, \ldots, s-1$) неотрицательные целые числа, меньшие десяти. В этих формулах мы записали число десять при помощи его двоичного разложения 1010 для того, чтобы подчеркнуть, что все операции проводятся в двоичной системе счисления. Каждое из заданных в двоичной системе счисления чисел δ_j, $j = 0, \ldots, s-1$, изображается одной из десятичных цифр в соответствии с таблицей (*).

Полученные при помощи этой таблицы десятичные цифры являются цифрами десятичного разложения числа N.

Заметим, что для каждой из цифр δ_j мы получали прежде всего ее двоичное разложение. Если мы воспользуемся четырехзначным двоичным разложением $\varepsilon_{j3}\varepsilon_{j2}\varepsilon_{j1}\varepsilon_{j0}$ для каждой из цифр δ_j, то последовательность $4s$ двоичных цифр

$$\varepsilon_{s-1,3}\varepsilon_{s-1,2}\varepsilon_{s-1,1}\varepsilon_{s-1,0}\varepsilon_{s-2,3}\cdots\varepsilon_{03}\varepsilon_{02}\varepsilon_{01}\varepsilon_{00}$$

представляет собой так называемое *двоично-десятичное* представление числа N.

Этот способ изображения чисел применяется для ввода десятичных чисел в двоичную машину, а также для вывода результатов из двоичной машины в десятичной системе счисления. Кроме того, довольно широко распространены машины, в которых для изображения чисел принята эта система на всех этапах работы. В таких машинах не всякий набор двоичных цифр

$$\varepsilon_{4s-1}\varepsilon_{4s-2}\cdots\varepsilon_1\varepsilon_0$$

оказывается допустимым. Необходимо выполнение условия

$$8\varepsilon_{4k+3} + 4\varepsilon_{4k+2} + 2\varepsilon_{4k+1} + \varepsilon_{4k} \leqslant 9$$

для $k = 0, 1, \ldots, s-1$.

Наряду с системой счисления, имеющей основанием 2, можно рассматривать систему, основанием которой является

любое целое число $p \geqslant 2$. В качестве цифр такой системы можно принять целые числа $0, 1, \ldots, p-1$. Каждому целому числу N можно поставить в соответствие такую конечную последовательность выбранных цифр

$$\beta_{s-1}\beta_{s-2}\cdots\beta_1\beta_0,$$

что

$$N = \beta_{s-1}p^{s-1} + \beta_{s-2}p^{s-2} + \ldots + \beta_1 p + \beta_0.$$

Указанная последовательность $\beta_{s-1}\ldots\beta_0$ называется представлением числа N в системе счисления с основанием p (p-код). При условии $\beta_{s-1} \neq 0$ представление определяется однозначно. Перевод из одной системы счисления в другую, т. е. получение представления числа N в той или иной системе счисления при условии, что задано представление этого числа в какой-либо другой системе, производится совершенно аналогично переводам из десятичной системы в двоичную и обратно.

Аналогично рассмотренной выше двоично-десятичной системе можно использовать систему счисления с основанием p, применяя для изображения каждой из цифр систему счисления с основанием $q < p$. Особого внимания заслуживает случай, когда $q^l = p$, где l — целое положительное число. В этом случае каждая из цифр $0, 1, 2, \ldots, (p-1)$ изображается при помощи l цифр из набора $0, 1, \ldots, (q-1)$, и любые l цифр из этого набора изображают одну из цифр $0, 1, 2, \ldots, (p-1)$. Таким образом любой sl-значный код из цифр системы с основанием q можно рассматривать как s-значный p-код.

Оба кода, как легко видеть, изображают одно и то же число, и установленное соответствие между sl-значными q-кодами и s-значными p-кодами оказывается взаимно однозначным: s-значный p-код можно рассматривать как сокращенную запись sl-значного q-кода. Такой сокращенный способ записи широко применяется в отношении двоичных кодов. В основном используются случаи, когда $p = 2^3$ или $p = 2^4$.

Условимся обозначать 10, 11, 12, 13, 14 и 15 соответственно буквами a, b, c, d, e и f. Тогда 12-разрядный двоичный код

111100101001

изобразится трехзначным шестнадцатеричным кодом

f29

или четырехзначным восьмеричным кодом

7451.

Рассмотренные системы цифрового изображения чисел могут быть обобщены следующим образом. Выберем возрастающую последовательность целых положительных чисел $1 = q_0, q_1, \ldots, q_k, \ldots$, каждый следующий член которой является целым кратным предыдущего. Тогда любое целое число N может быть представлено в виде суммы

$$N = \gamma_{s-1}q_{s-1} + \gamma_{s-2}q_{s-2} + \cdots + \gamma_1 q_1 + \gamma_0,$$

в которой коэффициенты γ_k для $k = 0, \ldots, s-1$ являются целыми числами, удовлетворяющими условиям

$$0 \leqslant \gamma_k \leqslant \frac{q_{k+1}}{q_k} - 1, \qquad k = 0, 1, \ldots, s-1.$$

Последовательность $\gamma_{s-1}\gamma_{s-2}\cdots\gamma_1\gamma_0$ может рассматриваться как представление числа N в системе счисления, определенной числами $\{q_k\}$.

При условии $\gamma_{s-1} \neq 0$ последовательность коэффициентов $\gamma_{s-1}\gamma_{s-2}\cdots\gamma_1\gamma_0$ однозначно определяется числом N.

Системы счисления с основанием p в обычном смысле этого слова являются частным случаем рассмотренной общей системы, определяемой числами $q_k = p^k$, $k = 0, 1, \ldots$

При использовании системы, определяемой числами $\{q_k\}$, можно изображать все возникающие «цифры» γ_k при помощи одной и той же системы счисления с каким-либо постоянным основанием p. Каждая «цифра» γ_k заменяется набором цифр системы с основанием p, и поэтому вместо кода $\gamma_{s-1}\gamma_{s-2}\cdots\gamma_1\gamma_0$ возникает код $\beta_{r-1}\beta_{r-2}\cdots\beta_1\beta_0$, где $0 \leqslant \beta_k \leqslant p-1$, $k = 0, \ldots, r-1$. В том случае, когда для каждого индекса k цифру γ_k независимо от ее значения условимся изображать в системе с основанием p одним и тем же количеством цифр, мы будем иметь возможность по возникшему коду $\beta_{r-1}\beta_{r-2}\cdots\beta_1\beta_0$ однозначно восстановить код $\gamma_{s-1}\gamma_{s-2}\cdots\gamma_1\gamma_0$.

В частном случае если $q_{k+1} : q_k = p^{r_k}$, где r_k — целое положительное число, и мы условимся использовать r_k-значное

представление цифры γ_k в системе счисления с основанием p, то между r-значными p-кодами, где $r = r_{s-1} + r_{s-2} + \cdots + r_1 + r_0$ и s-значными кодами $\gamma_{s-1}\gamma_{s-2}\cdots\gamma_1\gamma_0$, где $0 \leqslant \gamma_k \leqslant p^{r_k} - 1$, $k = 0, \ldots, s-1$, установится естественное взаимно однозначное соответствие. Код $\gamma_{s-1}\gamma_{s-2}\cdots\gamma_1\gamma_0$ можно использовать как сокращенную запись r-значного p-кода $\beta_{r-1}\beta_{r-2}\cdots\beta_1\beta_0$.

Такой способ сокращенной записи для двоичных кодов применяется, например, в том случае, когда значность кода не позволяет разбить его достаточно удобно на некоторое число частей равной длины. Например, если в рассмотренном выше 12-разрядном двоичном коде

111100101001

по тем или иным причинам желательно выделить младший разряд, то остальные 11 разрядов нельзя было бы разбить на группы равной длины, отличной от единицы или одиннадцати. Если выделить справа две группы по четыре разряда и оставшиеся три разряда разбить на две группы из двух и одного разряда, то это будет соответствовать системе счисления, порожденной последовательностью 1, 2, 32, 512, 2048, 4096, ... Рассматриваемый двоичный код запишется цифрами

13941,

где следует иметь в виду, что крайняя правая цифра 1 изображает один двоичный разряд, а следующая за ней цифра (4) — четыре двоичных разряда 0100 и т. д. Иными словами, вторая и третья цифры (4 и 9) в этом коде являются шестнадцатеричными цифрами, в то время как четвёртая цифра (3) — четверичной цифрой, а пятая (1) — двоичной цифрой.

До сих пор мы рассматривали способы изображения неотрицательных чисел. При изображении чисел, имеющих различные знаки, их необходимо снабжать соответственно знаком плюс или минус. Наиболее простой способ изображения знака заключается в добавлении к коду абсолютной величины числа еще одного «знакового» разряда. Чтобы не вводить лишних символов, удобно обозначать знаки плюс и минус цифрами 0 и 1. Какой из этих двух цифр изображать знак «+», а какой — знак «—» в значительной степени

безразлично. Ниже под числом N мы будем понимать целое число произвольного знака, а под изображением числа N в той или иной системе счисления — код числа $|N|$, дополненный знаковым разрядом, имеющим соответствующее значение.

Для изображения чисел вида $\frac{N}{2^m}$ в двоичной системе счисления вводится понятие *двоичной запятой*, аналогичной обычной запятой в десятичной системе счисления. Число $\frac{N}{2^m}$ представляется при помощи n-значного кода ($n > m$), изображающего число $|N|$, снабженного запятой, помещенной между цифрами ε_m и ε_{m-1}, т. е. кодом

$$\varepsilon_{n-1}\varepsilon_{n-2}\ldots\varepsilon_m,\ \varepsilon_{m-1}\ldots\varepsilon_0$$

с соответствующей цифрой знакового разряда.

Так, например, число $1{,}125 = \frac{9}{8}$ изобразится двоичным кодом

$$1{,}001,$$

а число 2^{-9} — кодом

$$0{,}000\,000\,001.$$

Равенство $\frac{2^r N}{2^{r+n}} = \frac{N}{2^n}$ показывает, что приписывание любого числа нулей справа от кода, снабженного запятой, не изменяет его значения. В кодах целых чисел удобно считать, что запятая стоит правее крайней правой цифры.

Если мы ограничимся числами вида $\frac{N}{2^m}$, где m фиксировано, а $|N|$ меньше, чем 2^n, то для изображения любого из них нам достаточно будет воспользоваться n-значным двоичным кодом

$$\varepsilon_{n-1}\varepsilon_{n-2}\ldots\varepsilon_0$$

с добавлением знакового разряда. Фиксированное значение m определяет положение запятой. Такая система изображения называется системой с *фиксированной запятой*. Система характеризуется двумя параметрами — разрядностью и положением запятой, определяемой числом m.

Система *плавающей запятой* заключается в том, что число $\frac{N}{2^r}$ (где r может изменяться, а $|N|$ по-прежнему меньше, чем 2^n) задается при помощи изображения числа $\frac{N}{2^m}$, где m фиксировано, и изображения разности $m - r$, указывающей на соответствующий сдвиг запятой в ту или иную сторону от ее места в изображении числа $\frac{N}{2^m}$. Направление сдвига запятой определяется знаком разности $m - r$, а число разрядов, на которое следует сдвинуть запятую — абсолютной величиной $|m - r|$. Обычно в системе плавающей запятой ограничиваются числами $\frac{N}{2^r}$, где r удовлетворяет условию

$$|m - r| < 2^s.$$

В этом случае система, помимо параметров m и n, характеризуется третьим параметром s. Числа изображаются в этой системе $(1 + s + 1 + n)$-значными двоичными кодами. Разность $m - r$ обычно называется *порядком* изображаемого числа, а число $\frac{|N|}{2^m}$ — его *мантиссой*.

Наиболее часто используются системы, в которых $m = n$. Число s обычно выбирается около $\log_2 n$.

Нетрудно видеть, что некоторые числа допускают в системе плавающей запятой изображения при помощи различных кодов. Так, например, в системе $n = m = 5$ и $s = 2$ число 1,5 допускает изображения с порядками 1, 2 и 3; коды мантисс будут в этих случаях соответственно

$$11000,\ 01100 \text{ и } 00110.$$

Изображение определяется однозначно условием, что либо величина мантиссы не меньше чем 2^{n-1-m}, либо порядок равен $-(2^s - 1)$. Изображение, в котором мантисса не меньше чем 2^{n-1-m}, называется *нормальным*. Среди рассмотренных выше изображений числа 1,5 нормальным является изображение с порядком 1 и мантиссой 11000. Число 2^{-5} изображается в рассмотренной системе порядком -3 с мантиссой 01000. Нормального изображения числа 2^{-6} в этой системе не имеет.

Арифметические действия над числами как с плавающей, так и с фиксированной запятой сводятся к операциям над целыми числами, дополненным процедурой учета положения запятой, аналогично операциям в десятичной системе счисления.

Перевод десятичной дроби из десятичной системы в двоичную при наличии двоичного арифметического устройства сводится, как и для целых чисел (см. выше), к вычислению значения многочлена, с той разницей, что кроме умножения на десять, необходимо производить операции умножения на одну десятую. Получение десятичных цифр числа, заданного двоичным кодом и меньшего чем единица, сводится к последовательному умножению на десять и отделению целой и дробной частей произведения. Для сведения случая любого числа к переводу числа меньшего, чем единица, достаточно исходное число умножить на соответствующую отрицательную степень десяти. Величина степени, на которую было умножено переводимое число, определит перенос десятичной запятой в полученной после перевода десятичной дроби.

Мы позволим себе не останавливаться на этом более подробно, так как ниже вопросу перевода из двоичной системы в десятичную и обратно посвящены специальные программы, снабженные подробным описанием алгоритма.

Рассмотренные выше способы изображения двоичных чисел с плавающей и фиксированной запятой применимы для представления чисел с любым основанием p.

ЧАСТЬ I

ГЛАВА I

ОПИСАНИЕ МАШИНЫ М-2

§ 1. Состав и структурная схема машины

Машина М-2 — цифровая универсальная вычислительная машина. Основными устройствами машины являются: внутреннее запоминающее устройство, арифметическое устройство, устройство управления, внешнее запоминающее устройство, входные устройства и выходное устройство (рис. 1).

Внутреннее запоминающее устройство предназначено для хранения программы, исходных данных задачи и промежуточных результатов вычислений во время работы машины по данной программе.

Арифметическое устройство предназначено для выполнения арифметических операций (сложение, вычитание, умножение и деление), а также ряда логических и вспомогательных операций.

Устройство управления предназначено для автоматического управления вычислительным процессом в соответствии с программой решения задачи.

Внешнее запоминающее устройство предназначено для хранения частей программы, групп промежуточных результатов и другого материала, обращение к которому во время решения задачи происходит сравнительно редко.

Входные устройства предназначены для ввода программы и исходных данных задачи во внутреннее запоминающее устройство машины.

Выходное устройство предназначено для вывода результатов вычислений из машины.

На рис. 2 приведена структурная схема машины М-2. Здесь прямоугольниками схематично изображены основные устройства машины, двойными стрелками — пути передачи

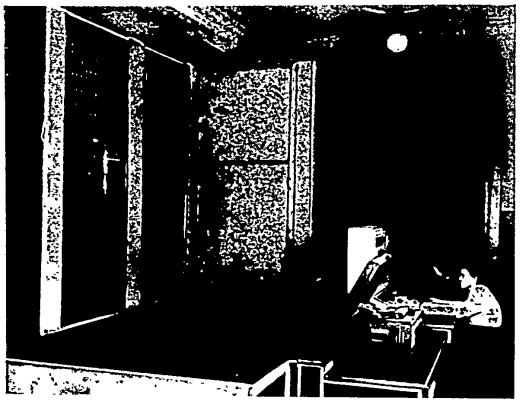


Рис. 1. Общий вид машины М-2. Слева видны стойки устройства управления и арифметического устройства. В закрытом шкафу находится магнитный барабан. Справа — пульт управления, рядом с которым расположены внешнее запоминающее устройство (на переднем плане) и выходное устройство.

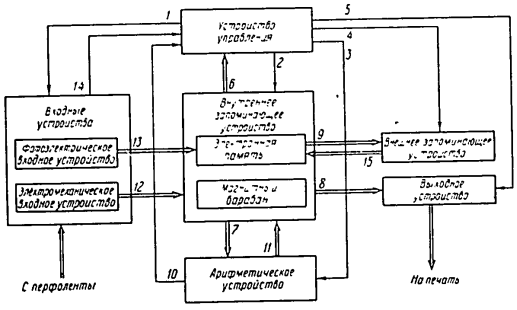


Рис. 2. Структурная схема машины М-2.

кодов, обычными стрелками — пути следования управляющих сигналов. Ниже, при описании устройств машины, роль каждой стрелки будет пояснена.

§ 2. Характеристика отдельных устройств машины

Внутреннее запоминающее устройство. Внутреннее запоминающее устройство машины состоит из

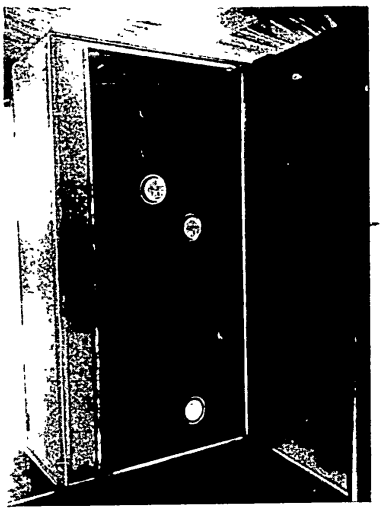


Рис. 3. Запоминающее устройство на электронно-лучевых трубках.

двух частей: электростатического запоминающего устройства (электронная память) и запоминающего устройства на магнитном барабане.

Электронная память машины (рис. 3) состоит из 512 ячеек, занумерованных от 512 до 1023. В каждой ячейке

памяти может храниться 34-разрядный двоичный код (на рис 4. приведена нумерация разрядов кода). Разряды всех 512 кодов, имеющие одинаковые номера, хранятся в одной

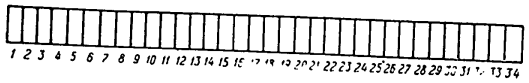


Рис. 4. Нумерация разрядов.

электронно-лучевой трубке (рис. 5). Поскольку код состоит из 34 разрядов, то электростатическое запоминающее устройство содержит 34 трубки. Запись и считывание кодов параллельны, т. е. производятся одновременно по всем разрядам.

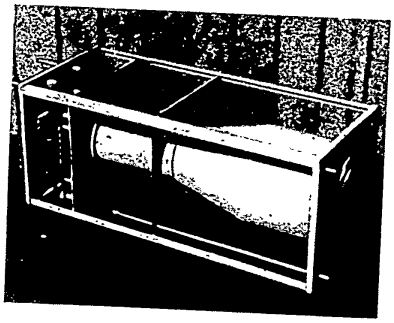


Рис. 5. Электронно-лучевая трубка.

Время выборки (и записи) кода из электронной памяти составляет 50 *мксек*. При использовании только электронной памяти такое время выборки обеспечивает скорость работы машины около 2000 операций в секунду.

Более медленным запоминающим устройством машины является магнитный барабан, также рассчитанный на хранение 512 кодов. Ячейки барабана занумерованы от 0 до

511. Коды записываются по образующим барабана. Ячейки с последовательными номерами находятся на расстоянии $^1/_8$ окружности барабана. Так же как и в электростатическом запоминающем устройстве, запись и считывание кодов параллельны. Скорость вращения барабана — 3000 оборотов в минуту. Время выборки целиком определяется временем ожидания кода, т. е. тем временем, которое необходимо, чтобы нужный код подошел под считывающие головки барабана. Это время в среднем равно времени поворота барабана на полоборота и составляет 10 *мсек*. При работе только с магнитным барабаном скорость работы машины составляет в среднем 25 операций в секунду.

Внутреннее запоминающее устройство связано со всеми остальными устройствами машины.

Вся необходимая для решения данной задачи программная и числовая информация поступает во внутреннее запоминающее устройство машины с входных устройств (рис. 2 стрелки *12* и *13*). В устройство управления из оперативной памяти поступает код очередной выполняемой команды (стрелка *6*), а в арифметическое устройство — коды, над которыми нужно выполнить операцию (стрелка *7*). Результат выполнения операции в арифметическом устройстве пересылается обратно в оперативную память машины (стрелка *11*). Из устройства управления в оперативную память поступают управляющие сигналы с указанием адресов, по которым нужно производить обращение к памяти (стрелка *2*).

В процессе решения задачи может происходить обмен информацией между внутренним и внешним запоминающими устройствами (стрелки *9* и *15*), а также вывод необходимого материала (стрелка *8*). Из схемы на рис. 2 видно, что с фотоэлектрическим входным устройством и с внешним запоминающим устройством связана только электронная часть внутреннего запоминающего устройства.

Арифметическое устройство. Арифметическое устройство состоит из четырех 34-разрядных регистров (*A*, *B*, *C*, *E*) — устройств, предназначенных для хранения и преобразования кодов во время выполнения операции арифметическим устройством машины.

На регистрах *B*, *C* и *E* выполнен сумматор арифметического устройства. Некоторые регистры, кроме своих прямых

функций, связанных с выполнением операций, несут еще ряд дополнительных. Так, через регистр *A* проходят все коды при передаче их из одного запоминающего устройства в другое, а также при вводе и выводе. Кроме того, в регистр *A* поступает код очередной выполняемой команды. Для временного хранения кода команды при выполнении операции используется также регистр *B*. Эти функции регистров *A* и *B* никак не связаны с работой арифметического устройства, поэтому на рис. 2 стрелки *6*, *8*, *9*, *12*, *13* и *15* проведены между соответствующими устройствами, минуя арифметическое устройство.

Можно было бы (как это и сделано на многих машинах) иметь специальный регистр, не входящий в состав арифметического устройства, через который бы осуществлялись все передачи кодов, и специальный (командный) регистр для хранения команд в устройстве управления. Поэтому можно считать, что функционально арифметическое устройство связано только с оперативной памятью машины и с устройством управления.

От устройства управления в арифметическое устройство поступают указания, какую операцию нужно выполнить (рис. 2 стрелка *3*), а из оперативной памяти — коды, над которыми нужно произвести операцию (стрелка *7*).

Результат выполнения операции арифметическим устройством пересылается на хранение обратно в оперативную память машины (стрелка *11*).

В зависимости от результатов выполнения некоторых операций (операций условного перехода и арифметических операций) арифметическое устройство может воздействовать на дальнейший ход вычислений (стрелка *10*), изменяя естественный порядок выполнения команд.

Устройство управления. В состав устройства управления входят два десятиразрядных регистра: *пусковой* регистр и *селекционный* регистр.

В пусковом регистре (счетчике команд) всегда фиксируется номер выполняемой команды. В селекционном регистре фиксируются адреса, по которым происходит обращение к памяти. Регистры *A* и *B* используются для хранения команды во время ее выполнения.

В процессе решения задачи очередная команда, номер которой указан в счетчике команд, выбирается в устройство управления из внутреннего запоминающего устройства машины (рис. 2 стрелка *6*). Команда расшифровывается устройством управления и в зависимости от кода выполняемой операции устройство управления настраивает на ее выполнение те или иные устройства машины (стрелки *1—5*). После выполнения очередной команды к содержимому счетчика команд, как правило, добавляется единица. Однако некоторые команды сами могут содержать информацию о произвольном изменении содержимого счетчика команд. Сюда относятся операции условного перехода и операции, содержащие информацию о безусловной передаче управления. В первом случае состояние счетчика команд изменяется в зависимости от результата выполнения операции в арифметическом устройстве машины (стрелка *10*), во втором — независимо от результата выполнения операции.

Первоначальное состояние счетчика команд (номер команды, с которой нужно начать вычисления), как правило, устанавливается с пульта управления машиной (рис. 6) с помощью специальных тумблеров.

На характер работы устройства управления оказывают влияние, кроме арифметического устройства, также и входные устройства (стрелка *14*). Подробнее об этом будет сказано при описании входных устройств.

Внешнее запоминающее устройство. Внешнее запоминающее устройство машины выполнено на магнитной ленте (рис. 7).

В машине имеется возможность записи на магнитную ленту групп (блоков) из различного количества кодов, находящихся в электронной памяти машины (рис. 2 стрелка *9*), а также возможность передать с магнитной ленты в электронную память любой из записанных на ней блоков (стрелка *15*). Кроме того, имеется возможность продвинуть ленту на величину одного или нескольких блоков в том или ином направлении. Блоки, выводимые на ленту, машиной не помечаются. По команде считывания с магнитной ленты считывается группа кодов, начиная с кода, стоящего непосредственно под считывающим устройством. Это представляет определенные трудности при программировании, так как расположение блоков на ленте надо предусматривать в программе. Ввод с ленты заключается в том, что вначале лента продвигается с таким расчетом, чтобы начало блока оказалось

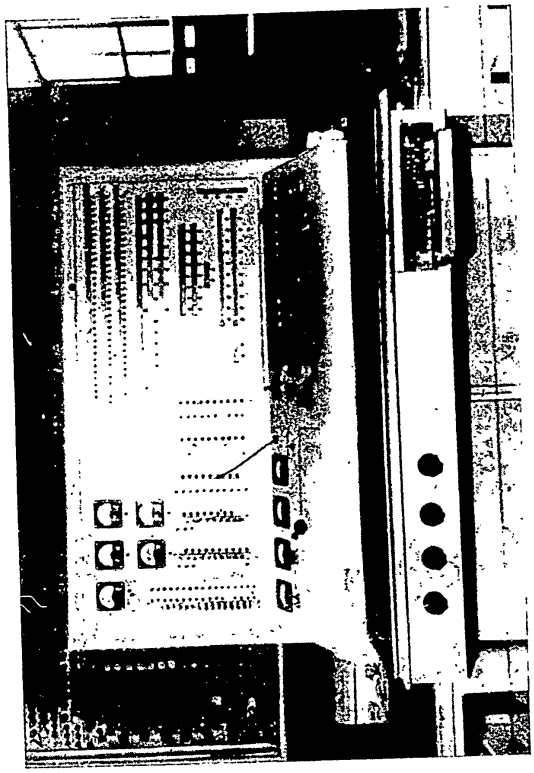


Рис. 6. Пульт управления.

под устройством считывания, а затем происходит передача блока в электростатическое запоминающее устройство. Запись на магнитную ленту производится верно только на те участки ленты, на которых до этого ничего не было записано. Поэтому перед началом решения задачи, требующей

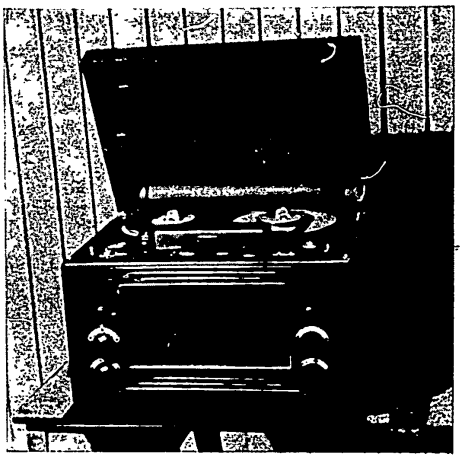


Рис. 7. Внешнее запоминающее устройство на магнитной ленте.

использования магнитной ленты, необходимо стереть материал на тех участках ленты, на которые будет производиться запись в процессе решения задачи. Информация, записанная на ленту, сохраняется длительное время и может быть использована для продолжения решения задачи. Из схемы на рис. 2 видно, что в отличие от внутренней памяти, внешнее запоминающее устройство с арифметическим устройством машины непосредственно не связано.

Входные устройства. Входные устройства (рис. 8), фотоэлектрическое (быстрый ввод) и электромеханическое (медленный ввод), осуществляют ввод материала в оперативную память машины.

Ввод с фотоэлектрического входного устройства может быть осуществлен только в электронную память машины (рис. 2 стрелка *13*). Ввод с электромеханического входного

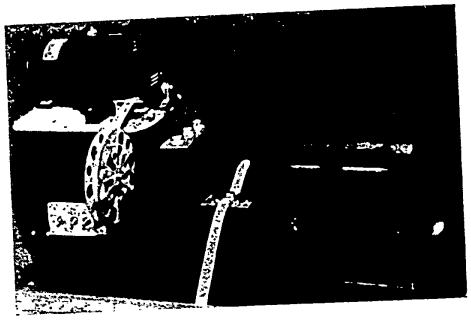


Рис. 8. Входные устройства: электромеханическое (справа) и фотоэлектрическое (слева).

устройства может быть осуществлен и в электронную, и в магнитную части оперативной памяти (стрелка *12*).

Для ввода используется перфорированная пятипозиционная бумажная лента. Ввод осуществляется или с помощью программы по адресам, указанным в командах ввода (стрелка *1*), или с пульта управления машины включением специального тумблера. В последнем случае перед каждым кодом должен быть указан его адрес.

Все коды и адреса, отперфорированные на ленте, оканчиваются специальными служебными знаками, различными для кодов и для адресов. На эти знаки реагирует устройство управления (рис. 2 стрелка *14*): при вводе кода происходит запись его в память машины по адресу, указанному

в селекционном регистре устройства управления, а при вводе адреса меняется состояние селекционного регистра.

На ввод одного кода с помощью электромеханического входного устройства требуется примерно 1,7 секунды, а с помощью фотоэлектрического — 0,03 секунды.

В машине имеется также возможность ввести любой код в любую ячейку оперативной памяти с пульта управления.

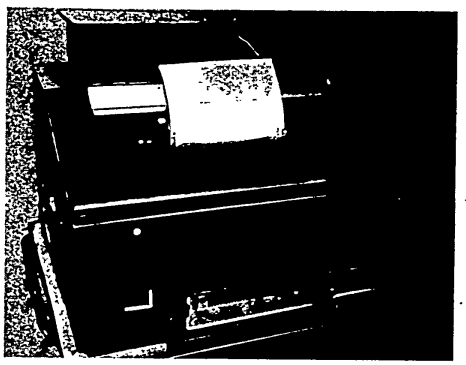


Рис. 9. Выходное устройство.

В этом случае адрес кода фиксируется на пульте управления специальными тумблерами, а сам код набирается вручную на клавишах, расположенных на пульте.

Выходное устройство. Выходное устройство (рис. 9) используется для вывода необходимого материала из оперативной памяти машины (рис. 2 стрелка *8*). Выводимые коды печатаются с помощью стандартного телетайпа на широкой бумажной ленте в три или четыре столбца в зависимости от положения специального тумблера. Телетайп печатает шестнадцатеричными цифрами последовательные четверки двоичных цифр, начиная с первых разрядов кода. При этом шестнадцатеричные цифры 10, 11, 12, 13, 14 и 15 изобра-

жаются телетайпом соответственно буквами a, b, c, d, e и f. Последние два двоичных разряда кода печатаются особо: 33-й разряд кода печатается после шестнадцатеричных цифр, соответствующих первым 32 разрядам кода, 34-й разряд — перед ними. При этом печатается знак «+», если в 33-м (или в 34-м) разряде кода стоит 1, и знак «—», если в 33-м (или в 34-м) разряде кода стоит 0. Например, код

1001 1010 1011 1101 0111 0000 1001 0101 10

напечатается так:

—9abd7095+

Печать последовательная, цифра за цифрой; время печати одного кода составляет 2,5 секунды.

§ 3. Представление чисел и команд в машине. Принятые обозначения

Представление чисел. Среди элементарных операций машины М-2 имеются операции, интерпретирующие коды как числа в системе фиксированной запятой, а также операции, интерпретирующие коды как числа в системе плавающей запятой.

В первом случае код

$$\alpha_1\alpha_2\alpha_3 \ldots \alpha_{33}\alpha_{34}, \quad \alpha_i = 0;\ 1 \tag{I. 1}$$

воспринимается как число

$$x = (2\alpha_{34} - 1)\sum_{k=1}^{33} \alpha_k 2^{-k}. \tag{I. 2}$$

Отсюда следует, что все числа в системе фиксированной запятой удовлетворяют неравенству

$$|x| \leqslant 1 - 2^{-33}.$$

Если в процессе вычислений результат некоторой операции окажется по абсолютной величине больше величины $1-2^{-33}$, машина остановится (такой останов называют остановом по переполнению разрядной сетки машины или просто остановом по переполнению). Следовательно, в том случае, когда вычислительный процесс реализуется с помощью элементарных операций машины, воспринимающих коды как числа в системе фиксированной запятой, необходимо так организовать вычисления, чтобы все числа, участвующие в них, и результаты выполнения операций не выходили за пределы интервала (—1, 1).

Число равно нулю, если равны нулю все α_i, $i=1, 2, \ldots, 33$. Нуль может иметь как положительный, так и отрицательный знак.

33 двоичных разряда, используемых для изображения цифровой части числа, позволяют вести вычисления примерно с десятью десятичными знаками.

В том случае, когда при выполнении некоторых операций коды интерпретируются как числа в системе плавающей запятой, код (I. 1) воспринимается как число

$$x = \varepsilon_x \cdot 2^p \cdot X, \tag{I. 3}$$

где

$$p = \sum_{k=1}^{6} \alpha_k 2^{6-k} - 32 \tag{I. 4}$$

есть порядок числа x, а

$$X = \sum_{k=8}^{33} \alpha_k 2^{7-k} \tag{I. 5}$$

его мантисса;

$$\varepsilon_x = (2\alpha_{34} - 1) \tag{I. 6}$$

изображает знак числа. При этом код (I. 1) будет рассматриваться как число в системе плавающей запятой лишь в том случае, если $\alpha_7 = 0$. В противном случае машина остановится.

Все числа, для которых $\alpha_8 = 1$, называются *нормализованными* числами; отличные от нуля числа, для которых $\alpha_8 = 0$, называются *ненормализованными*.

Из (I. 5) следует, что для нормализованных чисел

$$\frac{1}{2} \leqslant |X| \leqslant 1 - 2^{-26}. \tag{I. 7}$$

Число x будем называть *нормализованным нулем*, если $p+32=0$ и $X=0$. Нуль может иметь как положительный, так и отрицательный знак. Таким образом, нуль имеет нули во всех разрядах цифровой части числа и, следовательно, одинаково изображается как в системе фиксированной запятой, так и в системе плавающей запятой.

Из формулы (1.4 видно, что порядки чисел удовлетворяют условию

$$-32 < p < 31. \qquad (1.8)$$

Величина

$$a = p + 32 = \sum_{k=1}^{6} \alpha_k 2^{6-k}$$

называется *условным порядком числа*. Условный порядок числа есть величина положительная и всегда заключена в пределах

$$0 \leqslant a \leqslant 63.$$

Как и для случая чисел в системе фиксированной запятой, 34-й разряд используется для изображения знака числа, причем, как видно из формулы (1.6), положительному знаку соответствует 1 в этом разряде, а отрицательному знаку соответствует 0.

Из формул (1.7) и (1.8) следует, что диапазон изменения нормализованных чисел, исключая нуль, следующий:

$$2^{-31} \cdot \frac{1}{2} \leqslant |x| \leqslant 2^{31}\left(1 - 2^{-26}\right).$$

Если в результате выполнения некоторой операции получится число, большее чем $2^{32}(1-2^{-26})$, машина остановится по переполнению; если же в результате выполнения некоторой операции получается число, меньшее чем $2^{-31} \cdot \frac{1}{2}$, то оно заменяется нулем.

На рис. 10 показано распределение разрядов при изображении чисел.

Интересно отметить еще следующее обстоятельство. Из формулы (1.4), а также из рис. 10 видно, что для изображения порядков чисел в системе плавающей запятой используются разряды, являющиеся старшими разрядами чисел в системе фиксированной запятой. Поэтому если для двух кодов x и y, рассматриваемых как числа в системе фиксированной запятой, справедливо неравенство

$$x \leqslant y,$$

то это неравенство будет справедливо для кодов x и y также в том случае, если их рассматривать как числа в

системе плавающей запятой. Верно и обратное утверждение. Впрочем, оба эти замечания справедливы лишь в случае нормализованных чисел, о чем подробнее будет сказано при разборе операций сравнения.

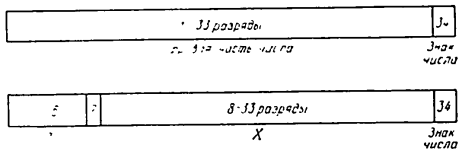


Рис. 10. Распределение разрядов при изображении чисел: вверху — для чисел в системе фиксированной запятой, внизу — для чисел в системе плавающей запятой.

Представление команд. Машина М-2 является *трехадресной* машиной. Первый адрес команды образуют разряды с 21-го по 30-й, второй адрес — разряды с 11-го по 20-й и третий адрес — разряды с 1-го по 10-й. Для изображения номера (кода) операции используются разряды с 31-го по 34-й. В случае арифметических операций в адресах команды стоят адреса кодов, над которыми производятся операции, и адрес результата. На рис. 11 приведено распределение разрядов в команде.

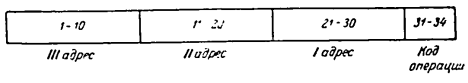


Рис. 11. Распределение разрядов при изображении команды.

Кодирование чисел и команд. Кодирование чисел и команд на машине М-2 производится, как правило, поразному.

Числа обычно кодируются следующим образом: 32 старших двоичных разряда числа записываются шестнадцатеричными цифрами, а два последних разряда — четверичной цифрой.

Например, число

1000 0101 0000 0000 0000 0000 0000 0001

(единица в системе плавающей запятой) запишется так:

850000001.

Первые восемь цифр здесь шестнадцатеричные, последняя — четверичная.

При кодировании команд два старших двоичных разряда каждого адреса записываются четверичной цифрой, а следующие две четверки двоичных цифр — шестнадцатеричными. Код операции записывается одной шестнадцатеричной цифрой. Например, команда

10 0010 0011 10 0010 0010 10 0010 0001 1001

запишется так:

2.23 2.22 2.21 9.

Здесь точками отделяются четверичные цифры от двух соседних шестнадцатеричных.

Так же, как правило, кодируются и константы, вводимые в машину вместе с программой.

Принятые обозначения. Введем некоторые обозначения, которые будут использоваться в дальнейшем.

Адреса команды — первый, второй и третий — будут обозначаться соответственно *IA*, *IIA*, *IIIA*.

Код, стоящий в первом адресе команды, обозначается буквой *C*, код, стоящий во втором адресе, — буквой *B*, в третьем адресе — буквой *A*.

Под записью $ne_I(ne_{II}, ne_{III})$ будем понимать n единиц первого (второго, третьего) адреса.

Под символом (α) будем понимать код, хранящийся в ячейке памяти с номером α, а под символом $<x>$ — ячейку, где хранится код x.

В тех случаях, когда нужно отметить, что в ячейке α хранится вполне определенный код $x = \alpha_1\alpha_2\alpha_3 \ldots \alpha_{34}$, мы будем пользоваться или записью $(\alpha) = x$, или записью $(\alpha) : \alpha_1\alpha_2\alpha_3 \ldots \alpha_{34}$. Первый способ используется обычно в тех случаях, когда x — число, второй — когда x — команда (или произвольный код).

Запись $(\alpha) \Rightarrow \beta$ означает, что содержимое ячейки с номером α помещается в ячейку β запоминающего устройства.

Тот факт, что число k изображается кодом x в системе фиксированной или плавающей запятой, обозначается соответственно таким образом:

$$x = (k)_\phi \quad \text{или} \quad x = (k)_\pi.$$

Переменные команды, адреса и коды операций помечаются значком «*».

§ 4. Система операций машины М-2

Ниже описаны элементарные операции, имеющиеся в машине. Кроме того, отдельно приведены список всех операций машины (табл. I), а также таблица, содержащая время выполнения основных операций (табл. II).

А. ОПЕРАЦИЯ «ПЕРЕКЛЮЧЕНИЕ» (КОД ОПЕРАЦИИ 0)

Машина М-2 может работать в трех режимах: в режиме плавающей запятой, в режиме фиксированной запятой и в режиме фиксированной запятой с двойной точностью (точнее с удвоенным числом разрядов).

Операции с кодами *8*, *9*, *d* и *c* в режиме плавающей запятой выполняются как операции умножения, деления, сложения и вычитания над числами в системе плавающей запятой. Эти же операции в режиме фиксированной запятой выполняются как операции умножения, деления, сложения и вычитания над числами в системе фиксированной запятой.

Режим фиксированной запятой с двойной точностью отличается от режима фиксированной запятой лишь тем, что в этих режимах по-разному выполняются операции с кодами *8* и *9*.

Все остальные операции (кроме операций с кодами *8*, *9*, *d* и *c*) выполняются всегда одинаково, независимо от того, в каком режиме работает машина.

Переключение машины с одного режима работы на другой осуществляется с помощью операции переключения.

Вид переключения задается содержимым разрядов с 16-го по 19-й команды переключения.

Таблица 1

Список операций машины М-2

III А	II А	I А	Код операции	Содержание команды
–	*0.00*	*C*	*0*	Безусловная передача управления команде с адресом *C*
. .	*0.0c*	*C*	*0*	Переключение на режим фиксированной занятой с двойной точностью и передача управления команде с адресом *C*.
.	*0.12*	*C*	*0*	Переключение на режим плавающей занятой и передача управления команде с адресом *C*.
	0.18	*C*	*0*	Переключение на режим фиксированной занятой и передача управления команде с адресом *C*.
A	[*B*]	*0.00*	*1*	Перемотка магнитной ленты на величину одного стандартного блока в прямом направлении и передача управления команде с адресом *A*.
A	[*B*]	*0.01*	*1*	Перемотка магнитной ленты на величину одного стандартного блока в обратном направлении и передача управления команде с адресом *A*.
A	*B*	*0.02*	*1*	Перемотка магнитной ленты на величину одного или нескольких блоков в прямом направлении и передача управления команде с адресом *A*. Общее число кодов в блоках равно $3.ff - B + 1$, где *B* — адрес электронной памяти.
A	*B*	*0.03*	*1*	Перемотка магнитной ленты на величину одного или нескольких блоков в обратном направлении и передача управления команде с адресом *A*. Общее число кодов в блоках равно $3.ff - B + 1$, где *B* — адрес электронной памяти.
A	*B*	*0.00*	*2*	Считывание с магнитной ленты одного стандартного блока в группу ячеек электронной памяти с последовательными номерами, начиная с ячейки *B*, и передача управления команде с адресом *A*.

Продолжение табл. 1

III А	II А	I А	Код операции	Содержание команды
A	*B*	*2.01*	*2*	Ввод одного кода в ячейку *B* электронной памяти с фотоэлектрического входного устройства и передача управления команде с адресом *A*.
A	*B*	*0.02*	*2*	Считывание с магнитной ленты одного или нескольких блоков в группу ячеек электронной памяти, начиная с ячейки *B* и кончая ячейкой *3.ff*, и передача управления команде с адресом *A*.
A	*B*	*0.03*	*2*	Ввод одного кода в ячейку *B* внутреннего запоминающего устройства с электромеханического входного устройства и передача управления команде с адресом *A*.
A	*B*	*0.00*	*3*	Запись на магнитную ленту одного стандартного блока из группы ячеек электронной памяти с последовательными номерами, начиная с ячейки *B*, и передача управления команде с адресом *A*.
A	*B*	*0.02*	*3*	Запись на магнитную ленту одного нестандартного блока из группы ячеек электронной памяти с последовательными номерами, начиная с ячейки *B*, и передача управления команде с адресом *A*.
A	*B*	*0.03*	*3*	Печать одного кода из ячейки *B* внутреннего запоминающего устройства и передача управления команде с адресом *A*.
A	*B*	*C*	*4*	Перенос кода (*C*) в ячейку *B* и передача управления команде с адресом *A*.
—	*B*	*C*	*5*	Останов машины. Коды (*B*) и (*C*) можно прочесть на пульте управления машины.
A	*B*	*C*	*6*	Сравнение чисел (*B*) и (*C*) с учетом знаков. Если $(B) \geqslant (C)$, то управление передается следующей по порядку команде; если $(B) < (C)$, то управление передается команде с адресом *A*.

Продолжение табл. I

III A	II A	I A	Код операции	Содержание команды
A	B	C	7	Сравнение абсолютных величин чисел (B) и (C). Если $\|(B)\| \geqslant \|(C)\|$, то управление передается следующей по порядку команде; если $\|(B)\| < \|(C)\|$, управление передается команде с адресом A.
A	B	C	8	Деление. $(B):(C) \Rightarrow A$. Операция выполняется по-разному, в зависимости от режима работы машины.
A	B	C	9	Умножение. $(B) \times (C) \Rightarrow A$. Операция выполняется по-разному, в зависимости от режима работы машины.
A	B	C	a	Специальное вычитание. $(B)-(C) \Rightarrow A$. Выполняется, как вычитание чисел в системе фиксированной запятой.
A	B	C	b	Специальное сложение. $(B)+(C) \Rightarrow A$. Выполняется, как сложение чисел в системе фиксированной запятой.
A	B	C	c	Вычитание $(B)-(C) \Rightarrow A$. Операция выполняется по-разному, в зависимости от режима работы машины.
A	B	C	d	Сложение. $(B)+(C) \Rightarrow A$. Операция выполняется по-разному, в зависимости от режима работы машины.
A	B	C	e	Присвоение знака $\|(B)\| \operatorname{sign}(C) \Rightarrow A$.
A	B	C	f	Поразрядное логическое умножение кодов (B) и (C). Результат записывается по адресу A.

Примечания. 1. Черта в адресе команды означает, что содержимое этого адреса не влияет на характер выполнения данной операции.

2. [B] означает произвольный адрес электронной памяти.

Таблица II

Время выполнения элементарных операций машины при использовании электронной памяти *)

Операция	Число операций в секунду	Время выполнения операций в *мксек*
Сложение с плавающей запятой	720— 1860	537,5—1387,5
Вычитание с плавающей запятой	720— 1860	537,5—1387,5
Умножение с плавающей запятой	675— 3080	325,0—1475,0
Деление с плавающей запятой	660— 3080	325,0—1512,5
Сложение с фиксированной запятой	2120— 2970	337,5— 462,5
Вычитание с фиксированной запятой	2120— 2970	337,5— 462,5
Умножение с фиксированной запятой	605— 1270	787,5—1650,0
Деление с фиксированной запятой	590— 605	1650,0—1700,0
Умножение с фиксированной запятой с удвоенным количеством разрядов	580— 1175	850,0—1725,0
Деление с фиксированной запятой с удвоенным количеством разрядов	565— 580	1712,5—1775,0
Переключение	10000—14000	87,5— 100,0
Перенос числа	4200— 5000	200,0— 237,5
Логическое умножение	3080— 3640	275,0— 325,0
Перемена знака	2850— 3330	300,0— 350,0
Сравнение алгебраическое	2420— 2850	350,0— 412,5
Сравнение по модулю	2420— 2740	375,0— 412,5

*) Включая время обращения к памяти.

При этом присутствие единицы в одном из этих разрядов имеет следующий смысл.

Единица в 16-м разряде означает переключение машины на работу с нормальным количеством разрядов; единица в 17-м разряде означает переключение машины на работу с числами в системе фиксированной запятой; единица в 18-м разряде означает переключение машины на работу с удвоенным количеством разрядов и, наконец, присутствие единицы в 19-м разряде означает переключение машины на работу с числами в системе плавающей запятой.

Комбинируя единицы в разрядах 16—19 можно получать разные переключения. Содержимое разрядов с 11-го по 15-й, а также содержимое разряда 20 не влияют на вид переключения. Будем считать, что в этих разрядах стоят нули. Тогда комбинациями во втором адресе команды переключения, осуществляющими переключение машины с одного режима работы на другой, являются, например, следующие:

Содержимое *IIA*	Вид переключения
0.12	Переключение на режим плавающей запятой (ПЗ).
0.0c	Переключение на режим фиксированной запятой с двойной точностью (ФД).
0.18	Переключение на режим фиксированной запятой (ФЗ).

Мы будем пользоваться только этими комбинациями.

Первый адрес в команде переключения используется для безусловной передачи управления. Содержимое третьего адреса не влияет на выполнение операции. Операция переключения с кодом *0.00* во втором адресе означает безусловную передачу управления команде с номером *C* без изменения режима работы машины.

После выполнения какого-либо переключения машина работает на соответствующем режиме до тех пор, пока не будет сделано другое переключение.

Б. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Сделаем вначале несколько замечаний, относящихся ко многим арифметическим операциям:

— если при выполнении некоторой операции произойдет переполнение, то запись результата не производится;

— результат арифметической операции над нормализованными числами всегда нормализован;

— при выполнении операций над числами в системе плавающей запятой нуль в результате имеет всегда положительный знак, в то время как в случае операций над числами в системе фиксированной запятой может получаться в результате как положительный, так и отрицательный нуль.

Операция сложения с фиксированной запятой (код операции *d*). Сложение чисел x и y в режиме фиксированной запятой осуществляется по следующим формулам:

$$x + y = \operatorname{sign} x \cdot (|x| + |y|), \tag{I. 9}$$

если числа x и y имеют одинаковые знаки, и

$$x + y = \begin{cases} \operatorname{sign} x \cdot (|x| - |y|) & \text{при } |x| \geqslant |y|, \\ \operatorname{sign} y \cdot (|y| - |x|) & \text{при } |y| > |x|, \end{cases} \tag{I. 10}$$

если числа x и y имеют разные знаки.

При этом $x = (B)$, $y = (C)$; результат операции помещается в ячейку A внутреннего запоминающего устройства. Из формулы (I. 10) видно, что знак нуля в результате совпадает со знаком числа (B).

Никаких погрешностей операция сложения с фиксированной запятой не вносит.

Операция вычитания с фиксированной запятой (код операции *c*). Так как $x - y = x + (-y)$, то вычитание чисел x и y происходит по формуле (I. 9), когда x и y различных знаков, и по формулам (I. 10), когда числа x и y одинаковых знаков.

Все сказанное в отношении операции сложения с фиксированной запятой относится и к операции вычитания.

Операция умножения с фиксированной запятой (код операции *9*). Умножение чисел x и y в режиме фиксированной запятой происходит по формуле

$$xy = (\operatorname{sign} x \cdot \operatorname{sign} y)\,|x| \cdot |y|.$$

Здесь $x=(B)$ — множимое, $y=(C)$ — множитель. Погрешность δ абсолютной величины произведения заключена в пределах

$$-2^{-33}<\delta\leqslant 0.$$

При выполнении операции с кодом *9* в режиме фиксированной запятой с двойной точностью в ячейке A получаются 33 младших разряда произведения, в ячейке $A+1$ получаются старшие разряды произведения. Знак (A) при этом совпадает со знаком произведения.

Операция деления с фиксированной запятой (код операции *8*). Деление чисел x и y в режиме фиксированной запятой происходит по формуле

$$\frac{x}{y}=(\operatorname{sign}x\cdot\operatorname{sign}y)\frac{|x|}{|y|},$$

где $x=(B)$ — делимое, $y=(C)$ — делитель.

Результат деления абсолютных величин x и y получается всегда с недостатком, погрешность δ модуля частного находится в пределах

$$-2^{-33}<\delta\leqslant 0.$$

При работе с удвоенным количеством разрядов в ячейке A получается частное, в ячейке $A+1$ получается модуль остатка от деления со знаком частного.

При умножении и делении чисел в режиме фиксированной запятой с двойной точностью операции выполняются несколько медленнее, так как требуется одно дополнительное обращение к запоминающему устройству (отсылка остатка от деления или младших разрядов произведения).

Операция сложения с плавающей запятой (код операции *d*). Сложение чисел $x=2^{p_1}X$ и $y=2^{p_2}Y$ осуществляется согласно следующих формул:

$$z=x+y=2^{\max(p_1,\,p_2)}(X'+Y'), \tag{I.11}$$

$$\left.\begin{aligned}&X'=X,\ Y'=2^{p_2-p_1}Y,\ \text{если } p_1\geqslant p_2,\\&X'=2^{p_1-p_2}X,\ Y'=Y,\ \text{если } p_2>p_1.\end{aligned}\right\} \tag{I.12}$$

Сложение мантисс X' и Y' выполняется так же, как в случае сложения чисел в системе фиксированной запятой по формулам (I.9) и (I.10). Нормализация результата z производится по-разному, в зависимости от того, по каким формулам, (I.9) или (I.10), выполнялось сложение мантисс.

В первом случае ввиду того, что $1\leqslant|X'+Y'|<2$, нормализация осуществляется сдвигом кода мантиссы $X'+Y'$ на один разряд вправо и добавлением единицы к порядку числа z.

Во втором случае $|X'+Y'|\leqslant\frac{1}{2}$ и нормализация осуществляется последовательными сдвигами кода $X'+Y'$ на один разряд влево, если не выполняется условие $\alpha_8=1$; при каждом сдвиге порядок числа z уменьшается на единицу. Если после 26 сдвигов влево $\alpha_8=0$, то результат z полагается равным нулю.

Погрешность δ модуля мантиссы результата заключена в пределах

$$-\frac{3}{4}\cdot 2^{-26}\leqslant\delta<1\cdot 2^{-26}.$$

Операция вычитания с плавающей запятой (код операции *c*). Вычитание чисел $x=2^{p_1}X$ и $y=2^{p_2}Y$ выполняется по формулам, аналогичным формулам (I.11) и (I.12):

$$z=x-y=2^{\max(p_1,\,p_2)}(X'-Y'), \tag{I.11'}$$

$$\left.\begin{aligned}&X'=X,\ Y'=2^{p_2-p_1}Y,\ \text{если } p_1\geqslant p_2,\\&X'=2^{p_1-p_2}X,\ Y'=Y,\ \text{если } p_2>p_1.\end{aligned}\right\} \tag{I.12'}$$

Разность $X'-Y'$ находится по формулам (I.9) или (I.10), в зависимости от того, разных или одинаковых знаков числа x и y. Все замечания, относящиеся к сложению с плавающей запятой, справедливы и в случае вычитания.

Погрешности сложения и вычитания с плавающей запятой. Рассмотрим вначале случай, когда сложение или вычитание приводит к формуле (I.9).

Погрешность может возникнуть как при выравнивании порядков чисел x и y в процессе преобразования мантисс, согласно формул (I.12) и (I.12'), так и при нормализации результата.

Результат сдвига мантиссы одного из чисел на $|p_1-p_2|$ разрядов вправо округляется. Тем самым допускается погрешность δ_1 абсолютной величины мантиссы

$$-\frac{1}{2}\cdot 2^{-26}<\delta_1\leqslant\frac{1}{2}\cdot 2^{-26}.$$

Если сумма мантисс по абсолютной величине меньше единицы, то нормализации не требуется, и окончательной ошибкой будет $\delta = \delta_1$. Если сумма мантисс по абсолютной величине больше единицы, то ее необходимо сдвинуть на один разряд вправо.

В зависимости от того, что стоит в 26-м разряде непормализованной мантиссы — нуль или единица, ошибка δ_2 от сдвига ее на один разряд вправо будет или нулем, или равна $-\frac{1}{2}\cdot 2^{-26}$.

Вместе с тем, в результате сдвига вдвое уменьшится ошибка δ_1, так что общая ошибка будет удовлетворять неравенству

$$-\frac{3}{4}\cdot 2^{-26} < \delta \leqslant \frac{1}{4}\cdot 2^{-26}.$$

Таким образом, при выполнении операции сложения или вычитания в рассматриваемом случае может возникнуть погрешность δ:

$$-\frac{3}{4}\cdot 2^{-26} < \delta \leqslant \frac{1}{2}\cdot 2^{-26}.$$

Исследуем теперь случай, когда сложение или вычитание приводит к формуле (I.10). Можно предположить, что уменьшаемое больше вычитаемого. Здесь могут представиться три возможности

$$|p_1 - p_2| = 0, \quad |p_1 - p_2| = 1, \quad |p_1 - p_2| > 1.$$

Рассмотрим каждую из них в отдельности.

а) $|p_1 - p_2| = 0$. Так как выравнивания порядков нет, то результат вычитания точный, и нормализация никаких ошибок не вносит. Ошибка сложения (вычитания) в этом случае равна нулю.

б) $|p_1 - p_2| = 1$. Так как разность порядков равна по абсолютной величине единице, то при выравнивании порядков мантисса вычитаемого сдвигается на один разряд вправо и к ее порядку прибавляется единица. Если мантисса вычитаемого до сдвига имела в 26-м разряде нуль, то при сдвиге никакой погрешности не возникает. Сложение и вычитание в этом случае выполняются точно. Если же в 26-м разряде мантиссы вычитаемого до выравнивания порядков была единица, то при сдвиге вправо мантисса вычитаемого приобретает ошибку, равную $-\frac{1}{2}\cdot 2^{-26}$ (так как результат сдвига не округляется), а разность мантисс — ошибку $+\frac{1}{2}\cdot 2^{-26}$. В этом случае из мантиссы разности вычитается $1\cdot 2^{-26}$, в результате чего ошибка результата до нормализации будет равна $-\frac{1}{2}\cdot 2^{-26}$. Это будет погрешностью сложения (вычитания), если нормализации не требуется.

Нормализация результата выполняется следующим образом.

После первого сдвига влево разности мантисс, когда ошибка возрастет в два раза и станет равной $-1\cdot 2^{-26}$, в 26-й разряд разности добавляется единица. Добавление к мантиссе $1\cdot 2^{-26}$ компенсирует ошибку. Следовательно, после первого сдвига мантисса будет точной. При последующих сдвигах (если они необходимы) в младшие разряды вводятся нули. Таким образом, ошибка δ результата в случае (I.10) будет равна или нулю, или $\frac{1}{2}\cdot 2^{-26}$.

в) $|p_1 - p_2| > 1$. Поскольку порядки отличаются больше чем на единицу, то при выравнивании порядков мантисса вычитаемого сдвинется вправо по крайней мере на два разряда. А это значит, что разность мантисс будет всегда больше $\frac{1}{4}$, следовательно, при нормализации (если она нужна) будет сделан всего один сдвиг влево. Рассмотрим два случая: когда старшая из отброшенных цифр мантиссы после выравнивания порядков равна нулю и когда она равна единице.

В первом случае отброшенные разряды мантиссы вычитаемого составляют величину, меньшую $\frac{1}{2}\cdot 2^{-26}$. Значит, ошибка модуля разности до нормализации неотрицательна и меньше половины младшего разряда мантиссы. Если нормализации результата не требуется, то это — окончательная погрешность результата. Если нормализация необходима, то при сдвиге мантиссы влево на один разряд ошибка возрастет в два раза, т. е.

$$0 \leqslant \delta < 1\cdot 2^{-26}.$$

Во втором случае отброшенные разряды составляют величину, большую или равную $\frac{1}{2}\cdot 2^{-26}$, а значит, результат вычитания мантисс приобретает положительную ошибку δ_1:

$$\frac{1}{2}\cdot 2^{-26} \leqslant \delta_1 < 1\cdot 2^{-26}.$$

Вычитание единицы 26-го разряда из мантиссы до нормализации сдвигает границы возможной ошибки на 2^{-26}:

$$-\frac{1}{2}\cdot 2^{-26} \leqslant \delta_2 < 0.$$

Если нормализации результата не требуется, то ошибка δ_2 — окончательная ошибка результата. Если требуется нормализовать результат, то ошибка δ_2 удваивается. Но в этом случае в 26-й разряд мантиссы нормализованного результата добавляется единица, так что ошибка попадает в полуинтервал

$$[0;\ 1\cdot 2^{-26}).$$

Рассмотрение всех трех возможных случаев показывает, что ошибка сложения (вычитания) в том случае, когда оно приводит к формулам (I.10), заключена в пределах

$$-\frac{1}{2}\cdot 2^{-26} \leqslant \delta < 1\cdot 2^{-26},$$

Таким образом, относительная ошибка абсолютной величины результата сложения (вычитания) двух чисел в системе плавающей запятой удовлетворяет следующему неравенству.

$$-\frac{3}{4}\cdot 2^{-26} < \delta < 1\cdot 2^{-26}.$$

Умножение с плавающей запятой (код операции 9). Умножение чисел $x = 2^{p_1}X$ и $y = 2^{p_2}Y$ осуществляется по следующей формуле:

$$z = xy = (\operatorname{sign} x \cdot \operatorname{sign} y)\, 2^{p_1+p_2}\,(|X|\cdot|Y|).$$

Умножение мантисс происходит лишь в том случае, если выполняется неравенство

$$-31 \leqslant p_1 + p_2 \leqslant 31.$$

Если $p_1+p_2 > 31$, то машина останавливается. Отсюда видно, что остановка машины может быть даже в том случае, когда результат в действительности меньше чем 2^{31}.

Например, при умножении двух чисел x и y для $x = y = \frac{1}{2}\cdot 2^{16}$ сумма порядков $(p_1+p_2 = 16+16 = 32)$ будет больше 31, хотя произведение меньше чем 2^{31}.

В случае, когда $p_1+p_2 < -31$, результат полагается равным нулю.

После сложения порядков мантиссы умножаются так же, как и в случае умножения с фиксированной запятой. Незначительные изменения сделаны с целью уменьшения ошибок результата. Ошибка абсолютной величины мантиссы произведения удовлетворяет неравенству

$$-1\cdot 2^{-26} \leqslant \delta < 0.$$

Деление с плавающей запятой (код операции 8). Деление двух чисел $x = 2^{p_1}X$ и $y = 2^{p_2}Y$ начинается с проверки того, не являются ли нулями делитель или делимое. В первом случае машина останавливается, во втором случае (при условии, что делитель не равен нулю) результат полагается равным нулю. После этого деление осуществляется согласно формулы

$$z = \frac{x}{y} = (\operatorname{sign} x \cdot \operatorname{sign} y)\, 2^{p_1-p_2}\left(\frac{|X|}{|Y|}\right).$$

Деление мантисс производится лишь в том случае, если

$$-32 \leqslant p_1 - p_2 \leqslant 31.$$

При $p_1 - p_2 > 31$ машина останавливается, при $p_1 - p_2 < -32$ результат полагается равным нулю. Деление мантисс происходит так же, как и в случае деления с фиксированной запятой. Некоторые незначительные изменения в процессе деления сделаны для того, чтобы уменьшить ошибку результата. Ошибка абсолютной величины мантиссы частного заключена в пределах

$$-1\cdot 2^{-26} < \delta \leqslant 0.$$

Замечание об операциях с ненормализованными числами. Все операции с плавающей запятой рассчитаны на нормализованные числа. Однако иногда возникает необходимость точно знать, что получится в результате той или иной операции с плавающей запятой над ненормализованными числами.

В случае умножения и деления с плавающей запятой ненормализованные числа воспринимаются арифметическим узлом как нули. При операциях сложений и вычитаний с плавающей запятой дело обстоит иначе. Из замечаний, сделанных выше относительно нормализации результатов сложения, видно, что в случае, когда сложение мантисс происходит по формуле (I. 9), результат сложения ненормализованных чисел может быть ненормализованным; в том же случае, когда сложение мантисс осуществляется по формулам (I. 10), результат сложения ненормализованных чисел всегда нормализован.

В. ОПЕРАЦИИ УСЛОВНОГО ПЕРЕХОДА

В машине имеются две операции, которые могут осуществлять условный переход: операция алгебраического сравнения и операция сравнения по абсолютной величине.

Алгебраическое сравнение (код операции 6). При выполнении этой операции коды (B) и (C) воспринимаются арифметическим узлом машины как числа в системе фиксированной запятой.

Операция заключается в том, что вначале производится алгебраическое сравнение чисел (B) и (C), а затем, в зависимости от того, будет ли выполняться условие $(B) \geqslant (C)$ или не будет, управление передается или следующей по номеру команде, или команде A (точнее, или к содержимому пускового регистра добавляется единица, или в пусковом регистре устанавливается A).

Сравнение по абсолютной величине (код операции *7*). Так же, как и в случае алгебраического сравнения, при выполнении операции сравнения по абсолютной величине коды (B) и (C) рассматриваются как числа в системе фиксированной запятой. Отличие в операциях заключается в том, что при выполнении последней сравниваются абсолютные величины чисел. При выполнении условия $|(B)| \geqslant |(C)|$ вслед за командой сравнения будет выполняться следующая по номеру команда, при невыполнении — команда с номером A (точнее, при выполнении условия $|(B)| \geqslant |(C)|$ к содержимому пускового регистра добавляется единица, при невыполнении — в пусковом регистре устанавливается A).

Замечание, сделанное в конце § 3 относительно сравнения чисел в системе фиксированной запятой и чисел в системе плавающей запятой, говорит о том, что коды (B) и (C) могут быть также числами в системе плавающей запятой.

Сравнивать можно и команды и вообще произвольные наборы двоичных цифр. Нужно только помнить, что их коды будут при сравнении восприниматься как числа в системе фиксированной запятой, а содержимое 34-го разряда будет рассматриваться как знак соответствующего числа.

Для случая чисел в системе плавающей запятой сравнение правильно выполняется только для нормализованных чисел. При сравнении ненормализованных чисел может быть получен неверный результат. Например, число $x = 2^{10} \cdot \frac{1}{2}$ больше числа $y = 2^{11} \cdot \frac{1}{8}$, а результат сравнения покажет обратное, так как у второго числа в разрядах порядка стоит большее двоичное число. Последнее замечание относится к обеим операциям условного перехода.

Г. ОПЕРАЦИИ ДЛЯ РАБОТЫ С ВНЕШНИМ ЗАПОМИНАЮЩИМ УСТРОЙСТВОМ

Для работы с магнитной лентой в машине М-2 предусмотрено три операции: операция записи на магнитную ленту, операция считывания с магнитной ленты и операция перемотки магнитной ленты.

Операция записи на магнитную ленту (код операции *3*). Существуют две модификации операции записи на ленту. Операция с кодом *3* и комбинацией *0.00* в первом адресе команды означает запись на ленту блока из 16 кодов из ячеек электростатического запоминающего устройства с последовательными номерами. Номер первой ячейки указывается во втором адресе команды.

Операция с кодом *3* и комбинацией *0.02* в первом адресе означает запись на магнитную ленту блока кодов из ячеек электронной памяти с последовательными номерами, причем номер первой ячейки указывается во втором адресе команды, а номер последней — всегда *3.ff*.

Последние блоки будем называть нестандартными в отличие от блоков из 16 кодов, которые назовем стандартными.

Запись на магнитную ленту может быть осуществлена только из электронной памяти машины и только на те участки ленты, где до этого ничего не было записано. После выполнения любой из операций записи на магнитную ленту будет выполняться команда с номером A.

Операция перемотки магнитной ленты (код операции *1*). Операция выполняется по-разному в зависимости от кода, стоящего в первом адресе команды.

Если в первом адресе команды стоит код *0.00*, то операция с кодом *1* означает перемотку магнитной ленты в прямом направлении на величину одного стандартного блока. Второй адрес команды не используется, но для правильного выполнения операции необходимо, чтобы в нем был указан какой-либо адрес электронной памяти.

Если в первом адресе команды стоит код *0.01*, то осуществляется перемотка ленты на величину одного стандартного блока в обратном направлении. Во втором адресе должен стоять любой адрес электронной памяти.

С помощью команды перемотки с кодом *0.02* (*0.03*) в первом адресе можно осуществить перемотку магнитной ленты на величину одного или нескольких блоков в прямом (обратном) направлении. Блоки могут быть как стандартными, так и нестандартными. Общее число кодов в блоках не должно превышать 512. Во втором адресе команды указывается адрес B электронной памяти, такой, чтобы число $3.ff - B + 1$ было равно общему числу кодов в блоках.

После выполнения команды перемотки управление передается команде *A*.

Операция считывания с магнитной ленты (код операции *2*). Операция с кодом *2* и комбинацией *0.00* в первом адресе осуществляет считывание одного стандартного блока с магнитной ленты в любую группу ячеек с последовательными номерами. Номер первой ячейки группы указывается во втором адресе команды.

Операция с кодом *0.02* в первом адресе осуществляет считывание одного или нескольких блоков в группу ячеек электронной памяти с последовательными адресами. Адрес первой ячейки группы указывается во втором адресе команды, адрес последней ячейки группы — всегда *3.ff*. После выполнения команды считывания осуществляется безусловная передача управления команде *A*.

Д. ОПЕРАЦИИ ВВОДА И ВЫВОДА

Операция ввода (код операции *2*). Операция ввода с комбинацией *2.01* в первом адресе означает ввод одного кода с перфоленты с помощью фотоэлектрического входного устройства.

Ввод может быть осуществлен только в электронную память машины. Если перед кодом на перфоленте указан адрес этого кода, то код введется по этому адресу, в противном случае — по адресу *B*.

Операция ввода с комбинацией *0.03* в первом адресе означает ввод одного кода с перфоленты с помощью электромеханического входного устройства в любую ячейку внутреннего запоминающего устройства. Так же, как и в предыдущем случае код с перфоленты вводится по адресу, указанному на перфоленте; если же адрес перед кодом не указан — в ячейку *B*.

После выполнения операции ввода осуществляется безусловная передача управления команде с адресом *A*.

Операция вывода (код операции *3*, содержимое первого адреса *0.03*). При выполнении этой операции происходит печать кода (*B*) и осуществляется безусловная передача управления команде *A*.

В каком виде осуществляется печать, было сказано в § 1 при описании выходного устройства.

Е. ОСТАЛЬНЫЕ ОПЕРАЦИИ

Операция специального сложения (код операции *b*) и операция специального вычитания (код операции *a*). Эти операции выполняются так же, как операции сложения и вычитания с фиксированной запятой и не зависят от режима работы машины.

Операция логического умножения (код операции *f*). При выполнении операции с кодом *f* происходит поразрядное логическое умножение кодов. Например, результатом умножения кодов $\alpha_1\alpha_2\ldots\alpha_{34}$, $\alpha_i = 0; 1$ и $\beta_1\beta_2\ldots\beta_{34}$, $\beta_i = 0; 1$ будет код $\gamma_1\gamma_2\ldots\gamma_{34}$, где $\gamma_i = 1$, если $\alpha_i = 1$ и $\beta_i = 1$, и $\gamma_i = 0$ в остальных случаях.

Операция присвоения знака (код операции *e*). В результате выполнения операции получается код, у которого 33 первых разряда совпадают с соответствующими разрядами кода (*B*), а 34-й разряд — с 34-м разрядом кода *C*. Результат помещается в ячейку *A*.

Перенос числа (код операции *4*). Операция заключается в том, что в ячейку *B* помещается код, хранящийся в ячейке *C*. Содержимое ячейки *C* при этом не портится. Третий адрес используется для безусловной передачи управления команде *A*.

Операция «стоп» (код операции *5*). Операция с кодом *5* означает останов машины. На пульте можно прочесть содержимое ячеек *B* и *C*.

Третий адрес команды не используется.

§ 5. Подготовка ленты для ввода

Как уже указывалось, для ввода материала в машину используется пятипозиционная перфорированная бумажная лента. Подготовка ленты — необходимый этап работы при решении задач на М-2. Перфорирование ленты производится на буквопечатающем аппарате системы «Телетайп» (рис. 12); одновременно происходит печать перфорируемых знаков. На клавишном устройстве телетайпа имеются клавиши для изображения на ленте шестнадцатеричных, четверичных и двоичных цифр, служебных знаков *V* и *S*, а также клавиши «возврат каретки», «перевод рулона», «бланк пробела» для управления печатью.

Каждый знак изображается на ленте вполне определенной комбинацией пробивок. Каждая комбинация пробивок образует одну строку по ширине ленты (рис. 13). В таблице III приведено соответствие знаков комбинациям пробивок.

Как видно из табл. III, наличие пробивки в первой позиции означает, что пробивки в данной строке изображают шестнадцатеричную цифру. При этом во второй позиции

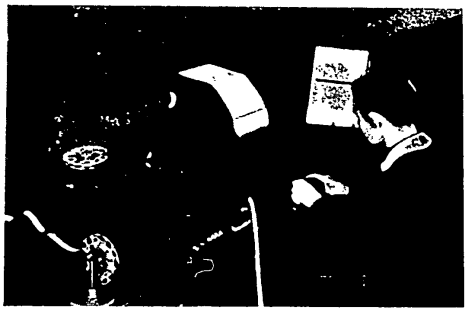


Рис. 12 Устройство для подготовки перфоленты.

изображается младшая цифра двоичного числа, соответствующего данной шестнадцатеричной цифре, в пятой позиции — старшая двоичная цифра. Наличие пробивки в позиции означает, что данная двоичная цифра равна единице, отсутствие пробивки означает, что двоичная цифра равна нулю.

Если в первой позиции пробивки нет, а во второй и третьей есть, то комбинация пробивок в строке изображает четверичную цифру, причем пятая позиция соответствует старшей двоичной цифре, четвертая — младшей. Если нет пробивок в первых двух позициях, а есть в третьей и четвертой, то пробивка в пятой позиции означает двоичную цифру 1, а отсутствие пробивки — двоичную цифру 0.

Входные устройства вместо шестнадцатеричных и четверичных цифр вводят в регистр A арифметического узла,

а затем в запоминающее устройство двоичные эквиваленты этих цифр. Поэтому совершенно безразлично, какими цифрами, шестнадцатеричными, четверичными или теми и другими одновременно, записан и отперфорирован код.

Знак S означает конец адреса на ленте, а знак V — конец кода на ленте. Если коды желательно ввести по адресам, указанным в программе, то на ленте нужно отперфори-

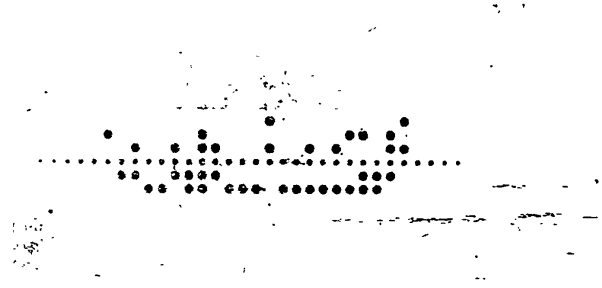


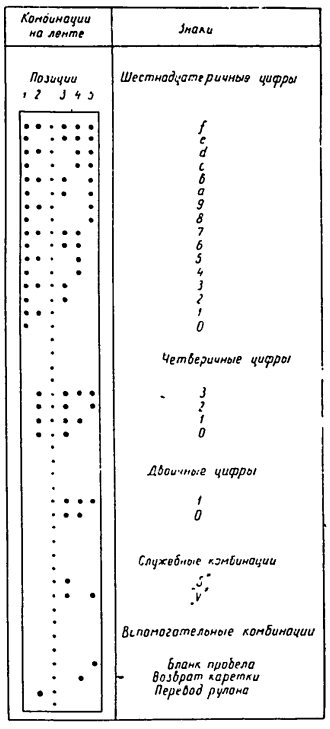
Рис. 13. Перфолента.

ровать только сами коды; после каждого кода на ленте должна быть нанесена пробивка, соответствующая знаку V. Если нужно адреса кодов указать на ленте, то сначала перфорируется адрес каждого кода, дополненный знаками a и S, затем сам код со знаком V.

Код с входного устройства попадает в регистр A арифметического узла и по знаку V записывается или по адресу, указанному в команде ввода, или по адресу, отперфорированному на ленте. В случае отсутствия знака V после кода записи не произойдет. Дело в том, что код вдвигается в регистр A, начиная со старших разрядов, до тех пор, пока не воспримется входным устройством знак V. Запишется тот код, который стоит в этот момент в регистре A, т. е. 34 двоичных разряда, предшествующих знаку V. То же самое относится и к записи адреса по знаку S. Знак a

Таблица III

Кодировка знаков на перфоленте



нужен лишь для сдвига адреса в регистре A на 4 разряда влево. Вместо a можно брать любую шестнадцатеричную цифру.

Таким образом, если при перфорировании ленты была сделана ошибка в коде, а знак V не поставлен, то пробивку кода можно повторить и после него поставить знак V. Это же относится и к перфорированию адресов.

Посторонние пробивки между знаком V предыдущего кода и следующим кодом, между знаком S адреса и кодом, а также непосредственно перед адресом, вообще говоря, не страшны. Однако лучше посторонних пробивок избегать.

Иногда посторонние пробивки могут испортить все дело. Предположим, что в процессе работы вводится один код с фотоэлектрического устройства, причем на ленте сразу после знака V имеются посторонние пробивки. Счет после ввода этого кода должен продолжаться. Так как лента при вводе с фотоэлектрического входного устройства протягивается с большой скоростью, то после ввода кода она еще по инерции продвигается на 3—5 строк. Посторонние пробивки, расположенные после знака кода, попадая в регистр A, изменяют его содержимое, т. е. портят некоторые данные, необходимые для выполнения текущей операции (регистр A участвует в выполнении операций в арифметическом узле). Вычисления сбиваются.

На ленте недопустимы случайные пробивки S и V, так как в этом случае будут произведены записи по неверным адресам, или записи неверных кодов.

Кроме устройства, осуществляющего перфорацию ленты, имеются также устройства для дублирования и сверки готовых лент.

Контроль правильности перфорации может быть осуществлен или сличением двух лент, отперфорированных независимо друг от друга, или считыванием с ленты.

На одну ленту можно поочередно то дублировать данные с другой ленты, то перфорировать, что позволяет удобно монтировать ленту из готовых кусков, внося на нее с клавишного устройства необходимые добавления.

КРАТКАЯ ХАРАКТЕРИСТИКА ЦИФРОВОЙ УНИВЕРСАЛЬНОЙ ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ М-2

Система счисления	двоичная.
Количество двоичных разрядов	34.
Система команд	трехадресная.
Представление чисел	в системе фиксированной запятой и в системе плавающей запятой.
Диапазон изменения чисел (исключая нуль)	в системе фиксированной запятой: $2^{-31} \leqslant \lvert x \rvert \leqslant 1-2^{-33}$; в системе плавающей запятой: $2^{-31}\frac{1}{2} \leqslant \lvert x \rvert \leqslant 2^{31}(1-2^{-26})$.
Внутреннее запоминающее устройство	электростатическое и магнитный барабан на 512 кодов каждое.
Внешнее запоминающее устройство	магнитная лента; объем 50 000 кодов, скорость считывания 30 кодов в секунду.

Продолжение

Входные устройства	электромеханическое со временем ввода одного числа 1,7 сек.; фотоэлектрическое со временем ввода одного числа 0,03 сек.
Выходное устройство	телетайп; скорость печати 21 числа в минуту.
Скорость работы машины	при использовании только электростатического запоминающего устройства около 2000 операций в секунду; при использовании только магнитного барабана около 25 операций в секунду.

ГЛАВА II

ОСОБЕННОСТИ ПРОГРАММИРОВАНИЯ НА МАШИНЕ М-2

§ 1. Использование режимов работы

Как известно, машина М-2 может работать в трех режимах: в режиме плавающей запятой, в режиме фиксированной запятой и в режиме фиксированной запятой с двойной точностью. Тот или иной режим машины задается программным путем с помощью команды переключения и сохраняется до тех пор, пока другая команда переключения его не изменит. Так как ряд команд по-разному выполняется в разных режимах, то и программа, не содержащая внутри себя команд переключения, может выполняться по-разному в зависимости от того, какой последний режим работы машины был задан.

В режиме плавающей запятой мантисса чисел *), участвующих в вычислениях, представляется с относительной точностью до 2^{-26}, что примерно соответствует $1{,}6 \cdot 10^{-8}$. Эта точность выдерживается для мантисс всех чисел, заключенных по абсолютной величине в пределах от 2^{-32} до 2^{+31}, т. е. примерно в пределах от $2{,}5 \cdot 10^{-10}$ до $2 \cdot 10^{+9}$.

Такого диапазона изменения и точности представления чисел достаточно для решения большинства практических задач. Это представляет значительные удобства при программировании, так как дает возможность фактически не учитывать диапазона изменения величин, участвующих в вычислениях.

*) Напомним, что для всех чисел, отличных от нуля, мантисса по абсолютной величине заключена между $1/2$ и 1.

Однако в режиме плавающей запятой менее удобно производить логическую обработку кодов и, в частности, формирование и преобразование команд, что, как известно, занимает значительное место при программировании различных задач. Поэтому иногда приходится, решая задачу в режиме плавающей запятой, переходить на фиксированную запятую. По той же причине для решения задач логического характера используются в основном режимы фиксированной запятой или фиксированной запятой с двойной точностью.

В режиме фиксированной запятой числа, участвующие в вычислениях, могут быть представлены, вообще говоря, с большей точностью, чем при работе с плавающей запятой, так как в последнем случае часть разрядов отводится для изображения порядка числа. Например, числа, близкие к единице, представляются с точностью до 2^{-33}, т. е. примерно с точностью до $1{,}2 \cdot 10^{-10}$. Такая абсолютная точность представления сохраняется для всех чисел, но относительная точность будет уменьшаться вместе с уменьшением их абсолютных величин. Так, для чисел порядка 10^{-6} сохраняется всего лишь около 4 значащих десятичных цифр. Существенной особенностью представления чисел в режиме фиксированной запятой является то обстоятельство, что все числа, участвующие в вычислениях в этом режиме, должны быть по абсолютной величине меньше единицы.

Эти особенности вносят целый ряд трудностей в программирование и подготовку задач для решения их в режиме фиксированной запятой. Однако существуют некоторые методы, позволяющие в какой-то степени избежать этих трудностей.

Один из таких методов — метод плавающих масштабов — успешно используется в вычислительном центре МГУ. Этот метод будет подробно изложен в § 6 настоящей главы.

Иногда при решении какой-либо задачи можно получить хорошие результаты, используя для вычислений как первый, так и второй режимы: в режиме фиксированной запятой можно производить часть вычислений с несколько повышенной точностью, если точности чисел с плавающей запятой в этой части вычислений недостаточно, а в режиме плавающей запятой — остальные вычисления, как правило, по более простой программе.

Например, если требуется найти корень уравнения $x = f(x)$ итерациями с точностью до 9 верных десятичных знаков, то можно в режиме плавающей запятой получить приближение с точностью до шести-семи верных десятичных знаков, а затем на фиксированной запятой произвести необходимые уточнения. На первый взгляд может показаться, что этот прием нерационален, так как вместо одной программы нужно иметь две, работающие в разных режимах, но на самом деле таким образом часто можно получить выигрыш и в количестве занятых ячеек и в скорости счета. Дело в том, что если нужна большая точность вычислений, а участвующие в них величины меняются в широком диапазоне, то решение придется вести в режиме фиксированной запятой по довольно сложной программе с введением для ряда величин масштабных множителей и автоматическим изменением их по ходу вычислений. Если же достаточно хорошее приближение уже найдено по программе в режиме плавающей запятой, то окончательное уточнение можно провести по программе в режиме фиксированной запятой уже с постоянными масштабами, так как получаемые приближения будут мало отличаться друг от друга и поэтому легко будет подобрать постоянные масштабные множители.

Обе эти программы будут сравнительно простыми и в ряде случаев могут вместе занимать меньшее количество ячеек памяти, чем первая более громоздкая программа. Точно так же скорость счета в этом случае может очень заметно возрасти, если мы имеем дело с достаточно большим числом итераций.

Третий режим — режим фиксированной запятой с двойной точностью — в основном предназначен для проведения вычислений с удвоенным и вообще с многократно увеличенным числом знаков, что, как известно, без применения этого специального режима требует чрезмерного усложнения программы [2].

В этом же режиме удобно производить логическую обработку кодов, поэтому данный режим работы широко используется при решении задач логического характера. Подробнее об использовании этого режима для логической обработки кодов будет рассказано в § 5 настоящей главы.

§ 2. Преобразование команд

Как уже указывалось выше, один и тот же код расшифровывается машиной по-разному в зависимости от того, используется ли этот код в качестве команды или в качестве числа.

В последнем случае при выполнении некоторых операций один и тот же код может восприниматься либо как число в системе плавающей запятой, либо как число в системе фиксированной запятой.

Если требуется изменить какую-либо команду, то удобно рассматривать ее как число в системе фиксированной запятой и пользоваться теми операциями, которые осуществляют работу над числами в этой системе. Иногда для этого приходится изменять режим работы машины.

Команда $n\,k\,l\,p$ эквивалентна числу в системе фиксированной запятой, абсолютная величина которого есть

$$n2^{-10}+k2^{-20}+l2^{-30}+\left[\frac{p}{2}\right]2^{-33}, \qquad (\text{II. 1})$$

где n, k, l и p всегда положительны, так как они являются номерами ячеек памяти и номером операции, а $\left[\frac{p}{2}\right]$ означает целую часть от числа $\frac{p}{2}$. Так как знаковый разряд кода используется в командах для кодирования номера операции, то каждая команда, рассматриваемая как число, будет положительной или отрицательной в зависимости от четности номера операции, а именно: положительной, если номер операции нечетный, и отрицательной, если номер операции четный. С учетом этих замечаний преобразование команд производится по общим правилам действий в системе фиксированной запятой.

Например, если мы хотим из команды

$$3.70\quad 2.22\quad 0.10\quad 4$$

получить команду

$$3.70\quad 2.22\quad 0.11\quad 4,$$

т. е. увеличить первый адрес команды на единицу, то, используя представление команды в виде числа в системе фиксированной запятой, это можно сделать двумя способами: либо вычесть (операция a) из этой команды число 2^{-30}, записанное в системе фиксированной запятой, т. е. вычесть код

$$0.00\quad 0.00\quad 0.01\quad 1,$$

либо прибавить (операция b) к этой команде код

$$0.00\quad 0.00\quad 0.01\quad 0,$$

изображающий -2^{-30} в системе фиксированной запятой.

Другой пример. Нужно получить из команды

$$\alpha\quad \beta\quad \gamma\quad 9$$

команду

$$\alpha+k\quad \beta+m\quad \gamma+n\quad 9.$$

Для этого достаточно из первой команды вычесть (операция a) константу

$$k\quad m\quad n\quad 0.$$

Если бы нам нужно было из той же команды получить команду

$$\alpha+k\quad \beta-m\quad \gamma+n\quad 9,$$

где $k\neq 0$, то пришлось бы из первой команды вычесть константу

$$k-1\quad \overline{m}\quad n\quad 0,$$

где $\overline{m}=1024-m$, т. е. дополнение m до единицы следующего адреса. Действительно, для получения требуемой команды необходимо, учитывая положительный знак первоначальной команды, прибавить (операция b) к ней число

$$k2^{-10}-m2^{-20}+l2^{-30}=$$
$$=(k-1)2^{-10}+(2^{10}-m)2^{-20}+l2^{-30},$$

или вычесть (операция a) это число, взятое с обратным знаком, которое в системе фиксированной запятой и примет вид указанной выше константы.

Приведенные примеры изменения адресов команды, т. е. ее переадресации, позволяют сделать несколько методических указаний о подборе констант для переадресации.

В том случае, когда все изменяемые адреса какой-либо команды нужно одновременно увеличить или уменьшить, то константа переадресации должна содержать в каждом адресе столько единиц, на сколько нужно изменить соответствующий адрес преобразуемой команды, и иметь знак плюс или минус (1 или 0 в 34-м разряде) в зависимости от знака преобразуемой команды, направления изменения адресов и кода операции команды переадресации.

Другой случай, когда в преобразуемой команде направление изменения адресов различное, можно свести к первому случаю, если представить величину, на которую нужно изменить данную команду, в виде (II. 1), взяв коэффициенты при 2^{-10}, 2^{-20} и 2^{-30} со своими знаками, а затем, применяя дополнительный код, преобразовать эту запись так, чтобы коэффициенты при данных степенях двойки были одного знака. Именно так мы и поступили в последнем из вышеприведенных примеров.

Некоторый произвол в выборе знака константы переадресации позволяет при изменении некоторых команд на одну и ту же величину в разных направлениях использовать только константу с каким-либо определенным знаком, а направление изменения команды учитывать при выборе операции, с помощью которой будет производиться указанное изменение команды. Это позволяет при составлении программ экономить число констант.

Перейдем теперь к преобразованию команды, связанному не только с изменением ее адресной части, но и кода операции. При этом для лучшего понимания этого вопроса рассмотрим сначала изменение команды, у которой во всех трех адресах стоят нули. Такая команда эквивалентна некоторому числу $q \cdot 2^{-33}$ в системе фиксированной запятой, где q — целое число, удовлетворяющее неравенству $-7 \leqslant q \leqslant 7$. Например, команда *0.00 0.00 0.00 a* эквивалентна числу $-5 \cdot 2^{-33}$.

Преобразование таких команд будет понятно из рассмотрения следующих примеров, причем код операции записывается в двух системах счисления: шестнадцатеричной и двоичной и для простоты записи опускается адресная часть команды:

1. $+\begin{array}{l} 9 = 1001 \sim +4 \cdot 2^{-33} \\ 3 = 0011 \sim +1 \cdot 2^{-33} \end{array}$
$+5 \cdot 2^{-33} \sim 1011 = b.$

2. $+\begin{array}{l} 9 = 1001 \sim +4 \cdot 2^{-33} \\ 4 = 0100 \sim -2 \cdot 2^{-33} \end{array}$
$+2 \cdot 2^{-33} \sim 0101 = 5.$

3. $-\begin{array}{l} 7 = 0111 \sim +3 \cdot 2^{-33} \\ 6 = 0110 \sim -3 \cdot 2^{-33} \end{array}$
$+6 \cdot 2^{-33} \sim 1101 = d.$

Рассмотрим более сложный пример на формирование команд. Пусть нам нужно из команды

$$N \quad \alpha \quad \beta \quad 7$$

сформировать команду

$$L \quad \alpha \quad \beta \quad 4.$$

Для этого необходимо первоначальную команду взять со знаком формируемой команды и добавить к ней некоторую константу. (Знак первоначальной команды определяется из тех соображений, чтобы величины α и β, которые заранее могут быть неизвестны, не фигурировали в искомой константе.) Требуемую константу легко получить как решение уравнения

$$\varepsilon a + x = b,$$

где a и b — числа в системе фиксированной запятой, эквивалентные первоначальной и формируемой командам, ε принимает значения $+1$ или -1, а x — искомая константа.

Так как в данном случае $\varepsilon = -1$, то получим

$$x = a + b = N2^{-10} + \alpha 2^{-20} + \beta 2^{-30} + 3 \cdot 2^{-33} - \\ - (L2^{-10} + \alpha 2^{-20} + \beta 2^{-30} + 2 \cdot 2^{-33}) = \\ = (N - L)\, 2^{-10} + 1 \cdot 2^{-33}.$$

Здесь возможны два случая: $N \geqslant L$ и $N < L$. В первом случае, когда знаки коэффициентов при степенях двойки одинаковы, искомая константа имеет вид

$$N - L \quad 0.00 \quad 0.00 \quad 3.$$

Во втором случае, когда знаки коэффициентов при степенях двойки противоположны, последнее соотношение можно привести к виду

$$x = -\left[(L - N - 1)\,2^{-10} + (2^{10} - 1)\,2^{-20} + (2^{10} - 1)\,2^{-30} + 7 \cdot 2^{-33}\right],$$

т. е. искомая константа будет иметь вид

$$L - N - 1 \quad 3.ff \quad 3.ff \quad e.$$

Формируемая команда как в первом, так и во втором случае получается вычитанием первоначальной команды из найденной константы.

В этом примере мы опять свели действие над командами к действиям над эквивалентными им числами в системе фиксированной запятой.

Довольно часто при составлении программ для машины М-2 применяется изменение знака команд — преобразование команд, связанное с особенностями кодирования операций в машине. Дело в том, что все операции разбиты на пары, в каждой из которых номера операций отличаются только знаковым разрядом. При этом пары образуют такие операции, как умножение (операция *9*) и деление (операция *8*), сложение (операции *b* и *d*) и вычитание (операции *a* и *c*), алгебраическое сравнение (операция *6*) и сравнение по модулю (операция *7*). Это дает возможность из одной операции такой пары получить другую только изменением знака. Основываясь на этом, можно, например, легко получить из команды, образующей сумму двух чисел, команду, образующую разность этих чисел. Для этого достаточно преобразуемую команду вычесть из нуля.

С помощью операции *e* (перенос числа с присвоением знака другого числа) можно той или иной команде присвоить знак любого числа. Это обстоятельство следующим образом можно использовать, например, при вычислении значений $\sin x$. Сначала можно получить в какой-либо ячейке α значение $\sin|x|$, а для учета знака x использовать переменную команду N:

$$(N)^*: \quad \alpha \quad G \quad \alpha \quad b^*,$$

где (G): *0.00 0.00 0.00 0*. Этой команде предварительно присваивается знак x с помощью команды L:

$$(L): \quad N \quad N \quad \beta \quad e,$$

где $(\beta) = x$. В результате этого команда N будет производить вычитание из нуля значения $\sin|x|$, если $x < 0$, и сложение с нулем значения $\sin|x|$ в противном случае.

Встречаются при программировании и более сложные способы формирования команд, требующие применения операций умножения и деления в режиме фиксированной запятой. Указать все возможные способы формирования переменных команд довольно трудно. С некоторыми из них можно познакомиться при разборе программ, приведенных во II части. Некоторые возможности будут указаны ниже в § 5 этой главы, когда будут рассматриваться различные приемы обработки кодов в режиме фиксированной запятой с двойной точностью. Здесь же ограничимся еще одним примером.

Пусть в какой-либо программе, работающей в режиме плавающей запятой, нужно по двум переменным командам

$$(M)^*: \quad \alpha \quad \alpha \quad \beta + i^* \quad d,$$
$$(M+1)^*: \quad N \quad \alpha \quad \gamma + i^* \quad 7$$

сформировать следующую команду L:

$$(L)^*: \quad \alpha \quad \beta + i^* \quad \gamma + i^* \quad b.$$

Для этого нужно поочередно извлечь из этих команд величины

$$0.00 \quad 0.00 \quad \beta + i \quad 0,$$
$$0.00 \quad 0.00 \quad \gamma + i \quad 0,$$

установить временно режим фиксированной запятой и затем из первой величины делением ее на число 2^{-10} получить величину

$$0.00 \quad \beta + i \quad 0.00 \quad 0.$$

После этого известным уже способом при помощи операций *a* и *b* легко сформировать указанную команду L. По окончании формирования этой команды нужно снова уста-

новить режим плавающей запятой. Часть программы, осуществляющая такое формирование команды, будет иметь следующий вид:

N	δ	M	G	f	Извлечение $\beta+i$
$N+1$	L	$M+1$	G	f	Извлечение $\gamma+i$
$N+2$	0.00	0.18	$N+3$	0	Режим ФЗ
$N+3$	δ	δ	$G+1$	8	Сдвиг $\beta+i$ во II адрес
$N+4$	L	L	δ	b	Формирование (L)
$N+5$	L	$G+2$	L	a	
$N+6$	0.00	0.12	$N+7$	0	Режим ПЗ
G	0.00	0.00	$3.ff$	0	Выделитель $1A$
$G+1$	0.01	0.00	0.00	1	$(2^{-10})_\phi$
$G+2$	α	0.00	0.00	b	Константа для формирования

§ 3. Циклы

Возможность преобразования команд позволяет многократно использовать отдельные части программы, видоизменяя их в случае необходимости. Многократное обращение к этим частям реализуется с помощью команд условной и безусловной передачи управления. Особенно важны для образования циклов команды условной передачи управления.

Если заранее известно число повторений в данном цикле, то для обеспечения этого числа повторений можно устроить счетчик, к которому с каждым повторением добавляется единица *), и в конце цикла выясняется, выполнено ли нужное число повторений. Рассмотрим в качестве примера вычисление полинома

$$a_0x^n+a_1x^{n-1}+\ldots+a_{n-1}x+a_n=$$
$$=(\ldots((a_0x+a_1)x+a_2)x+\ldots+a_{n-1})x+a_n$$

*) В качестве единицы счетчика можно использовать произвольное число, имеющее представление в машине, например единицу первого адреса.

с помощью схемы Горнера в режиме плавающей запятой. Пусть коэффициент a_i этого полинома расположен в ячейке $A+l$, а значение x — в ячейке β. Ячейка γ используется для хранения состояния счетчика, ячейка α используется в качестве рабочей.

Тогда программа, реализующая данный алгоритм, будет выглядеть так *):

	N	$N+1$	γ	G	4	$0\Rightarrow$ в счетчик γ
	$N+1$	$N+2$	$N+4$	$G+1$	4	Восстановление ($N+4$)
	$N+2$	$N+3$	α	A	4	$a_0\Rightarrow\alpha$
	$N+3$	α	α	β	9	$(\alpha)\,x\Rightarrow\alpha$
	$N+4^*$	α	α	$A+i^*$	d	$(\alpha)+a_i\Rightarrow\alpha$
при $i<n$	$N+5$	γ	γ	$G+2$	b	Добавление 1 к счетчику
	$N+6$	$N+4$	$N+4$	$G+2$	b	Переадресация ($N+4$)
	$N+7$	$N+3$	γ	$G+3$	7	Сравнение на окончание цикла
	G	0.00	0.00	0.00	0	
	$G+1$	α	α	$A+1$	d	
	$G+2$	0.00	0.00	0.01	1	
	$G+3$	0.00	0.00	$n+1$	$0.$	

При рассмотрении данной программы легко убедиться в том, что в ней функции счетчика числа повторений γ косвенным образом дублируются переменной командой $N+4$. Поэтому программу можно упростить, если для сравнения на окончание цикла использовать конечный вид переменной

*) В программах, приводимых в этой главе, для краткости опущены некоторые команды (например, команды останова или печати), которые не существенны для понимания рассматриваемых здесь приемов программирования.

команды. Ниже приводится эта программа с указанным упрощением:

	$N+1$	$N+2$	$N+4$	$G+1$	4	Восстановление $(N+4)$
	$N+2$	$N+3$	α	A	4	$a_0 \Rightarrow \alpha$
при $i \leqslant n$	$N+3$	α	α	$\beta \cdot 9$	$(\alpha)\, x \Rightarrow \alpha$	
	$N+4$	α	α	$A+i$	d	$(\alpha)+a_i \Rightarrow \alpha$
	$N+5$	$N+4$	$N+4$	$G+2$	b	Переадресация $(N+4)$
	$N+6$	$N+3$	$N+4$	$G+3$	7	Сравнение на окончание цикла
	$G+1$	α	α	$A+1$	d	
	$G+2$	0.00	0.00	0.01	1	
	$G+3$	α	α	$A+n+1$	d.	

При введении дополнительного счетчика числа повторений цикла мы всегда добавляем к циклу одну команду для изменения состояния счетчика. Это увеличивает количество операций, выполняемых при одном обращении к циклу, а значит, и время работы цикла. Кроме того, вне цикла вместе с командами, восстанавливающими первоначальное состояние переменных команд, нужно иметь дополнительную команду, восстанавливающую первоначальное состояние счетчика, а также константу, указывающую на число повторений данного цикла (правда, аналогичную константу в виде конечного состояния переменной команды необходимо иметь и в случае, когда функции счетчика числа повторений выполняет переменная команда). Поэтому при использовании в качестве счетчика переменной команды мы сэкономим по сравнению с введением дополнительного счетчика три ячейки: ячейку для счетчика, команду, восстанавливающую первоначальное состояние счетчика, и команду в цикле, изменяющую состояние счетчика.

Переменные команды содержатся в большинстве циклов, число повторений в которых заранее известно. Исключение могут составить некоторые простейшие итерационные циклы с заранее известным числом итераций, как, например, цикл для получения величины x^n.

Наличие в цикле хотя бы одной переменной команды позволяет, как правило, использовать ее в качестве счетчика таким образом, как это было показано в предыдущем примере. Правда, могут быть случаи, когда переменная команда зависит и от других частей программы. Такие переменные команды несколько сложнее использовать в качестве счетчиков. Однако нужно иметь в виду, что команды сравниваются как эквивалентные им числа, так что разряды второго адреса будут старше разрядов первого адреса, а разряды третьего старше разрядов второго. Поэтому, например, если в данном цикле изменяется третий адрес переменной команды, не зависящий от других частей программы, то ее все же можно использовать в качестве счетчика. Если же переменный адрес, зависящий от какой-либо другой части программы, занимает разряды, старшие по отношению к адресам, зависящим от данного цикла, и в цикле нет других переменных команд, которые можно было бы удобно использовать в качестве счетчика, то в этом случае либо приходится все же вводить счетчик, либо в других частях программы вместе с изменением данной переменной команды соответственно изменять и конечный вид этой команды, используемый при сравнении на окончание работы цикла.

Например, команду

$$\alpha+j \quad \beta \quad \gamma+l \quad d$$

можно использовать в качестве счетчика в цикле по параметру j без вспомогательных усложнений программы, а для того, чтобы использовать эту команду в качестве счетчика в цикле по параметру i, необходимо вместе с изменением данной команды по j соответственно изменять по j и ее конечный вид:

$$\alpha+j \quad \beta \quad \gamma+n \quad d,$$

где n — последнее значение, принимаемое i в этом цикле. (В зависимости от того, производится ли переадресация по i этой переменной команды до или после ее выполнения, в цикле эта команда в конечном виде может выполняться или не выполняться.)

Приведем пример более сложной зависимости адресов переменной команды от параметров:

$$\alpha+ni+j \quad \beta \quad \gamma+j \quad d,$$

где $j < n$. Эту команду можно использовать в качестве счетчика в цикле по параметру i, так как зависимость от j старшего адреса несущественна.

Иногда число повторений в цикле меняется по ходу вычислений, хотя и остается известным заранее перед каждым обращением к нему. В этом случае при использовании переменной команды в качестве счетчика нужно при каждой смене числа повторений изменять конечный вид переменной команды.

Параметр цикла может непосредственно в качестве числа участвовать в вычислениях, производимых данным циклом. Тогда этот параметр-число можно также использовать в качестве счетчика. Однако здесь нужно иметь в виду, что если этот параметр будет представляться приближенным числом, то он в качестве счетчика не всегда будет давать нужный результат. Так, например, если $^1/_3$ в двоичной форме представляется в виде приближенного числа с недостатком, то сумма $^1/_3 + ^1/_3 + ^1/_3$ будет несколько меньше единицы, поэтому при сравнении ее с точной единицей мы не получим равенства, что может привести к неправильной работе цикла, использующего $^1/_3$ в качестве единицы счетчика повторений. Для правильной работы цикла в этом случае надо соответственно подобрать и константу для сравнения.

Если число повторений какого-либо цикла заранее неизвестно, т. е. оно существенным образом зависит от результатов вычислений в данном цикле, то часто приходится производить ряд дополнительных вычислений, подготовляющих проверку того или иного условия, например оценку точности полученных результатов. При составлении таких циклов для работы в режиме плавающей запятой нужно осторожно подходить к заданию абсолютной погрешности вычислений, учитывая, что работа в режиме плавающей запятой обеспечивает только постоянную относительную погрешность. Абсолютная же погрешность больших величин может быть большой.

Во всех циклах, в которых функции счетчика выполняют переменные команды, сравнение на окончание работы по циклу целесообразно производить с помощью операции 7 (сравнение по модулю). Это дает возможность при программировании не думать о знаке той или иной команды, участвующей в сравнении, что уменьшает возможность ошибки.

Из сказанного выше видно, что существенным элементом всякого цикла является константа, участвующая в сравнении на окончание работы цикла. Часто число таких констант можно сократить. Дело в том, что эти константы можно задавать довольно грубо, поэтому иногда одна и та же константа может использоваться в двух или нескольких циклах. Так, например, из двух констант

$$(G): \alpha + n \quad \beta \quad \beta \quad 9,$$
$$(G+1): \alpha + n \quad \gamma \quad \gamma \quad d,$$

участвующих в сравнении в разных циклах, достаточно оставить одну наименьшую. Иногда в качестве константы можно использовать какую-либо команду, оказавшуюся подходящей для этой цели.

На машине М-2 есть команды, на правильность работы которых не влияет состояние некоторых их разрядов. Эти разряды можно использовать для задания какой-нибудь дополнительной информации. Так, например, в командах переключения (операция *0*) такими будут старшие 15 разрядов; эти разряды можно использовать для задания констант, младшие разряды которых не влияют на характер выполнения программы. Этот прием использовался, например, в программе для перевода из двоичной системы счисления в десятичную, помещенной во II части.

§ 4. Использование передач управления

Интересной особенностью машины М-2 является совмещение операции безусловной передачи управления с рядом других операций.

Это позволяет при составлении программ почти не пользоваться дополнительными командами для осуществления безусловных передач управления, а использовать для этой цели имеющиеся возможности в других командах программы, например, при установлении того или иного режима работы, при восстановлении переменной команды или засылке какой-либо величины в рабочую ячейку и т. д.

Особенно большие возможности дает сочетание в одной команде безусловной передачи управления и переноса числа (операция *4*).

Так как восстановление переменных команд, а также засылка различных величин в рабочие ячейки, производится в основном с помощью этой операции, то практически всегда имеется возможность без введения дополнительных команд начинать работу цикла не с первой его команды, а с какой-нибудь средней, что позволяет избежать выполнения вхолостую некоторых команд и сэкономить некоторое количество тактов. Например, приведенную выше программу для вычисления полинома степени n по схеме Горнера можно переписать в следующем виде:

	$N+1$	α	β	A	9	$a_0 x \Rightarrow \alpha$
	$N+2$	$N+4$	$N+4$	$G+1$	4	Восстановление $(N+4)$
при $l \leqslant n$	$N+3$	α	α	β	9	$(\alpha)\, x \Rightarrow \alpha$
при $l \leqslant n$	$N+4^*$	α	α	$A+i^*$	d	$(\alpha)+a_i \Rightarrow \alpha$
при $l \leqslant n$	$N+5$	$N+4$	$N+4$	$G+2$	b	Переадресация $(N+4)$
при $l \leqslant n$	$N+6$	$N+3$	$N+4$	$G+3$	7	Сравнение на окончание цикла
	$G+1$	α	α	$A+1$	d	
	$G+2$	0.00	0.00	0.01	1	
	$G+3$	α	α	$A+n+1$	d	

Эти изменения позволяют сократить на один такт время выполнения программы.

Благодаря тому, что в машине М-2 имеется ряд команд, осуществляющих безусловную передачу управления, можно иногда заметно сократить количество ячеек памяти, занятых программой, за счет выполнения переменных команд разных частей программы в одних и тех же ячейках.

Если, например, в цикле имеются несколько переменных команд, расположенных подряд, причем их можно поставить в таком порядке, чтобы последняя из них содержала безусловную передачу управления, то эту группу команд не обязательно помещать вместе с циклом, в котором они работают. Для их хранения можно использовать свободную в этот момент группу рабочих ячеек, куда засылается исходный вид переменных команд при восстановлении или формировании. Для того чтобы включить эти команды в работу

цикла, достаточно в цикле иметь команду безусловной передачи управления первой из них (что часто можно сделать, не затрачивая для этого дополнительной команды), а последняя переменная команда должна реализовать возврат в нужное место цикла. Если каждая из переменных команд содержит безусловную передачу управления, то рабочие ячейки для их хранения могут быть расположены в произвольном порядке, нужно только исходный вид переменных команд задать таким образом, чтобы осуществлялся необходимый порядок их работы.

Если переменные команды какого-либо цикла восстанавливаются при каждом обращении к этому циклу, то ячейки, где расположены эти команды, можно также использовать в качестве рабочих ячеек в других частях программы.

Например, если нужно в цикле осуществлять поочередно перенос чисел из некоторых ячеек $\beta+l$, $l=0, 1, \ldots, n$, в рабочую ячейку α и имеется рабочая ячейка L, то можно перед обращением к данному циклу заслать в ячейку L команду

$$N \quad \alpha \quad \beta \quad 4$$

и начать с нее работу цикла. Здесь (N) — команда цикла, которая должна выполняться вслед за командой, помещенной в ячейку L. В конце цикла нужно поставить две команды:

$(N+k-1)$:	L	L	G	a	Изменение (L)
$(N+k)$:	L	L	$G+1$	7	Сравнение на окончание цикла

Здесь

(G):	0.00	0.00	0.01	1,
$(G+1)$:	N	α	$\beta+n+1$	4.

В этом цикле данная переменная команда выполняет функции счетчика. Обращение к ней происходит с помощью команды сравнения $(N+k)$ в случае, если еще не сделано нужное число повторений.

Таким образом, на месте одной переменной команды можно выполнять указанным образом несколько переменных команд, если состояние предыдущей переменной команды не влияет на дальнейшие вычисления.

Рассмотрим еще несколько приемов использования операции переноса числа.

При решении задач часто оказывается необходимым некоторый участок программы в первый раз пройти несколько иначе, чем в последующие. Это достигается следующим образом. Пусть данный участок программы начинается с ячейки N. В первый раз в нее можно поместить команду:

$$(N):\quad L\quad N\quad G\quad 4,$$

где (L) — команда, которая в первый раз должна выполняться вслед за командой N, а (G) — команда, передающая управление той команде, с которой в дальнейшем должна начинаться работа по данному участку программы. Команда (N), выполняясь первый раз, заменяет себя командой (G) и передает управление ячейке L. Порядок следующего прохождения этого участка программы теперь уже будет зависеть от вида новой команды, которая будет находиться в ячейке N.

Если соответствующим образом подобрать команду (G), то можно осуществить обращение поочередно то в одно место программы, то в другое. Пусть при выполнении некоторого участка программы нам нужно обращаться то к команде (L), то к команде (M). Для этого в некоторую ячейку N первоначально поместим команду

$$(N):\quad L\quad N\quad G\quad 4,$$

а (G) выберем следующим образом:

$$(G):\quad M\quad N\quad G+1\quad 4,$$

где

$$(G+1):\quad L\quad N\quad G\quad 4.$$

При многократном прохождении участка программы, содержащего команду (N), в ячейке N будут находиться то команда

$$M\quad N\quad G+1\quad 4,$$

то команда

$$L\quad N\quad G\quad 4.$$

При выполнении каждой из этих команд в ячейку N засылается другая команда, благодаря чему осуществляется «переключение» передачи управления.

Такое переключение можно сделать не только двухпозиционным, но и вообще n-позиционным, для чего нужно заготовить n констант:

$$\begin{array}{rcccc}
(G): & L_0 & N & G+1 & 4, \\
(G+1): & L_1 & N & G+2 & 4, \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
(G+i): & L_i & N & G+i+1 & 4, \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
(G+n-1): & L_{n-1} & N & G & 4.
\end{array}$$

Первоначальное состояние ячейки N во всех случаях необходимо восстанавливать в начале программы. Для осуществления n-позиционного переключателя необходимо $n+2$ ячейки оперативной памяти: n ячеек для констант, ячейку, где будет выполняться переменная команда, осуществляющая переключение, и ячейку для команды восстановления первоначального состояния переключателя.

Для двухпозиционного переключателя при таком способе требуется 4 ячейки памяти, но оказывается, что его можно осуществить несколько компактнее с помощью всего лишь двух команд:

$$\begin{array}{rcccc}
(N): & N+1 & G & N+1 & a \\
(N+1)^{*}: & L & L_1 & L_2 & 7^{*},
\end{array}$$

где

$$(G):\quad 0.00\quad 0.00\quad 0.00\quad 0,$$

а (L_1) и (L_2) — какие-либо два кода, удовлетворяющие следующим условиям:

а) $|(L_1)| \geqslant |(L_2)|$,

б) $(L_1) < (L_2)$.

Такие коды в программе практически всегда можно подобрать, а нуль обычно всегда имеется в памяти. Поэтому на осуществление такого переключателя требуется три ячейки вместо четырех (кроме ячеек N и $N+1$, еще требуется одна ячейка для команды, которая восстанавливает исходный вид $(N+1)$ присвоением ей определенного знака), но зато будет выполняться каждый раз одна лишняя операция.

§ 5. Использование режима фиксированной запятой с двойной точностью

Как уже указывалось при описании системы операций машины, результат умножения в режиме фиксированной запятой с двойной точностью получается с удвоенным количеством

разрядов, а при делении, кроме частного, получается и остаток. Это обстоятельство существенно облегчает программирование в том случае, когда необходимо выдержать высокую точность результата вычислений (более десяти знаков), так как позволяет построить сравнительно компактные подпрограммы для выполнения операций с удвоенным и вообще с многократно увеличенным числом разрядов *). На машинах, не имеющих такого рода операций, вести вычисления с бо́льшим числом знаков, чем то, на которое рассчитана машина, практически невозможно из-за чрезмерного усложнения программы.

Кроме того, режим фиксированной запятой с двойной точностью используется иногда и в тех случаях, когда большой точности вычислений не требуется. Этот режим часто помогает сократить число команд программы, что в особенности важно для стандартных программ. Использование режима фиксированной запятой с двойной точностью при выделении целой и дробной частей числа, при переводе чисел из десятичной системы в двоичную и обратно позволило значительно упростить соответствующие подпрограммы и сократить количество тактов, необходимых для их работы.

Наиболее распространенные приемы использования этого режима связаны с разбиением кода на две части. Если, например, нужно отделить условный порядок числа $x = 2^p X$, записанного в системе плавающей запятой, от его мантиссы, то достаточно выполнить в режиме фиксированной запятой с двойной точностью следующую команду:

$$\beta \quad \alpha \quad G \quad 9,$$

где

$$(G) = \left(2^{-27}\right)_\Phi, \quad (\alpha) = (x)_\Pi = (\operatorname{sign} X \cdot [(32+p) \cdot 2^{-6} + |X| \cdot 2^{-7}])_\Phi.$$

В результате выполнения этой команды в ячейке β получится величина $\left(\frac{1}{2} X\right)_\Phi$, а в ячейке $\beta + 1$ получится величина

$$\left(\operatorname{sign} X \cdot (32+p)\, 2^{-33}\right)_\Phi.$$

*) Правда, некоторое неудобство на машине М-2 составляет тот факт, что в режиме фиксированной запятой с двойной точностью по третьему адресу команды записываются младшие разряды результата умножения, а не старшие.

Рассмотрим другой пример. Если в этом режиме умножить некоторое число $(X)_\Phi$ на $\left(10 \cdot 2^{-33}\right)_\Phi$, то в ячейке, указанной по третьему адресу команды умножения, получится величина $([10 \cdot X])_\Phi$, а в следующей по порядку ячейке — величина $\left([10 \cdot X] \cdot 2^{-33}\right)_\Phi$, где фигурные и квадратные скобки обозначают целую и дробную части числа. Таким путем можно выделять десятичные цифры числа $(X)_\Phi$.

Иногда одной командой в режиме фиксированной запятой с двойной точностью можно одновременно выполнить умножение или деление и очистить какую-либо ячейку.

Например, при выполнении команды

$$\beta \quad \alpha \quad G \quad 8,$$

где (α): *0.00 0.00 3.ff 0* и $(G) = \left(2^{-10}\right)_\Phi$,
в ячейке β получается число *0.00 3.ff 0.00 0*, т. е. производится сдвиг величины *3.ff* из первого адреса во второй и одновременно засылается нуль в ячейку $\beta + 1$.

Подобные приемы широко используются в стандартных подпрограммах, помещенных во II части.

§ 6. Метод плавающих масштабов

Выше уже указывалось, что при решении задач в режиме фиксированной запятой все участвующие в вычислениях числа должны быть меньше единицы по абсолютной величине. Поэтому при решении большинства практических задач для некоторых величин y_i приходится вводить масштабные множители μ_i. В этом случае в вычислениях участвуют величины $\mu_i y_i = Y_i$, а иногда и μ_i, причем в процессе решения задачи должны выполняться неравенства $|Y_i| < 1$ и $0 < \mu_i < 1$.

Некоторые величины x_i, участвующие в вычислениях, могут быть по абсолютной величине настолько малы, что будут изображаться как числа в системе фиксированной запятой с весьма низкой относительной точностью. Для таких величин в целях повышения относительной точности вычислений приходится вводить масштабные множители $\frac{1}{\nu_i}$. В этом случае в вычислениях участвуют величины $\frac{x_i}{\nu_i} = X_i$, а иногда и ν_i, причем в процессе решения задачи должны выполняться неравенства $0 < r \leqslant |X| < 1$ и $0 < \nu_i < 1$, где r выбирается из соображений точности.

Поскольку в практических задачах мы, как правило, не можем заранее точно определить диапазон изменения той или иной величины, то масштабные множители подбираются обычно довольно грубо, что сильно влияет на точность вычислений. Но даже и в том случае, если диапазон изменения величины известен точно, но достаточно велик, то масштабные множители приходится вводить, исходя из наибольшего абсолютного значения данной величины. Ясно, что такой масштабный множитель вносит большую относительную погрешность, когда величина в ходе своего изменения становится сравнительно небольшой.

Введение масштабных множителей, естественно, усложняет и программирование, так как при составлении программы приходится учитывать, с какими масштабными множителями получаются не только интересующие нас величины, но и все промежуточные результаты. Кроме того, при выборе масштабных множителей необходимо предусмотреть, чтобы ни один из промежуточных результатов не превосходил по абсолютной величине единицы. Для этого иногда приходится существенно видоизменять исходные формулы.

Для повышения точности вычислений было бы целесообразно всю задачу разбить на этапы так, чтобы на каждом из них диапазон изменения каждой из величин y_i был невелик, и на каждом этапе подобрать для величин свои масштабные множители.

Здесь мы опять сталкиваемся с той трудностью, что заранее можем не знать характера поведения некоторых величин, и поэтому для них трудно заранее подбирать масштабные множители. Но эту задачу уже можно поручить машине, составив программу таким образом, чтобы машина сама производила подбор этих множителей для каждого этапа вычислений.

Такой метод решения задач в режиме с фиксированной запятой — *метод плавающих масштабов* — применяется в вычислительном центре МГУ.

Сущность этого метода заключается в следующем. Часть величин, участвующих в решении задачи, представляется в виде

$$y_i = 2^{p_i} Y_i, \tag{II.2}$$

где p_i — целые положительные, отрицательные или равные нулю числа, а Y_i удовлетворяют неравенству

$$|Y_i| < 1.$$

Масштабными множителями в этом случае будут величины 2^{-p_i}.

Величины y_i хранятся в этом случае в двух ячейках. В одной ячейке хранится величина p_i, которая называется *масштабом*, или *порядком* y_i и представляется в машине условным числом, например количеством единиц первого адреса. В другой ячейке хранится величина Y_i, которая называется *мантиссой* y_i и представляется обычным числом в системе фиксированной запятой.

Решение задачи разбивается на такие этапы, в каждом из которых масштабы величин, участвующих в вычислениях, остаются постоянными. Определенная часть программы при переходе от одного этапа к другому производит исследование мантисс и при некоторых условиях производит изменение масштабов некоторых величин.

Условия, которые определяют смену масштабов, заключаются в том, что величины Y_i должны после смены масштабов удовлетворять неравенству

$$0 < r \leqslant |Y_i| < R \leqslant 1, \tag{II.3}$$

где значение R подбирается из тех соображений, чтобы в процессе вычислений на очередном этапе не произошло переполнения, а значение r определяется из соображений точности вычислений. Если какая-либо величина y_i в процессе своего изменения становится сравнительно небольшой по абсолютной величине, то иногда не нужно проверять выполнения условия $r \leqslant |Y_i|$, так как часто точное представление слишком малой величины не повышает точности вычислений.

Проверка условия (II.3) для величины Y_i и необходимое изменение масштаба p_i мы называем *нормализацией* *) величины y_i. При изменении масштаба величины y_i соответственно изменяется и ее мантисса Y_i.

При учете порядков основные арифметические операции выполняются следующим образом.

Пусть имеются две величины: $x = 2^p X$ и $y = 2^q Y$. Их произведение находится по формуле $x \cdot y = 2^{p+q}(X \cdot Y)$,

*) Следует отметить, что приведенное понятие нормализации является более общим по сравнению с нормализацией чисел, производимой при выполнении арифметических операций в режиме плавающей запятой.

после чего иногда необходимо произвести нормализацию $x \cdot y$. Частное $\frac{x}{y}$ находится по формуле $\frac{x}{y} = 2^{p-q}\left(\frac{X}{Y}\right)$, причем масштабы p и q должны быть подобраны таким образом, чтобы $\left|\frac{X}{Y}\right| < 1$. Сумма и разность этих величин находятся по одной из формул

$$x \pm y = 2^p\left(X \pm 2^{-(p-q)}Y\right), \quad \text{если} \quad p \geqslant q,$$
$$x \pm y = 2^q\left(2^{-(q-p)}X \pm Y\right), \quad \text{если} \quad p < q.$$

В этом случае масштабы p и q должны быть подобраны так, чтобы не произошло переполнения при вычислении выражений, стоящих в круглых скобках. После сложения или вычитания по одной из этих формул иногда приходится нормализовать результаты.

Если учитывать порядки при каждой выполняемой операции, то этот метод, по сути дела, сведется к программному введению плавающей запятой, с той только разницей, что в форме (II. 2) представляются не все величины, а лишь часть из них, для которых необходимо введение масштабов. Ясно, что при таком способе программа получится громоздкой, так как каждая операция будет реализоваться целым рядом команд.

Однако многие формулы можно преобразовать к такому виду, который позволяет запрограммировать эти формулы с помощью более простых приемов, чем программное введение плавающей запятой. Так, например, все программы для вычисления элементарных функций с помощью метода плавающих масштабов несущественно отличаются в ту или иную сторону от аналогичных программ, составленных для режима плавающей запятой, и в отношении скорости счета и в отношении объема оперативной памяти, но при этом они обеспечивают бо́льшую точность вычислений. Этот факт имеет большое значение в связи с тем, что при решении практических задач значительная доля вычислений осуществляется по стандартным подпрограммам.

К сказанному выше следует еще добавить, что определенные части программы (иногда довольно значительные), осуществляющие управление счетом и контроль полученных результатов, не вызывают усложнения при применении метода плавающих масштабов. В силу этого наличие в программе усложненных частей может и не привести к значительному увеличению количества ячеек, занятых всей программой, и замедлению счета по сравнению с теми же характеристиками программы решения данной задачи в режиме плавающей запятой. При этом также нужно учесть тот факт, что скорость выполнения операций в режиме фиксированной запятой в среднем несколько выше, чем в режиме плавающей запятой, что в известной степени компенсирует увеличение времени вычислений, вызванное тем, что при методе плавающих масштабов программа получается более громоздкой.

Рассмотрим для примера интегрирование систем дифференциальных уравнений методом Рунге — Кутта.

Всю программу, по которой осуществляется решение таких задач, можно разбить на четыре части: 1) управление счетом и контроль правильности вычислений, 2) вычисление правых частей системы, 3) осуществление очередного шага интегрирования и 4) контроль и смена масштабов после выполнения каждого шага интегрирования.

Метод Рунге — Кутта с видоизменением Гилла [4] для системы

$$\frac{dy_i}{dt} = f_i(y_1, \ldots, y_n), \quad i = 1, 2, 3, \ldots, n,$$

на одном шаге интегрирования выражается следующими формулами:

$$\left.\begin{aligned} k_{ij} &= 2^m h f_i(y_{1j}, \ldots, y_{nj}),\\ r_{i,\,j+1} &= \delta_j(k_{ij} - q_{ij}),\\ y_{i,\,j+1} &= y_{ij} + 2^{-m} r_{i,\,j+1},\\ q_{i,\,j+1} &= q_{ij} + 3\cdot 2^m(2^{-m} r_{i,\,j+1}) - \delta_j k_{ij} \end{aligned}\right\} \text{для } j = 0, 1, 2, \left.\begin{aligned} k_{i3} &= 2^m h f_i(y_{13}, \ldots, y_{n3}),\\ r_{i4} &= \delta_3(k_{i3} - 2q_{i3}),\\ y_{i4} &= y_{i3} + 2^{-m} r_{i4},\\ q_{i4} &= q_{i3} + 3\cdot 2^m(2^{-m} r_{i4}) - 3\delta_3 k_{i3}, \end{aligned}\right\} \quad \text{(II. 4)}$$

где

$$\delta_0 = \frac{1}{2}, \quad \delta_1 = 1 - \sqrt{\frac{1}{2}}, \quad \delta_2 = 1 + \sqrt{\frac{1}{2}}, \quad \delta_3 = \frac{1}{6}.$$

В этих формулах h является величиной шага интегрирования, а y_{i0} и q_{i0} равны соответственно y_{i4} и q_{i4} предыдущего шага интегрирования. В начальный момент величины y_{i0} равны соответствующим начальным данным, а $q_{i0}=0$. Множитель 2^m вводится для учета ошибок округления.

Эти формулы были преобразованы к виду, удобному для программирования методом плавающих масштабов.

Величины y_{ij} представляются в виде

$$y_{ij}=2^{p_i}Y_{ij}.$$

В таком же виде можно представить и величины r_{ij}

$$r_{ij}=2^{p_i}R_{ij};$$

тогда из формул (II. 4)

$$y_{i,j+1}=2^{p_i}Y_{i,j+1}=2^{p_i}(Y_{ij}+2^{-m}R_{i,j+1})$$

и, следовательно,

$$Y_{i,j+1}=Y_{ij}+2^{-m}R_{i,j+1}.$$

В соответствии с этим после замены

$$k_{ij}=2^{p_i}K_{ij},$$
$$q_{ij}=2^{p_i}Q_{ij}$$

и некоторых элементарных преобразований формулы (II. 4) примут вид:

$$\left.\begin{aligned}
K_{ij}&=2^{m+p_h+l_{ij}-p_i}HF_{ij},\\
R_{i,j+1}&=(K_{ij}-Q_{ij})-\sigma_j(K_{ij}-Q_{ij}),\\
Y_{i,j+1}&=Y_{ij}+2^{-m}R_{i,j+1},\\
Q_{i,j+1}&=Q_{ij}+3\cdot 2^m(2^{-m}R_{i,j+1})-K_{ij}+\sigma_jK_{ij}
\end{aligned}\right\}\text{для } j=0,1,2,$$

$$\begin{aligned}
K_{i3}&=2^{m+p_h+l_{i3}-p_i}HF_{i3},\\
R_{i4}&=(K_{i3}-2Q_{i3})-\sigma_3(K_{i3}-Q_{i3}),\\
Y_{i4}&=Y_{i3}+2^{-m}R_{i4},\\
Q_{i4}&=Q_{i3}+3\cdot 2^m(2^{-m}R_{i4})-3K_{i3}+3\sigma_3K_{i3}.
\end{aligned}$$

В этих соотношениях $h=2^{p_h}H$, $f_i(y_{1j},\ldots,y_{nj})=2^{l_{ij}}F_{ij}$, $\sigma_j=1-\delta_j$, причем

$$|\sigma_j|<1,\ |H|<1 \text{ и } |F_{ij}|<1.$$

Величины Y_{ij} после очередной смены масштабов должны удовлетворять неравенству (II. 3), например неравенству $\frac{1}{3}\leqslant|Y_{ij}|<\frac{3}{1}$ при $p_i>p_{i0}$, а при $p_i=p_{i0}$ достаточно выполнение неравенства $|Y_{ij}|<R$, т. е. $|Y_{ij}|<\frac{3}{4}$.

Величины p_h, H, p_{i0} и m подбираются из тех соображений, чтобы в вычислениях, необходимых для реализации одного шага интегрирования не могло произойти переполнения. Величины p_{i0} вводятся для того, чтобы не производилась нормализация величин y_i в том случае, когда они становятся близкими к нулю.

Указанные условия обеспечивают точность вычислений в восемь-девять верных десятичных знаков.

Величины K_{ij} подсчитываются по программе, осуществляющей вычисление правых частей. Сложность этой части программы зависит от вида правых частей системы, но, как правило, здесь в значительной степени используются стандартные подпрограммы для вычисления элементарных функций. Часть программы, осуществляющая один шаг интегрирования, незначительно отличается от аналогичной программы для режима плавающей запятой, так как в этой части программы участвуют только мантиссы величины. Осуществление же контроля и смены масштабов практически не влияет на скорость счета, так как обращение к этой части программы производится довольно редко.

Было проведено сравнение программ, составленных для решения одной системы дифференциальных уравнений в режиме плавающей запятой и в режиме фиксированной запятой с помощью метода плавающих масштабов. Правые части этой системы составляли рациональные выражения от тригонометрических функций.

Результаты сравнения следующие: при решении этой задачи в режиме фиксированной запятой количество ячеек памяти, занятых программой, было на 20% больше, чем в программе для режима плавающей запятой, а количество тактов, необходимых для ее решения, больше на 16%.

Поскольку увеличение числа тактов небольшое, то различие во времени получилось незначительным. Точность же вычислений в режиме фиксированной запятой была примерно на два десятичных знака больше, чем в режиме плавающей запятой.

§ 7. Рациональное использование оперативной памяти

Размещение в памяти машины исходных данных, промежуточных и конечных результатов, а также самой программы является одним из основных моментов программирования.

Наличие в машине М-2 двух видов оперативной памяти: «быстрой» (электронной памяти) и «медленной» (магнитного барабана) вносит целый ряд особенностей в программирование на этой машине. Эти особенности связаны с желанием распределить в памяти программу вместе с рабочими ячейками и исходными данными таким образом, чтобы обеспечивалась максимально возможная скорость счета по выбранному алгоритму решения задачи.

Это желание будет заведомо выполнено, если программу вместе с исходными данными и рабочими ячейками удастся полностью разместить в электронной памяти. Конечно, это не всегда удается сделать, но для целого ряда «критических» в этом отношении задач можно подобрать такие алгоритмы, которые позволяют составить достаточно компактную программу. С точки зрения объема вычислений эти алгоритмы, как правило, уже не будут наилучшими, но фактическая скорость решения задачи на машине зависит не только от количества выполняемых операций, но и от того, с какой скоростью они выполняются. Поэтому иногда целесообразно для решения задачи выбрать алгоритм, требующий, может быть, большего количества вычислений по сравнению с другими алгоритмами, но для которого можно составить программу с использованием только электронной памяти. Однако при выборе таких алгоритмов всегда нужно проверять, достигаем ли мы этим путем фактического ускорения счета. Некоторые алгоритмы хотя и реализуются достаточно компактной программой, требуют такого большого объема вычислений, что рациональнее проводить решение задачи по другим алгоритмам даже с использованием магнитного барабана.

При использовании магнитного барабана нужно иметь в виду тот факт, что если на нем помещена только программа, а все числа и рабочие ячейки, используемые этой программой, расположены в электронной памяти, то в этом случае обеспечивается скорость вычислений до 400 операций

в секунду*), тогда как при размещении на магнитном барабане чисел и рабочих ячеек будет выполняться в секунду от 25 до 100 операций в зависимости от числа обращений к барабану (напомним, что на каждое обращение к барабану требуется в среднем 10 *мсек*).

Поэтому при недостатке места в электронной памяти целесообразно на магнитном барабане размещать прежде всего такие части программы, по которым производится незначительная доля вычислений; при этом рабочие ячейки желательно оставлять в электронной памяти. Точно так же на барабане можно хранить таблицы значений каких-либо величин, обращение к которым при решении данной задачи производится сравнительно редко.

При решении некоторых задач может не уместиться в электронной памяти и такая часть программы, по которой проводится значительная доля вычислений. Хранение ее во время работы на магнитном барабане может существенно повлиять на скорость счета. Но если работа по этой части программы производится после сравнительно больших вычислений по другим частям программы, то в этом случае ее можно временно хранить на магнитном барабане, а для работы переносить в электронную память на место отработавших частей, которые отсылаются на магнитный барабан. По окончании работы этой части нужно произвести обратное перемещение программы. Такое перемещение осуществляется с помощью дополнительной программы, которую целесообразно помещать в электронной памяти.

*) Это достигается тем, что ячейки с последовательными адресами, как указывалось в предыдущей главе, расположены друг от друга на расстоянии $\frac{1}{8}$ окружности барабана. Такое размещение ячеек на барабане подобрано из тех соображений, чтобы за время между прохождением под считывающими головками двух команд с последовательными номерами как раз успевали произойти необходимые обращения к электронной памяти и выполниться самая продолжительная операция машины. В этом случае фактически не тратится времени на ожидание выборки очередной команды.

ГЛАВА III

СИСТЕМА СТАНДАРТНЫХ ПОДПРОГРАММ

§ 1. Понятие о стандартных программах и подпрограммах

Составление программы для решения более или менее сложной задачи при выполнении всех требований, предъявляемых к программе, является весьма трудоемким делом. Между тем, довольно часто приходится решать однотипные задачи: интегрировать различные системы обыкновенных дифференциальных уравнений, вычислять определенные интегралы, обращать матрицы различных порядков и т. д. Ясно, что программы решения различных задач одного и того же типа будут отличаться лишь в деталях, если решение ведется одним и тем же методом. Поэтому, зафиксировав определенный алгоритм решения задач данного класса, можно один раз составить такую программу, реализующую этот алгоритм, которую затем можно было бы использовать для решения любой конкретной задачи данного класса. Такие программы будем называть *стандартными программами*. Наличие готовых стандартных программ позволяет значительно экономить труд программиста при решении задач данного класса, так как их не нужно составлять заново каждый раз.

Для некоторых классов задач стандартные программы могут реализовать выбранный алгоритм решения полностью, и для их использования в конкретной задаче достаточно иметь еще небольшую обслуживающую часть программы, которая задает всю необходимую информацию для стандартной программы, если это требуется, меняет параметры самой задачи, если нужно решить несколько вариантов, обрабатывает полученные результаты и т. д. Основные же вычисления полностью выполняются стандартной программой. Однако в стандартной программе не всегда удается учесть все особенности решения конкретных задач данного класса. В этом случае составляется такая стандартная программа, которая реализует только части алгоритма, являющиеся общими для всех задач этого класса. Такую стандартную программу можно использовать при решении конкретной задачи только совместно с другими частями программы, которые дополняют стандартную программу и которые меняются от задачи к задаче, например вычисление правых частей при интегрировании систем обыкновенных дифференциальных уравнений или вычисление подынтегральной функции при численных квадратурах. Для составления этих дополнительных частей программы и подключения их к стандартной программе, а также для составления ведущей части программы требуются определенные, иногда довольно серьезные усилия программиста, которые существенно зависят от того, как сделана стандартная программа. Поэтому при составлении стандартных программ стремятся к тому, чтобы сделать их применение к конкретным задачам по возможности более простым и удобным.

Стандартные программы могут зависеть от параметров, характеризующих конкретную задачу, например, порядок матрицы при ее обращении, порядок интегрируемой системы обыкновенных дифференциальных уравнений и т. д. В этом случае специальная часть стандартной программы подготавливает ее для работы при заданных значениях параметров, так что при различных значениях параметров фактически получаются разные программы.

Стандартные программы обычно требуют задания информации, которая не изменяет программы по существу, а служит для конкретизации данной задачи (например, значение верхнего и нижнего пределов интегрирования при численных квадратурах) и связи с дополнительными частями программы, совместно с которыми работает данная стандартная программа (например информация о том, где расположена программа вычисления подынтегральной функции).

При решении сколько-нибудь сложной задачи довольно трудно составить сразу общую программу ее решения. Но после того, как для решения задачи выбран определенный численный метод, решение сводится к вычислениям, ведущимся в определенной последовательности по ряду формул или групп формул. Как правило, каждая формула или группа

формул представляет собой некоторый самостоятельный этап вычислений, поэтому и программирование естественным образом распадается на ряд отдельных этапов. Сначала для каждого отдельного этапа вычислений составляется своя программа. Составление программы для одного этапа, естественно, оказывается более простой задачей. Общая же программа получается соединением между собой отдельных составленных программ. Если какой-либо самостоятельный этап вычислений требует для своей реализации довольно громоздкой программы, то его целесообразно расчленить на более мелкие составные части.

В различных частях задачи может потребоваться реализация одной и той же функциональной зависимости, поэтому было бы нерационально составлять программу для реализации этой зависимости каждый раз, как только она встретится — это только удлинило бы программу. Целесообразно иметь эту программу только в одном месте и использовать ее по мере надобности в любой части задачи.

Таким образом, мы приходим к понятию *подпрограммы*, понимая под этим часть программы, которая реализует определенный алгоритм (некоторую функциональную зависимость) и которая может быть использована в различных местах общей программы.

Решение различных задач также может содержать общие части. Очень часто в самых различных задачах приходится, например, переводить числа из одной системы счисления в другую, вычислять тригонометрические функции и т. д. Для реализации таких часто встречающихся алгоритмов можно однажды составить программу и затем пользоваться ею как готовой подпрограммой в любой задаче, где встречаются аналогичные вычисления, независимо от того, к какому классу принадлежит данная задача. Для этого выбирается определенная система их использования, и подпрограммы, удовлетворяющие всем требованиям выбранной системы, называются *стандартными подпрограммами*.

Стандартные подпрограммы реализуют отдельные алгоритмы и используются одинаково просто в любой задаче. С точки зрения удобства использования можно отметить стандартные подпрограммы с одним аргументом. Для использования такой стандартной подпрограммы достаточно в основной программе написать одну команду обращения к ней

(может быть, необходимо иметь еще одну дополнительную команду для задания аргумента в определенной ячейке памяти), поэтому при расписывании программы с точки зрения программиста можно считать, что в машине имеется элементарная операция, эквивалентная по своему содержанию соответствующей стандартной подпрограмме. Таким образом, стандартные подпрограммы расширяют набор элементарных операций машины программным путем, что существенно облегчает программирование, так как, используя больший набор элементарных операций, можно короче и проще описать алгоритм решения задачи.

Следует также отметить, что вновь составленная программа требует известного количества машинного времени и труда программиста для ее отладки. Наличие развитой библиотеки стандартных подпрограмм позволяет резко сократить эти затраты, так как в этом случае необходимо проверить лишь правильность использования стандартных программ и подпрограмм в данной задаче и те части программы, которые составлены программистом заново.

Удобство и простота подключения стандартных подпрограмм к основной программе зависят от выбранной системы их использования. Наиболее существенными здесь являются вопросы размещения стандартных подпрограмм в памяти машины, обращение к ним и удобство ввода подпрограмм в машину. Более подробно эти вопросы будут освещены в последних параграфах настоящей главы.

§ 2. Логические схемы программ

Как уже упоминалось выше, после того как выбран численный алгоритм, решение задачи сводится к вычислениям, ведущимся в определенной последовательности по ряду формул. Однако для составления программы недостаточно выписать только ту последовательность команд, которая с помощью имеющихся в машине элементарных операций реализует счет по данным формулам. При решении задач на цифровых вычислительных машинах с программным управлением очень важным является то обстоятельство, что один и тот же участок программы в процессе вычислений может использоваться многократно, например путем введения циклов. Но для того, чтобы при каждом новом повторении цикла можно

было получать все новые и новые результаты, часто бывает необходимо определенным образом подготовить команды цикла для нового повторения. Такая подготовка может заключаться, например, в изменении вида команд, выбирающих исходные данные для счета по формулам или отсылающих полученные результаты на хранение в память машины, т. е. в переадресации команд. Для этого, кроме команд, реализующих счет по формулам, в программе необходимо иметь команды, подготавливающие новое повторение цикла. Поскольку повторение должно произойти строго определенное количество раз, то необходимо еще иметь команды, которые бы проверяли, сделано ли достаточное количество повторений, и в зависимости от этого направляли дальнейший ход вычислений.

Часто переменные команды цикла следует возвратить к исходному виду, т. е. уничтожить результаты переадресации. Это требуется, например, в случае, когда данный цикл входит в состав другого цикла. Следовательно, в программе необходимы дополнительные команды, которые восстанавливают исходный вид переменных команд.

Все указанные выше дополнительные команды не участвуют непосредственно в счете, а выполняют функции управления программой. Кроме того, в программе нужны обслуживающие команды для ввода материала в машину, обмена информацией между внешней и внутренней памятью, перевода чисел из одной системы счисления в другую, вывода результатов и т. д.

Таким образом, при составлении программы необходимо реализовать некий расширенный алгоритм, который наряду с алгоритмом решения задачи включает в себя функции обслуживания и управления программой.

Составление программ для вычислений по определенным формулам или для реализации отдельных простых алгоритмов обычно не вызывает особых затруднений. Наиболее трудной задачей при программировании является как раз обеспечение правильного управления программой, т. е. своевременное восстановление и переадресацию переменных команд, выработку логических условий, правильную условную или безусловную передачу управления и т. д.

Если составлять программу непосредственно, то наряду с четким представлением выбранного метода решения задачи программист должен помнить и всю структуру программы

в целом, чтобы обеспечить правильное управление программой. Сделать это бывает довольно трудно, особенно в сложных задачах, поэтому, прежде чем составлять программу, желательно иметь какое-то краткое, легко обозримое описание программы с помощью минимального количества обозначений, в котором была бы отражена вся структура программы и схема ее управления.

Для этой цели пользуются *логической схемой*, которая представляет собой описание расширенного алгоритма с помощью операторов. Под *оператором* здесь понимается какая-либо совокупность последовательно выполняющихся команд программы, которая выполняет в программе некоторую самостоятельную функцию с точки зрения структуры программы и которую поэтому при составлении логической схемы можно записать одним символом — условным обозначением данного оператора. Запись в логической схеме этой совокупности команд одним символом позволяет более четко и обозримо выразить структуру программы, фиксируя внимание на основных ее моментах и опуская несущественные детали.

Операторы, выполняющие в программе однородные функции (счёт по формулам, переадресация переменных команд и т. д.), называются *однотипными* и в логической схеме обозначаются одним и тем же символом, снабженным индексом для отличия одного оператора данного типа от другого. Количество различных типов операторов, вообще говоря, зависит от соглашения. Обычно различают следующие типы операторов: арифметический оператор, оператор засылки, оператор переадресации, оператор восстановления, оператор формирования и логический оператор.

Арифметический оператор реализует вычисления или некоторые логические преобразования над кодами по данной формуле или группе формул.

Оператор засылки служит для занесения какой-либо информации в определенные ячейки памяти.

Оператор переадресации служит для изменения переменных команд программы путем их переадресации, т. е. путем прибавления некоторых величин к адресам команды. В частности, по окончании какого-либо цикла вычислений оператор переадресации может служить для возвращения переменных команд в исходное состояние (обратная переадресация).

Оператор восстановления служит для восстановления исходного вида переменных команд перед повторным обращением к какому-либо циклу. В отличие от оператора переадресации здесь восстановление осуществляется путем засылки заранее запасенных констант на нужные места. Функции операторов восстановления сходны с функциями операторов засылки, но поскольку они встречаются довольно часто и играют важную роль в управлении программой, то их целесообразно выделить в самостоятельный тип.

Оператор формирования служит для формирования исходного вида переменных команд и констант сравнения, если вид этих команд и констант заранее не известен, а определяется в ходе вычислений.

Логический оператор служит для проверки выполнения некоторого логического условия и передачи управления тому или иному оператору в зависимости от результата проверки.

Для перечисленных выше типов операторов мы будем употреблять следующие условные обозначения:

A — арифметический оператор,
$З$ — оператор засылки,
F — оператор переадресации,
$О$ — оператор восстановления,
$Ф$ — оператор формирования,
p (или q) — логический оператор.

Кроме того, мы будем пользоваться операторами безусловной передачи управления и останова машины, обозначая их соответственно символами ω и Ω.

Функции некоторых частей программы могут быть таковы, что их трудно описать с помощью перечисленных выше стандартных операторов, например функции обмена информацией между оперативной и внешней памятью машины, ввода и вывода материала и т. д.

При составлении логической схемы для описания таких частей программы употребляют нестандартные операторы. Вообще говоря, можно было бы увеличить количество стандартных операторов для описания некоторых других функций программы, не прибегая к нестандартным операторам, но увеличение количества употребляемых стандартных операторов не всегда оказывается целесообразным. Обычно это количество сводится к минимуму и в случае необходимости пользуются нестандартными операторами, которые могут иметь самый различный смысл.

В сложных задачах для сокращения логической схемы целесообразно ввести так называемые обобщенные операторы. Под обобщенным оператором будем понимать часть логической схемы, выполняющей некоторую самостоятельную функцию, в которой управление извне получает только первый оператор, а выход осуществляет только последний оператор. Обобщенные операторы удобно использовать для сокращения записи логической схемы путем объединения нескольких операторов в один — обобщенный. Содержание логической схемы от этого не меняется, но она становится более краткой и наглядной.

При написании логической схемы программы символы, обозначающие операторы, пишутся в одну строку в той же последовательности, в которой работают операторы в программе. Если какой-либо оператор передает управление не соседнему стоящему справа в логической схеме оператору, то такая передача управления указывается стрелкой. При логических операторах в скобках обычно указывается, какое логическое условие они проверяют, и стрелкой указывается, какому оператору передается управление в случае невыполнения этого условия (в случае выполнения условия управление передается следующему оператору логической схемы).

Если в каком-либо операторе, входящем в цикл, имеются переменные команды, которые изменяются по определенному закону при каждом новом повторении цикла, то такой оператор называется зависящим от параметра.

Следует обратить внимание на то, что в данном случае понятие параметра тесно связано с понятием цикла: параметр служит для характеристики состояния цикла в данный момент. Для этого каждому повторению цикла ставится в соответствие целое число, являющееся значением параметра, так, чтобы совокупность значений параметра являлась монотонной последовательностью (обычно в качестве такой последовательности берется отрезок натурального ряда). Тогда каждое значение параметра однозначно определяет состояние цикла, поскольку его переменные команды изменяются по определенному закону вместе с изменением параметра.

Параметр используется в основном для управления программой, поэтому в явном виде он может и не фигурировать

Явное значение параметра, как правило, бывает нужно только для выработки логических условий на окончание цикла или для формирования некоторых кодов, зависящих от параметра каким-либо сложным образом.

Если в программе имеется несколько циклов, вложенных друг в друга, то могут быть команды, меняющиеся при повторении каждого из этих циклов. Операторы, содержащие такие команды, будут зависеть сразу от нескольких параметров.

В логической схеме при каждом операторе, зависящем от параметров, в скобках перечисляются все параметры, от которых он зависит.

При операторах восстановления, формирования и переадресации в скобках указывается, по какому параметру производится восстановление, формирование или переадресация. Это определяет и область действия такого оператора — он воздействует на все операторы логической схемы, зависящие от данного параметра (предполагается, что сами операторы восстановления, формирования и переадресации от параметров не зависят, так что путаницы в обозначениях не возникает).

Вместе с логической схемой программы должно быть дано объяснение всех входящих в нее операторов, которые недостаточно объяснены в самой логической схеме.

В качестве примера рассмотрим следующую задачу. Пусть имеется квадратное уравнение относительно x

$$x^2 + r(\alpha)\, x + s(\beta) = 0,$$

где $r(\alpha)$ и $s(\beta)$ — заданные функции.

Требуется найти все действительные корни квадратного уравнения при любых сочетаниях заданных значений

$$\alpha_i \ (i = 1, 2, \ldots, k) \quad \text{и} \quad \beta_j \ (j = 1, 2, \ldots, l).$$

Разместим заданные значения α_i в ячейках $a+1, a+2, \ldots, a+k$, значения β_j — в ячейках $b+1, b+2, \ldots, b+l$ и будем при фиксированном значении α_m пробегать все значения β_j $(j = 1, 2, \ldots, l)$. (Предполагается, что α_i и β_j введены в память машины в двоичной системе счисления.)

В данном случае в программе будут переменные команды для выборки β_j, входящие в цикл по параметру j, и переменные команды для выборки α_i, входящие в цикл по параметру i, причем первый цикл содержится во втором.

Поскольку нас интересуют только действительные корни уравнения, то в случае комплексных корней будем печатать вместо них какие-то условные числа.

Логическая схема программы будет выглядеть следующим образом:

$$\overline{{}^{\vee}A_1(i)\,\Gamma_1(i)^{\vee}A_2(j)\,F_2(j)\,K_1\overline{p_1(j>l)}!O(j)\,p_2(i>k)}!\,\Omega.$$

Арифметические операторы A_1 и A_2 вычисляют соответственно $r(\alpha_i)$ и $s(\beta_j)$ по заданным формулам и запоминают их в определенных ячейках памяти.

Обобщенный оператор K_1, используя найденные значения r и s, решает квадратное уравнение и печатает найденные значения корней x_1 и x_2, используя стандартные подпрограммы извлечения квадратного корня и перевода чисел из двоичной системы в десятичную. В случае комплексных корней печатаются условные числа.

Операторы $F_1(i)$ и $F_2(j)$ переадресуют переменные команды в операторах A_1 и A_2 по параметрам i и j соответственно.

Операторы $p_1(j>l)$ и $p_2(i>k)$ проверяют выполнение указанных в скобках логических условий и в случае их невыполнения передают управление по стрелке.

Оператор $O(j)$ восстанавливает исходный вид переменной команды выбора β_j в операторе A_2, причем исходный вид переменной команды таков, что она должна сначала выполниться, а затем переадресовываться.

Оператор Ω означает останов машины.

Обобщенный оператор K_1 с точки зрения структуры программы выполняет самостоятельную функцию, но фактически он будет выражаться через несколько других операторов. Более подробно его можно, например, представить следующим образом:

$$\mathcal{З}A_3 p(\Delta \geqslant 0)!\overline{A_4}^{\vee}A_5.$$

Оператор $\mathcal{З}$ предварительно засылает условные числа в ячейки, где помещаются корни уравнения x_1 и x_2.

Оператор A_3 вычисляет дискриминант квадратного уравнения.

Оператор A_4 находит корни уравнения x_1 и x_2 и помещает их в те ячейки, где первоначально были помещены условные числа.

Оператор A_5 осуществляет перевод найденных корней из двоичной системы в десятичную и печатает их.

Логическое условие $p(\Delta \geqslant 0)$ в случае $\Delta < 0$ обходит оператор A_4 и передает управление сразу оператору A_5, который в этом случае отпечатает условные числа (например, числа, заведомо бо́льшие, чем все возможные значения корней).

Логическая схема вместе с объяснением всех входящих в нее операторов полностью описывает расширенный алгоритм решения задачи. Однако следует отметить, что логическая схема, вообще говоря, не определяет однозначно программу, которая будет реализовать этот алгоритм, так как каждый оператор может быть расписан в терминах элементарных операций машины по-разному.

Логическая схема позволяет в значительной мере сэкономить труд программиста на первом этапе составления программы, так как, используя небольшое количество различных операторов, гораздо проще охватить всю задачу в целом и правильно построить управление программой. Логическая схема облегчает проверку правильности построения программы и позволяет легко внести необходимые исправления в схему программы, тогда как исправление ошибки в самой программе очень часто бывает весьма сложным делом. Она также позволяет быстрее и легче разобраться в ранее составленной программе (своей или чужой). Поэтому все стандартные программы и подпрограммы, приводимые ниже, будут сопровождаться их логическими схемами.

Наличие логической схемы программы значительно упрощает и ее расписывание, так как программисту уже не нужно помнить всю задачу в целом, а можно сосредоточить свое внимание на расписывании каждого оператора логической схемы в отдельности. Эту часть работы можно значительно упростить и сделать ее чисто технической, если провести четкое разграничение операторов различных типов и достаточно формализовать их расписывание. В этом случае расписывание программы для каждого типа оператора будет производиться по совершенно определенным правилам, поэтому можно составить специальную программирующую программу и расписывание логической схемы в программу поручить самой машине, задав всю необходимую информацию о каждом из входящих в нее операторов.

При расписывании логической схемы в программу труд программиста может быть значительно сэкономлен при наличии заранее составленных и проверенных стандартных программ и подпрограмм, так как в этом случае не нужно расписывать в программу те части логической схемы, для которых имеются готовые подпрограммы, а достаточно подключить их к основной программе. Логические схемы подпрограммы, разумеется, незачем каждый раз приводить полностью, а достаточно ввести их в логическую схему программы в качестве обобщенных операторов. Ясно, что наибольший эффект достигается при развитой библиотеке стандартных подпрограмм, так как в этом случае для многих задач расписывание логической схемы в программу в значительной мере сводится к ее монтажу из готовых подпрограмм.

§ 3. Размещение стандартных подпрограмм

В некоторых машинах имеется специальное запоминающее устройство для хранения на фиксированных местах наиболее часто встречающихся стандартных подпрограмм, а также используемых в них констант. В этом случае для их использования требуется лишь несколько заранее отведенных рабочих ячеек оперативной памяти.

На машинах, не имеющих такого специального запоминающего устройства, в том числе и на машине М-2, подпрограммы приходится вводить в ячейки оперативной памяти. Способы размещения подпрограмм в памяти могут быть различными. При одном из них каждой подпрограмме отводится фиксированное место памяти и подпрограммы, участвующие в решении данной задачи, предварительно вводятся с заранее приготовленных перфолент на свои места. Такой способ имеет свои удобства, так как каждая подпрограмма будет всегда стоять на одном и том же месте памяти, имея фиксированный вход. Последнее обстоятельство, безусловно, удобно при обращении к стандартным подпрограммам. Однако такой способ размещения доставляет при программировании серьезные неудобства другого рода. Основное неудобство связано с тем обстоятельством, что программист не имеет возможности распределять всю память машины по своему усмотрению. В его распоряжении остается

только часть памяти, не занятая подпрограммами, используемыми в решении данной задачи, причем эти подпрограммы могут оказаться разбросанными по запоминающему устройству, что иногда вызывает затруднения в использовании оставшейся части памяти. Кроме того, при таком способе размещения емкость оперативной памяти накладывает ограничение на общее количество стандартных подпрограмм в библиотеке, которые можно разместить в памяти без пересечений.

Более удобен способ, при котором каждая подпрограмма может быть поставлена на любое место в памяти. Такой способ позволяет программисту более свободно распоряжаться памятью и использовать ее более рационально, причем распределение памяти между подпрограммами можно производить уже после составления основной программы. Количество подпрограмм в библиотеке при таком способе размещения уже не зависит от емкости оперативной памяти, что позволяет иметь более обширную библиотеку. От емкости оперативной памяти зависит лишь количество стандартных подпрограмм, используемых при решении одной задачи. Однако при наличии в машине внешнего запоминающего устройства снимается и это ограничение, так как основная программа и стандартные подпрограммы могут храниться во внешнем запоминающем устройстве и вызываться для работы в оперативную память по частям.

Наибольшая свобода в распределении памяти получается при такой системе, когда каждая стандартная подпрограмма совершенно автономна, т. е. содержит все необходимые ей константы и для нее выделены определенные рабочие ячейки. Но при этом в ряде случаев память используется нерационально, так как в решении задачи может участвовать целый ряд стандартных подпрограмм, а рабочие ячейки каждой из них используются только во время работы данной подпрограммы, в то время как одни и те же ячейки могли бы обслуживать не одну, а несколько подпрограмм. Кроме того, в различных подпрограммах могут использоваться одни и те же константы. Если вместе с подпрограммой вводить и все необходимые ей константы, то может оказаться, что одна и та же константа будет введена несколько раз в различные ячейки памяти. Между тем достаточно было бы иметь такую константу только в одной

ячейке памяти, используя ее в случае надобности в любой стандартной подпрограмме.

Для более экономного использования памяти целесообразно выделить определенную группу рабочих ячеек, общую для всех стандартных подпрограмм, а также группу ячеек для хранения «стандартных констант», т. е. констант, необходимых для работы нескольких стандартных подпрограмм или часто встречающихся при решении различных задач. В принятой на машине М-2 системе для стандартных констант отведены ячейки *2.00—2.1f*, а для рабочих ячеек стандартных подпрограмм отведена группа ячеек *2.20—2.2a*.

Перед решением задачи все стандартные константы вводятся на свои места в памяти с заранее приготовленной перфоленты. Однако в решении какой-либо конкретной задачи обычно участвует лишь несколько стандартных подпрограмм, поэтому используются не все введенные стандартные константы, а только часть из них. Ячейки памяти, в которые были введены остальные константы, можно использовать в основной программе для других целей. Для этого нужно только знать, какие стандартные константы используются в подпрограммах, участвующих в решении данной задачи.

Выделенную группу рабочих ячеек для стандартных подпрограмм также можно использовать в основной программе, но при этом следует иметь в виду, что хранящаяся в них информация может быть затерта во время работы какой-либо стандартной подпрограммы.

Стандартные подпрограммы для вычисления функций от одной переменной величины в режиме плавающей запятой, как правило, имеют стандартные ячейки для аргумента (*2.20*) и результата (*2.21*). Это удобно при пользовании стандартными подпрограммами. Аргумент в ячейке *2.20* при этом сохраняется, так что его можно без предварительных засылок использовать при последовательном вычислении нескольких функций для одного и того же значения аргумента. В некоторых случаях выполнение этих правил привело бы к дополнительным командам пересылок в стандартных подпрограммах, введение которых практически не оправдывается, например в случае, когда какая-либо стандартная подпрограмма используется главным образом в качестве вспомогательной части для других стандартных подпрограмм. Это

обстоятельство имеет место также в некоторых подпрограммах, рассчитанных на метод плавающих масштабов, где величины задаются в двух ячейках памяти. В этих случаях, а также для подпрограмм, выдающих два результата или более, необходимая информация о размещении аргумента и результатов дается в инструкции пользования подпрограммой.

Известно несколько способов ввода стандартных подпрограмм на любое место памяти. Простейший из них состоит в том, что подпрограммы составляются и расписываются на бланках в условных адресах, например в буквах, и при решении каждой конкретной задачи расписываются заново на желаемые ячейки памяти. Этот ручной способ, применявшийся до того, как были разработаны автоматизированные способы, требует много времени и является источником ошибок при расписывании и перфорации стандартных подпрограмм.

Один из известных автоматизированных способов заключается в том, что каждая стандартная подпрограмма составляется для определенного места запоминающего устройства. Для каждой подпрограммы составляется своя обслуживающая часть, которая вводится в машину ранее самой подпрограммы и по заданной информации осуществляет ввод этой подпрограммы на нужное место памяти, перерабатывая адреса стандартной подпрограммы, зависящие от ее расположения в памяти. Обслуживающая часть нужна только в момент ввода, поэтому при решении задачи места в памяти она не занимает. Этот способ имеет несомненные преимущества перед ручным способом, но и он не совсем удобен, так как к каждой стандартной подпрограмме надо составлять свою обслуживающую часть и вводить ее вместе с подпрограммой. Кроме того, программист не только должен знать количество ячеек, занятых самой подпрограммой, но учитывать и количество ячеек, занимаемых обслуживающей частью, чтобы в момент ввода обслуживающей части предотвратить затирание уже введенной информации.

В вычислительном центре МГУ принят другой способ автоматизации ввода.

Каждая стандартная подпрограмма составляется на ячейки памяти, начиная с *3.00*, и в таком виде фиксируется на перфоленте. Ввод любой стандартной подпрограммы на заданное место памяти с соответствующей переработкой адресов подпрограммы осуществляется универсальной программой ввода, которая наряду с этим выполняет еще и некоторые другие функции.

§ 4. Обращение к стандартным подпрограммам

Стандартная подпрограмма предназначена для того, чтобы ею можно было пользоваться как готовой частью программы в любом месте задачи, где требуется реализация данного алгоритма. Поэтому надо иметь возможность обращения к стандартной подпрограмме из любого места программы с последующим возвратом к ее очередной команде. На некоторых машинах такая возможность предусмотрена конструктивно, например введением двух видов управления или специальных команд передачи управления. На других машинах — в том числе и на М-2 — эту возможность приходится реализовать программным путем.

Известны различные способы включения подпрограмм программным путем. Самый простой способ заключается в том, что перед обращением к стандартной подпрограмме в ее выход засылается команда безусловной передачи управления нужной команде основной программы. При этом способе необходимо хранить вид этой команды возврата как константу и затратить команду для ее засылки. Поскольку обращение к одной и той же стандартной подпрограмме может производиться из различных мест основной программы, то для каждого нового обращения нужно иметь свою константу для команды возврата и написать команду засылки этой константы в выход подпрограммы. Основное неудобство здесь заключается в том, что необходимо знать не только номер ячейки начала подпрограммы для обращения к ней, но и номер ячейки, куда следует заслать команду возврата. Это неудобство сказывается меньше, если стандартные подпрограммы стоят на фиксированных местах, поскольку в этом случае вход и выход всех подпрограмм фиксирован и для них можно составить постоянную таблицу, которой можно пользоваться при программировании. Если же каждая стандартная подпрограмма может быть поставлена на любое место памяти, то при решении каждой новой задачи нужно заново вычислять номер ячейки, где находится выход из подпрограммы, что практически очень неудобно, так как эти

вычисления требуют дополнительной затраты труда программиста и являются источником ошибок.

В принятой на машине М-2 системе для обращения к стандартным подпрограммам применяется так называемая обратная связь, которая заключается в следующем. Перед командой непосредственного обращения к подпрограмме выполняется вспомогательная команда, с помощью которой в определенной ячейке памяти, называемой ячейкой обратной связи, запоминается адрес команды обращения к подпрограмме. При выполнении каждой стандартной подпрограммы по информации, хранящейся в ячейке обратной связи, формируется команда возврата в основную программу. Эта команда выполняется по окончании работы подпрограммы и реализует возврат в основную программу.

Особенности некоторых операций, имеющихся на машине М-2, позволяют реализовать обратную связь весьма экономным способом. Так, например, совмещение в одной операции (код операции *4*) функций переноса кода из одной ячейки в другую и передачи управления позволяет обойтись без вспомогательной команды для занесения информации в ячейку обратной связи и реализовать обращение к стандартной подпрограмме одной командой основной программы

$$(n):\quad m\quad \beta\quad n\quad 4,$$

где n — адрес команды обращения в основной программе, m — начало стандартной подпрограммы. Команда (n) переносит изображающий ее код в ячейку обратной связи β (тем самым в β фиксируется, в частности, адрес команды, осуществляющей уход из основной программы) и передает управление ячейке m, т. е стандартной подпрограмме.

Если после выполнения стандартной подпрограммы передать управление команде с адресом $\alpha+1$, а содержимое ячеек $\alpha+1$ и $\alpha+2$ будет

$$(\alpha+1):\quad \alpha+3\quad \beta\quad c_1\quad f,$$

$$(\alpha+2):\quad \alpha+3\quad \alpha+3\quad c_2\quad b,$$

где (c_1): *0.00 0.00 3.ff 0* — выделитель первого адреса, (c_2): *0.00 0.00 0.01 0* — единица первого адреса, то в результате будет реализован возврат к команде основной программы с адресом $n+1$.

Действительно, команда $\alpha+1$ (логическое умножение) выделяет первый адрес кода, хранящегося в ячейке обратной связи β, и результат выделения

0.00 0.00 n 0

помещает в ячейку $\alpha+3$. Команда $\alpha+2$ к полученному результату добавляет единицу первого адреса, помещая результат

0.00 0.00 n+1 0

в ячейку $\alpha+3$, и передает управление ячейке $\alpha+3$, где уже сформирована команда безусловной передачи управления команде $n+1$.

Особенности кода, изображающего команду переключения и передачи управления, позволяют и здесь сформировать команду возврата с помощью всего лишь двух команд.

Ясно, что после выполнения любой стандартной подпрограммы должны быть выполнены одни и те же три команды $\alpha+1$, $\alpha+2$ и $\alpha+3$, поэтому они вынесены в стандартные константы (ячейки *2.10*, *2.11*, *2.12*) и образуют так называемый стандартный выход. В каждой стандартной подпрограмме имеется команда безусловной передачи управления стандартному выходу (ячейке *2.10*), который и реализует возвращение в основную программу. Поскольку команда возврата в ячейке *2.12* формируется заново при каждом обращении к стандартному выходу, то ячейка *2.12* используется и в качестве ячейки обратной связи β.

Стандартным выходом можно пользоваться и для более сложного разветвления процесса вычислений.

Пусть, например, из основной программы нужно обратиться к подпрограмме A, в которой в свою очередь содержится обращение к подпрограмме B, и после выполнения подпрограммы A надо вернуться в основную программу. Такое многоступенчатое обращение можно осуществить с помощью стандартного выхода следующим образом.

К подпрограмме A обращаемся командой n основной программы, имеющей вид

$$(n):\quad a\quad \beta\quad n\quad 4,$$

где a — адрес первой команды подпрограммы A. Ячейка обратной связи β, в которой запоминается код команды

обращения n, должна быть отлична от *2.12*, и ее содержимое должно сохраняться до момента возврата в основную программу.

Обращение к подпрограмме B осуществляется командой m подпрограммы A с ячейкой обратной связи *2.12*:

$$(m):\ b\quad 2.12\quad m\quad 4,$$

где b — адрес первой команды подпрограммы B.

Если по окончании работы подпрограммы B управление передать стандартному выходу, то он осуществит возврат к команде $m+1$ подпрограммы A.

Для возвращения в основную программу нужно по окончании работы подпрограммы A переслать содержимое промежуточной ячейки обратной связи 3 в ячейку *2.12* и передать управление стандартному выходу, который и реализует возврат к команде $n+1$ основной программы.

Такой прием используется, в частности, если в какой-либо стандартной подпрограмме имеется обращение к другой стандартной подпрограмме. В таких случаях необходимо знать, какая ячейка используется в качестве промежуточной ячейки обратной связи.

При данном способе обращения к стандартным подпрограммам остается то неудобство, что для написания команды обращения к стандартной подпрограмме надо знать истинный адрес первой команды подпрограммы. Поэтому если распределение памяти между подпрограммами производится после составления основной программы, то при ее составлении команды обращения к подпрограммам нельзя сразу написать в окончательном виде, так как в этот момент еще неизвестно, где будет расположена та или иная стандартная подпрограмма. Вследствие этого в командах обращения вместо истинных адресов первых команд стандартных подпрограмм приходится временно ставить условные обозначения или оставлять их пустыми, а конкретные адреса писать лишь после того, как будет проведено распределение памяти и каждой подпрограмме будет отведено определенное место. Это особенно неудобно, если внутри одной стандартной подпрограммы имеется обращение к другой стандартной подпрограмме. В этом случае при обращении к первой подпрограмме требуется задать дополнительную информацию о том, где расположена вторая, что, естественно, делает

использование таких сложных стандартных подпрограмм менее удобным.

Это неудобство, однако, можно обойти и автоматизировать обращение к стандартным подпрограммам следующим образом. Каждой стандартной подпрограмме присваивается определенный номер, и при составлении программы в командах обращения к стандартным подпрограммам пишется этот номер вместо конкретного адреса входа в подпрограмму. После составления программы и распределения памяти машине задается в виде таблицы информация о размещении стандартных подпрограмм в памяти. Вслед за этим специальная обслуживающая программа, используя эту информацию, просматривает всю программу и заменяет номера стандартных подпрограмм соответствующими истинными адресами.

В приводимой системе ввиду отсутствия обслуживающей программы наиболее трудным моментом является обеспечение взаимосвязи двух подпрограмм, когда одна из них используется внутри другой. Если бы обращение из одной подпрограммы к другой осуществлялось непосредственно, то команда обращения была бы переменной и зависела от расположения в памяти второй подпрограммы. Засылать нужный вид этой команды было бы неудобно, так как каждый раз пришлось бы вычислять номер ячейки, где она находится при данном расположении подпрограмм. С целью устранения этой трудности в стандартной подпрограмме вместо переменной команды обращения к другой подпрограмме (если такое имеется) пишется постоянная команда обращения к некоторой фиксированной ячейке памяти, в которой должна быть помещена команда обращения к другой подпрограмме. Введение такой промежуточной ячейки связи между подпрограммами позволяет иметь переменную команду обращения всегда в одной и той же ячейке, что упрощает ее формирование

Иногда эта промежуточная команда, кроме обращения к другой стандартной подпрограмме, совмещает в себе еще какую-либо функцию, поэтому в инструкции должно быть указано, в каком виде следует задавать эту команду. Если промежуточная ячейка связи не используется в программе для других целей, то достаточно задать ее содержимое только один раз.

§ 5. Система ввода

Для рационального использования машинного времени и более удобного обслуживания машины большое значение имеет хорошо разработанная система ввода. Необходимость такой системы вызывается следующими обстоятельствами:

а) Как при отладке, так и при решении задачи надо быть уверенным в том, что ввод материала в машину произведен правильно, поэтому необходимо проконтролировать правильность ввода.

б) В соответствии с выбранной системой стандартных подпрограмм надо иметь возможность поставить каждую стандартную подпрограмму на любое место памяти. Желательно иметь единую программу, которая бы осуществляла ввод любой стандартной подпрограммы на заданное место с одновременной переработкой адресов, зависящих от расположения подпрограммы в запоминающем устройстве.

в) Как правило, вводимый материал неизбежно распадается на отдельные части, особенно при использовании стандартных программ и подпрограмм, которые вводятся в различные места памяти. Поэтому возникает необходимость обеспечения наиболее полной автоматизации ввода различных частей материала.

Конструктивные особенности конкретной машины могут предъявить дополнительные требования к указанным выше общим требованиям. На машине М-2, например, ввод с перфоленты при помощи фотоввода может быть осуществлен только в быструю (электронную) память. Поэтому к системе ввода из-за этого добавляется еще одна функция: она должна обеспечить ввод материала сначала в электронную память, а затем при надобности — автоматический перенос его на указанное место магнитного барабана.

В вычислительном центре МГУ разработана универсальная стандартная подпрограмма «Быстрый ввод», которая решает весь комплекс задач, связанных как с общими требованиями системы, так и с конструктивными особенностями машины М-2.

Подпрограмма «Быстрый ввод», может работать в двух режимах. Первый режим используется для ввода материала на заданное место памяти и контроля правильности ввода, когда вводимый материал является или основной программой, расписанной в истинных адресах, или числовым материалом, т. е. когда не требуется никакой переработки адресов. Второй режим используется для ввода на любое место памяти программ, расписанных на последовательные ячейки памяти, начиная с некоторого зафиксированного номера, с соответствующей переработкой адресов, зависящих от расположения программы в памяти, и контроля правильности ввода.

Контроль правильности ввода осуществляется при помощи контрольного суммирования вводимого материала и сравнения полученной контрольной суммы с известной, которая вводится вместе с материалом. Контрольное суммирование на различных машинах можно осуществить по-разному, в зависимости от имеющихся элементарных операций.

Возможность переработки адресов программы, зависящих от ее расположения в памяти, достигается следующим путем. Если заранее не известно, на каком месте памяти будет работать программа, то она предварительно расписывается в условных адресах. При вводе программы на какое-либо конкретное место памяти эти условные адреса перерабатываются по определенным правилам. В настоящей системе для стандартных рабочих ячеек и стандартных констант отведены ячейки памяти, начиная с *2.00*, и условные адреса для них совпадают с истинными адресами. Исходный же вид стандартных подпрограмм имеет адреса команд, начиная с *3.00*. Таким образом, адреса, зависящие от места подпрограммы в памяти, оказываются «мечеными» — все они больше или равны *3.00*. Правило переработки условных адресов получается очень простое: если программа вводится в ячейки памяти, начиная с номера $a_н$, то к каждому «меченому» адресу надо прибавить величину $a_н - 3.00$.

Но вместе со стандартной подпрограммой иногда приходится вводить и некоторые константы, которые используются только в этой подпрограмме и которые поэтому нецелесообразно включать в число стандартных. Эти константы, естественно, не зависят от того, в каком месте памяти стоит подпрограмма, но они могут записываться в двоичной системе таким образом, что при трактовке кодов, изображающих эти числа, как команд, могут оказаться «меченые» адреса. Для того чтобы такие константы не

перерабатывались при вводе по общим правилам, необходимо задать специальную информацию, по которой стандартная подпрограмма «Быстрый ввод» могла бы отличить их от перерабатываемых команд. Чтобы информация была короче, удобнее все такие константы поместить подряд, в ячейках памяти с последовательными номерами. Более удобно расположить их в конце подпрограммы, так как в противном случае относительный номер команды входа в подпрограмму зависел бы от количества имеющихся в ней констант, что было бы неудобно при обращении к стандартным подпрограммам.

Для ввода в машину какой-либо части материала с помощью стандартной подпрограммы «Быстрый ввод» необходимо задать следующую информацию: номер ячейки памяти, начиная с которой нужно поместить вводимый материал; общее количество вводимого материала (по числу занимаемых ячеек памяти); количество констант во вводимом материале, которые не должны перерабатываться при вводе, и контрольная сумма. Вся эта, а также дополнительная информация, связанная с особенностями конкретной машины, перфорируется стандартным образом отдельно от вводимого материала в виде так называемой «шапки», которая помещается перед вводимой частью материала. «Быстрый ввод» сначала вводит шапку и по содержащейся в ней информации осуществляет ввод основного материала. Естественно, что перед каждой отдельно вводимой частью материала должна быть своя шапка.

Для того чтобы осуществить автоматический ввод различных частей материала, перед вводом каждой из них надо задать «Быстрому вводу» информацию о том, что делать после ввода данной части материала. Для этого в шапке задается еще адрес команды, которой надо передать управление по окончании ввода. Таким образом, весь процесс ввода вместе с его контролем оказывается автоматизированным.

Стандартная подпрограмма «Быстрый ввод» вводится в машину с помощью небольшой программы «Первоначальный ввод — Б», команды которой перфорируются вместе с их адресами и вводятся в память не программным путем, а включением специального тумблера. Функция первоначального ввода заключается в том, что он вводит стандартную

подпрограмму «Быстрый ввод» в ячейки памяти, начиная с *3.00*, и передает ему управление. Весь дальнейший процесс ввода осуществляется уже «Быстрым вводом».

Заметим, что стандартная подпрограмма «Быстрый ввод», как и всякая стандартная подпрограмма, может быть поставлена на любое место памяти. Если его нужно поместить не с *3.00*, а в другом месте памяти, то после первого экземпляра подпрограммы «Быстрый ввод» в самом начале вводимого материала, следует поместить его второй экземпляр, который введется по общим правилам на место, указанное в его шапке.

Наряду с подпрограммами «Первоначальный ввод — Б» и «Быстрый ввод» имеются аналогичные подпрограммы «Первоначальный ввод — М» и «Медленный ввод», предназначенные для обслуживания медленного (электромеханического) ввода. «Медленный ввод» выполняет те же функции, что и «Быстрый ввод». Подготовка материала к вводу одинакова для обеих подпрограмм ввода.

Все стандартные подпрограммы хранятся в виде готовых перфолент, на которых отперфорирована также и часть шапки с информацией о количестве вводимого материала и контрольная сумма. При подготовке к решению задачи все части программы, а также стандартные подпрограммы должны быть отперфорированы на одной ленте. Составленные заново части программ, естественно, должны быть отперфорированы вручную, а для нанесения стандартных подпрограмм на общую ленту надо вручную отперфорировать только недостающую часть шапки, которая в каждом конкретном случае своя. Остальная часть шапки и сама подпрограмма просто дублируется с готовой перфоленты на общую.

ЧАСТЬ II

В этой части приведено большинство стандартных подпрограмм и программ, которые использовались при решении задач на машине М-2 вычислительным центром МГУ. Библиотека стандартных программ включает ряд обслуживающих программ, подпрограммы для вычисления большинства элементарных функций и стандартные программы, реализующие некоторые численные методы.

Напомним кратко основные характеристики выбранной системы стандартных подпрограмм.

Для всех стандартных подпрограмм библиотеки зафиксирована группа стандартных рабочих ячеек *2.20 — 2.2a*.

Наиболее часто встречающиеся константы выделены в массив стандартных констант, для которых зафиксирована группа ячеек памяти *2.00—2.1f*.

Стандартные подпрограммы для вычисления функций от одной переменной величины в режиме плавающей запятой, как правило, имеют стандартные ячейки для аргумента (*2.20*) и результата (*2.21*). Аргумент в ячейке *2.20* сохраняется. Там, где эти правила не выполнены, в краткой характеристике программы делается на этот счет оговорка.

Каждая стандартная подпрограмма составляется на ячейки памяти, начиная с *3.00*, и в таком виде фиксируется на перфоленте. Константы, вводимые вместе с подпрограммой, располагаются в последних ее строках. С помощью программы ввода каждая стандартная подпрограмма может быть поставлена на любое место памяти по информации, заданной программе ввода.

Информация для программы ввода задается в виде шапки, состоящей из трех кодов. В первом коде шапки задается информация о том, куда нужно ввести данную стандартную подпрограмму при решении конкретной задачи. Во втором коде содержится информация об общем количестве строк

в подпрограмме и количестве строк, подлежащих переработке при вводе. Если подпрограмма вводится в ячейки памяти, начиная с $a_н$, то переработка заключается в том, что все адреса, имеющие вид *3.00* $+k$, заменяются на $a_н + k$. Третий код шапки содержит контрольную сумму, которая используется для контроля правильности ввода.

Обращение к стандартным подпрограммам производится с обратной связью. Команда обращения имеет вид

$$(n):\quad m\quad 2.12\quad n\quad 4,$$

где n — адрес команды обращения в основной программе,

m — начало стандартной подпрограммы.

После выполнения стандартной подпрограммы управление передается стандартному выходу — ячейки *2.10 — 2.12*. Команды *2.10* и *2.11* формируют в ячейке *2.12* команду

$$0.00\quad 0.00\quad n+1\quad 0,$$

которая и реализует возврат к следующей команде основной программы. Если из какой-либо стандартной подпрограммы есть обращение к другой стандартной подпрограмме, то при обращении к первой из них для обратной связи используется ячейка *2.2a*, а не *2.12*. В таких случаях ячейка обратной связи указывается в краткой характеристике программы.

Все приводимые стандартные подпрограммы выполнены в соответствии с этими требованиями системы.

Описание подпрограмм построено по общему плану: излагается используемый алгоритм, приводится логическая схема программы, краткая характеристика программы и сама программа.

Все программы снабжены достаточно полными комментариями. Некоторые программы, представляющие интерес с точки зрения использования различных приемов программирования или особенностей операций, имеющихся на машине М-2, а также наиболее сложные подпрограммы сопровождаются подробным объяснением их работы.

Группы команд программы, соответствующие одному оператору, выделяются горизонтальными линиями.

Переменные команды и переменные адреса отмечаются звездочкой.

Вместе с программой приводятся и последние два кода шапки, которые являются постоянными для данной программы.

При написании логических схем программ используется символика, принятая в § 2 главы III. Дополнительно введем оператор обращения к стандартному выходу θ и оператор ρ — изменение режима работы машины. Для простых арифметических операторов иногда используется запись в виде формул, заключенных в квадратные скобки.

При описании каждой программы в пункте «Краткая характеристика программы» даются все сведения (по возможности без отсылок к остальному тексту), необходимые для ее использования при решении задач.

В количество тактов, приводимое в краткой характеристике, включены такты, требующиеся для работы стандартного выхода.

При обращении к любой из подпрограмм управление нужно передавать первой команде подпрограммы, если на этот счет нет специальных оговорок.

ГЛАВА IV

ОБСЛУЖИВАНИЕ ВВОДА

§ 1. Стандартная подпрограмма «Быстрый ввод»

Назначение подпрограммы «Быстрый ввод»

Стандартная подпрограмма «Быстрый ввод» предназначена для следующих целей:

а) Для ввода материала с перфоленты в оперативную память машины при помощи фотоэлектрического входного устройства в случае, когда коды отперфорированы на ленте без указания их адресов.

б) Для реализации ввода материала на магнитный барабан, поскольку при помощи фотоэлектрического входного устройства непосредственный ввод может быть осуществлен только в электронную память.

в) Для контроля правильности ввода материала.

г) Для ввода программ, расписанных в условных адресах, на любое заданное место памяти с соответствующей переработкой адресов программы, зависящих от ее расположения в памяти.

д) Для обеспечения автоматизации ввода отдельных частей материала, находящегося на перфоленте.

Подпрограмма занимает *0.38* (56) ячеек памяти и первоначально составлена для ячеек *3.00 — 3.37*. При работе подпрограммы используются рабочие ячейки *2.20 — 2.2а*.

Подготовка материала к вводу

Как указывалось, все стандартные подпрограммы расписаны на ячейки памяти, начиная с *3.00*, а стандартные рабочие ячейки и стандартные константы имеют меньшие адреса,

поэтому при вводе должны перерабатываться все адреса, большие или равные *3.00*, за исключением кодов, изображающих константы, которые вводятся вместе с подпрограммой и которые расположены в конце ее.

Стандартная подпрограмма «Быстрый ввод» может ввести материал только в группу ячеек памяти с последовательными адресами.

Вся необходимая информация о вводимом материале задается в шапке, которая должна быть отперфорирована перед данной частью материала. Шапка состоит из трех кодов:

$$\begin{array}{ll} n \quad k \quad\quad l \quad 0 & \text{— первый код,} \\ m_1 \; m-1 \; 0\,0\,0 \quad 0 & \text{— второй код,} \\ \Sigma_1 & \text{— третий код.} \end{array}$$

Первый код шапки содержит следующую информацию. Число n, указанное в третьем адресе, означает адрес ячейки, начиная с которой нужно поместить вводимый материал. Во втором адресе указывается число k, означающее адрес ячейки, начиная с которой материал предварительно вводится в электронную память.

В первом адресе число l означает адрес команды, которой надо передать управление в случае правильного ввода данной части материала. В частности, l может быть адресом команды подпрограммы «Быстрый ввод», если вслед за вводом данной части материала нужно ввести следующую часть, расположенную на той же перфоленте вслед за первой. Число l может быть адресом команды, с которой начинает работу основная программа, если после ввода данной части материала нужно автоматически перейти к решению задачи. Если после ввода материала желательно остановить машину, то l должно быть адресом команды «останов», которая обычно имеется в основной программе.

Примечание 1. Необходимость задания двух чисел n и k вызвана тем обстоятельством, что с помощью фотоэлектрического входного устройства можно осуществить ввод только в электронную память. Если вводимый материал требуется поместить на магнитный барабан, то его необходимо предварительно ввести в электронную память, а затем с помощью программы перенести на нужное место магнитного барабана. Если вводимый материал нужно поместить в электронную память, то можно положить $k = n$.

Примечание 2. Если имеется пересечение между группами ячеек, куда материал вводится предварительно и куда он ставится окончательно, то «Быстрый ввод» будет работать правильно только в случае, если $n \leqslant k$.

Второй код шапки содержит информацию о количестве вводимого материала и необходимом режиме работы подпрограммы «Быстрый ввод». Здесь число m означает количество вводимого материала (по числу ячеек), а m_1 — количество первых кодов вводимого материала, которые необходимо переработать при вводе. Подпрограмма «Быстрый ввод» всегда начинает работу во втором режиме — ввод с переработкой материала. После переработки первых m_1 кодов производится переключение на первый режим и остальные коды вводятся без переработки. Этим достигается то, что константы, вводимые вместе со стандартной подпрограммой и расположенные в конце ее, не подвергаются изменению в процессе ввода. В частности, если весь материал следует вводить без переработки, то надо положить $m_1 = 0$.

Третий код шапки означает контрольную сумму Σ_1 вводимого материала, которая определяется в ходе отладки.

При перфорации шапки на ленту между первым и вторым ее кодами, а также между последним кодом шапки и началом материала необходимо оставлять промежутки на ленте в 10—15 строк.

Для решения задачи весь вводимый материал (программа или части программы, используемые в решении задачи стандартные подпрограммы и числовой материал) должен быть смонтирован на одной перфоленте. Монтаж всех частей материала выглядит следующим образом. В начале ленты должна находиться подпрограмма «Быстрый ввод» с шапкой, первый код которой в этом случае всегда должен быть *0.00 0.00 3.00 0* (остальные два кода имеют стандартный вид), затем на ленте располагаются остальные части материала, причем перед каждой из них должна быть отперфорирована своя шапка, имеющая стандартный вид. Порядок расположения различных частей материала на перфоленте (а следовательно, и порядок ввода) определяется программистом.

Процесс ввода начинается с ввода в машину вспомогательной программы «Первоначальный ввод — Б». Эта программа перфорируется на отдельной ленте вместе с адресами при кодах и вводится с пульта управления по адресам

2.30—2.41. Затем на фотоэлектрическое входное устройство ставится лента, на которой отперфорированы все части вводимого материала, и управление передается программе «Первоначальный ввод — Б» (команде с адресом *2.30*), который вводит подпрограмму «Быстрый ввод» с общей перфоленты в ячейки памяти, начиная с *3.00*, и передает ей управление. «Быстрый ввод» проконтролирует ввод самого себя и передаст управление ячейке с адресом $l = 3.00$, т. е. снова «Быстрому вводу», который и осуществляет весь дальнейший процесс ввода в соответствии с информацией, содержащейся в шапках, расположенных перед каждой отдельной частью вводимого материала.

Если подпрограмму «Быстрый ввод» нужно поставить в какое-либо другое место памяти, то, как уже упоминалось выше, кроме первого его экземпляра, на общей ленте надо иметь второй экземпляр, шапка которого составляется уже по общим правилам.

Логическая схема

$$A_0 \rho \Phi_1 A_1 \Phi_2 A_2(k) \rightarrow$$

$$\rightarrow \omega_1 \overset{\vee}{\underset{\rightarrow}{F(\ )}} A_3(k, i)\, p_1(m_1 \geqslant 1)\, \overline{A_4} \vee A_5(n, i)\, p_2(i \geqslant m - 1) \vee$$

$$\rightarrow p_3(\Sigma_1 = \Sigma_2) \overline{A_6 \omega_2} \overset{\vee}{\underset{\rightarrow}{\Omega}} 2.2a.$$

Подпрограмма начинает работать с оператора A_0, который вводит первый код шапки *n k l 0* в ячейку *2.2a*. Затем оператор ρ устанавливает режим фиксированной запятой с двойной точностью и передает управление оператору Φ_1, который формирует в стандартных рабочих ячейках электронной памяти оператор A_1, имеющий вид:

2 20 2.21 2.20 2 01 2
2.21 3.04 2.29 2 01 2,

и передает ему управление. Оператор A_1 вводит второй и третий коды шапки, после чего оператор Φ_2 обрабатывает всю введенную информацию, формируя операторы $A_2(k)$, $A_3(k, 0)$ и $A_5(n, 0)$, в которых имеются переменные команды и константы. Кроме того, в ячейке *2.2a* формируется команда безусловной передачи управления по адресу l. Оператор $A_2(k)$,

представляющий собой цикл из трех команд, получив управление от оператора Φ_2, осуществляет предварительный ввод очередных m кодов с перфоленты в ячейки электронной памяти с последовательными номерами, начиная с номера k.

Примечание 1. При пользовании фотоэлектрическим входным устройством перфолента движется настолько быстро, что если коды отперфорированы на ленте без промежутков между ними, то невозможно осуществить ввод по одному коду с последующим остановом ленты, так как после сигнала останова ленты она еще некоторое время продолжает двигаться по инерции и часть следующего кода может пройти под считывающим устройством. Между тем, при непрерывном движении перфоленты в промежуток времени между вводом двух соседних кодов машина вполне успевает выполнить по крайней мере три команды, стоящие в электронной памяти, если в них нет обращения к магнитному барабану. Это обстоятельство используется в данной программе: операторы A_1 и A_2 всегда формируются в ячейках электронной памяти независимо от того, в какой части памяти — электронной или на магнитном барабане — расположена сама программа «Быстрый ввод». Это позволяет всегда осуществлять ввод двух последних кодов шапки (оператор A_1), а также основного материала (оператор A_2) при непрерывном движении перфоленты.

По изложенной только что причине инерционного движения ленты необходимо иметь промежутки на ленте между первым и вторым кодами шапки, а также между шапкой и основным материалом: после ввода первого кода шапки выполняются три команды программы (операторы ρ и Φ_1), которые могут находиться на магнитном барабане, а после ввода последних двух кодов шапки работает оператор Φ_2, состоящий из целой серии команд, поэтому в обоих случаях после ввода лента останавливается.

Примечание 2. Операторы $A_2(k)$ и ω_1 формируются оператором Φ_2 в следующем виде:

2.24 2.25 2.25 2.23 a
3.12 → 2.25 2.26 k* 2.01 2*
2.26 2.24 2.25 2.21 7
3.15 ← 2.27 3.15 2.27 2.22 4.

От оператора Φ_2 управление передается команде *2.25* оператора A_2, которая вводит код $R_i (i = 0, 1, \ldots, m-1)$ в ячейку $k+i$. В ячейке *2.21* оператором Φ_2 (команда *3.02*) сформирована константа сравнения в виде

(*2.21*): *2.26* $k+m-1$ *2.01* *2.*

Пока не будет введен последний код, команда *2.26* передает управление команде *2.24*, переадресующей команду *2.25*. По окончании ввода оператор ω_1 (команда *2.27*) очищает ячейку *2.27* (в ячейку *2.22* был занесен нуль командой *3.08*) и передает управление оператору A_3.

После того как осуществлен предварительный ввод материала, начинает работать цикл по i, обрабатывающий поочередно каждый введенный код R_i, где параметр i означает порядковый номер введенных кодов.

Оператор ω_1, работающий после оператора A_2 и находящийся еще вне цикла, передает управление оператору A_3, минуя оператор $F(\ )$, так как исходный вид переменных команд, зависящих от i, сформирован в таком виде, что они должны сначала выполняться, а затем переадресовываться.

Оператор $A_3(k, i)$ осуществляет контрольное суммирование введенного материала. Контрольное суммирование производится следующим образом: каждый очередной код расщепляется на две части, затем обе части сложением в режиме фиксированной запятой добавляются к накапливаемой контрольной сумме Σ_2, причем часть кода со старшими разрядами предварительно сдвигается в сторону младших разрядов, так что, хотя суммирование и ведется в режиме фиксированной запятой, переполнения не произойдет. Ячейка *2.27*, в которой получается контрольная сумма Σ_2, предварительно очищается оператором ω_1.

По окончании работы оператора A_3 оператор p_1 проверяет содержимое счетчика (ячейки *2.20*, в которую оператор A_1 заносит код, содержащий m_1 единиц в третьем адресе, где число m_1 есть количество кодов, подлежащих переработке при вводе), и если содержимое счетчика равно нулю, то управление передается оператору A_5, минуя оператор A_4. Оператор A_5 переносит очередной обрабатываемый код в ту ячейку, где он должен находиться окончательно. Если содержимое счетчика еще отлично от нуля, то оператор p_1 передает управление оператору A_4, который перерабатывает «меченые»

адреса данного кода (если в нем таковые имеются) и передает управление оператору A_5, который переносит этот переработанный код на свое место в памяти. Наряду с переработкой кода оператор A_4 вычитает единицу из третьего адреса счетчика. Таким образом, оператор p_1 передает управление непосредственно оператору A_4 ровно m_1 раз, пока содержимое счетчика не будет равно нулю. Получение нуля в счетчике означает, что все коды, подлежащие переработке, уже переработаны, и после этого управление от оператора p_1 всегда будет передаваться оператору A_5, минуя оператор A_4, что и означает переключение «Быстрого ввода» на первый режим работы — ввод без переработки материала.

Следует обратить внимание, что при использовании счетчика в данной части программы фактически использован прием, аналогичный использованию переменной команды цикла в качестве счетчика (см. гл. II § 3). Оператор A_1 на самом деле в ячейку *2.20* заносит не код, содержащий m_1 единиц третьего адреса, а целиком второй код шапки, т. е.

m_1 $m-1$ *0.00* *0.*

Так как при изменении счетчика меняется его третий адрес, то наличие во втором адресе числа $m-1$ не оказывает влияния на его работу. Этот прием позволяет сэкономить одну команду, которая была бы нужна для гашения второго адреса.

После того как очередной код перенесен на свое место оператором A_5, оператор p_2 проверяет, все ли m кодов введенного материала уже поставлены на свое место. Если нет, то управление передается на повторение цикла по i, т. е. оператору $F(i)$, который переадресует операторы $A_3(k, i)$ и $A_5(n, i)$ по параметру i и передает управление оператору A_3. Когда в цикле по i будет обработан последний код введенного материала, оператор p_2 передает управление оператору p_3. К этому моменту весь материал в окончательном виде размещен на своем месте и в ячейке *2.27* получена контрольная сумма Σ_2.

Оператор p_3 сравнивает введенную контрольную сумму Σ_1 и вновь полученную Σ_2. Несовпадение контрольных сумм означает, что либо материал с перфоленты был неправильно введен в машину, либо в шапке была отперфорирована неправильная контрольная сумма Σ_1. В обоих случаях оператор

p_3 передает управление оператору Ω — останов машины с вызовом на пульт управления обеих контрольных сумм *). При совпадении контрольных сумм управление передается оператору A_6, который выводит на печать код

n 0.00 l 0,

сформированный оператором Φ_2 в ячейке *2.2a*, и передает управление оператору ω_2. Оператор ω_2 передает управление команде, стоящей в ячейке *2.2a*, которая в свою очередь означает безусловную передачу управления команде с адресом *l*, указанным в шапке. В программе функции операторов A_6 и ω_2 совмещены в одной команде.

Команда безусловной передачи управления в ячейке *2.2a* сформирована в указанном выше виде для того, чтобы при одной печати вывести оба числа *n* и *l* (для контроля их набивки на перфоленте). Это можно сделать, так как в операции переключения содержимое третьего адреса команды роли не играет.

Программа

	0.36	*0.37*	*0.00*	*0*		
	0.00	*0.63*	*0.c9*	*2*		
3.00	*3.01*	*2.2a*	*2.01*	*2*	*n k l 0⇒2.2a*	A_0
3.01	*0.00*	*0.0c*	*3.02*	*0*	ФД	ρ
3.02	*3.03*	*2.20*	*3.30*	*4*	Формирование A_1	Φ_1
3.03	*2.20*	*2.21*	*3.31*	*4*		
3.04	*2.22*	*2.2a*	*3.37*	*f*	$(k \cdot 2^{-20})_{\Phi} \Rightarrow 2.22$	Φ_2
3.05	*2.2a*	*2.2a*	*2.22*	*a*	*n 0.00 l 0⇒2.2a*	
3.06	*2.25*	*3.33*	*2.22*	*b*	Формирование $A_2(k)$	
3.07	*3.15*	*3.2a*	*2.22*	*a*	Формирование $A_3(k, 0)$	

*) Это обстоятельство используется для определения контрольной суммы в ходе отладки. Пока контрольная сумма неизвестна, вместо нее в шапке можно отперфорировать любой код, например нуль, но тогда при вводе этой части материала с помощью подпрограммы «Быстрый ввод» произойдет останов машины. Полученную в машине правильную контрольную сумму можно прочесть на пульте управления и отперфорировать ее в шапке.

3.08	*2.21*	*3.37*	*3.2c*	*8*	$((2^{-10}-1) \cdot 2^{-10})_{\Phi} \Rightarrow$ $\Rightarrow 2.21$; $0 \Rightarrow 2.22$	
3.09	*2.21*	*2.2a*	*2.21*	*f*	$(n \cdot 2^{-10})_{\Phi} \Rightarrow 2.21$	
3.0a	*3.24*	*3.2b*	*2.21*	*a*	Формирование $A_5(n, 0)$	
3.0b	*2.27*	*3.36*	*2.21*	*a*	*3.00 —n 0.00 0.00 0⇒* $\Rightarrow 2.27$	
3.0c	*2.27*	*2.27*	*3.2e*	*9*	*0.00 0.00 3.00—n 0⇒* $\Rightarrow 2.28$	Φ_2
3.0d	*2.21*	*2.20*	*3.37*	*f*	$((m-1) \cdot 2^{-20})_{\Phi} \Rightarrow$ $\Rightarrow 2.21$	
3.0e	*2.21*	*2.25*	*2.21*	*b*	*2.26 k+m—1 2.01 2⇒* $\Rightarrow 2.21$	
3.0f	*3.10*	*2.23*	*3.2e*	*4*	$(2^{-20})_{\Phi} \Rightarrow 2.23$	
3.10	*3.11*	*2.24*	*3.32*	*4*	Формирование $A_2(k)$	
3.11	*3.12*	*2.26*	*3.34*	*4*		
3.12	*2.25*	*2.27*	*3.35*	*4*		
3.13	*3.15*	*3.15*	*3.2e*	*b*	Переадресация $A_3(k, i)$ и $A_5(n, l)$	$F(i)$
3.14	*3.24*	*3.24*	*3.2c*	*b*		
*3.15**	*2.26*	*3.15**	*3.22*	*b*	$(k+i) \Rightarrow 2.26$	A_3
3.16	*2.23*	*2.26*	*3.2e*	*9*	Накапливание Σ_2	
3.17	*2.27*	*2.27*	*2.24*	*b*		
3.18	*2.23*	*2.23*	*3.2c*	*9*		
3.19	*2.27*	*2.27*	*2.24*	*b*		
3.1a	*3.24*	*2.20*	*3.2c*	*7*	$p(m_1 \geqslant 1)$	p_1
3.1b	*2.20*	*2.20*	*3.2c*	*b*	$m_1 - 1$	A_4
3.1c	*3.22*	*3.22*	*2.26*	*e*	$\lvert(3.22)\rvert \cdot \operatorname{sign} R_i \Rightarrow 3.22$	
3.1d	*2.24*	*2.26*	*3.2f*	*9*	$R \cdot (2^{-3})_{\Phi} \Rightarrow 2.25$; остаток $\Rightarrow 2.24$	
3.1e	*3.20*	*2.23*	*2.28*	*4*	*0.00 0.00 3.00—n 0⇒* $\Rightarrow 2.23$	
3.1f	*2.23*	*2.23*	*3.2c*	*8*		

3.20	2.24	2.25	3.2c	9	Выделение jA (j = I, II, III)	
3.21	3.23	2.24	3.36	7	$p(\lvert jA \rvert \geqslant 3.00 \rvert)$	A_4
3.22*	2.26	2.26	2.23	b*	Изм. «меченого» адреса	
3.23	3.1f	3.2d	2.25	7		
3.24*	3.24*	2.26	2.22	b		A_5
3.25	3.13	3.15	2.2*	7	$p(i \geqslant m-1)$	p_2
3.26	3.29	2.27	2.29	6	$p(\Sigma_1 = \Sigma_2)$	p_3
3.27	3.29	2.29	2.27	6		
3.28	2.2a	2.2a	0.03	3	Печать (2.2a)	A_6, ω_2
3.29	0.00	2.29	2.27	5	Останов, вызов Σ_1 и Σ_2	Ω
3.2a	2.26	0.00	2.22	b	A_3 (0, 0)	
3.2b	0.00	2.26	2.22	b	A_5 (0, 0)	
3.2c	0.01	0.00	0.00	1	$(2^{-10})_\phi$	
3.2d	0.00	0.00	0.5f	f		
3.2e	0.00	0.01	0.00	1	$(2^{-20})_\phi$	
3.2f	0.80	0.00	0.00	1	$(2^{-3})_\phi$	
3.30	2.21	2.20	2.01	2	A_1	
3.31	3.04	2.29	2.01	2		
3.32	2.25	2.25	2.23	a	$A_2(0)$	
3.33	2.26	0.00	2.01	2		
3.34	2.24	2.25	2.21	7		
3.35	3.15	2.27	2.22	4		
3.36	3.00	0.00	0.00	0		
3.37	0.00	3.ff	0.00	0	Выделитель IIA	

§ 2. Программа «Первоначальный ввод — Б»

Программа «Первоначальный ввод — Б» предназначена для ввода стандартной подпрограммы «Быстрый ввод» в ячейки памяти *3.00—3.37*. Контроль ввода осуществляется самой

подпрограммой «Быстрый ввод»; программа «Первоначальный ввод — Б» только подготовляет ее для соответствующей работы и передает управление команде *3.15* (оператору A_3) этой подпрограммы. Для того чтобы подпрограмма «Быстрый ввод» могла проконтролировать ввод самой себя, на месте переменных команд должны быть отперфорированы те коды, которые записаны в приведенной только что программе.

Программа занимает ячейки памяти *2.30—2.41* и вводится по адресам, отперфорированным у каждой команды. Вход программы — ячейка *2.30*. После ввода первого экземпляра стандартной подпрограммы «Быстрый ввод» программа «Первоначальный ввод — Б» больше не нужна, и ячейки памяти, в которых она расположена, могут быть использованы для других целей. Если же в процессе работы эта программа не забивается другим материалом, то ее можно использовать многократно без предварительного ввода.

Логическая схема

$$A_1 A_2 \Phi \rho \omega^{1} \to 3.15$$

Здесь A_1 — ввод первого кода шапки подпрограммы «Быстрый ввод» в ячейку *2.2a*; A_2 — ввод с помощью цикла последних двух кодов шапки и подпрограммы «Быстрый ввод»; Φ — формирование констант и исходного вида команд «Быстрого ввода», необходимых для его работы без переработки материала; ρ — установление режима фиксированной запятой с двойной точностью.

Программа

2.30	2.31	2.2a	2.01	2	0.00 0.00 3.00 0 $\Rightarrow$ 2.2a	A_1
2.31	2.32	2.32	2.3d	4	Завершение ввода	A_2
2.32*	2.33	2.fe*	2.01	2		
2.33	2.32	2.32	2.3e	a		
2.34	2.32	2.32	2.3f	7		

2.35	2.20	2.fe	3.37	f	Формирование констант сравнения	Ф
2.36	2.21	2.40	2.20	a		
2.37	2.38	3.15	2.40	4	Восстановление команд 3.15 и 3.24	
2.38	2.39	3.24	2.41	4		
2.39	2.22	2.22	2.22	a	$0 \Rightarrow 2.22$	
2.3a	2.27	2.27	2.27	a	$0 \Rightarrow 2.27$	
2.3b	2.3c	2.29	2.ff	4	$\Sigma_1 \Rightarrow 2.29$	
2.3c	0.00	0.0c	3.15	0	ФД	ρ, ω
2.3d	2.33	2.fe	2.01	2	Константы и исходный вид команд	
2.3e	0.00	0.01	0.00	1		
2.3f	2.33	3.38	2.01	2		
2.40	2.26	3.00	2.22	b		
2.41	3.00	2.26	2.22	b		

ГЛАВА V

СТАНДАРТНЫЕ ПОДПРОГРАММЫ ДЛЯ РЕЖИМА ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Подпрограммы этой главы используются при работе в режиме плавающей запятой. Они удовлетворяют всем предъявляемым к стандартным программам требованиям, изложенным в главе III.

§ 1. Стандартные константы

	0.00	0.1f	0.00	0	
	0.00	3.42	1.b4	5	
2.00	0.00	0.00	0.00	0	-0
2.01	2.16	1.21	3.ed	5	$\left(\frac{\pi}{2}\right)_{\text{п}}$
2.02	2.00	0.00	0.00	1	$(2^{-1})_{\Phi}$
2.03	1.00	0.00	0.00	1	$(2^{-2})_{\Phi}$
2.04	0.66	1.99	2.66	7	$(10^{-1})_{\Phi}$
2.05	0.01	0.00	0.00	1	$(2^{-10})_{\Phi}$, e_{III}
2.06	0.00	0.01	0.00	1	$(2^{-20})_{\Phi}$, e_{II}
2.07	0.00	0.00	0.01	1	$(2^{-30})_{\Phi}$, e_{I}
2.08	0.00	0.00	0.00	3	$(2^{-33})_{\Phi}$
2.09	3.f0	0.00	0.00	0	Выделитель порядка
2.0a	2.05	2.2e	1.0c	3	$(\ln 2)_{\text{п}}$
2.0b	0.07	3.ff	3.ff	f	Выделитель мантиссы
2.0c	2.14	0.00	0.00	1	$(1)_{\text{п}}$
2.0d	2.04	0.00	0.00	1	$(2^{-1})_{\text{п}}$
2.0e	0.00	0.00	3.ff	0	Выделитель *1A*

2.0f	*0.10*	*0.00*	*0.00*	*1*	$(2^{-6})_{\Phi}$
2.10	*2.12*	*2.12*	*2.0e*	*f*	Стандартный выход
2.11	*2.12*	*2.12*	*2.07*	*a*	
2.12	*0.00*	*0.00*	*0.00*	*0*	
2.13	*0.01*	*0.00*	*0.01*	*1*	$e_{\text{I}}+e_{\text{III}}$
2.14					
2.15	*0.00*	*0.00*	*0.08*	*1*	$(2^{-27})_{\Phi}$
2.16	*0.08*	*0.00*	*0.00*	*1*	$(2^{-4})_{\Phi}$
2.17					
2.18	*2.15*	*2.a0*	*2.79*	*b*	$(\sqrt{2})_{\Phi}$
2.19	*2.45*	*0.00*	*0.00*	*1*	$(10)_{\text{п}}$
2.1a	*0.00*	*0.00*	*0.00*	*5*	$(2^{-32})_{\Phi}$
2.1b	*0.40*	*0.00*	*0.00*	*1*	$(2^{-4})_{\Phi}$
2.1c	*1.ff*	*3.ff*	*3.ff*	*e*	$(2^{-33}-2^{-1})_{\Phi}$
2.1d	*0.00*	*2.22*	*2.02*	*4*	
2.1e					
2.1f					

Ячейки *2.14*, *2.17*, *2.1e*, *2.1f* не используются.

Ячейки *2.10*—*2.12* образуют стандартный выход, к нему происходит обращение после работы любой подпрограммы; константы *2.10*—*2.12*, а также используемые в стандартном выходе константы *2.07*, *2.0e*, поэтому всегда упоминаются при характеристике подпрограмм.

Ячейка *2.1d* используется для связи с подпрограммой выделения целой и дробной частей числа при обращении к ней из другой стандартной подпрограммы.

§ 2. Стандартная подпрограмма для перевода чисел из десятичной системы счисления в двоичную

В том случае, когда для решения задачи требуется небольшое количество исходных данных, перевод их в двоичную систему не представляет труда и может быть осуществлен на руках. Если же задача требует большой числовой информации, сам перевод чисел является уже довольно трудоемкой задачей и должен быть передан или машине, или специальному устройству, предназначенному для этой цели.

В машине М-2 нет устройства, осуществляющего перевод чисел из десятичной системы в двоичную. Эту задачу решает сама машина по специальной подпрограмме.

Подпрограмма рассчитана на перевод семизначных нормализованных десятичных чисел x в двоичные числа в системе плавающей запятой.

Здесь

$$x=10^{p}\cdot X,$$

$$X=0,\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7,\quad \alpha_i=0,\ 1,\ \ldots,\ 9,\quad \alpha_1\neq 0.$$

Подготовка чисел для перевода

Все десятичные числа, которые необходимо перевести в двоичную систему с помощью подпрограммы, должны быть предварительно выписаны на бланках в следующем виде:

$$\varepsilon_x\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7\,|\,p\,|\,\varepsilon_p. \qquad (V.1)$$

Здесь ε_x означает знак числа, α_i — десятичные цифры мантиссы ($\alpha_1\neq 0$), p — десятичный однозначный порядок числа, ε_p — знак порядка.

При подготовке ленты с числовой информацией ε_x и ε_p наносятся на ленту с помощью двоичной клавиатуры (1 соответствует положительному знаку, 0 — отрицательному знаку), α_i и p — с помощью шестнадцатеричной клавиатуры. Так как входные устройства вводят в память машины двоичные эквиваленты шестнадцатеричных цифр, то десятичное число, записанное и отперфорированное в виде (V.1), в ячейке памяти будет изображаться кодом числа

$$\left(2\bar{\varepsilon}_p-1\right)\left(\bar{\varepsilon}_x\cdot 2^{-1}+\alpha_1\cdot 2^{-5}+\ldots+\alpha_7\cdot 2^{-29}+|p|\cdot 2^{-33}\right)$$

в системе фиксированной запятой. Здесь $\bar{\varepsilon}_x$ и $\bar{\varepsilon}_p$ — коды знаков числа и порядка.

Описание алгоритма. Логическая схема программы

Перевод осуществляется следующим образом. Вначале согласно формуле

$$(X)_{\Phi}=\sum_{i=1}^{7}\left(\alpha_i 10^{-i}\right)_{\Phi} \qquad (V.2)$$

десятичная мантисса представляется в системе фиксированной запятой, затем по известной величине $(X)_\phi$ определяется $(X)_\text{п}$ и, наконец, учитывается десятичный порядок числа x:

$$(x)_\text{п} = [(10)_\text{п}]^p \cdot (X)_\text{п}. \qquad \text{(V. 3)}$$

Ниже приводится логическая схема программы.

$$\rho_1 H \downarrow \overline{A_1 p\,(n \geqslant 7)}^{\uparrow} A_2 \rho_2 A_3 0.$$

Оператор ρ_1 переключает машину на режим фиксированной запятой с двойной точностью.

Оператор H подготовляет для работы арифметические операторы A_1, A_2 и A_3.

Оператор A_1 по известной величине y_{n-1} определяет y_n:

$$y_n = [y_{n-1} + (-2^{-1}\alpha_{8-n})_\phi](10^{-1})_\phi; \quad y_0 = 0. \qquad \text{(V. 4)}$$

Операторы A_1 и $p\,(n \geqslant 7)$ вычисляют по схеме Горнера значение многочлена, отличающегося от многочлена (V. 2) множителем 2^{-1}, т. е. вычисляют величину $y_7 = (-2^{-1}X)_\phi$.

Оператор A_2 вычисляет величину $(-|X|)_\text{п}$.

Оператор ρ_2 переключает машину на режим плавающей запятой.

Оператор A_3 вычисляет $(x)_\text{п}$.

Краткая характеристика программы

Число ячеек запоминающего устройства, занятых программой, 24.

Число тактов: максимальное 82, минимальное 45, в среднем (при $|p| = 4$) 63; при $x = 0$ требуется 19 тактов работы машины.

Относительная ошибка $(x)_\text{п}$ меньше $5 \cdot 10^{-8}$.

Используемые ячейки со стандартными константами: *2.00*, *2.02*, *2.04*, *2.07*, *2.0b*, *2.0e* — *2.12*, *2.19*, *2.1b*, *2.1c*.

Используемые стандартные рабочие ячейки *2.20*—*2.25*.

Число x в десятичной системе помещается в ячейку *2.20*.

Число x в системе плавающей запятой $(x)_\text{п}$ получается тоже в ячейке *2.20*.

Числа для перевода должны быть подготовлены так, как сказано в начале этого параграфа.

Программа

	0.15	*0.17*	*0.00*	*0*		
	0.00	*1.82*	*1.14*	*5*		
3.00	*0.00*	*0.0c*	*3.01*	*0*	Ф1	ρ_1
3.01	*3.02*	*2.22*	*2.00*	*4*	$y_0 = 0 \Rightarrow 2.22$	H
3.02	*3.10*	*3.10*	*2.20*	*e*	$\operatorname{sign} x \cdot \|(3.10)\| \Rightarrow 3.10$	
3.03	*2.23*	*2.1c*	*2.20*	*f*	$\bar{e}_x = \bar{e}_p = 0$	
3.04	*2.23*	*2.1b*	*2.23*	*9*	Выделение кодов	
3.05	*3.06*	*2.25*	*2.23*	*4*	p и α_i	
3.06	*2.23*	*2.1b*	*2.24*	*9*	$y_n \Rightarrow 2.22$	A_1
3.07	*2.22*	*2.22*	*2.23*	*d*		
3.08	*2.21*	*2.22*	*2.04*	*9*		
3.09	*3.06*	*2.00*	*2.24*	*7*	$p\,(n \geqslant 7)$	p
3.0a	*3.0d*	*2.24*	*3.17*	*4*	$(-\|X\|)_\text{п}$	A_2
3.0b	*2.24*	*2.24*	*2.0f*	*d*		
3.0c	*2.21*	*2.22*	*2.02*	*9*		
3.0d	*3.0b*	*2.0b*	*2.22*	*7*		
3.0e	*2.22*	*2.22*	*2.24*	*c*		
3.0f	*0.00*	*0.12*	*3.12*	*0*	ПЗ	ρ_2
*3.10**	*2.22*	*2.22*	*2.19*	*9**	$(2.22) \times (10)_\text{п} \Rightarrow 2.22$	A_3
3.11	*2.25*	*2.25*	*2.1b*	*b*	$p - 1$	
3.12	*3.10*	*2.00*	*2.25*	*7*	$q(p = 0)$	
3.13	*2.22*	*2.22*	*2.00*	*c*	Нормализация	
3.14	*3.16*	*2.20*	*2.02*	*7*	$p(x \geqslant 0)$	
3.15	*2.22*	*2.22*	*2.02*	*e*	$(x)_\text{п}$	
3.16	*2.10*	*2.20*	*2.22*	*4*		0

3.17	*1.d0*	*0.00*	*0.00*	*1*	Константа

Стандартные константы, используемые в программе:

2.00	*0.00*	*0.00*	*0.00*	*0*	-0
2.02	*2.00*	*0.00*	*0.00*	*1*	$(2^{1})_{\phi}$
2.04	*0.66*	*1.99*	*2.66*	*7*	$(10^{1})_{\phi}$.
2.0b	*0.07*	*3.ff*	*3.ff*	*f*	Выделитель мантиссы
2.0f	*0.10*	*0.00*	*0.00*	*1*	$(2^{-6})_{\phi}$
2.19	*2.45*	*0.00*	*0.00*	*1*	$(10)_{n}$
2.1b	*0.40*	*0.00*	*0.00*	*1*	$(2^{-4})_{\phi}$
2.1c	*1.ff*	*3.ff*	*3.ff*	*e*	

Пояснения к программе

Команда *3.00* оператора ρ_1 осуществляет переключение машины на режим фиксированной запятой с двойной точностью.

Команда *3.01* оператора H помещает в ячейку *2.22*, где всегда хранится y_n (V. 4), величину $y_0 = 0$.

Команда *3.02*, учитывая знак десятичного порядка числа, подготавливает для работы оператор A_3.

Команда *3.03* гасит 1-й и 34-й разряды кода десятичного числа: $\bar{\varepsilon}_x = \bar{\varepsilon}_p = 0$. Присваивая отрицательные знаки числу и его порядку, мы, во-первых, избавляемся от некоторых переменных команд в операторах A_1 и A_3, а во-вторых, от необходимости вводить счетчик в цикле *3.06*—*3.09*. (Знаки числа и порядка будут учтены в операторе A_3.)

Команда *3.04* расщепляет код десятичного числа x на две части, отделяя код десятичного порядка от кода десятичных цифр мантиссы. В результате выполнения этой операции в ячейке *2.23* получается число $-(p \cdot 2^{1})_{\phi}$, а в ячейке *2.24* — число $-(\alpha_1 2^{-9} + \alpha_2 2^{-13} + \ldots + \alpha_7 2^{-33})_{\phi}$.

Команда *3.05* отсылает число $(-p2^{-4})_{\phi}$ на хранение в ячейку *2.25*.

Команда *3.06* оператора A_1 при первом выполнении выделяет код младшей десятичной цифры мантиссы α_7: $-(\alpha_7 2^{-4})_{\phi}$. При этом код в ячейке *2.24* будет сдвинут на четыре разряда вправо, и тем самым подготовлен для выделения следующей десятичной цифры:

$$(2.24) = -(\alpha_1 \cdot 2^{13} + \alpha_2 2^{-17} + \ldots + \alpha_6 \cdot 2^{-33})_{\phi}.$$

Команда *3.07* добавляет $(-\alpha_7 \cdot 2^{1})_{\phi}$ к содержимому ячейки *2.22* (V. 4):

$$y_0 + (-\alpha_7 2^{1})_{\phi} \Rightarrow 2.22.$$

Команда *3.08*, умножая (*2.22*) на $(10^{1})_{\phi}$, получает y_1:

$$(2.22) \cdot (10^{1})_{\phi} = y_1 = [y_0 - (\alpha_7 2^{-4})_{\phi}] \cdot (10^{1})_{\phi} \Rightarrow 2.22.$$

Команда *3.09* передает управление команде *3.06*, так как $(2.24) \neq 0$ ($\alpha_1 \neq 0$). При следующем выполнении цикла в ячейке *2.22* будет получен y_2, затем y_3 и так далее, пока после очередного выполнения цикла не получится нуль в ячейке *2.24*, т. е. до тех пор, пока не будет вычислен

$$y_7 = (-2^{-1}X)_{\phi}.$$

Для того чтобы из $(-2^{1}X)_{\phi}$ получить $(-|X|)_{n}$, необходимо код числа $(-2^{-1}X)_{\phi}$ сдвинуть в разряды мантиссы (так, чтобы старшая двоичная цифра попала в 8-й разряд), а в разрядах порядка установить порядок, равный

$$-7 + k + 4 = -3 + k,$$

где k — количество разрядов, на которое нужно сдвинуть вправо мантиссу, а 4 учитывает деление $(-2^{-1}X)_{\phi}$ на $(-2^{1})_{\phi}$. Вычисление $(-|X|)_{n}$ осуществляет оператор A_2, а именно:

команда *3.0a* пересылает в рабочую ячейку *2.24* константу (*3.17*) — число с нулевой мантиссой и порядком $p = -3$ — и передает управление команде *3.0d*;

команда *3.0d* передает управление или следующей по порядку команде, или команде *3.0b* — первой команде цикла — в зависимости от того, сделано нужное число сдвигов кода $(-2^{1}X)_{\phi}$ или нет;

команда *3.0b* прибавляет к (*2.24*) единицу порядка;

команда *3.0c* осуществляет сдвиг на один разряд вправо.

В результате работы цикла в разрядах с 8-го по 33-й ячейки *2.22* получается мантисса числа $(-|X|)_{n}$, а в разрядах с 1-го по 6-й — его условный порядок.

Команда *3.0e* из порядка и мантиссы формирует $(-|X|)_{\text{п}}$.

Команда *3.0f* (оператор p_2) переключает машину на режим плавающей запятой и передает управление команде *3.12*.

Команда *3.12* оператора A_3 реализует выход из цикла *3.10—3.12*, осуществляющего вычисление $(-|x|)_{\text{п}}$ в соответствии с формулой (V.3). Домножение $(-|X|)_{\text{п}}$ на $(10^p)_{\text{п}}$ производится последовательным умножением $(-|X|)_{\text{п}}$ на $(10^p)_{\text{п}}$ p раз, если $p>0$ и последовательным делением, если $p<0$. Это умножение (или деление) осуществляется командой *3.10*. Код операции этой команды (*8* или *9*) формируется командой *3.02* оператора H в зависимости от знака p.

Команды *3.11* и *3.12* обеспечивают повторение цикла ровно p раз.

Нормализация $(-|x|)_{\text{п}}$ командой *3.13* фактически необходима лишь в случае $x=0$, когда в операторе A_2 получается не нормализованное число $(-|X|)_{\text{п}}$.

Команда *3.14* выясняет знак x и в случае $x\geqslant 0$ числу $(-|x|)_{\text{п}}$ присваивается положительный знак (команда *3.15*).

Следует отметить, что часть вычислений по данной программе ведется в режиме фиксированной запятой с двойной точностью. Применение этого режима позволило написать довольно компактную программу для вычисления $(X)_{\Phi}$. Это обстоятельство и явилось решающим фактором при выборе алгоритма перевода.

§ 3. Стандартная подпрограмма для перевода чисел из двоичной системы счисления в десятичную

Данная подпрограмма используется при выводе результатов вычислений на печать.

Описание алгоритма и требования, предъявляемые к программе

Целое число p, удовлетворяющее соотношению

$$\left.\begin{aligned}&(x)_{\text{п}}=[(10)_{\text{п}}]^p(X)_{\text{п}},\\&\frac{1}{10}\leqslant(|X|)_{\text{п}}<1,\end{aligned}\right\}\qquad\text{(V.5)}$$

является десятичным порядком x.

Следовательно, десятичный порядок легко получить последовательным умножением (или делением) $(x)_{\text{п}}$ на $(10)_{\text{п}}$ до тех пор, пока абсолютная величина результата умножения (или деления) не попадет в пределы промежутка $\left[\frac{1}{10},\ 1\right)$.

Если теперь $(X)_{\text{п}}$ умножить на $(10)_{\text{п}}$ и выделить целую часть, то получим старшую десятичную цифру мантиссы α_1; умножив дробную часть на $(10)_{\text{п}}$ и вновь выделив целую часть, получим следующую цифру мантиссы α_2. Продолжая этот процесс, можно получить все необходимые цифры мантиссы.

То обстоятельство, что цифры мантиссы получаются в системе плавающей запятой, незначительно усложняет дело. Однако применение алгоритма в таком виде неудобно из других соображений. Дело в том, что среди элементарных операций М-2 нет операции выделения целой и дробной частей числа, поэтому на реализацию этого алгоритма потребовалось бы много тактов работы машины. Исходя из этого, цифры десятичной мантиссы получаются несколько иначе. Число X представляется в системе фиксированной запятой. Умножая $(X)_{\Phi}$ на $10\cdot 2^{-33}$ в режиме фиксированной запятой с двойной точностью, получаем одновременно и целую часть от числа $10\cdot X$ в виде $(\alpha_1\cdot 2^{-33})_{\Phi}$ и дробную часть. Умножая затем дробную часть на $10\cdot 2^{-33}$, получаем $(\alpha_2\cdot 2^{-33})_{\Phi}$ и дробную часть от $10^2\cdot X$. Продолжая этот процесс, можно получить столько цифр десятичной мантиссы, сколько необходимо.

Прежде чем объяснять логическую схему программы, необходимо сделать некоторые замечания относительно требований, предъявляемых к программе. Желательно, чтобы десятичные цифры печатались в таком виде, в каком они вводятся в машину, т. е. в виде

$$\varepsilon_x\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7|p|\varepsilon_p,\qquad\text{(V.6)}$$

где ε_x и ε_p — знаки числа и порядка, α_i — десятичные цифры мантиссы, p — десятичный порядок числа. В соответствии с этим, а также с замечанием относительно печати кодов, сделанным в главе I при описании выходного устройства (см. стр. 40), необходимо, чтобы двоичные коды чисел α_i,

$|p|$, а также коды $\bar{\varepsilon}_x$ и $\bar{\varepsilon}_p$ были расположены в ячейке памяти следующим образом:

$$\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7\,|\,p\,|\,\bar{\varepsilon}_p\bar{\varepsilon}_x.$$

Распределение кодов по разрядам ячейки следующее:
1—4 5—8 9—12 13—16 17—20 21—24 25 28 29 - 32 33 34

Логическая схема

Логическая схема программы выглядит следующим образом:

$$\overline{\mathcal{З}p(x=0)\,\uparrow\,\overline{H0\uparrow A_1\downarrow p(|y_n|\geqslant 1)\uparrow A_2 p(|z_k|\leqslant 1-\varepsilon)\uparrow\rho_1 A_3 A_1\rho_2\omega}}.$$

Оператор $\mathcal{З}$ очищает ячейку *2.21*, в которой будет формироваться десятично-кодированное число.

Оператор H осуществляет печать десятичного числа.

Оператор A_1 по известной величине y_{n-1} находит y_n

$$y_n = y_{n-1}(10)_n \Rightarrow \langle y_{n-1}\rangle,\ y_0 = (x)_n \qquad (V.7)$$

и запоминает — n. Операторы A_1 и $p(|y_n|\geqslant 1)$ вычисляют величину

$$y_{n_1} = (X)_n (10)_n,\quad 1 \leqslant |y_{n_1}| < 10, \qquad (V.8)$$

а также

$$-n_1 = p - 1. \qquad (V.9)$$

Оператор A_2 по z_{k-1} определяет z_k

$$z_k = \frac{z_{k-1}}{(10)_n} \Rightarrow \langle z_{k-1}\rangle,\quad z_0 = y_{n_1} \qquad (V.10)$$

и запоминает величину $-n_1 + k$. Операторы A_2 и $p(|z_k|\leqslant\leqslant 1-\varepsilon)$ вычисляют

$$z_{k_1} = (X)_n = 2^{p_1} X_1,\ \frac{1-\varepsilon}{10} < (|X|)_n \leqslant 1-\varepsilon, \qquad (V.11)$$

а также

$$p = -n_1 + k_1 \qquad (V.12)$$

в ячейке *2.21* в единицах 32-го разряда. Величина $\varepsilon > 0$ выбрана из тех соображений, чтобы в дальнейшем, при округлении мантиссы, не могло произойти переполнение.

Оператор ρ_1 переключает машину на режим фиксированной запятой с двойной точностью.

Оператор A_3 устанавливает в соответствующих разрядах ячейки *2.21* коды $\bar{\varepsilon}_x$ и $\bar{\varepsilon}_p$, получает $(X)_\phi$ и округляет ее.

Оператор A_4, используя алгоритм выделения десятичных цифр, описанный в начале настоящего параграфа, получает в ячейке *2.25* семь старших десятичных цифр мантиссы (в разрядах с 1-го по 28-й) и формирует из (*2.25*) и (*2.21*) десятично-кодированное число.

Оператор ρ_2 переключает машину на режим плавающей запятой и осуществляет безусловную передачу управления оператору H.

Краткая характеристика программы

Число занятых программой ячеек памяти 32.

Число тактов: максимальное 87, минимальное 50, в среднем ($|p|=4$) 63.

Ячейки со стандартными константами, используемые в подпрограмме: *2.00*, *2.02*, *2.07*, *2.08*, *2.0c*, *2.0e* —*2.12*, *2.19* — —*2.1b*.

Используемые стандартные рабочие ячейки: *2.20*—*2.26*.

Точность: относительная ошибка десятичного числа равна $5\cdot 10^{-8}$.

Число x, которое нужно перевести в десятичную систему, необходимо поместить в ячейку *2.20*. Во время работы программы (*2.20*) портится.

Полученное в ячейке *2.21* десятичное число печатается.

Программа

	0.1c	*0.1f*	*0.00*	*0*		
	0.02	*0.7b*	*3.fc*	*f*		
3.00	*3.01*	*2.21*	*2.00*	*4*		*З*
3.01	*3.05*	*2.00*	*2.20*	*7*	$p(x=0)$	*p*
3.02	*2.10*	*2.21*	*0.03*	*3*	Печать числа	*H0*

3.03	2.20	2.20	2.19	9	$y_n \Rightarrow 2.20$	A_1
3.04	2.21	2.21	2.1a	a	$(-n2^{-32})_\Phi \Rightarrow 2.21$	
3.05	3.03	2.20	2.0c	7	$p(\lvert y_n \rvert \geqslant 1)$	p
3.06	2.20	2.20	2.19	8	$z_k \Rightarrow 2.20$	A_2
3.07	2.21	2.21	2.1a	b	$\lvert(-n_1+k)2^{-32}\rvert_\Phi \Rightarrow 2.21$	
3.08	3.06	3.1e	2.20	7	$p(\lvert z_k \rvert \leqslant 1-\varepsilon)$	p
3.09	0.1f	0.0c	3.0a	0	ФД	ρ_1
3.0a	3.0c	2.21	2.00	6	Образование $\bar{\varepsilon}_x$ и $\bar{\varepsilon}_y$	A_3
3.0b	2.21	2.21	2.08	b		
3.0c	2.21	2.21	2.20	e		
3.0d	2.23	2.20	3.1c	9	$(X_1)_\Phi \Rightarrow 2.23$	
3.0e	2.20	2.20	2.02	e	$(\lvert X \rvert)_n \Rightarrow 2.20$	
3.0f	3.12	2.25	2.00	4		
3.10	2.20	2.20	2.0f	b		
3.11	2.22	2.23	2.02	9	$(X)_\Phi \Rightarrow 2.23$	
3.12	3.10	2.20	2.02	7		
3.13	2.26	3.1d	2.23	e	Округление $(X)_\Phi$	
3.14	2.23	2.26	2.23	b		
3.15	2.23	2.23	3.1f	9	$\operatorname{sign} x \cdot (\alpha_1 2^{-33})_\Phi \Rightarrow 2.24$	A_4
3.16	2.25	2.24	2.25	b		
3.17	2.25	2.25	2.1b	8	$[(2.25) \cdot 2^4]_\Phi \Rightarrow 2.25$	
3.18	3.15	2.25	3.09	7	$p(i \geqslant 7)$	
3.19	2.25	2.25	2.02	8	Формирование десятичного числа	
3.1a	2.21	2.25	2.21	b		
3.1b	0.00	0.12	3.02	0	П З	ρ_2, ω
3.1c	0.00	0.00	0.10	1	$(2^{-26})_\Phi$	
3.1d	0.00	0.00	0.35	b	$\varepsilon = (0{,}5 \cdot 10^{-7})_\Phi$	
3.1e	2.07	3.ff	3.ff	7	$(1-\varepsilon)_n$	
3.1f	0.00	0.00	0.01	5	$(10 \cdot 2^{-33})_\Phi$	

Стандартные константы, используемые в программе:

2.00	0.00	0.00	0.00	0	—0
2.02	2.00	0.00	0.00	1	$(2^{-1})_\Phi$
2.08	0.00	0.00	0.00	3	$(2^{-33})_\Phi$
2.0c	2.14	0.00	0.00	1	$(1)_n$
2.0f	0.10	0.00	0.00	1	$(2^{-6})_\Phi$
2.19	2.45	0.00	0.00	1	$(10)_n$
2.1a	0.00	0.00	0.00	5	$(2^{-32})_\Phi$
2.1b	0.40	0.00	0.00	1	$(2^{-4})_\Phi$

Пояснения к программе

Команда *3.00* (оператор *З*) очищает ячейку *2.21*, в которой будет формироваться десятичное число.

Команда *3.01* проверяет на равенство нулю число x. При $x=0$ переводить x в десятичную систему не нужно и управление передается команде *3.02*, осуществляющей печать десятичного числа. При $x \neq 0$ управление передается команде *3.05*, проверяющей выполнение условия $|y_n| \geqslant 1$ (при первом выполнении команды *3.05* $y_n = x$). При невыполнении этого условия управление передается команде *3.03* — первой команде цикла *3.03—3.05*.

Команда *3.03* оператора A_1 вычисляет по формуле (V.7) величину y_n, а команда *3.04* запоминает в ячейке *2.21* величину $-n$ в виде условного числа $-(n \cdot 2^{-32})_\Phi$. В результате работы цикла вычисляется величина $y_{n_1} = (10X)_n$, а также $-(n_1 \cdot 2^{-32})_\Phi = [(p-1) \cdot 2^{-32}]_\Phi$ (см. формулы (V.7), (V.8), (V.9)).

Команда *3.06* оператора A_2 вычисляет z_k, а команда *3.07* — величину $[(-n_1+k) \cdot 2^{-32}]_\Phi$, добавляя $(2^{-32})_\Phi$ к (*2.21*).

Команда *3.08* передает управление или вновь первой команде цикла *3.06—3.08*, или команде *3.09* в зависимости от того, выполняется условие $|z_k| > 1-\varepsilon$ или нет. В результате работы цикла вычисляются $(X)_n = 2^{p_1} X_1$ и $(p2^{-33})_\Phi$ (см. формулы (V.10), (V.11) и (V.12)).

Команда *3.09* (оператор ρ_1) переключает машину на режим фиксированной запятой с двойной точностью.

Команда *3.0a* оператора A_3 выясняет знак десятичного порядка p. Если $p \geqslant 0$, управление передается команде *3.0b*; если $p < 0$ — команде *3.0c*.

Команда *3.0b* устанавливает в 33-м разряде ячейки *2.21* код $\bar{\varepsilon}_p = 1$.

Команда *3.0c* присваивает (*2.21*) знак x. После выполнения команды *3.0c* в ячейке *2.21* сформирован код

$$\operatorname{sign} x \cdot \left(|p| \cdot 2^{-37} + \bar{\varepsilon}_p \cdot 2^{-33}\right)_\Phi.$$

Команда *3.0d* выделяет код мантиссы X_1 числа $(X)_\Pi$, причем, поскольку это выделение осуществляется умножением $(X)_\Pi$ на $\left(2^{-26}\right)_\Phi$ в режиме фиксированной запятой с двойной точностью, то старшая цифра мантиссы попадает в первый разряд ячейки *2.23*.

Команда *3.0e* получает абсолютную величину $(X)_\Pi$.

Команда *3.0f* очищает ячейку *2.25* и передает управление команде *3.12*.

Для того чтобы получить теперь $(X)_\Phi$, достаточно (*2.23*), т. е. $(X_1)_\Phi$ умножить на $\left(2^{p_1}\right)_\Phi$ или, что то же, сдвинуть (*2.23*) вправо на $|p_1|$ разрядов. Это и делают команды цикла *3.10—3.12*.

Команда *3.10* добавляет единицу к условному порядку числа $(X)_\Pi$.

Команда *3.11* сдвигает $(X_1)_\Phi$ на один разряд вправо.

Команда *3.12* обеспечивает повторение цикла $|p_1|$ раз.

Команды *3.13* и *3.14* производят округление $(X)_\Phi$.

Команда *3.15*, умножая $(X)_\Phi$ на $\left(10 \cdot 2^{-33}\right)_\Phi$, при первом выполнении получает в ячейке *2.24* $\operatorname{sign} x \cdot \left(\alpha_1 \cdot 2^{-33}\right)_\Phi$, а в ячейке *2.23* $(\{10X\})_\Phi$.

Команда *3.16* добавляет $\operatorname{sign} x \cdot \left(\alpha_1 \cdot 2^{-33}\right)_\Phi$ к (*2.25*), а команда *3.17* сдвигает (*2.25*) на четыре разряда влево. При следующем повторении цикла в ячейке *2.23* получается $\operatorname{sign} x \cdot \left(\alpha_2 \cdot 2^{-33}\right)_\Phi$, в ячейке *2.24* $(\{10^2X\})_\Phi$, а в ячейке *2.25* $\operatorname{sign} x \cdot \left(\alpha_1 \cdot 2^{-29} + \alpha_2 \cdot 2^{-33}\right)_\Phi$.

Команда *3.18* обеспечивает повторение цикла 7 раз. В результате работы цикла в ячейке *2.25* получается число

$$\operatorname{sign} x \cdot \left(\alpha_1 \cdot 2^{-5} + \alpha_2 \cdot 2^{-9} + \ldots + \alpha_7 \cdot 2^{-29}\right)_\Phi.$$

Команда *3.19* сдвигает (*2.25*) на один разряд влево и, наконец, команда *3.1a* формирует из (*2.21*) и (*2.25*) десятично-кодированное число.

Команда *3.1b* переключает машину на режим плавающей запятой и передает управление оператору H.

Заметим, что в команде *3.18* для сравнения используется код команды *3.09*. То обстоятельство, что для операции переключения несущественно содержимое разрядов третьего адреса, позволило использовать команду *3.09* в качестве константы для сравнения, взяв нужное содержимое третьего адреса (см. гл. II, § 3).

§ 4. Стандартная подпрограмма для выделения целой и дробной частей числа

Описание алгоритма

Подпрограмма вычисляет целую и дробную части положительного числа x согласно их обычному математическому определению, а целую и дробную части отрицательного числа — согласно формулам

$$\{x\} = -\{|x|\}, \quad [x] = x - \{x\}.$$

Целая часть, кроме того, представляется в виде условного числа $[x]_{усл} = \left([x] \cdot 2^{-33}\right)_\Phi$.

Если $x = 2^p \cdot X$, где $p > 0$ и $\frac{1}{2} \leqslant X < 1$, то для получения дробной части x отделяются старшие p разрядов мантиссы X и младшие разряды сдвигаются на p разрядов влево. Полученной мантиссе приписывается нулевой порядок и в случае необходимости результат нормализуется. Целая часть x получается вычитанием полученной дробной части из значения x.

Для получения $[x]_{усл}$ старшие p разрядов мантиссы помещаются в младшие разряды.

Логическая схема

$$A_1 p\,(|x| \geqslant 1)^{\downarrow} \overline{A_2 A_3 A_4 A_5 \uparrow A_6 0 \downarrow} \, \underline{\Im\omega}\,|\,.$$

Оператор A_1 производит переключение на режим фиксированной запятой с двойной точностью, получает величину

$(2^{-32})_\phi$ и заносит нуль в ячейку *2.25*, где будет помещена величина $[x]_{усл}$.

Оператор A_2 получает величину $\left(-\frac{1}{2}|X|\right)_\phi$ и порядок p в виде $(-p\cdot 2^{-33})_\phi$.

Оператор A_3 образует величину $(2^{-32+p})_\phi$.

Оператор A_4 формирует $[-|x|]_{усл}$ и $\{-|x|\}$ и производит переключение на режим плавающей запятой.

Оператор A_5 получает нормализованную величину

$$\{x\} = \operatorname{sign} x \cdot \{|x|\}$$

и присваивает величине $[x]_{усл}$ требуемый знак.

Оператор A_6 получает нормализованную целую часть путем вычитания из x полученной дробной части.

Оператор $З$ пересылает x в ячейку, где должна находиться $\{x\}$.

Краткая характеристика программы

Количество занятых ячеек *21*.

Число тактов: максимальное 43; минимальное (для $|x| < 1$) 10; в среднем (для $p = 6$) 32 такта.

Точность сохраняется, но при больших положительных p у дробной части остается мало значащих цифр.

Используемые стандартные константы: *2.00*, *2.02*, *2.03*, *2.07*, *2.0c*, *2.0e*, *2.10—2.12*, *2.15*, *2.16*, *2.1c*, *2.1d*.

Используемые стандартные рабочие ячейки: *2.21—2.28*.

Аргумент $x \Rightarrow 2.23$.

Функции $\{x\} \Rightarrow 2.24$; $[x] \Rightarrow 2.23$; $[x]_{усл} \Rightarrow 2.25$.

Ячейка обратной связи *2.12*.

Дополнительные сведения. При использовании данной подпрограммы необходимо, чтобы в ячейке *2.1d* находилась команда: (*2.1d*): *N 2.22 2.02 4*, где *N* — адрес первой команды данной подпрограммы. Содержимое ячейки *2.1d* достаточно задать один раз, если эта ячейка не используется для других целей.

Обращение к данной подпрограмме производится командой вида

(*n*): *2.1d 2.12 n 4*.

Программа

(При выполнении команды *2.1d* в ячейку *2.22* засылается $\left(\frac{1}{2}\right)_\phi$.)

	0.15	*0.14*	*0.00*	*0*		
	0.00	*1.8f*	*0.41*	*0*		
3.00	*0.00*	*0.0c*	*3.01*	*0*	ФД	A_1
3.01	*2.25*	*2.07*	*2.03*	*9*	$(2^{-32})_\phi \Rightarrow 2.26$; $0 \Rightarrow 2.25$	
3.02	*3.14*	*2.23*	*2.0c*	*7*	$p(\lvert x\rvert \geqslant 1)$	p
3.03	*2.24*	*2.23*	*2.1c*	*f*		A_2
3.04	*2.24*	*2.24*	*2.15*	*9*	$\left(-\frac{1}{2}\lvert X\rvert\right)_\phi \Rightarrow 2.24$; $(-p\cdot 2^{-33})_\phi \Rightarrow 2.25$	
3.05	*3.07*	*2.28*	*2.24*	*4*	Получение величины $(2^{-32+p})_\phi$ в ячейке *2.26*	A_3
3.06	*2.21*	*2.22*	*2.22*	*9*		
3.07	*2.24*	*2.25*	*2.02*	*9*		
3.08	*3.0a*	*2.24*	*2.02*	*7*		
3.09	*2.26*	*2.26*	*2.22*	*8*		
3.0a	*3.06*	*2.00*	*2.25*	*7*		
3.0b	*2.24*	*2.28*	*2.26*	*9*	$[-\lvert x\rvert]_{усл} \Rightarrow 2.25$; $(\{\lvert x\rvert\})_\phi \Rightarrow 2.24$	A_4
3.0c	*2.26*	*2.24*	*2.16*	*9*	$-\{\lvert x\rvert\} \Rightarrow 2.24$	
3.0d	*2.24*	*2.27*	*2.02*	*a*		
3.0e	*0.00*	*0.12*	*3.0f*	*0*	ПЗ	
3.0f	*2.24*	*2.24*	*2.00*	*c*	Нормализация $-\{\lvert x\rvert\}$	A_5
3.10	*2.24*	*2.24*	*2.23*	*e*	$\{x\} \Rightarrow 2.24$	
3.11	*2.25*	*2.25*	*2.23*	*e*	$[x]_{усл} \Rightarrow 2.25$	
3.12	*2.23*	*2.23*	*2.24*	*c*	$[x] \Rightarrow 2.23$	A_6
3.13	*0.00*	*0.12*	*2.10*	*0*	0	0
3.14	*3.12*	*2.24*	*2.23*	*4*		З, ω

Стандартные константы, используемые в программе:

2.00	*0.00*	*0.00*	*0.00*	*0*	—0
2.02	*2.00*	*0.00*	*0.00*	*1*	$(2^{-1})_{\Phi}$
2.03	*1.00*	*0.00*	*0.00*	*1*	$(2^{-2})_{\Phi}$
2.07	*0.00*	*0.00*	*0.01*	*1*	$(2^{-30})_{\Phi}$
2.0c	*2.14*	*0.00*	*0.00*	*1*	$(1)_{\text{п}}$
2.15	*0.00*	*0.00*	*0.08*	*1*	$(2^{-27})_{\Phi}$
2.16	*0.08*	*0.00*	*0.00*	*1*	$(2^{-7})_{\Phi}$
2.1c	*1.ff*	*3.ff*	*3.ff*	*e*	

Пояснения к программе

Команда *3.03* гасит знаковый разряд у кода, изображающего аргумент, и старший разряд кода, который служит для изображения знака порядка.

К моменту работы оператора A_3 в ячейке *2.25* находится величина

$$\left(-p\cdot 2^{-33}\right)_{\Phi},\quad \text{где } p=\sum_{i=0}^{4}2^{i}\varepsilon_i \quad (\varepsilon_i=0;\ 1).$$

После команды *3.05* начинает работать цикл по i. Команда *3.06* последовательно получает в ячейке *2.22* величины 2^{-2^i}. Первый раз эта команда обходится, так как в ячейке *2.22* командой *2.0d* уже получена величина 2^{-2^i} при $i=0$, т. е. $\frac{1}{2}$.

Команда *3.07* сдвигает код, стоящий в ячейке *2.25*, на один разряд вправо и выделяет очередной двоичный разряд ε_i, помещая его в старший разряд ячейки *2.24*. Если $\varepsilon_i=1$, то команда *3.09* умножает содержимое ячейки *2.26*, где первоначально находилась величина $\left(2^{-32}\right)_{\Phi}$, на величину 2^{2^i} (фактически производится деление на величину 2^{-2^i}).

Если в ячейке *2.25* есть еще $\varepsilon_k\neq 0$, то команда *3.0a* передает управление команде *3.06* — на повторение цикла по i. Когда будут учтены все значащие разряды порядка p, в ячейке *2.26* будет получена величина $\left(2^{-32+p}\right)_{\Phi}$ и команда *3.0a* передает управление команде *3.0b*.

Команда *3.0b* в ячейке *2.25* получает величину

$$[-|x|]_{\text{усл}}=\left(-\frac{1}{2}|X|\right)_{\Phi}\left(2^{-32+p}\right)_{\Phi}=\left(-|X|\cdot 2^{-33+p}\right)_{\Phi},$$

а в ячейке *2.24* — величину $\{|x|\}$ в виде числа в системе фиксированной запятой.

Команда *3.0c* сдвигает полученную в ячейке *2.24* величину на 7 разрядов вправо, помещая результат в ячейку *2.27*, а команда *3.0d* приформировывает к этому коду нулевой порядок и его условном изображении, в результате чего в ячейке *2.24* получается $\{|x|\}$ в системе плавающей запятой.

§ 5. Стандартная подпрограмма для вычисления $\sin x$ и $\cos x$

Описание алгоритма

Сущность выбранного алгоритма состоит в том, что из аргумента x сначала выделяется целое кратное 2π, а затем, используя обычные формулы приведения, задача сводится к вычислению косинуса угла, расположенного в первой четверти. Косинус приведенного угла вычисляется с помощью полинома восьмой степени.

Аргумент x можно представить в виде

$$x=\operatorname{sign}x\cdot|x|=\operatorname{sign}x\cdot\left(2k\pi+\alpha_1\pi+\beta_1\frac{\pi}{2}+t_1\right),$$

где $\alpha_1=0;\ 1,\ \beta_1=0;\ 1,\ 0\leqslant t_1<\frac{\pi}{2}$. (Значения α_1 и β_1 вместе определяют четверть, в которой находится аргумент $|x|$ после выделения из него целого кратного 2π.)

Тогда

$$\begin{aligned}\sin|x|&=\sin\left(2k\pi+\alpha_1\pi+\beta_1\frac{\pi}{2}+t_1\right)=\sin\left(\alpha_1\pi+\beta_1\frac{\pi}{2}+t_1\right)=\\&=(-1)^{\alpha_1}\left[\sin\beta_1\frac{\pi}{2}\cdot\cos t_1+\cos\beta_1\frac{\pi}{2}\cdot\sin t_1\right]=\\&=(-1)^{\alpha_1}[\beta_1\cos t_1+(1-\beta_1)\sin t_1].\end{aligned}$$

Окончательно получаем:

$$\begin{aligned}\sin x&=\operatorname{sign}x\cdot\sin|x|=\\&=\operatorname{sign}x\cdot[(-1)^{\alpha_1}\beta_1\cos t_1+(-1)^{\alpha_1}(1-\beta_1)\sin t_1].\end{aligned}\qquad (V.13)$$

Проводя аналогичные выкладки, получим:

$$\cos x = (-1)^{\alpha_1}(1-\beta_1)\cos t_1 - (-1)^{\alpha_1}\beta_1 \sin t_1. \quad (V.14)$$

Соотношения (V. 13) и (V. 14) означают не что иное, как обычные формулы приведения. В них значения $\sin x$ и $\cos x$ выражаются через синус и косинус угла t_1, расположенного в первой четверти.

Для составления стандартной подпрограммы формулы (V. 13) и (V.14) целесообразно преобразовать так, чтобы в правые части формул входил, например, только косинус. Для этого положим

$$t_2 = \frac{\pi}{2} - t_1,\ \beta_2 = 1 - \beta_1,\ \alpha_2 = 0 \quad \text{при} \quad \alpha_1 = \beta_1$$

$$\text{и } \alpha_2 = 1 \quad \text{при} \quad \alpha_1 \neq \beta_1,$$

тогда будем иметь

$$\sin x = \operatorname{sign} x \cdot \left[(-1)^{\alpha_1}\beta_1 \cos t_1 + (-1)^{\alpha_2}\beta_2 \cos t_2\right], \quad (V.15)$$

$$\cos x = (-1)^{\alpha_1}(1-\beta_1)\cos t_1 + (-1)^{\alpha_2}(1-\beta_2)\cos t_2, \quad (V.16)$$

где по-прежнему $\alpha_i = 0;\ 1,\ \beta_i = 0;\ 1,\ 0 \leqslant t_i < \frac{\pi}{2}$ $(i = 1, 2)$.

В справедливости формул (V. 15) и (V. 16) нетрудно убедиться, произведя указанную замену.

В стандартной подпрограмме $\cos t$ вычисляется через многочлен 4-й степени относительно t^2:

$$\cos t \approx a_0 + a_1 t^2 + a_2 t^4 + a_3 t^6 + a_4 t^8, \quad (V.17)$$

где

$$\begin{aligned} a_0 &= \ \ 0{,}999\,999\,95, \\ a_1 &= -1{,}233\,698\,19, \\ a_2 &= \ \ 0{,}253\,650\,64, \\ a_3 &= -0{,}020\,810\,46, \\ a_4 &= \ \ 0{,}000\,858\,11. \end{aligned}$$

Этот многочлен получен свертыванием с помощью полиномов Чебышева [3] отрезка степенного ряда для $\cos t$, содержащего 7 членов разложения, и обеспечивает вычисление $\cos t$ для $t \in [0;\ 1]$ с точностью до $5 \cdot 10^{-8}$.

Вычисление полинома в подпрограмме осуществляется по схеме Горнера.

Логическая схема

$$\Phi K \overset{\downarrow}{A_1} p \underset{\uparrow}{\underline{A_2}} 0.$$

Оператор Φ формирует в операторе A_1 команду для учета знака x при вычислении $\sin x$.

Обобщенный оператор K вычисляет $x_1 = \left| x : \frac{\pi}{2} \right|$ и, используя подпрограмму «Выделение целой и дробной частей числа» (см. § 4 этой главы), вычисляет $[x_1]$, $t_1 = \{x_1\}$, α_1 и β_1.

Оператор A_1 вычисляет по полиному (V. 17) значение $\cos t_i$, умножает $\cos t_i$ на $(-1)^{\alpha_i}$ и посылает в ячейку, где получается $\cos x$, величину

$$(-1)^{\alpha_i}(1-\beta_i)\cos t_i,$$

а в ячейку, где получается $\sin x$ — величину

$$\operatorname{sign} x \cdot (-1)^{\alpha_i}\beta_i \cos t_i.$$

Оператор p является логическим условием, осуществляющим передачу управления стандартному выходу, если подготовлен выход из подпрограммы. В противном случае управление передается оператору A_2.

Оператор A_2 вычисляет $t_2 = 1 - t_1$, β_2 и α_2, подготавливает выход из подпрограммы и передает управление оператору A_1.

Более подробно о работе всех операторов будет сказано в пояснениях к программе.

Краткая характеристика программы

Количество занятых ячеек 29.

Число тактов: максимальное 85 (из них 43 на выделение $[x_1]$ и $\{x_1\}$); минимальное 50 (из них 10 на выделение $[x_1]$ и $\{x_1\}$); в среднем 73 такта (для $2\pi \leqslant |x| \leqslant 4\pi$).

Точность $5 \cdot 10^{-8}$.

Используемые стандартные константы: *2.00—2.03*, *2.06—2.08*, *2.0c*, *2.0e*, *2.10—2.12*, *2.15*, *2.16*, *2.1a*, *2.1c*, *2.1d*.

Используемые стандартные рабочие ячейки: *2.20—2.26*, *2.2a*.

Аргумент $x \Rightarrow$ *2.20*.

Функция $\sin x \Rightarrow$ *2.21*, $\cos x \Rightarrow$ *2.22*.

Ячейка обратной связи *2.2a*.

Дополнительные сведения: требуется наличие подпрограммы выделения целой и дробной частей числа.

Программа

	0.18	*0.1c*	*0.00*	*0*		
	0.03	*0.81*	*1.71*	*d*		
3.00	*3.13*	*3.13*	*2.20*	*e*	*(3.13)* | sign $x \Rightarrow$ *3.13*	Ф
3.01	*2.23*	*2.20*	*2.01*	*8*	$x : \frac{\pi}{2} \Rightarrow$ *2.23*	
3.02	*2.23*	*2.23*	*2.01*	*e*	$x_1 = \left\lfloor x : \frac{\pi}{2} \right\rfloor \Rightarrow$ *2.23*	*K*
3.03	*2.1d*	*2.12*	*3.03*	*4*	$[x_1] \Rightarrow$ *2.25*; $t_1 = \{x_1\} \Rightarrow$ *2.24*	
3.04	*2.23*	*2.24*	*2.24*	*9*		
3.05	*2.26*	*3.1c*	*2.23*	*9*		
3.06	*2.26*	*2.26*	*3.1b*	*d*		
3.07	*2.26*	*2.26*	*2.23*	*9*	Вычисление полинома (V.17) по схеме Горнера $\cos t_i \Rightarrow$ *2.23*	
3.08	*2.26*	*2.26*	*3.1a*	*d*		
3.09	*2.26*	*2.26*	*2.23*	*9*		
3.0a	*2.26*	*2.26*	*3.19*	*d*		
3.0b	*2.26*	*2.26*	*2.23*	*9*		
3.0c	*2.23*	*2.26*	*3.18*	*d*		*A_1*
3.0d	*2.26*	*2.25*	*2.1a*	*f*	$\alpha_i \Rightarrow$ *2.26*	
3.0e	*3.10*	*2.26*	*2.1a*	*7*	$\lvert(2.26)\rvert \geqslant (2^{-32})_\phi$	
3.0f	*2.23*	*2.00*	*2.23*	*c*	$-\cos t_i \Rightarrow$ *2.23*	
3.10	*2.26*	*2.25*	*2.08*	*f*	$\beta_i \Rightarrow$ *2.26*	
3.11	*3.13*	*2.00*	*2.26*	*7*	$0 \geqslant \lvert(2.26)\rvert$	
3.12	*3.14*	*2.22*	*2.23*	*4*	$\cos x \Rightarrow$ *2.22*	
*3.13**	*2.21*	*2.00*	*2.23*	*b**	$\sin x \Rightarrow$ *2.21*	
3.14	*2.10*	*2.06*	*2.12*	*7*		*p*

3.15	*2.25*	*2.25*	*2.08*	*b*	$[x_1] + (2^{-33})_\phi \Rightarrow$ *2.25*	*A_2*
3.16	*2.24*	*2.0c*	*2.24*	*c*	$t_2 = 1 - t_1 \Rightarrow$ *2.24*	
3.17	*3.04*	*2.12*	*2.2a*	*4*		
3.18	*2.07*	*3.ff*	*3.ff*	*b*	$a_0 =$ 0.999 999 95	
3.19	*2.14*	*3.bd*	*0.e9*	*4*	$a_1 = -$ 1.233 698 19	
3.1a	*1.f4*	*0.3b*	*3.3f*	*b*	$a_2 =$ 0.253 650 64	
3.1b	*1.b5*	*1.4f*	*1.5a*	*6*	$a_3 = -$ 0.020 810 46	
3.1c	*1.67*	*0.1e*	*1.75*	*f*	$a_4 =$ 0.000 858 11	

Стандартные константы, используемые в программе:

2.00	*0.00*	*0.00*	*0.00*	*0*	-0
2.01	*2.16*	*1.21*	*3.ed*	*5*	$\left(\frac{\pi}{2}\right)_\text{п}$
2.06	*0.00*	*0.01*	*0.00*	*1*	$(2^{-20})_\phi$
2.08	*0.00*	*0.00*	*0.00*	*3*	$(2^{-33})_\phi$
2.0c	*2.14*	*0.00*	*0.00*	*1*	$(1)_\text{п}$
2.1a	*0.00*	*0.00*	*0.00*	*5*	$(2^{-32})_\phi$

2.1d — ячейка связи с подпрограммой выделения целой и дробной частей числа.

Пояснения к программе

Перед обращением к данной подпрограмме аргумент x должен быть помещен в ячейку *2.20*. Непосредственное обращение к подпрограмме производится командой основной программы с адресом n:

$$(n) : a_\text{п}\ 2.2a\ n\ 4,$$

где $a_\text{п}$ — истинный адрес команды *3.00* подпрограммы.

Команда *3.00* (оператор Ф) присваивает знак x коду, изображающему команду *3.13*, формируя операцию фиксированного сложения (код операции *b*) при $x \geqslant 0$ или операцию фиксированного вычитания (код операции *a*) при $x < 0$.

Команда *3.01* обобщенного оператора K делит аргумент x на $\frac{\pi}{2}$, а команда *3.02* присваивает полученному частному положительный знак, получая $x_1 = \left\lfloor x : \frac{\pi}{2} \right\rfloor$ в ячейке *2.23*.

Командой *3.03* производится обращение к подпрограмме выделения целой и дробной частей числа, которая вычисляет $t_1 = \{x_1\}$ в виде нормализованного числа и помещает его в ячейку *2.24*. В ячейку *2.25* помещается целая часть x_1 в виде условного числа $([x_1] \cdot 2^{33})_\phi$. Код, находящийся в ячейке *2.25*, используется затем в качестве логической шкалы: наличие единицы (нуля) в 33-м разряде кода означает, что β_i равно единице (нулю), а наличие единицы (нуля) в 32-м разряде означает, что α_i равно единице (нулю).

Команды *3.04*—*3.0c* оператора A_1 реализуют вычисление аппроксимирующего полинома (V. 17) для $\cos t_i$, помещая $\cos t_i$ в ячейку *2.23*. Программа вычислений по схеме Горнера составлена без цикла. Это увеличивает количество команд, но дает экономию в тактах.

Команды *3.0d* и *3.0e* исследуют 32-й разряд логической шкалы, что эквивалентно исследованию α_i. При $\alpha_i = 1$ выполняется команда *3.0f*, которая вычитанием из нуля изменяет знак у $\cos t_i$ на обратный. При $\alpha_i = 0$ команда *3.0f* обходится, т. е. знак $\cos t_i$ не меняется. Таким образом, команды *3.0d*—*3.0f* вычисляют величину $(-1)^{\alpha_i} \cos t_i$, помещая ее в ячейку *2.23*.

Команды *3.10* и *3.11* исследуют 33-й разряд логической шкалы, что эквивалентно исследованию β_i. Если $\beta_i = 0$, то соответствующее слагаемое в правой части формулы (V. 15) будет равно нулю и его учитывать не следует. В этом случае управление передается команде *3.12*, которая содержимое ячейки *2.23* переносит в ячейку *2.22*. Выполнение команды *3.12* соответствует получению величины

$$(-1)^{\alpha_i}(1-\beta_i)\cos t_i.$$

Заметим, что в силу соотношения $\beta_2 = 1 - \beta_1$ в правых частях формул (V. 15) и (V. 16) может быть отлично от нуля только одно из слагаемых, поэтому при выполнении команды *3.12* в ячейке *2.22* уже получен $\cos x$; команда *3.12* передает управление команде *3.14* (оператор p), обходя команду *3.13*.

Если $\beta_i = 1$, то команда *3.11* передает управление команде *3.13*, минуя команду *3.12*. Команда *3.13*, сформированная оператором Ф, вычисляет в ячейке *2.21* значение $\sin x$ с учетом знака x, после чего управление передается оператору p. После выполнения команды *3.13* в ячейке *2.21* получается величина

$$\operatorname{sign} x \cdot (-1)^{\alpha_i} \beta_i \cos t_i.$$

Команда *3.14* (логическое условие p) сравнивает по абсолютной величине содержимое ячейки *2.12* с константой $(2^{-20})_\phi$. В начале подпрограммы (в операторе K) имеется обращение к стандартной подпрограмме выделения целой и дробной частей числа, после которой работает стандартный выход. В результате его работы в ячейке *2.12* формируется команда, код которой, рассматриваемый как число *), будет меньше величины $(2^{-20})_\phi$, поэтому команда *3.14* первый раз передаст управление команде *3.15* оператора A_2.

Команда *3.15* оператора A_2 сложением в режиме фиксированной запятой добавляет к содержимому ячейки *2.25* единицу младшего разряда, формируя тем самым логическую шкалу для α_2 и β_2. Команда *3.16* вычисляет $t_2 = 1 - t_1$. Команда *3.17* пересылает содержимое ячейки обратной связи данной подпрограммы в ячейку *2.12* стандартного выхода и передает управление команде *3.04* оператора A_1, который работает точно так же, как и в первый раз, только вместо величин t_1, α_1 и β_1 будут фигурировать величины t_2, α_2 и β_2. Находящийся теперь в ячейке *2.12* код, рассматриваемый как число, будет больше величины $(2^{-20})_\phi$, поэтому при повторном выполнении команды *3.14* управление передается стандартному выходу.

§ 6. Стандартная подпрограмма для вычисления e^x

Описание алгоритма

Элементарные преобразования

$$e^x = 2^{\log_2 e^x} = 2^{x \cdot \frac{1}{\ln 2}} = 2^{\left[\frac{x}{\ln 2}\right]} \cdot 2^{\left\{\frac{x}{\ln 2}\right\}} =$$

$$= 2^{\left[\frac{x}{\ln 2}\right]} \cdot e^{\left\{\frac{x}{\ln 2}\right\} \ln 2} = 2^{\left[\frac{x}{\ln 2}\right]} \cdot (e^z)^2,$$

$$z = \left\{\frac{x}{\ln 2}\right\} \cdot \frac{\ln 2}{2}, \quad |z| \leqslant \frac{\ln 2}{2}$$

*) Напомним (см. гл. III, § 4), что при работе стандартного выхода в ячейке *2.12* формируется команда

0.00 0.00 s 0,

где s есть адрес команды, которой передается управление из стандартного выхода.

сводят задачу к вычислению e^z при z, удовлетворяющих неравенству

$$|z| \leqslant \frac{\ln 2}{2}.$$

Функция e^z представляется дробно-рациональной функцией

$$e^z \approx \frac{z^3 + 12z^2 + 60z + 120}{-z^3 + 12z^2 - 60z + 120}; \qquad (V.18)$$

при этом допускается ошибка, не превосходящая $0{,}9 \cdot 10^{-8}$.

Соотношение (V.18) вытекает из разложения [1]:

$$e^z = \frac{z+2}{-z+2} + \sum_{n=1}^{\infty} \frac{(-1)^n 2 \cdot z^{2n+1}}{P_n(-z)\,P_{n+1}(-z)}.$$

$$P_n(z) = b_n z^n + b_{n-1} z^{n-1} + \ldots + b_1 z + b_0,$$

где $b_i = \frac{(2n-i)!}{i!\,(n-i)!}$ для $i = 0, 1, \ldots, n$. Для интервала $|z| \leqslant \frac{\ln 2}{2}$ ряд в этом разложении знакопеременный, поэтому, ограничиваясь $n = 3$, получим

$$e^z = \frac{z^3 + 12z^2 + 60z + 120}{-z^3 + 12z^2 - 60z + 120} + \delta,$$

где

$$|\delta| \leqslant \left| \frac{(-1)^3 2z^7}{P_3(-z)\,P_4(-z)} \right| \leqslant 0{,}9 \cdot 10^{-8}.$$

Из соображений точности и скорости вычисления e^z выражение (V.18) преобразуется к следующему виду:

$$e^z \approx \frac{12(z^2+10) + z(z^2+60)}{12(z^2+10) - z(z^2+60)}. \qquad (V.19)$$

Логическая схема

$$Kp([y] \geqslant -32)\downarrow З\omega \uparrow \rho_1 \Phi \rho_2 A \downarrow 0.$$

Оператор K вычисляет $y = \frac{x}{\ln 2}$, $[y]$ и $\{y\}$. Используется целая часть, записанная в условных единицах (см. § 4 этой главы).

Оператор $З$ помещает нуль в ячейку результата.

Оператор ρ_1 осуществляет переключение машины на режим фиксированной запятой.

Оператор Φ формирует $2^{[y]}$.

Оператор ρ_2 осуществляет переключение машины на режим плавающей запятой.

Оператор A подсчитывает z, e^z по формуле (V.19) и получает окончательный результат e^x.

Краткая характеристика программы

Количество занятых программой ячеек 24.

Число тактов: максимальное число тактов 53 (из них 31 на выделение дробной и целой частей $y = \frac{x}{\ln 2}$), минимальное число тактов 31 (из них 9 на выделение $[y]$ и $\{y\}$), в среднем (при $4 \leqslant x < 8$) на вычисление e^x требуется 49 тактов работы машины.

Точность результата $5 \cdot 10^{-8}$.

Аргумент $x \Rightarrow$ *2.20*.

Функция $e^x \Rightarrow$ *2.21*.

Ячейка обратной связи *2.2a*.

Используемые стандартные константы: *2.00, 2.02, 2.03, 2.07, 2.0a, 2.0c, 2.0e, 2.10–2.12, 2.15, 2.16, 2.19, 2.1c, 2.1d*.

Используемые стандартные рабочие ячейки: *2.20–2.28, 2.2a*.

Дополнительные сведения: требуется наличие подпрограммы выделения целой и дробной частей числа.

Программа

	0.0e	0.17	0.00	0		
	0.00	3.bc	3.fc	d		
3.00	2.23	2.20	2.0a	8	$y = \frac{x}{\ln 2}$	K
3.01	2.1d	2.12	3.01	4	$[y] \Rightarrow 2.25$, $\{y\} \Rightarrow 2.24$	
3.02	3.04	3.16	2.25	6	$p([y] \geqslant -32)$	P

Declassified in Part - Sanitized Copy Approved for Release @ 50-Yr 2014/03/19 : CIA-RDP81-01043R004100010010-0

3.03	3.13	2.21	2.00	4	$0 \Rightarrow 2.21$	З, ω
3.04	0.00	0.18	3.05	0	ФЗ	γ_1
3.05	2.22	2.25	2.15	8	$2^{[y]}$	Ф
3.06	2.22	2.22	2.0c	b		
3.07	0.00	0.12	3.08	0	ПЗ	γ_2
3.08	2.23	2.24	3.17	9	z	
3.09	2.24	2.23	2.23	9		
3.0a	2.25	2.24	3.15	d		
3.0b	2.21	2.24	2.19	d		
3.0c	2.23	2.23	2.25	9	Вычисление e^z	
3.0d	2.21	2.21	3.14	9	по формуле (V.19)	A
3.0e	2.25	2.21	2.23	d		
3.0f	2.24	2.21	2.23	c		
3.10	2.21	2.25	2.24	8		
3.11	2.21	2.21	2.21	9	$(e^z)^2$	
3.12	2.21	2.22	2.21	9	$e^x = 2^{[y]}(e^z)^2$	
3.13	2.10	2.12	2.2a	4		0
3.14	2.46	0.00	0.00	1	$(12)_п$	
3.15	2.67	2.00	0.00	1	$(60)_п$	
3.16	0.00	0.00	0.04	0	$(-2^{-2s})_ф$	
3.17	1.f5	2.2e	1.0c	3	$\left(\frac{\ln 2}{2}\right)_п$	

Стандартные константы, используемые в программе:

2.00	0.00	0.00	0.00	0	-0
2.0a	2.05	2.2e	1.0c	3	$(\ln 2)_п$
2.0c	2.14	0.00	0.00	1	$(1)_п$
2.15	0.00	0.00	0.08	1	$(2^{-2s})_ф$
2.19	2.45	0.00	0.00	1	$(10)_п$

2.1d — ячейка для связи с подпрограммой выделения целой и дробной частей числа

§ 7. Стандартная подпрограмма для вычисления $\sqrt{x}$

Описание алгоритма

Так как $\sqrt{x} = \sqrt{2^p X} = \sqrt{2^p}\sqrt{X}$, то задача сводится к вычислению $\sqrt{X}$ и $\sqrt{2^p}$. Нахождение $\sqrt{2^p}$ не представляет труда, так как

$$\sqrt{2^p} = \begin{cases} 2^n, & \text{если} \quad p = 2n, \\ 2^n\sqrt{2}, & \text{если} \quad p = 2n+1. \end{cases}$$

Квадратный корень из мантиссы находится последовательными приближениями, причем очередное приближение y_{n+1} для $\sqrt{X}$ выражается через предыдущее формулой

$$y_{n+1} = \frac{1}{2}\left(\frac{X}{y_n} + y_n\right). \qquad (V.20)$$

Для промежутка $\left[\frac{1}{2}, 1\right)$, в котором заключена мантисса X, оказалось достаточно удобным за начальное приближение выбрать величину

$$y_0 = \frac{1}{2} + \frac{1}{2}X.$$

В худшем случае, для $X = \frac{1}{2}$, при таком выборе начального приближения уже y_3 отличается от $\sqrt{X}$ меньше чем на 10^{-8}.

В среднем для вычисления $\sqrt{X}$ с точностью 10^{-8} требуется две итерации.

Соотношение (V.20) получается, если решать уравнение $f(y) = y^2 - X = 0$ методом Ньютона. Отсюда же следует и оценка для числа итераций. Действительно, для метода Ньютона быстрота сходимости последовательных приближений определяется неравенством

$$|y_{n+1} - y| \leqslant \frac{\max f''(y)}{\min f'(y)}(y_n - y)^2,$$

а для нашего случая коэффициент

$$\frac{\max f''(y)}{\min f'(y)} \leqslant 2.$$

Отсюда видно, что каждая итерация при вычислении $\sqrt{X}$ примерно удваивает число верных значащих цифр.

Логическая схема

Обозначим $2^n\sqrt{X}$ через $\varphi(x)$, тогда

$$\sqrt{x}=\begin{cases}\varphi(x), & \text{если } p=2n,\\ \varphi(x)\sqrt{2}, & \text{если } p=2n-1.\end{cases}$$

Логическая схема программы выглядит следующим образом:

$$\rho_1 A_1 p\,(x\neq 0)\,\overline{A_2}\,\rho_2 q\cdot\overline{A_3}\,\theta.$$

Оператор ρ_1 переключает машину на режим фиксированной запятой с двойной точностью.

Оператор A_1 формирует $\left(\frac{1}{2}X\right)_\phi$ и подготовляет вычисление величины $32+n$ — условного порядка функции $\varphi(x)$. При этом в ячейке *2.22* получается $(2^{-1})_\phi$ или 0 в зависимости от того, нечетным или четным является p.

Оператор A_2 формирует условный порядок $\varphi(x)$, вычисляет начальное приближение y_0, получает $y=(\sqrt{X})_\phi$ по формуле

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

и, наконец, формирует функцию $\varphi(x)$ из условного порядка $32+n$ и $(\sqrt{X})_\phi$.

Оператор ρ_2 переключает на режим плавающей запятой.

Оператор q проверяет четность порядка p. В случае нечетного p управление передается следующей команде.

Оператор A_3 умножает $\varphi(x)$ на $\sqrt{2}$.

Краткая характеристика программы

Количество занятых ячеек памяти 18.

Число тактов: максимальное 31, минимальное 21, в среднем (считая, что для вычисления $(\sqrt{X})_\phi$ требуется две итерации) 25.

Точность: ошибка результата не превосходит $3{,}2\cdot 10^{-8}$.

Аргумент $x\Rightarrow 2.20$.

Функция $\sqrt{x}\Rightarrow 2.21$.

Используемые стандартные константы: *2.02*, *2.03*, *2.07*, *2.0e—2.12*, *2.15*, *2.16*, *2.18*.

Используемые стандартные рабочие ячейки: *2.20—2.28*.

Программа

	0.10	*0.11*	*0.00*	*0*		
	0.00	*2.6f*	*3.23*	*5*		
3.00	*0.00*	*0.4c*	*3.01*	*0*	ФД *)	ρ_1
3.01	*2.21*	*2.20*	*2.15*	*9*	$\left(\frac{1}{2}X\right)_\phi\Rightarrow 2.21$; $((32+p)\,2^{-32})_\phi\Rightarrow 2.22$	A_1
3.02	*2.22*	*2.22*	*2.02*	*9*	$((16+n)\,2^{-33})_\phi\Rightarrow 2.23$	
3.03	*2.23*	*2.23*	*2.15*	*8*	$((16+n)\,2^{-6})_\phi\Rightarrow 2.23$	
3.04	*3.0e*	*2.20*	*2.0f*	*7*	$p\,(x\neq 0)$	p
3.05	*2.23*	*2.23*	*2.03*	*b*	$((32+n)\,2^{-6})_\phi\Rightarrow 2.23$	A_2
3.06	*2.25*	*2.21*	*2.02*	*b*	y_0	
3.07	*2.26*	*2.21*	*2.25*	*8*	Вычисление $(\sqrt{X})_\phi$	
3.08	*2.27*	*2.25*	*2.02*	*9*		
3.09	*2.27*	*2.28*	*2.26*	*a*		
3.0a	*2.25*	*2.25*	*2.27*	*a*		
3.0b	*3.07*	*3.00*	*2.27*	*7*		
3.0c	*2.25*	*2.25*	*2.16*	*9*	$\varphi(x)$	
3.0d	*2.24*	*2.26*	*2.23*	*b*		
3.0e	*0.00*	*0.12*	*3.0f*	*0*	ПЗ	ρ_2
3.0f	*3.11*	*2.22*	*2.03*	*7*		q
3.10	*2.24*	*2.24*	*2.18*	*9*	$\varphi(x)\cdot\sqrt{2}$	A_3
3.11	*2.10*	*2.21*	*2.24*	*4*		θ

*) Используется также в качестве константы сравнения.

Стандартные константы, используемые в программе:

2.02	*2.00*	*0.00*	*0.00*	*1*	$(2^{-1})_ф$
2.03	*1.00*	*0.00*	*0.00*	*1*	$(2^{-2})_ф$
2.0f	*0.10*	*0.00*	*0.00*	*1*	$(2^{-6})_ф$
2.15	*0.00*	*0.00*	*0.08*	*1*	$(2^{-21})_ф$
2.16	*0.08*	*0.00*	*0.00*	*1*	$(2^{-7})_ф$
2.18	*2.15*	*2.a0*	*2.79*	*b*	$(\sqrt{2})_п$

§ 8. Стандартная подпрограмма для вычисления arctg x

Описание алгоритма

Для $|x| \leqslant 1$ arctg x вычисляется по следующей формуле

$$\operatorname{arctg} x \approx x \sum_{i=1}^{8} a_i x^{2i},$$

где

$$\begin{aligned} a_0 &= \ \ 1{,}00000000, \\ a_1 &= -0{,}33333061, \\ a_2 &= \ \ 0{,}19992355, \\ a_3 &= -0{,}14201562, \\ a_4 &= \ \ 0{,}10632794, \\ a_5 &= -0{,}07486925, \\ a_6 &= \ \ 0{,}04248576, \\ a_7 &= -0{,}01594163, \\ a_8 &= \ \ 0{,}00281805. \end{aligned}$$

Этот полином получен из разложения arctg x по многочленам Чебышева [3] и обеспечивает точность $5 \cdot 10^{-8}$. Полином $\sum_{i=1}^{8} a_i x^{2i}$ вычисляется по схеме Горнера. Если $|x| > 1$, то используется формула

$$\operatorname{arctg} x = \frac{\pi}{2} - \operatorname{arctg}\left(\frac{1}{x}\right).$$

Логическая схема

$$З_1 O(n)\, p(|x| \leqslant 1) \overbrace{!З_{2\omega} \downarrow A_1 \uparrow} A_2 \overbrace{\downarrow A_3(n) F(n) p(n \geqslant 8)!} A_4 0.$$

Оператор $З_1$ помещает a_8 в ячейку *2.23*, где будет накапливаться значение полинома.

Оператор $З_2$ засылает нуль в ячейку *2.21*.

Оператор A_1 получает $-\frac{1}{x}$ и засылает $\frac{\pi}{2}$ в ячейку *2.21*.

Оператор A_2 вычисляет z^2; $z = x$, если $|x| \leqslant 1$ и $z = -\frac{1}{x}$, если $|x| > 1$.

Оператор $A_3(n)$ по y_{n-1} определяет y_n:

$$y_n = y_{n-1} \cdot z^2 - a_{8-n}, \quad y_0 = -a_8.$$

В результате работы операторов $A_3(n)$, $F(n)$ и p вычисляется значение многочлена

$$-\sum_{i=0}^{8} a_i z^{2i}.$$

Оператор A_4 вычисляет

$$\operatorname{arctg} x = (2.21) - z \cdot \sum_{i=0}^{8} a_i z^{2i}.$$

Краткая характеристика программы

Число ячеек запоминающего устройства, занятых программой, 25.

Число тактов 43—44.

Точность $5 \cdot 10^{-8}$.

Используемые ячейки со стандартными константами: *2.00*, *2.01*, *2.07*, *2.0c*, *2.0e*, *2.10—2.12*.

Используемые стандартные рабочие ячейки: *2.20—2.23*.

Аргумент $x \Rightarrow 2.20$.

Функция arctg $x \Rightarrow 2.21$.

Программа

	0.10	*0.18*	*0.00*	*0*		
	0.01	*0.e0*	*0.c1*	*5*		
3.00	*3.01*	*2.23*	*3.18*	*4*	$y_0 = -a_8 \Rightarrow 2.23$	$З_1$

3.01	3.02	3.08	3.0e	4	Восстановление 3.08	O
3.02	3.04	2.0c	2.20	7	$p\,(x \leqslant 1)$	p
3.03	3.06	2.21	2.00	4	$0 \Rightarrow 2.21$	$\mathfrak{Z}_2, \omega$
3.04	2.20	3.10	2.20	8	$-\frac{1}{x}$	A_1
3.05	3.06	2.21	2.01	4	$\frac{\pi}{2} \Rightarrow 2.21$	
3.06	2.22	2.20	2.20	9	z^2	A_2
3.07	2.23	2.23	2.22	9	y_n	A_3
3.08*	2.23	2.23	3.17*	d		
3.09	3.08	3.08	2.07	a	Переадресация 3.08	F
3.0a	3.07	3.0f	3.08	7	$p\,(n \geqslant 8)$	p
3.0b	2.23	2.20	2.23	9	$\operatorname{arc\,tg} x \Rightarrow 2.21$	A_4
3.0c	2.21	2.21	2.23	c		
3.0d	0.00	0.00	2.10	0		0
3.0e	2.23	2.23	3.17	d	Константа восстановления	
3.0f	2.23	2.23	3.0f	d	Константа сравнения	
3.10	2.14	0.00	0.00	0	$-a_0$	Коэффициенты многочлена
3.11	1.f5	1.55	1.27	b	$-a_1$	
3.12	1.e6	1.97	0.61	2	$-a_2$	
3.13	1.e4	2.2d	2.45	5	$-a_3$	
3.14	1.d6	3.38	1.3a	e	$-a_4$	
3.15	1.d4	3.2a	2.86	3	$-a_5$	
3.16	1.c5	1.c0	2.c6	2	$-a_6$	
3.17	1.b4	0.53	0.03	5	$-a_7$	
3.18	1.85	3.15	3.7f	e	$-a_8$	

Стандартные константы, используемые в программе:

2.00	0.00	0.00	0.00	0	-0
2.01	2.16	1.21	3.ed	5	$\left(\frac{\pi}{2}\right)_п$
2.07	0.00	0.00	0.01	1	$(2^{-30})_ф$
2.0c	2.14	0.00	0.00	1	$(1)_п$

§ 9. Стандартная подпрограмма для вычисления $\ln x$

Описание алгоритма

В основе вычисления $\ln x$ лежит равенство

$$\ln x = p \ln 2 + \ln X.$$

Для вычисления $\ln X$ используется ряд

$$\ln X = 2 \sum_{k=0}^{\infty} \frac{1}{2k+1} \left(\frac{X-1}{X+1}\right)^{2k+1}.$$

С целью ускорения сходимости исходную формулу целесообразно преобразовать, положив $y = \lambda X$:

$$\ln x = p \ln 2 - \ln \lambda + \ln y,$$

где $\ln y$ вычисляется с помощью ряда

$$\ln y = 2 \sum_{k=0}^{\infty} \frac{1}{2k+1} \left(\frac{y-1}{y+1}\right)^{2k+1}.$$

Множитель λ подбирается из условия, чтобы величина $u = \frac{y-1}{y+1}$ при изменении X от $\frac{1}{2}$ до 1 лежала в промежутке, симметричном относительно нуля, что дает $\lambda = \sqrt{2}$ и $|u| < 0{,}172$. Тогда будем иметь:

$$\ln x = \left(p - \frac{1}{2}\right) \ln 2 + 2 \sum_{k=0}^{\infty} \frac{1}{2k+1} u^{2k+1}.$$

Если взять пять членов ряда, то погрешность не будет превосходить $0{,}6 \cdot 10^{-9}$.

Далее полагаем

$$\ln y \approx 2\left(u+\frac{1}{3}u^3+\frac{1}{5}u^5+\frac{1}{7}u^7+\frac{1}{9}u^9\right)\approx$$
$$\approx 2\left(u+\frac{1}{3}u^3+\frac{1}{5}u^5+cu^7\right),$$

где $c=\frac{1}{7}+\frac{0^2}{9}$.

Погрешность такой замены не превосходит $0{,}3\cdot 10^{-8}$.

Если член cu^7 с помощью полинома Чебышева заменить полиномом пятой степени, то окончательно получим:

$$\ln x=\left(p-\frac{1}{2}\right)\ln 2+a_1u+a_2u^3+a_3u^5,$$

где

$$a_1=2{,}000\,000\,815,$$
$$a_2=0{,}666\,445\,069,$$
$$a_3=0{,}415\,054\,254.$$

Абсолютная погрешность вычисления $\ln y$ по этой формуле не превосходит $0{,}3\cdot 10^{-7}$.

Логическая схема

$$p\left(x\geqslant 2^{-32}\right)!\overline{A_1A_20}\downarrow\Omega.$$

Оператор A_1 получает мантиссу и порядок заданного аргумента в виде нормализованных чисел.

Оператор A_2 вычисляет $\ln x$ по указанной выше формуле.

При $x<2^{-32}$ машина останавливается.

Краткая характеристика программы

Количество занятых ячеек 28.

Число тактов 26.

Точность: абсолютная погрешность равна $0{,}3\cdot 10^{-7}$. Для значений x, близких к единице, получается большая относительная погрешность.

Используемые стандартные константы: *2.02, 2.07, 2.09—2.0e, 2.10—2.12, 2.16, 2.18, 2.1a.*

Используемые стандартные рабочие ячейки: *2.20—2.24.*

Аргумент $x\Rightarrow 2.20$.

Функция $\ln x\Rightarrow 2.21$.

Программа

	0.14	*0.1b*	*0.00*	*0*		
	0.03	*0.5b*	*1.ae*	*f*		
3.00	*2.1a*	*2.20*	*2.16*	*6*	$p\left(x\geqslant(2^{-7})_\Phi\right)$	*p*
3.01	*0.00*	*0.18*	*3.02*	*0*	Ф З	
3.02	*2.21*	*2.20*	*2.0b*	*f*	$(X)_\Pi\Rightarrow 2.21$	
3.03	*2.21*	*2.21*	*2.02*	*b*		
3.04	*2.22*	*2.20*	*2.09*	*f*	$-\left((32+p)\cdot 2^{-6}\right)_\Phi\Rightarrow 2.22$	
3.05	*2.22*	*2.22*	*2.16*	*9*	$-\left((32+p)\cdot 2^{-13}\right)_\Phi\Rightarrow 2.22$	A_1
3.06	*2.22*	*2.22*	*3.17*	*b*	$-(32+p)_\Pi\Rightarrow 2.22$	
3.07	*0.00*	*0.12*	*3.08*	*0*	П З	
3.08	*2.22*	*3.18*	*2.22*	*c*	$(p)_\Pi=-(32)_\Pi+(32+p)_\Pi\Rightarrow 2.22$	
3.09	*2.22*	*2.22*	*2.0d*	*c*	$\left(p-\frac{1}{2}\right)\ln 2\Rightarrow 2.22$	
3.0a	*2.22*	*2.22*	*2.0a*	*9*		
3.0b	*2.23*	*2.21*	*2.18*	*9*	$y=\sqrt{2}X\Rightarrow 2.23$	
3.0c	*2.24*	*2.23*	*2.0c*	*c*	$u=\frac{y-1}{y+1}\Rightarrow 2.23$	
3.0d	*2.23*	*2.23*	*2.0c*	*d*		
3.0e	*2.23*	*2.24*	*2.23*	*8*		
3.0f	*2.24*	*2.23*	*2.23*	*9*		A_2
3.10	*2.21*	*3.1b*	*2.24*	*9*		
3.11	*2.21*	*2.21*	*3.1a*	*d*	$\ln y\Rightarrow 2.21$	
3.12	*2.21*	*2.21*	*2.24*	*9*		
3.13	*2.21*	*2.21*	*3.19*	*d*		
3.14	*2.21*	*2.21*	*2.23*	*9*		
3.15	*2.21*	*2.21*	*2.22*	*d*	$\ln x\Rightarrow 2.21$	
3.16	*0.00*	*0.00*	*2.10*	*0*		0
3.17	*2.60*	*0.00*	*0.00*	*0*		
3.18	*2.64*	*0.00*	*0.00*	*0*	$(-32)_\Pi$	

3.19	*2.24*	*0.00*	*0.01*	*b*	$a_1 = 2{,}000\,000\,82$
3.1a	*2.05*	*1.53*	*2.12*	*7*	$a_2 = 0{,}666\,445\,07$
3.1b	*1.f6*	*2.90*	*0.fe*	*d*	$a_3 = 0{,}415\,054\,25$

Стандартные константы, используемые в программе:

2.02	*2.00*	*0.00*	*0.00*	*0*	$\left(\frac{1}{2}\right)_ф$
2.09	*3.f0*	*0.00*	*0.00*	*0*	Выделитель порядка
2.0a	*2.05*	*2.2e*	*1.0c*	*3*	$(\ln 2)_п$
2.0b	*0.07*	*3.ff*	*3.ff*	*f*	Выделитель мантиссы
2.0c	*2.14*	*0.00*	*0.00*	*1*	$(1)_п$
2.0d	*2.04*	*0.00*	*0.00*	*1*	$\left(\frac{1}{2}\right)_п$
2.16	*0.08*	*0.00*	*0.00*	*1*	$(2^{-1})_ф$
2.18	*2.15*	*2.a0*	*2.79*	*b*	$(\sqrt{2})_п$
2.1a	*0.00*	*0.00*	*0.00*	*5*	$(2^{-32})_ф$; стоп

ГЛАВА VI

СТАНДАРТНЫЕ ПОДПРОГРАММЫ ДЛЯ РЕЖИМА ФИКСИРОВАННОЙ ЗАПЯТОЙ

§ 1. Общие замечания. Стандартные константы

Приведенные здесь подпрограммы составлены для метода плавающих масштабов (см. гл. II, § 6).

При обращении к ним аргумент $x = 2^{p_x}X$ задается парой величин: мантиссой X, представленной в виде числа в системе с фиксированной запятой ($|X| < 1$), и порядка p_x, представленного в виде числа $(p_x \cdot 2^{-30})_ф$. Результат $y = 2^{p_y}Y$ выдается в таком же виде.

Ввиду того, что на машине М-2 нормализацию можно произвести только программным путем с затратой значительного количества тактов, поэтому аргумент x не предполагается обязательно нормализованным, а в тех подпрограммах, где по ряду причин важно иметь x в нормализованном виде, нормализация производится внутри подпрограммы. Результат y в подпрограммах также специально не нормализуется, но, как правило, он получается нормализованным по самому алгоритму вычисления соответствующей функции. Если же его нужно использовать в дальнейших вычислениях обязательно в нормализованном виде, то можно предварительно обратиться к подпрограмме «Нормализация». Однако в большинстве случаев результат выполнения той или иной подпрограммы можно без существенной потери точности использовать в вычислениях в том виде, в каком он выдается подпрограммой.

В приведенных ниже подпрограммах, так же как и для режима плавающей запятой, ячейки *2.20—2.2a* используются в качестве стандартных рабочих ячеек, а ячейки *2.00—2.1f* отведены под стандартные константы, причем среди этих

констант расположены команды *2.10—2.12*, образующие стандартный выход, а также команды *2.0e—2.0f*, осуществляющие связь с подпрограммой «Нормализация», и команда *2.19*, осуществляющая связь с подпрограммой выделения целой и дробной частей числа (с п п «$[x]$ и $\{x\}$»). Ячейки *2.1a—2.1f* константами не заняты. Ниже приводятся стандартные константы вместе с шапкой, необходимой для ввода этих констант.

	0.00	*0.19*	*0.00*	*0*	
	0.02	*3.86*	*3.20*	*9*	
2.00	*0.00*	*0.00*	*0.00*	1	$(+0)_\Phi$
2.01	*3.ff*	*3.ff*	*3.ff*	*f*	$(1-2^{-33})_\Phi$
2.02	*2.00*	*0.00*	*0.00*	*1*	$\left(\frac{1}{2}\right)_\Phi$
2.03	*1.00*	*0.00*	*0.00*	*1*	$\left(\frac{1}{4}\right)_\Phi$
2.04	*0.40*	*0.00*	*0.00*	*1*	$(2^{-4})_\Phi$
2.05	*0.01*	*0.00*	*0.00*	*1*	$(2^{-10})_\Phi$
2.06	*0.00*	*0.01*	*0.00*	*1*	$(2^{-20})_\Phi$
2.07	*0.00*	*0.00*	*0.01*	*1*	$(2^{-30})_\Phi$
2.08	*0.00*	*0.00*	*0.00*	*5*	$(2^{-32})_\Phi$
2.09	*0.00*	*0.00*	*0.00*	*3*	$(2^{-33})_\Phi$
2.0a	*2.80*	*0.00*	*0.00*	*1*	$\left(\frac{10}{16}\right)_\Phi$
2.0b	*3.33*	*0.cc*	*3.33*	*5*	$\left(\frac{8}{10}\right)_\Phi$
2.0c	*0.00*	*0.00*	*3.ff*	*0*	Выделитель *1A*
2.0d	*2.d4*	*0.4f*	*0.cd*	*1*	$\left(\frac{\sqrt{2}}{2}\right)_\Phi$
2.0e	*0.00*	*2.00*	*2.20*	*7*	Связь с п п «Нормализация»
2.0f	*2.10*	*2.21*	*2.18*	*4*	
2.10	*2.12*	*2.12*	*2.0c*	*f*	Стандартный выход
2.11	*2.12*	*2.12*	*2.07*	*a*	
2.12	*0.00*	*0.00*	*0.00*	*0*	
2.13	*3.24*	*0.fd*	*2.a8*	*9*	$\left(\frac{\pi}{4}\right)_\Phi$

2.14	*2.8b*	*3.98*	*0.db*	*b*	$\left(\frac{2}{\pi}\right)_\Phi$
2.15	*0.00*	*0.02*	*3.7e*	*7*	$(0{,}0000027434)_\Phi$
2.16	*0.00*	*0.00*	*0.03*	*1*	$(3\cdot 2^{-30})_\Phi$
2.17	*0.00*	*0.00*	*0.04*	*1*	$(4\cdot 2^{-30})_\Phi$
2.18	*0.00*	*0.00*	*0.20*	*0*	$(-32\cdot 2^{-30})_\Phi$
2.19	*0.00*	*2.23*	*2.02*	*4*	Связь с п/п «$[x]$ и $\{x\}$»

§ 2. Стандартная подпрограмма «Нормализация»

Описание алгоритма

Величина $x=2^{p_x}X$, где $|X|<1$ и $|p_x|<2^{30}$ приводится к виду $x=2^{p'_x}X'$, где X' удовлетворяет неравенству $\frac{1}{2}\leqslant|X'|<1$, если $p'_x>-32$ и $X\neq 0$; в противном случае полагается $X'=0$ и $p'_x=-32$.

При $p'_x\geqslant 32$ происходит останов машины с вызовом на пульт управления величин X' и p'_x.

Подпрограмма производит указанные действия в следующем порядке. Сначала выясняется, не равна ли нулю величина X. При $X=0$ на место p'_x засылается $p'_x=-32$.

В противном случае производятся, если это необходимо, последовательные сдвиги кода величины X на 4 разряда влево до тех пор, пока полученный код, изображающий величину X'', не будет удовлетворять неравенству

$$\frac{1}{16}\leqslant|X''|<1;$$

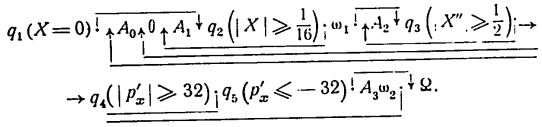
при каждом таком сдвиге величину p_x уменьшаем на 4 единицы (первый цикл).

По окончании этого цикла производятся последовательные сдвиги кода величины X'' на один разряд влево до тех пор, пока полученный код, изображающий величину X', не будет удовлетворять неравенству $\frac{1}{2}\leqslant|X'|<1$; при каждом таком сдвиге величину p_x уменьшаем на единицу (второй цикл). В результате этого получим величину p'_x.

Далее анализируется полученная величина p'_x: при $p'_x \leqslant -32$ полагается $p'_x = -32$ и $X' = 0$; при $p'_x \geqslant 32$ происходит останов машины.

Осуществление сдвигов на 4 разряда в первом цикле и на один разряд во втором цикле при общем сдвиге на небольшое число разрядов (до 10—12 разрядов) в среднем занимает меньшее число тактов, чем при других соотношениях количества сдвигов в первом и втором циклах: такой случай наиболее часто встречается при нормализации.

Логическая схема



Операторы $q_1(X=0)$ и A_0 осуществляются командами, помещенными среди стандартных констант, причем оператор A_0 производит засылку числа $(-32 \cdot 2^{-30})_\Phi$ в ячейку для порядка p'_x. Оператор A_1 осуществляет сдвиг на четыре разряда кода величины X и уменьшение величины p_x на четыре единицы. Оператор A_2 осуществляет сдвиг на один разряд кода величины X'' и уменьшение величины p_x на единицу. Оператор A_3 производит засылку нуля в ячейку для X'. Операторы A_1 и q_2 образуют первый цикл, а операторы A_2 и q_3 образуют второй цикл.

Краткая характеристика программы

Количество занятых ячеек 11.

Число тактов: максимальное 38, минимальное 5, в случае сдвига на 7 разрядов 20. Точность не теряется.

Используемые стандартные константы: *2.00*, *2.02*, *2.04*, *2.07*, *2.0c*, *2.0e*, *2.0f*, *2.10*, *2.11*, *2.12*, *2.17*, *2.18*.

Используемые стандартные рабочие ячейки: *2.20*, *2.21*.

Аргумент: $X \Rightarrow 2.20$; $p_x \Rightarrow 2.21$.

Результат: $X' \Rightarrow 2.20$; $p'_x \Rightarrow 2.21$.

Ячейка обратной связи *2.12*.

Дополнительные сведения: для использования данной подпрограммы необходимо, чтобы в ячейке *2.0e* хранилась команда (*2.0e*): $N+2$ *2.00 2.20 7*, где N — адрес первой команды данной подпрограммы.

Обращение к подпрограмме производится командой вида (*n*): *2.0e 2.12 n 4*.

Программа

	0.0b	*0.0a*	*0.00*	*0*	
	0.00	*1.cf*	*1.fc*	*0*	
3.00	*2.20*	*2.20*	*2.04*	*8*	A_1
3.01	*2.21*	*2.21*	*2.17*	*a*	
3.02	*3.00*	*2.20*	*2.04*	*7*	q_2
3.03	*0.00*	*0.00*	*3.06*	*0*	ω_1
3.04	*2.20*	*2.20*	*2.02*	*8*	A_2
3.05	*2.21*	*2.21*	*2.07*	*a*	
3.06	*3.04*	*2.20*	*2.02*	*7*	q_3
3.07	*2.10*	*2.21*	*2.18*	*7*	q_4, 0
3.08	*3.0a*	*2.18*	*2.21*	*6*	q_5
3.09	*2.0f*	*2.20*	*2.00*	*4*	A_3, ω_2
3.0a	*0.00*	*2.21*	*2.20*	*5*	Ω

Стандартные константы, используемые в подпрограмме:

2.00	*0.00*	*0.00*	*0.00*	*1*	$(+0)_\Phi$
2.02	*2.00*	*0.00*	*0.00*	*1*	$\left(\frac{1}{2}\right)_\Phi$
2.04	*0.40*	*0.00*	*0.00*	*1*	$(2^{-4})_\Phi$
2.07	*0.00*	*0.00*	*0.01*	*1*	$(2^{-30})_\Phi$
2.0e	*0.00*	*2.00*	*2.20*	*7*	Ячейки связи с п/п
2.0f	*2.10*	*2.21*	*2.18*	*4*	«Нормализация»
2.17	*0.00*	*0.00*	*0.04*	*1*	$(4 \cdot 2^{-30})_\Phi$
2.18	*0.00*	*0.00*	*0.20*	*0*	$(-32 \cdot 2^{-30})_\Phi$

§ 3. Стандартная подпрограмма выделения целой и дробной частей числа

Описание алгоритма и логическая схема подпрограммы

Подпрограмма вычисляет целую и дробную части положительного числа x согласно их обычному математическому определению и целую и дробную части отрицательного числа согласно формул:

$$\{x\} = -\{|x|\}, \quad [x] = x - \{x\}.$$

Дробная часть представляется числом в системе фиксированной запятой; целая часть представляется в виде условного числа, а именно:

$$[x]_{усл} = \left([x] \cdot 2^{-33}\right)_{\phi}.$$

При $p_x \geqslant 32$ производится останов по переполнению.

Логическая схема программы имеет следующий вид:

$$A_1 q\,(p_x \geqslant 0)\,\overline{!\,A_2 0\,\downarrow}\,A_3 0.$$

Оператор A_1 производит переключение на режим фиксированной запятой с двойной точностью и вычисляет величину $2^{-|p_x|-1}$ следующим способом. Сначала на место $2^{-|p_x|-1}$ засылается $^1{}_2$, затем выделяется младший разряд порядка, и если он равен единице, то содержимое ячейки для $2^{-|p_x|-1}$ умножается на $^1{}_2$; далее выделяется следующий разряд, и если он равен единице, то содержимое ячейки для $2^{-|p_x|-1}$ умножается на квадрат предыдущего множителя, и т. д. до тех пор, пока не будут выделены все разряды порядка, отличные от нуля.

Логическое условие $q\,(p_x \geqslant 0)$ в случае $p_x < 0$ передает управление оператору A_3, который получает

$$[x]_{усл} = 0 \quad \text{и} \quad \{x\} = X \cdot \left(2^{p}\right)_{\phi}.$$

Оператор A_2 в случае $p_x \geqslant 0$ сначала получает величину $\left(2^{-33+p}\right)_{\phi}$, а затем $[x]_{усл}$ и $\{x\}$ умножением с двойной точностью величины X на $\left(2^{-33 \cdot p}\right)_{\phi}$.

Краткая характеристика программы

Количество занятых ячеек 16.

Число тактов: максимальное 35, минимальное 14, в среднем (в случае $p_x = 10$) 28 тактов.

Для $p_x \geqslant 0$ точность сохраняется; для $p_x < 0$ абсолютная погрешность равна 2^{-33}.

Используемые стандартные константы: *2.00*, *2.02*, *2.04*, *2.07*, *2.09*, *2.0c*, *2.10—2.12*, *2.19*.

Используемые стандартные рабочие ячейки *2.20—2.28*.

Аргумент: $X \Rightarrow$ *2.20*; $p_x \Rightarrow$ *2.21*.

Функция: $\{x\} \Rightarrow$ *2.25*; $[x]_{усл} \Rightarrow$ *2.26*.

Ячейка обратной связи *2.12*.

Дополнительные сведения: при использовании данной подпрограммы необходимо, чтобы в ячейке связи *2.19* хранилась команда

(*2.19*): *N 2.23 2.02 4*,

где N — адрес первой команды данной подпрограммы;

обращение к подпрограмме производится командой вида (*n*): *2.19 2.12 n 4*.

Программа

При выполнении команды *2.19* в ячейку *2.23* засылается $\left(\frac{1}{2}\right)_{\phi}$.

	0.10	*0.0f*	*0.00*	*0*		
	0.01	*2.2b*	*0.85*	*9*		
3.00	*0.00*	*0.0c*	*3.01*	*0*	Режим ФД	A_1
3.01	*2.25*	*2.21*	*2.04*	*9*	$\left[\frac{p_x}{2}\right] \cdot 2^{-33} \Rightarrow$ *2.26*, $\left\{\frac{p_x}{2}\right\} \Rightarrow$ *2.25*	
3.02	*3.05*	*2.28*	*2.02*	*4*	$\left(\frac{1}{2}\right)_{\phi} \Rightarrow$ *2.28*	
3.03	*2.25*	*2.26*	*2.02*	*9*	Цикл, вычисляющий величину $2^{-\lvert p_x \rvert - 1}$	
3.04	*2.27*	*2.28*	*2.28*	*9*		
3.05	*3.07*	*2.25*	*2.02*	*7*		
3.06	*2.22*	*2.23*	*2.28*	*9*		
3.07	*3.03*	*2.00*	*2.26*	*7*		

3.08	*3.0d*	*2.21*	*2.00*	*6*	$q(p_x \neq 0)$	q
3.09	*2.27*	*2.09*	*2.23*	*8*	$[x] \Rightarrow 2.26$	A_2
3.0a	*2.27*	*2.27*	*2.02*	*9*	$[x] \Rightarrow 2.25$	
3.0b	*2.25*	*2.20*	*2.28*	*9*	$\{x\} \Rightarrow 2.25$	
3.0c	*0.00*	*0.10*	*2.10*	*0*	Режим Ф.З, 0	0
3.0d	*2.28*	*2.23*	*2.23*	*b*	$\{x\} = X \cdot 2^{p_x} \Rightarrow 2.25$	A_3
3.0e	*2.24*	*2.20*	*2.28*	*9*		
3.0f	*0.00*	*0.10*	*2.10*	*0*	Режим Ф З, 0	0

Стандартные константы, используемые в подпрограмме:

2.00 *0.00* *0.00* *0.00* *1* $(+0)_ф$

2.02 *2.00* *0.00* *0.00* *1* $\left(\frac{1}{2}\right)_ф$

2.04 *0.40* *0.00* *0.00* *1* $(2^{-4})_ф$

2.09 *0.00* *0.00* *0.00* *3* $(2^{-33})_ф$

2.19 ячейка связи с п п «$[x]$ и $\{x\}$»

Пояснения к программе

Команды *3.03*—*3.07* образуют цикл, вычисляющий значение $2^{-|p_x|-1} = 2^{-\sum_{i=0}^{k} 2^i \varepsilon_i - 1}$. Перед работой этого цикла командой *2.19* в ячейку *2.23* засылается $\frac{1}{2}$, соответствующая значению $2^{-\sum_{i=0}^{k} 2^i \varepsilon_i - 1}$ при $k=0$ ($p_x=0$); командой *3.00* устанавливается режим фиксированной запятой с двоичной точностью; командой *3.01* производится умножение величины $p_x \cdot 2^{-30}$ на 2^{-4}, в результате чего в ячейке *2.26* получается величина $\left[\frac{p_x}{2}\right] \cdot 2^{-33}$, а в ячейке *2.25* — выделенный младший разряд величины p_x, т. е. $\left\{\frac{p_x}{2}\right\} = \frac{\varepsilon_0}{2}$; командой *3.02* в ячейку *2.28* засылается $\frac{1}{2}$, соответствующая значению величины 2^{-2^k} при $k=0$.

После этого с команды *3.05* начинается работа цикла. По команде *3.05* производится сравнение выделенного разряда порядка ε_k (первый раз ε_0) с единицей (в действительности сравнение производится с $\frac{1}{2}$, так как выделяется $\frac{\varepsilon_k}{2}$). Если $\varepsilon_k = 1$, то выполняется команда *3.06*, производящая умножение предыдущего значения величины $2^{-\sum_{i=0}^{k} 2^i \varepsilon_i - 1}$ на величину 2^{-2^k}, тем самым получается новое значение величины $2^{-\sum_{i=0}^{k} 2^i \varepsilon_i - 1}$. Если $\varepsilon_k = 0$, а также после выполнения команды *3.06*, выполняется команда *3.07*, которая производит сравнение с нулем величины $\left[\frac{p_x}{2^{k+1}}\right] \cdot 2^{-33}$ (первый раз величины $\left[\frac{p_x}{2}\right] \cdot 2^{-33}$). Если эта величина равна нулю, то работа цикла прекращается и выполняется команда *3.08*; если же она отлична от нуля, то с команды *3.03* начинается очередное повторение цикла. Команда *3.03* производит умножение величины $\left[\frac{p_x}{2^{k+1}}\right] \cdot 2^{-33}$ на $\frac{1}{2}$, т. е. производит выделение очередной цифры порядка $\varepsilon_k \left(\frac{\varepsilon_k}{2} = \left\{\frac{p_x}{2^{k+1}}\right\}\right)$, и одновременно в ячейке *2.26* получает новое значение величины $\left[\frac{p_x}{2^{k+1}}\right] \cdot 2^{-33}$.

Команда *3.04* вычисляет новое значение величины 2^{-2^k} возведением в квадрат предыдущего значения этой величины и т. д.

После окончания цикла команда *3.08* выясняет знак порядка p_x.

В случае $p_x \geqslant 0$, выполняются команды *3.09* и *3.0a*, которые вычисляют величину $2^{-(33-p_x)}$, после чего команда *3.0b* умножает на эту величину мантиссу X. В результате этого в старших разрядах произведения (в ячейке *2.26*) окажется величина $[x] \cdot 2^{-33}$, а в младших разрядах произведения

(в ячейке *2.25*) окажется величина $\{x\}$. После этого производится обратное переключение на режим фиксированной запятой и передача управления стандартному выходу (команда *3.0c*). При $p_x \geqslant 32$ произойдет останов по переполнению на команде *3.09*.

В случае $p_x < 0$ выполняется команда *3.0d*, получающая величину 2^{p_x}, а затем — команда *3.0e*, производящая умножение мантиссы X на полученную величину 2^{p_x}. В результате этого в ячейке *2.25* (в старших разрядах произведения) окажется величина $\{x\}$. В ячейку *2.26* при выполнении команды *3.03* засылается нуль. Это соответствует $[x] = 0$ при $p_x < 0$.

Команда *3.0f* устанавливает режим фиксированной запятой и передает управление стандартному выходу.

§ 4. Стандартная подпрограмма для вычисления $\sqrt{x}$

Описание алгоритма

Аргумент x сначала нормализуется, так что либо $X = 0$, либо $\frac{1}{2} \leqslant |X| < 1$.

Если $X = 0$, то мантисса $\sqrt{x}$ равна нулю, порядок $\sqrt{x}$ равен -32; при $X \neq 0$ величина $\sqrt{x}$ находится по формулам:

$$\sqrt{x} = \sqrt{X \cdot 2^{p_x}} = \begin{cases} \sqrt{X} \cdot 2^n, & \text{если } p_x = 2n, \\ \sqrt{X} \cdot \frac{1}{\sqrt{2}} \cdot 2^{n+1}, & \text{если } p_x = 2n+1. \end{cases}$$

Квадратный корень из мантиссы находится последовательными приближениями по итерационной формуле

$$Y_{n+1} = \frac{1}{2}\left(\frac{X}{Y_n} + Y_n\right).$$

Начальное приближение Y_0 берется следующее:

$$Y_0 = \frac{1}{2}X + \frac{1}{2}.$$

Процесс последовательных приближений ведется до тех пор, пока не будет выполняться неравенство

$$|\Delta| = \left|\frac{1}{2}\left(\frac{X}{Y_n} - Y_n\right)\right| \leqslant 3 \cdot 2^{-30},$$

что обеспечивает точность вычислений $Y_{n+1} \approx \sqrt{X}$ до 2^{-33}.

Логическая схема

$$Kp_1(|X| > 0)\downarrow A_1 p_2 \downarrow A_2 \uparrow A_3 \downarrow \theta.$$

Обобщенный оператор K производит нормализацию аргумента x.

Оператор A_1 вычисляет величину $\sqrt{X}$.

Логическое условие p_2 в случае $\left\{\frac{p_x}{2}\right\} = 0$ передает управление оператору A_3.

Оператор A_2 в случае $\left\{\frac{p_x}{2}\right\} \neq 0$ домножает $\sqrt{X}$ на $\frac{1}{\sqrt{2}}$ и добавляет единицу к порядку величины x.

Оператор A_3 вычисляет порядок величины $\sqrt{x}$, равный $\left[\frac{p_x+1}{2}\right]$

Мантисса величины $\sqrt{x}$ в результате вычислений удовлетворяет неравенству

$$\frac{1}{2} \leqslant |X| < 1.$$

При $x < 0$ произойдет переполнение.

Краткая характеристика программы

Количество занятых ячеек 15.

Число тактов: максимальное 71 (из них 38 на нормализацию), минимальное (при $x = 0$) 11 (из них 5 на нормализацию), в среднем (если при нормализации происходит сдвиг на 7 разрядов и для вычисления $\sqrt{X}$ требуется две итерации) 41 — из них 20 на нормализацию.

Относительная погрешность равна 2^{-33}.

Используемые стандартные константы: *2.00*, *2.02*, *2.04*, *2.07*, *2.09*, *2.0c*—*2.12*, *2.15*, *2.17*, *2.18*.

Используемые стандартные рабочие ячейки: *2.20*—*2.24*, *2.2a*.

Аргумент: $X \Rightarrow 2.20$, $p_x \Rightarrow 2.21$.

Функция: мантисса $\sqrt{x} \Rightarrow 2.20$, порядок $\sqrt{x} \Rightarrow 2.21$.

Ячейка обратной связи *2.2a*.

Дополнительные сведения: требует наличия в памяти подпрограммы «Нормализация».

Программа

	0.0f	*0.0e*	*0.00*	*0*		
	0.01	*3.ce*	*2.4d*	*9*		
3.00	*2.0e*	*2.12*	*3.00*	*4*	Нормализация x	K
3.01	*3.0e*	*2.20*	*2.09*	*7*	$p_1(\lvert X\rvert \geqslant 2^{-33})$	p_1
3.02	*2.22*	*2.20*	*2.02*	*9*	$\frac{1}{2}\,X \Rightarrow 2.22$	A_1
3.03	*2.20*	*2.22*	*2.02*	*b*	$Y_0 \Rightarrow 2.20$	
3.04	*2.23*	*2.22*	*2.20*	*8*	$\frac{1}{2}\,X/Y_n \Rightarrow 2.23$	
3.05	*2.20*	*2.20*	*2.02*	*9*	$\frac{1}{2}\,Y_n \Rightarrow 2.20$	
3.06	*2.24*	*2.23*	*2.20*	*a*	$\Delta \Rightarrow 2.24$	
3.07	*2.20*	*2.23*	*2.20*	*b*	$Y_{n+1} \Rightarrow 2.20$	
3.08	*3.04*	*2.15*	*2.24*	*7*	$p(\lvert\Delta\rvert \leqslant 3\cdot 2^{-20})$	
3.09	*2.22*	*2.21*	*2.07*	*f*	$p_2\left(\left\lvert\left\{\frac{p_x}{2}\right\}\right\rvert > 0\right)$	p_2
3.0a	*3.0d*	*2.22*	*2.07*	*7*		
3.0b	*2.20*	*2.20*	*2.0d*	*9*	$\sqrt{X}\,\frac{1}{\sqrt{2}} \Rightarrow 2.20$	A_2
3.0c	*2.21*	*2.21*	*2.07*	*b*	$p_x + 1 \Rightarrow 2.21$	
3.0d	*2.21*	*2.21*	*2.02*	*9*	$\left[\frac{p_x+1}{2}\right] \Rightarrow 2.21$	A_3
3.0e	*2.10*	*2.12*	*2.2a*	*4*	$(2.2a) \Rightarrow 2.12;\ 0$	0

Стандартные константы, используемые в программе:

2.02	*2.00*	*0.00*	*0.00*	*1*	$\left(\frac{1}{2}\right)_\phi$
2.07	*0.00*	*0.00*	*0.01*	*1*	$(2^{-30})_\phi$
2.09	*0.00*	*0.00*	*0.00*	*3*	$(2^{-33})_\phi$
2.0d	*2.d4*	*0.4f*	*0.cd*	*1*	$\left(\frac{1}{\sqrt{2}}\right)_\phi$
2.15	*0.00*	*0.02*	*3.7e*	*7*	Константа для сравнения

§ 5. Стандартная подпрограмма для вычисления $\sin x$ и $\cos x$

Описание алгоритма *)

Аргумент x представляется в виде

$$x = \operatorname{sign} x \cdot |x| = \operatorname{sign} x \cdot \left(2k\pi + \alpha_1\pi + \beta_1\frac{\pi}{2} + t_1\right),$$

где $\alpha_1 = 0;\ 1,\ \beta_1 = 0;\ 1,\ 0 \leqslant t_1 < \frac{\pi}{2}$.

Тогда

$$\sin x = \operatorname{sign} x \cdot \sin|x| = \operatorname{sign} x \cdot [(-1)^{\alpha_1}\beta_1\cos t_1 + (-1)^{\alpha_1}(1-\beta_1)\sin t_1],$$
$$\cos x = (-1)^{\alpha_1}(1-\beta_1)\cos t_1 - (-1)^{\alpha_1}\beta_1\sin t_1.$$

Если положить $\beta_2 = 1-\beta_1$, $t_2 = \frac{\pi}{2} - t_1$, $\alpha_2 = 0$ при $\alpha_1 = \beta_1$ и $\alpha_2 = 1$ при $\alpha_1 \neq \beta_1$, то будем иметь:

$$\sin x = \operatorname{sign} x \cdot [(-1)^{\alpha_1}\beta_1\cos t_1 + (-1)^{\alpha_2}\beta_2\cos t_2],$$
$$\cos x = (-1)^{\alpha_1}(1-\beta_1)\cos t_1 + (-1)^{\alpha_2}(1-\beta_2)\cos t_2.$$

Порядки значений $\sin x$ и $\cos x$ принимаем постоянными и равными единице, а их мантиссы следующими:

$$\text{мантисса } \sin x = \frac{1}{2}\sin x,$$
$$\text{мантисса } \cos x = \frac{1}{2}\cos x.$$

Для упрощения подпрограммы и для удобства пользования ею результат не нормализуется, что является целесообразным в большинстве случаев. Если в какой-либо конкретной задаче необходимо произвести нормализацию, то это можно сделать в основной программе.

*) Алгоритм, используемый в данной подпрограмме, мало отличается от алгоритма для аналогичной подпрограммы в режиме плавающей запятой (гл. V, § 5), где дано подробное объяснение алгоритма и программы. Разница в алгоритме заключается только в том, что здесь косинус приведенного угла вычисляется через его синус и введен масштабный множитель $\frac{1}{2}$.

По данной подпрограмме вычисляются только мантиссы, для чего нужно сначала получить $\frac{1}{2}\cos t_i$ ($i=1; 2$).

Вычисление $\frac{1}{2}\cos t_i$ производится по формуле

$$\frac{1}{2}\cos t_i = \frac{1}{2} - \sin^2\frac{t_i}{2},$$

а $\sin\frac{t_i}{2}$ вычисляется с помощью полинома 9-й степени, полученного из ряда Тейлора для $\sin t$ с внесением поправки в последний коэффициент для учета остаточного члена, а именно:

$$\sin t \approx t - \frac{1}{3!}t^3 + \frac{1}{5!}t^5 - \frac{1}{7!}t^7 + c_4 t^9,$$

где $c_4 = 0{,}0000027434$.

Данный полином обеспечивает вычисление $\sin t$ для $t \in \left[0, \frac{\pi}{4}\right]$ с абсолютной погрешностью, не превышающей 2^{-33}. В программе полином вычисляется по схеме Горнера.

Логическая схема

$$\Phi K \downarrow \overline{A_1 p \uparrow A_2} \uparrow 0.$$

Оператор Φ формирует в операторе A_1 команду для учета знака x при вычислении $\frac{1}{2}\sin x$.

Обобщенный оператор K с помощью подпрограммы выделения целой и дробной частей числа вычисляет величины

$$\left[|x|\cdot\frac{2}{\pi}\right] \quad \text{и} \quad \left\{|x|\cdot\frac{2}{\pi}\right\}\frac{\pi}{4} = \frac{t_1}{2}.$$

Оператор A_1 по указанным в алгоритме формулам вычисляет $\frac{1}{2}\cos t_i$, умножает $\frac{1}{2}\cos t_i$ на $(-1)^{\alpha_i}$ и помещает в ячейку, где получается $\frac{1}{2}\sin x$, величину

$$\operatorname{sign} x \cdot \frac{1}{2}(-1)^{\alpha_i}\beta_i \cos t_i, \quad \text{если} \quad \beta_i = 1,$$

а в ячейку, где получается $\frac{1}{2}\cos x$, — величину

$$\frac{1}{2}(-1)^{\alpha_i}(1-\beta_i)\cos t_i, \quad \text{если} \quad \beta_i = 0.$$

Оператор p является логическим условием, осуществляющим передачу управления стандартному выходу, если выход из программы уже подготовлен. В противном случае управление передается оператору A_2.

Оператор A_2 вычисляет $t_2 = 1 - t_1$, β_2 и α_2 подготавливает выход из программы и передает управление оператору A_1.

Краткая характеристика программы

Количество занятых ячеек 31.

Число тактов: максимальное 84 (из них 35 на вычисление $[x]$ и $\{x\}$); минимальное 61 (из них 14 на вычисление $[x]$ и $\{x\}$); в среднем 68 (для $x = X \cdot 2^3$).

Абсолютная погрешность равна 2^{-32}.

Используемые стандартные константы: *2.00, 2.02, 2.04, 2.08, 2.09, 2.0c, 2.10—2.15, 2.19.*

Используемые стандартные рабочие ячейки: *2.20—2.28, — 2.2a.*

Аргумент: $X \Rightarrow$ *2.20*; $p_x \Rightarrow$ *2.21.*

Функции: $\frac{1}{2}\sin(X \cdot 2p_x) \Rightarrow$ *2.20*, $\frac{1}{2}\cos(X \cdot 2p_x) \Rightarrow$ *2.21.*

Ячейка обратной связи *2.2a.*

Дополнительные сведения: требует наличия в памяти подпрограммы выделения целой и дробной частей числа.

Программа

	0.1c	*0.1e*	*0.00*	*0*		
	0.03	*0.77*	*3.6c*	*9*		
3.00	*3.17*	*3.17*	*2.20*	*e*	$\lvert(3.17)\rvert \cdot \operatorname{sign} x \Rightarrow$ *3.17*	Φ
3.01	*2.20*	*2.20*	*2.14*	*9*	$x \cdot \frac{2}{\pi} \Rightarrow$ *2.20*	
3.02	*2.20*	*2.20*	*2.02*	*e*	$x_1 = \lvert x\rvert \cdot \frac{2}{\pi} \Rightarrow$ *2.20*	K
3.03	*2.19*	*2.12*	*3.03*	*4*		
3.04	*2.25*	*2.25*	*2.13*	*9*	$\frac{t_1}{2} = \left\{\lvert x\rvert \cdot \frac{2}{\pi}\right\} \cdot \frac{\pi}{4} \Rightarrow$ *2.25*	

3.05	2.22	2.25	2.25	9	Вычисление аппроксимирующего полинома по схеме Горнера: $\sin\frac{t_i}{2}\Rightarrow 2.24$	
3.06	2.24	2.15	2.22	9		
3.07	2.24	2.24	3.1c	b		
3.08	2.24	2.24	2.22	9		
3.09	2.24	2.24	3.1d	b		
3.0a	2.24	2.24	2.22	9		
3.0b	2.24	2.24	3.1e	b		
3.0c	2.24	2.24	2.22	9		
3.0d	2.24	2.24	2.25	9		
3.0e	2.24	2.24	2.25	b		
3.0f	2.24	2.24	2.24	9	$\frac{1}{2}\cos t_i\Rightarrow 2.24$	A_1
3.10	2.24	2.02	2.24	a		
3.11	2.27	2.26	2.08	f	$\alpha_i\Rightarrow 2.27$	
3.12	3.14	2.27	2.08	7	$\lvert(2.27)\rvert\geqslant\lvert 2^{-32}\rvert$	
3.13	2.24	2.00	2.24	a	$-\frac{1}{2}\cos t_i\Rightarrow 2.24$	
3.14	2.27	2.26	2.09	f	$\beta_i\Rightarrow 2.27$	
3.15	3.17	2.00	2.27	7	$\lvert 0\rvert\geqslant\lvert(2.27)\rvert$	
3.16	3.18	2.21	2.24	4	$\frac{1}{2}\cos x\Rightarrow 2.21$	
3.17*	2.20	2.00	2.24	b*	$\frac{1}{2}\sin x\Rightarrow 2.20$	
3.18	2.10	2.15	2.12	7	$p(\lvert(2.12)\rvert<3.2^{-20})$	p_1,θ
3.19	2.26	2.26	2.09	b	$[x_1]\cdot 2^{-33}+2^{-33}\Rightarrow 2.26$	A_2
3.1a	2.25	2.13	2.25	a	$\frac{1}{2}t_2=\frac{\pi}{4}-\frac{1}{2}t_1\Rightarrow 2.25$	
3.1b	3.05	2.12	2.2a	4		
3.1c	0.00	0.d0	0.34	0	$\left(-\frac{1}{7!}\right)_\phi$	Константы
3.1d	0.08	2.22	0.88	9	$\left(\frac{1}{5!}\right)_\phi$	
3.1e	0.aa	2.aa	2.aa	a	$\left(-\frac{1}{3!}\right)_\phi$	

Стандартные константы, используемые в программе:

2.00	0.00	0.00	0.00	1	$(+0)_\phi$
2.02	2.00	0.00	0.00	1	$\left(\frac{1}{2}\right)_\phi$
2.08	0.00	0.00	0.00	5	$(2^{-82})_\phi$
2.09	0.00	0.00	0.00	3	$(2^{-33})_\phi$
2.13	3.24	0.fd	2.a8	9	$\left(\frac{\pi}{4}\right)_\phi$
2.14	2.8b	3.98	0.db	b	$\left(\frac{2}{\pi}\right)_\phi$
2.15	0.00	0.02	3.7e	7	$(0{,}000\,002\,743\,4)_\phi$

§ 6. Стандартная подпрограмма для вычисления $\ln x$

Описание алгоритма

Функцию $\ln x$ можно представить следующим образом:

$$\ln x=\ln(2^{p_1}X_1)=p_1\ln 2+\ln X_1=$$
$$=\left(p_1-\frac{1}{2}\right)\ln 2+\ln(\sqrt{2}\cdot X_1).$$

Величина $\ln(\sqrt{2}\cdot X_1)$ вычисляется с помощью полинома седьмой степени:

$$\ln(\sqrt{2}\cdot X_1)\approx u+c_1u^3+c_2u^5+c_3u^7,$$

где

$$u=2\frac{X_1-\sqrt{\frac{1}{2}}}{X_1+\sqrt{\frac{1}{2}}}.$$

Этот полином получается из разложения $\ln(\sqrt{2}\cdot X_1)$ в ряд

$$\ln(\sqrt{2}\cdot X_1)=2\sum_{k=0}^{\infty}\frac{1}{2k+1}\left(\frac{\sqrt{2}X_1-1}{\sqrt{2}X_1+1}\right)^{2k+1},$$

если ограничиться $k=4$ и степень многочлена $P_9(u)$ понизить на единицу с помощью полиномов Чебышева.

Если значение $\ln(\sqrt{2}\cdot X_1)$ вычислено, то

$$\ln x = 2^{p_2}\cdot X_2 = 2^{p_2}[X_1' + 2^{-p_2}\cdot \ln(\sqrt{2}\cdot X_1)],$$

где X_1' — мантисса величины $\left(p_1 - \frac{1}{2}\right)\ln 2$, а p_2 — ее порядок.

Для всех значений $x > 0$, кроме интервала

$$\frac{1}{2} \leqslant x < 2,$$

величина X_2 удовлетворяет неравенству

$$\frac{1}{2}\ln 2 \leqslant |X_2| \leqslant \ln 2.$$

При x, принадлежащем интервалу

$$\frac{1}{2} \leqslant x < 2,$$

величина X_2 равна $\frac{1}{2}\ln x$, т. е. результат получается с постоянным масштабом.

При $x \leqslant 0$ производится останов машины по переполнению.

Логическая схема

$$KA_1A_2\omega\downarrow\overline{A_3\uparrow A_4q}\downarrow A_5\emptyset.$$

Оператор K нормализует аргумент x.

Оператор A_1 вычисляет величину $\ln(\sqrt{2}\cdot X_1)$.

Оператор A_2 получает $\left(p_1 - \frac{1}{2}\right)\cdot 2^{-30}$ и подготавливает для работы операторы A_3 и A_4.

Оператор A_3 накапливает p_2 (порядок величины $\ln x$) и величину $2^{-p_2}\cdot\ln(\sqrt{2}\cdot X_1)$.

Оператор A_4 получает величину 2^{-30+p_2}, необходимую для нахождения X_1'.

Оператор q проверяет выполнение неравенства

$$|2^{-30+p_2}| \geqslant \left|\left(p_1 - \frac{1}{2}\right)\cdot 2^{-30}\right|.$$

Оператор A_5 вычисляет величину X_2.

Краткая характеристика программы

Количество занятых ячеек 28.

Количество тактов: максимальное 83 (из них 38 на нормализацию); минимальное 33 (из них 8 на нормализацию; в среднем 61 (если $p_1 = 10$ и нормализация происходит на 7 разрядов).

Абсолютная погрешность X_2 равна 2^{-31}.

Используемые стандартные константы: *2.00, 2.02, 2.04, 2.07, 2.0c —2.12, 2.17, 2.18.*

Используемые стандартные рабочие ячейки: *2.20—2.23, 2.2a.*

Ячейка обратной связи *2.2a.*

Аргумент $x = 2^{p_1}X_1$: $X_1 \Rightarrow$ *2.20*, $p_1 \Rightarrow$ *2.21.*

Функция $\ln x = 2^{p_2}X_2$: $X_2 \Rightarrow$ *2.20*, $p_2 \Rightarrow$ *2.21.*

Дополнительные сведения: требуется наличие подпрограммы «Нормализация».

Программа

	0.16	*0.1b*	*0.00*	*0*		
	0.03	*3.ee*	*2.b7*	*3*		
3.00	*2.0e*	*2.12*	*3.00*	*4*	Нормализация x	K
3.01	*2.22*	*2.20*	*2.0d*	*a*	Вычисление u	
3.02	*2.23*	*2.22*	*2.02*	*9*		
3.03	*2.23*	*2.23*	*2.0d*	*b*		
3.04	*2.22*	*2.22*	*2.23*	*8*		
3.05	*2.23*	*2.22*	*2.22*	*9*	Вычисление многочлена	A_1
3.06	*2.20*	*2.23*	*3.1a*	*9*		
3.07	*2.20*	*2.20*	*3.19*	*b*		
3.08	*2.20*	*2.20*	*2.23*	*9*		
3.09	*2.20*	*2.20*	*3.18*	*b*		
3.0a	*2.20*	*2.20*	*2.23*	*9*		
3.0b	*2.20*	*2.20*	*2.22*	*9*		
3.0c	*2.20*	*2.20*	*2.22*	*b*		

3.0d	*2.22*	*2.07*	*2.02*	*9*	$(2^{-31})_\Phi \Rightarrow 2.22$	A_2
3.0e	*2.23*	*2.21*	*2.22*	*a*	$\left(p_1 - \frac{1}{2}\right) \cdot 2^{-30} \Rightarrow 2.23$	
3.0f	*3.12*	*2.21*	*2.00*	*4*	$0 \Rightarrow 2.21$; ω	ω
3.10	*2.21*	*2.21*	*2.07*	*b*		A_3
3.11	*2.20*	*2.20*	*2.02*	*9*		
3.12	*2.22*	*2.22*	*2.22*	*b*		A_4
3.13	*3.10*	*2.22*	*2.23*	*7*		q
3.14	*2.23*	*2.23*	*2.22*	*8*	Вычисление X_2	A_5
3.15	*2.23*	*2.23*	*3.1b*	*9*		
3.16	*2.20*	*2.20*	*2.23*	*b*		
3.17	*2.10*	*2.12*	*2.2a*	*4*	$(2.2a) \Rightarrow 2.12$	0
3.18	*0.55*	*1.55*	*1.aa*	*b*	$c_1 = (0{,}083\,333\,412\,8)_\Phi$	
3.19	*0.0c*	*3.2d*	*2.2e*	*f*	$c_2 = (0{,}012\,494\,607\,7)_\Phi$	
3.1a	*0.02*	*1.84*	*2.70*	*7*	$c_3 = (0{,}002\,323\,732\,1)_\Phi$	
3.1b	*2.c5*	*3.21*	*1.fe*	*1*	$(\ln 2)_\Phi$	

Стандартные константы, используемые в программе:

2.00	*0.00*	*0.00*	*0.00*	*1*	$(+0)_\Phi$
2.02	*2.00*	*0.00*	*0.00*	*1*	$\left(\frac{1}{2}\right)_\Phi$
2.07	*0.00*	*0.00*	*0.01*	*1*	$(2^{-30})_\Phi$
2.0d	*2.d4*	*0.4f*	*0.cd*	*1*	$\left(\frac{\sqrt{2}}{2}\right)_\Phi$

§ 7. Стандартная подпрограмма для вычисления e^x

Описание алгоритма

Вычисление e^x основано на следующих формулах *):

$$e^x = 2^{p'} (e^z)^2,$$

*) Элементарные преобразования, приводящие к данным формулам, аналогичны преобразованиям гл. V, § 6.

где

$$z = \frac{\ln 2}{2}\left\{\frac{x}{\ln 2}\right\} \quad \text{и} \quad p' = \left[\frac{x}{\ln 2}\right] \quad \text{при} \quad \left\{\frac{x}{\ln 2}\right\} \leqslant 0$$

или

$$z = \frac{\ln 2}{2}\left(\left\{\frac{x}{\ln 2}\right\} - 1\right) \quad \text{и} \quad p' = \left[\frac{x}{\ln 2}\right] + 1 \quad \text{при} \quad \left\{\frac{x}{\ln 2}\right\} \geqslant 0.$$

Здесь функции $[x]$ и $\{x\}$ понимаются так же, как и в подпрограмме «Выделение $[x]$ и $\{x\}$».

Функция e^z вычисляется с помощью степенного ряда

$$e^z = \sum_{k=0}^{\infty} \frac{z^k}{k!}.$$

Для каждого z берется столько членов ряда, чтобы последний используемый член ряда удовлетворял условию

$$\left|\frac{z^k}{k!}\right| \leqslant 2^{-28}.$$

Это обеспечивает точность вычислений до 2^{-28}.

Логическая схема

$$K3 \downarrow A_1 q\left(\left|\frac{z^k}{k!}\right| \leqslant 2^{-28}\right)! A_2 0.$$

Оператор K вычисляет $\left[\frac{x}{\ln 2}\right]$, $\left\{\frac{x}{\ln 2}\right\}$, p' и z.

Оператор $З$ подготавливает для работы оператор A_1.

Оператор A_1 вычисляет очередной член ряда и добавляет его к содержимому ячейки, где накапливается сумма ряда.

Оператор A_2 вычисляет $(e^z)^2$.

Краткая характеристика

Число занятых программой ячеек 22.

Число тактов работы машины: максимальное 105 (из них 35 на выделение $[x]$ и $\{x\}$); минимальное 34 (из них 14 на выделение $[x]$ и $\{x\}$); в среднем — для $p_\infty = 10 - 68$ (из них 28 на выделение $[x]$ и $\{x\}$).

Относительная погрешность равна 2^{-31}.

Используемые стандартные константы: *2.00—2.02, 2.04, 2.07, 2.09, 2.0c, 2.10—2.12, 2.17, 2.19.*

Используемые стандартные рабочие ячейки: *2.20—2.29*, *2.2a*.

Аргумент $x = 2^{p_x}X$: $X \Rightarrow 2.20$; $p_x \Rightarrow 2.21$.

Функция $y = e^x$: мантисса величины $e^x \Rightarrow 2.20$, порядок величины $e^x \Rightarrow 2.21$.

Ячейка обратной связи *2.2a*.

Дополнительные сведения: требуется подпрограмма выделения целой и дробной частей числа.

Программа

	0.11	*0.15*	*0.00*	*0*		
	0.01	*1.20*	*3.4d*	*1*		
3.00	*2.21*	*2.21*	*2.07*	*b*	$2^{-33} \cdot \left[\frac{x}{\ln 2}\right] \Rightarrow 2.26$	
3.01	*2.20*	*2.20*	*3.14*	*9*	$\left\{\frac{x}{\ln 2}\right\} \Rightarrow 2.25$	
3.02	*2.19*	*2.12*	*3.02*	*4*		
3.03	*2.21*	*2.26*	*3.13*	*8*	$2^{-30}\left[\frac{x}{\ln 2}\right] \Rightarrow 2.21$	
3.04	*2.22*	*2.25*	*3.15*	*9*	$\frac{\ln 2}{2}\left\{\frac{x}{\ln 2}\right\} \Rightarrow 2.22$	*K*
3.05	*3.08*	*2.25*	*2.00*	*6*	$p\left(\left\{\frac{x}{\ln 2}\right\} \geqslant 0\right)$	
3.06	*2.22*	*2.22*	*3.15*	*a*	$\frac{\ln 2}{2}\left(\left\{\frac{x}{\ln 2}\right\} - 1\right) \Rightarrow 2.22$	
3.07	*2.21*	*2.21*	*2.07*	*b*	$\left(\left[\frac{x}{\ln 2}\right] + 1\right) \cdot 2^{-30} \Rightarrow 2.21$	
3.08	*2.20*	*2.01*	*2.22*	*b*	$1 + z \Rightarrow 2.20$	
3.09	*3.0a*	*2.24*	*2.22*	*4*	$z \Rightarrow 2.24$	*З*
3.0a	*3.0b*	*2.27*	*2.04*	*4*	$2^{-4} \Rightarrow 2.27$	
3.0b	*2.27*	*2.27*	*2.04*	*b*	$(k-1) \cdot 2^{-4} + 2^{-4} \Rightarrow 2.27$	
3.0c	*2.29*	*2.04*	*2.27*	*8*	$\frac{1}{k} \Rightarrow 2.29$	
3.0d	*2.24*	*2.24*	*2.29*	*9*	$\frac{z^{k-1}}{k!} \Rightarrow 2.24$	*A_1*
3.0e	*2.24*	*2.24*	*2.22*	*9*	$\frac{z^k}{k!} \Rightarrow 2.24$	
3.0f	*2.20*	*2.20*	*2.24*	*b*		

3.10	*3.0b*	*2.17*	*2.24*	*7*	$q\left(\left\|\frac{z^k}{k!}\right\| \leqslant 2^{-28}\right)$	*q*
3.11	*2.20*	*2.20*	*2.20*	*9*	$(e^z)^2 \Rightarrow 2.20$	*A_2*
3.12	*2.10*	*2.12*	*2.2a*	*4*	$(2.2a) \Rightarrow 2.12$	*0*
3.13	*0.80*	*0.00*	*0.00*	*1*	$(2^{-3})_\phi$	
3.14	*2.e2*	*2.a3*	*2.ca*	*5*	$\left(\frac{1}{2\ln 2}\right)_\phi$	Константы
3.15	*1.62*	*3.90*	*2.ff*	*1*	$\left(\frac{\ln 2}{2}\right)_\phi$	

Стандартные константы, используемые в программе:

2.00	*0.00*	*0.00*	*0.00*	*1*	$(+0)_\phi$
2.01	*3.ff*	*3.ff*	*3.ff*	*f*	$(1-2^{-33})_\phi$
2.04	*0.40*	*0.00*	*0.00*	*1*	$(2^{-4})_\phi$
2.07	*0.00*	*0.00*	*0.01*	*1*	$(2^{-30})_\phi$
2.17	*0.00*	*0.00*	*0.04*	*1*	$(2^{-28})_\phi$

§ 8. Стандартная подпрограмма для перевода чисел из десятичной системы в двоичную

Так же, как и в случае плавающей запятой, десятичные числа $x = 10^p X$ записываются на бланках и наносятся на перфоленту в виде

$$\varepsilon_x \alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5 \alpha_6 \alpha_7 \,|\, p \,|\, \varepsilon_p,$$

где ε_x — знак числа, ε_p — знак десятичного порядка, α_i — десятичные цифры мантиссы X. В ячейке запоминающего устройства десятичное число будет изображаться кодом числа

$$(2\bar{\varepsilon}_p - 1)(\bar{\varepsilon}_x \cdot 2^{-1} + \alpha_1 \cdot 2^{-5} + \ldots + \alpha_7 \cdot 2^{-29} + |p| \cdot 2^{-33})$$

в системе фиксированной запятой. Здесь $\bar{\varepsilon}_x$ и $\bar{\varepsilon}_p$ — коды знаков числа и порядка.

Описание алгоритма

Мантисса X переводится в двоичную систему согласно формуле

$$(X)_\Phi = \sum_{i=1}^{7} (\alpha_i \cdot 10^{-i})_\Phi,$$

причем многочлен вычисляется по схеме Горнера.

Нахождение двоичного порядка основывается на справедливости соотношений:

$$x = X \cdot 10^p = \left[X \cdot \frac{10}{16}\right] \cdot 10^{p-1} \cdot 2^4 =$$

$$= \left[X \cdot \left(\frac{10}{16}\right)^2\right] \cdot 10^{p-2} \cdot 2^{4\cdot 2} = \ldots = \left[X \cdot \left(\frac{10}{16}\right)^p\right] 2^{4p} \text{для } p > 0$$

и

$$x = X \cdot 10^p = \left[X \cdot \frac{8}{10}\right] \cdot 10^{p+1} \cdot 2^{-3} =$$

$$= \left[X \cdot \left(\frac{8}{10}\right)^2\right] \cdot 10^{p+2} \cdot 2^{-3\cdot 2} = \ldots = \left[X \cdot \left(\frac{8}{10}\right)^{|p|}\right] \cdot 2^{-3|p|} \text{для } p \leqslant 0.$$

Произведя дальнейшие преобразования над x по этим группам формул, получаем двоичное изображение числа x в обычной форме, принятой для метода плавающих масштабов:

$$x = 2^{p_1} \cdot X',$$

где

$$p_1 = 4p, \quad X' = (X)_\Phi \cdot \left(\frac{10}{16}\right)_\Phi^p, \quad \text{если} \quad p > 0,$$

$$p_1 = -3|p|, \quad X' = (X)_\Phi \cdot \left(\frac{8}{10}\right)_\Phi^{|p|}, \quad \text{если} \quad p \leqslant 0.$$

После этого x нормализуется.

Логическая схема

$$\rho_1 A_1 3\omega_1 \; ¡ \overbrace{\downarrow A_2 \uparrow A_8 q_1 (i \geqslant 7)}^{} ! \; \rho_2 \rightarrow$$

$$\rightarrow \omega_2 \; ¡ \overbrace{\downarrow A_4 \uparrow A_5 q_2 (0 \geqslant p)}^{} ! \; \omega_3 \; ¡ \overbrace{\downarrow A_6 \uparrow q_3 (p \geqslant 0)}^{} ! \; K.$$

Оператор ρ_1 осуществляет переключение машины на режим фиксированной запятой с двойной точностью.

Оператор A_1 получает в ячейке *2.24* величину $(p \cdot 2^{-4})_\Phi$ и в ячейке *2.23* число

$$\operatorname{sign} x \cdot (\alpha_1 \cdot 2^{-9} + \alpha_2 \cdot 2^{-13} + \ldots + \alpha_7 \cdot 2^{-33}) = K_x.$$

Оператор 3 очищает ячейку *2.20*.

Оператор A_2 производит умножение (*2.20*) на $\left(\frac{1}{10}\right)_\Phi$ и произведение помещает в ячейку *2.20*.

Оператор A_3 получает величину $(2^{-4}\alpha_i)_\Phi$ и добавляет ее к (*2.20*).

В результате работы операторов $A_2 A_3 q\,(i \geqslant 7)$ вычисляется величина $\left(\frac{10}{16} X\right)_\Phi$ в ячейке *2.20* и очищается ячейка *2.23*.

Оператор ρ_2 осуществляет переключение машины на режим фиксированной запятой.

Оператор A_4 умножает (*2.20*) на $\left(\frac{10}{16}\right)_\Phi$ и произведение помещает в ячейку *2.20*.

Оператор A_5 вычитает единицу из десятичного порядка и накапливает двоичный порядок числа. Последовательность операторов $A_4 A_5 q\,(0 \geqslant p)$ вычисляет двоичный порядок $p_1 = 4p$ и двоичную мантиссу

$$X' = (X)_\Phi \cdot \left(\frac{10}{16}\right)_\Phi^p.$$

Оператор A_6 умножает (*2.20*) на $\left(\frac{8}{10}\right)_\Phi$, добавляет единицу к десятичному порядку и накапливает двоичный порядок числа. Последовательность операторов $A_6 q\,(p \geqslant 0)$ вычисляет двоичный порядок $p_1 = -3|p|$ и мантиссу

$$X' = (X)_\Phi \cdot \left(\frac{8}{10}\right)_\Phi^{|p|}.$$

Оператор K производит нормализацию двоичного числа и передает управление стандартному выходу.

Краткая характеристика программы

Число ячеек, занятых программой, 23.

Число тактов: максимальное 103 (из них 20 на нормализацию), минимальное при $x \neq 0$ — 51 (из них 8 на нормализацию); в среднем (при $p = 4$) 72 такта.

Относительная погрешность равна 2^{-24}; однозначные целые числа переводятся точно.

Используемые стандартные константы: *2.20*, *2.02*, *2.07*, *2.0a—2.0c*, *2.0e—2.12*, *2.16—2.18*.

Используемые стандартные рабочие ячейки: *2.20—2.25*.

Число в десятичной системе помещается в ячейку *2.20*.

Мантисса двоичного числа получается в ячейке *2.20*, порядок — в ячейке *2.21*.

Ячейка обратной связи *2.12*.

Дополнительные сведения: требуется наличие подпрограммы «Нормализация».

Программа

	0.14	*0.16*	*0.00*	*0*		
	0.00	*1.18*	*3.5c*	*9*		
3.00	*0.00*	*0.0c*	*3.01*	*0*	Режим ФД	ρ_1
3.01	*2.24*	*2.20*	*2.04*	*9*	$(p \cdot 2^{-4})_\Phi \Rightarrow 2.24$, $\lvert K_x \rvert \operatorname{sign} p \Rightarrow 2.25$	
3.02	*2.23*	*2.25*	*3.16*	*f*	$-\lvert K_x \rvert \Rightarrow 2.23$	
3.03	*3.05*	*2.20*	*2.02*	*7*	$K_x \Rightarrow 2.23$	A_1
3.04	*2.23*	*2.00*	*2.23*	*a*		
3.05	*3.07*	*2.20*	*2.00*	*4*	$0 \Rightarrow 2.20$	$3\omega_1$
3.06	*2.20*	*2.20*	*3.15*	*9*	$(2.20) \cdot \left(\frac{1}{10}\right)_\Phi \Rightarrow 2.20$	A_2
3.07	*2.22*	*2.23*	*2.04*	*9*	$(2.20) + (2^{-4}\alpha_i)_\Phi \Rightarrow 2.20$	A_3
3.08	*2.20*	*2.22*	*2.21*	*b*		
3.09	*3.06*	*2.00*	*2.23*	*7*	$q_1 (i \geqslant 7)$	q_1
3.0a	*0.00*	*0.10*	*3.0c*	*0*	Режим ФЗ	ρ_2, ω_2

3.0b	*2.20*	*2.20*	*2.0a*	*9*	$(2.20) \cdot \left(\frac{10}{16}\right)_\Phi \Rightarrow 2.20$	A_4
3.0c	*2.24*	*2.24*	*2.04*	*a*	$p - 1 \Rightarrow p$	
3.0d	*2.23*	*2.23*	*2.17*	*b*	$(2.23) + (4 \cdot 2^{-30})_\Phi \Rightarrow 2.23$	A_5
3.0e	*3.0b*	*2.00*	*2.24*	*6*	$q_2 (0 \geqslant p)$	q_2
3.0f	*0.00*	*0.00*	*3.13*	*0*		ω_3
3.10	*2.23*	*2.23*	*2.16*	*a*	$(2.23) - (3 \cdot 2^{-30})_\Phi \Rightarrow 2.23$	
3.11	*2.20*	*2.20*	*2.0b*	*9*	$(2.20) \cdot \left(\frac{8}{10}\right)_\Phi \Rightarrow (2.20)$	A_6
3.12	*2.24*	*2.24*	*2.04*	*b*	$p + 1 \Rightarrow p$	
3.13	*3.10*	*2.24*	*2.00*	*6*	$q_3 (p \geqslant 0)$	q_3
3.14	*2.0e*	*2.21*	*2.23*	*4*	Нормализация x	K
3.15	*0.66*	*1.99*	*2.66*	*7*	$\left(\frac{1}{10}\right)_\Phi$	
3.16	*0.1f*	*3.ff*	*3.ff*	*e*	$(2^{-33} - 2^{-5})_\Phi$	

Стандартные константы, используемые в программе

2.00	*0.00*	*0.00*	*0.00*	*1*	$(+0)_\Phi$
2.02	*2.00*	*0.00*	*0.00*	*1*	$\left(\frac{1}{2}\right)_\Phi$
2.04	*0.40*	*0.00*	*0.00*	*1*	$(2^{-4})_\Phi$
2.0c	*2.80*	*0.00*	*0.00*	*1*	$\left(\frac{10}{16}\right)_\Phi$
2.0b	*3.33*	*0.cc*	*3.33*	*5*	$\left(\frac{8}{10}\right)_\Phi$
2.16	*0.00*	*0.00*	*0.03*	*1*	$(3 \cdot 2^{-30})_\Phi$
2.17	*0.00*	*0.00*	*0.04*	*1*	$(4 : 2^{-30})_\Phi$

§ 9. Стандартная подпрограмма для перевода чисел из двоичной системы в десятичную

Программа предназначена для перевода числа $x = 2^{p_1}X_1$ к нормализованному десятичному виду $x = 10^{p_2}X_2$ и для его печати; при этом число печатается в том же виде, в каком оно вводится в машину, т. е. $\varepsilon_x\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7 \mid p_2 \mid \varepsilon_p$, где $\alpha_1 \neq \neq 0$ для $x \neq 0$.

Для этого необходимо сформировать число

$$(2\bar{\varepsilon}_x - 1) \cdot (\alpha_1 \cdot 2^{-4} + \alpha_2 \cdot 2^{-8} + \ldots + \alpha_7 \cdot 2^{-28} + |p_2| \cdot 2^{-32} + \bar{\varepsilon}_p \cdot 2^{-33}) \quad \text{(VI.1)}$$

в системе фиксированной запятой.

Описание алгоритма

Для получения десятичного порядка сначала проводятся следующие тождественные преобразования:

$$X_1 \cdot 2^{p_1} = \left[X_1 \cdot \left(\frac{10}{16}\right)_\Phi\right] \cdot 2^{p_1+4} \cdot 10^{-1} =$$

$$= \left[X_1 \cdot \left(\frac{10}{16}\right)_\Phi^2\right] \cdot 2^{p_1+4\cdot 2} \cdot 10^{-2} = \ldots$$

$$\ldots = \left[X_1 \cdot \left(\frac{10}{16}\right)_\Phi^k\right] \cdot 2^{p_1+4k} \cdot 10^{-k} \ldots$$

до тех пор, пока не будет выполнено условие $p_1 + 4k \geqslant 1$. (Если $p_1 \geqslant 1$, то $k = 0$.)

В результате этих преобразований число x будет представлено в виде $x = X_1^{(1)} \cdot 2^{p_1(1)} \cdot 10^{p_2(1)}$, где

$$X_1^{(1)} = X_1 \cdot \left(\frac{10}{16}\right)_\Phi^k, \quad p_1^{(1)} = p_1 + 4k, \quad p_1^{(1)} \geqslant 1, \quad p_2^{(1)} = -k.$$

После этого производятся преобразования:

$$X_1^{(1)} \cdot 2^{p_1^{(1)}} \cdot 10^{p_2^{(1)}} = \left[X_1^{(1)} \cdot \left(\frac{8}{10}\right)_\Phi\right] \cdot 2^{p_1^{(1)}-3} \cdot 10^{p_2^{(1)}+1} =$$

$$= \left[X_1^{(1)} \cdot \left(\frac{8}{10}\right)_\Phi^2\right] \cdot 2^{p_1^{(1)}-3\cdot 2} \cdot 10^{p_2^{(1)}+2} = \ldots$$

$$\ldots = \left[X_1^{(1)} \cdot \left(\frac{8}{10}\right)_\Phi^l\right] \cdot 2^{p_1^{(1)}-3l} \cdot 10^{p_2^{(1)}+l} = \ldots$$

до тех пор, пока не будет выполнено условие $p_1^{(1)} - 3l \leqslant 0$.

В результате этих преобразований число x будет представлено в виде $x = X_1^{(2)} \cdot 2^{p_1^{(2)}} \cdot 10^{p_2^{(2)}}$, где

$$X_1^{(2)} = X_1^{(1)} \cdot \left(\frac{8}{10}\right)_\Phi^l,$$

$$p_1^{(2)} = p_1^{(1)} - 3l, \quad -2 \leqslant p_1^{(2)} \leqslant 0, \quad p_2^{(2)} = l - k.$$

Далее, число $X_1^{(2)} \cdot 2^{p_1^{(2)}}$ представляем в системе фиксированной запятой:

$$X_1^{(3)} = \left(X_1^{(2)} \cdot 2^{p_1^{(2)}}\right)_\Phi.$$

После этого, умножая $X_1^{(3)}$ последовательно на 10 и вычитая при этом после каждого умножения единицу из порядка $p_2^{(2)}$, получим число x в форме $x = 10^{p_2} \cdot X_2$, где

$$X_2 = X_1^{(3)} 10^m, \quad \frac{1-\varepsilon}{10} \leqslant |X_2| \leqslant 1 - \varepsilon, \quad p_2 = p_2^{(2)} - m.$$

Величина ε выбирается из тех соображений, чтобы не могло произойти переполнения при округлении.

Выделение десятичных цифр X_2 производится в режиме фиксированной запятой с двойной точностью, так же как и в § 3 главы V, а именно: X_2 умножается на $(10 \cdot 2^{-33})_\Phi$, после чего получается в одной ячейке $(\alpha_1 \cdot 2^{-33})_\Phi$, а в другой — дробная часть $X_2 \cdot 10$, которая в свою очередь умножается на $(10 \cdot 2^{-33})_\Phi$, после чего получается в одной ячейке $(\alpha_2 \cdot 2^{-33})_\Phi$, а в другой — очередная дробная часть и так далее.

Логическая схема

$$Kq_1(X_1 = 0)\,\overline{!\,\omega \mid \uparrow A_1 \downarrow q_2(p_1^{(1)} \geqslant 1) \mid \uparrow A_2 q_3(0 \geqslant p_1^{(2)}) \mid A_3\rho_1 \rightarrow}$$

$$\overline{\rightarrow A_4 A_5 \downarrow A_6 q(l \geqslant 7) \mid \rho_2 A_7 \downarrow H0}.$$

Оператор K нормализует $x = X_1 \cdot 2^{p_1}$ и засылает нуль в ячейку, где будет получаться десятичный порядок и код (VI. 1).

Оператор A_1 накапливает величины $p_1^{(1)}$, $p_2^{(1)}$ и $X_1^{(1)}$.

Оператор A_2 накапливает величины $p_1^{(2)}$, $p_2^{(2)}$ и $X_1^{(2)}$.

Оператор A_3 получает $X_1^{(3)}$.

Оператор ρ_1 устанавливает режим фиксированной запятой с двойной точностью.

Оператор A_4 получает величину X_2, десятичный порядок p_2 и очищает ячейку *2.21*.

Оператор A_5 округляет число X_2.

Оператор A_6 выделяет очередную десятичную цифру в виде $(\alpha_i \cdot 2^{-33})_\phi$, добавляет ее к содержимому ячейки *2.21* и сдвигает (*2.21*) на четыре разряда влево.

Оператор ρ_2 устанавливает режим фиксированной запятой.

Оператор A_7 присваивает нужные знаки величинам p_2 и X_2 и формирует десятичное число в виде (VI. 1).

Оператор H осуществляет печать числа.

Краткая характеристика программы

Количество занятых ячеек 36.

Число тактов: максимальное 139 (из них 38 на нормализацию), минимальное — при $x \neq 0$ — 60 (из них 8 на нормализацию), в среднем ($p_1 = 10$ и нормализация на 7 разрядов) 93 — из них 20 на нормализацию.

Относительная погрешность равна $5 \cdot 10^{-8}$.

Используемые стандартные константы: *2.00*, *2.02*, *2.04*, *2.07—2.0c*, *2.0e — 2.12*, *2.16—2.18*.

Используемые стандартные рабочие ячейки: *2.20—2.25*, *2.2a*.

Ячейка обратной связи *2.2a*.

Аргумент $x = X_1 \cdot 2^{p_1}$: $X_1 \Rightarrow 2.20$, $p_1 \Rightarrow 2.21$.

Результат засылается в *2.25* и печатается.

Дополнительные сведения: требуется подпрограмма «Нормализация».

Программа

	0.1d	*0.23*	*0.00*	*0*		
	0.00	*3.d0*	*2.85*	*b*		
3.00	*2.0e*	*2.12*	*3.00*	*4*	Нормализация x	K
3.01	*3.02*	*2.25*	*2.00*	*4*	$0 \Rightarrow 2.25$	

3.02	*3.07*	*2.00*	*2.20*	*7*	$q_1(0 \geqslant \lvert X_1 \rvert)$	q_1
3.03	*3.20*	*2.12*	*2.2a*	*4*	$(2.2a) \Rightarrow 2.12$	ω
3.04	*2.21*	*2.21*	*2.17*	*b*	$p + 4 \Rightarrow p$	
3.05	*2.25*	*2.25*	*2.08*	*a*	$(2.25) - (2^{-32})_\phi \Rightarrow 2.25$	A_1
3.06	*2.20*	*2.20*	*2.0a*	*9*	$(2.20) \cdot \left(\frac{10}{16}\right)_\phi \Rightarrow 2.20$	
3.07	*3.04*	*2.21*	*2.07*	*6*	$q_2(p_1^{(1)} \geqslant 1)$	q_2
3.08	*2.21*	*2.21*	*2.16*	*a*	$p - 3 \Rightarrow p$	
3.09	*2.25*	*2.25*	*2.08*	*b*	$(2.25) + (2^{-32})_\phi \Rightarrow 2.25$	A_2
3.0a	*2.20*	*2.20*	*2.0b*	*9*	$(2.20) \cdot \left(\frac{8}{10}\right)_\phi \Rightarrow 2.20$	
3.0b	*3.08*	*2.00*	*2.21*	*6*	$q_3(0 \geqslant p_1^{(2)})$	q_3
3.0c	*3.0f*	*2.12*	*2.2a*	*4*	$(2.2a) \Rightarrow 2.12$	
3.0d	*2.20*	*2.20*	*2.02*	*9*		
3.0e	*2.21*	*2.21*	*2.07*	*b*	$(2.20) \cdot (2^{p_1^{(2)}})_\phi \Rightarrow 2.20$	A_3
3.0f	*3.0d*	*2.21*	*2.00*	*6*		
3.10	*0.1f*	*0.0c*	*3.13*	*0*	Режим ФД *)	p_1
3.11	*2.20*	*2.20*	*3.22*	*9*	$(10 \cdot (2.20))_\phi \Rightarrow 2.20$	
3.12	*2.25*	*2.25*	*2.08*	*a*	$(2.25) - (2^{-32})_\phi \Rightarrow 2.25$	A_4
3.13	*3.11*	*2.20*	*3.23*	*7*	$q\left(\lvert (2.20) \rvert \geqslant \frac{1-\varepsilon}{10}\right)$	
3.14	*2.23*	*3.21*	*2.20*	*e*	$(2.20) + (\operatorname{sign} x \cdot \varepsilon)_\phi \Rightarrow$	
3.15	*2.23*	*2.20*	*2.23*	*b*	$\Rightarrow 2.23$	A_5
3.16	*2.23*	*2.23*	*3.22*	*9*	$(\alpha_i \cdot 2^{-33})_\phi \Rightarrow 2.24$, $\{X_2 \cdot 10^4\} \Rightarrow 2.23$	
3.17	*2.21*	*2.21*	*2.24*	*b*	$(2.21) + (\alpha_i \cdot 2^{-33})_\phi \Rightarrow$ $\Rightarrow 2.21$	A_6
3.18	*2.21*	*2.21*	*2.04*	*8*	$(2.21) : (2^{-4})_\phi \Rightarrow 2.21$; $0 \Rightarrow 2.22$	

*) Используется в программе также в качестве константы.

3.19	3.16	2.21	3.10	7	$q_4 (l \geqslant 7)$	q_4
3.1a	0.00	0.18	3.1b	0	Режим ФЗ	ρ_2
3.1b	2.21	2.21	2.21	b	Сдвиг (2.21) на 1 разряд влево	
3.1c	3.1e	2.25	2.00	6	$q (p_2 \geqslant 0)$	
3.1d	2.25	2.25	2.09	b	$(\lvert p_2 \rvert \cdot 2^{-32})_\Phi + (2^{-33})_\Phi \Rightarrow \Rightarrow 2.25$	A_7
3.1e	2.25	2.25	2.21	e	Формирование кода (VI.1)	
3.1f	2.25	2.21	2.25	b		
3.20	2.10	2.25	0.03	3.	Печать кода (VI.1)	$H,0$
3.21	0.00	0.00	0.35	b	$\varepsilon = (5 \cdot 10^{-8})_\Phi$	
3.22	0.00	0.00	0.01	5	$(10 \cdot 2^{-33})_\Phi$	
3.23	0.66	1.99	2.61	1	$\left(\frac{1-\varepsilon}{10}\right)_\Phi$	

Стандартные константы, используемые в программе:

2.00	0.00	0.00	0.00	1	$(+0)_\Phi$
2.02	2.00	0.00	0.00	1	$(2^{-1})_\Phi$
2.04	0.40	0.00	0.00	1	$(2^{-4})_\Phi$
2.07	0.00	0.00	0.01	1	$(2^{-30})_\Phi$
2.08	0.00	0.00	0.00	5	$(2^{-32})_\Phi$
2.09	0.00	0.00	0.00	3	$(2^{-38})_\Phi$
2.0a	2.80	0.00	0.00	1	$\left(\frac{10}{16}\right)_\Phi$
2.0b	3.33	0.cc	3.33	5	$\left(\frac{8}{10}\right)_\Phi$
2.16	0.00	0.00	0.03	1	$(3 \cdot 2^{-30})_\Phi$
2.17	0.00	0.00	0.04	1	$(4 \cdot 2^{-30})_\Phi$

ГЛАВА VII

НЕКОТОРЫЕ СТАНДАРТНЫЕ ПРОГРАММЫ ДЛЯ РЕЖИМА ПЛАВАЮЩЕЙ ЗАПЯТОЙ

§ 1. Программа для решения систем обыкновенных дифференциальных уравнений методом Рунге — Кутта

При численном интегрировании обыкновенных дифференциальных уравнений ставится цель получить решение с заданной точностью на всем отрезке интегрирования. Для достижения этой цели необходимо, чтобы на каждом шаге погрешность была достаточно мала. В различных точках t_i одной и той же интегральной кривой заданная точность обеспечивается, вообще говоря, при различных значениях шага интегрирования h_i. Если интегрирование ведется с постоянным шагом на всем отрезке, то для обеспечения заданной точности вычислений в качестве постоянного шага h приходится брать наименьший из h_i. При этом может оказаться, что такой мелкий шаг на самом деле необходим только на небольшом участке, а в остальных частях отрезка интегрирования возможен значительно более крупный шаг. Поэтому при интегрировании с постоянным шагом мы сделаем излишне большое количество шагов, вследствие чего затратим больше времени на решение задачи и можем потерять точность из-за накапливания ошибок округления. В этих случаях целесообразно несколько усложнить программу с тем, чтобы она автоматически выбирала шаг интегрирования в зависимости от хода интегральных кривых и чтобы в каждой точке отрезка интегрирования этот шаг был максимально возможным при заданной точности, что и сделано в данной программе.

Для пользования программой система дифференциальных уравнений должна быть приведена к виду

$$\frac{dx_i}{dt} = f_i(t,\ x_1,\ x_2,\ \ldots,\ x_n) \quad (i = 1,\ 2,\ \ldots,\ n) \qquad \text{(VII. 1)}$$

или, переходя к векторным обозначениям,

$$\frac{dx}{dt} = f(t,\ x).$$

Программа рассчитана на решение систем любого порядка (ограничение на порядок системы накладывает лишь емкость оперативной памяти машины).

Значение $x(t+h)$ вычисляется по формуле Рунге — Кутта:

$$x(t+h) = x(t) + m,$$

где

$$\begin{aligned} m &= \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4, \\ k_1 &= hf[t,\ x(t)], \\ k_2 &= hf\left[t + \frac{h}{2},\ x(t) + \frac{1}{2}k_1\right], \\ k_3 &= hf\left[t + \frac{h}{2},\ x(t) + \frac{1}{2}k_2\right], \\ k_4 &= hf[t+h,\ x(t) + k_3]. \end{aligned} \qquad \text{(VII. 2)}$$

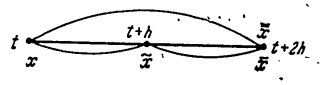
Описание алгоритма выбора шага

Процесс численного интегрирования состоит из последовательных этапов, каждый из которых заключается в том, что по известным значениям $x(t)$ и исходной величине шага h_0 с помощью формул (VII. 2) вычисляются значения $x(t+h)$, причем шаг h ($h \leqslant h_0$) выбирается так, чтобы он обеспечивал заданную точность вычислений ε; наряду с этим определяется исходная величина шага h_0 для следующего этапа. Каждый этап реализуется по следующему алгоритму.

а) По значениям x в точке t по формулам (VII. 2) находится решение $\bar{\bar{x}}_1$ в точке $t + 2h_0$.

б) По значениям x в точке t по формулам (VII. 2) находится решение $\tilde{x}_1$ в точке $t + h_0$.

в) По значениям $\tilde{x}_1$ в точке $t + h_0$ по тем же формулам (VII. 2) вычисляется решение $\bar{x}_1$ в точке $t + 2h_0$.



г) По полученным значениям $\bar{x}_1$ и $\bar{\bar{x}}_1$ находится погрешность вычислений на данном шаге по какому-либо правилу, например

$$\delta_1 = \max_i |\bar{x}_{i1} - \bar{\bar{x}}_{i1}|.$$

д) Значение δ_1 сравнивается с заданной допустимой погрешностью ε. Если $\delta_1 \geqslant \varepsilon$, то шаг h_0, возможно, не обеспечивает требуемой точности в данной точке отрезка интегрирования. В этом случае шаг h_0 делится пополам и, исходя из значения x в точке t, с шагом $\frac{h_0}{2}$ снова вычисляется $\tilde{x}_2$ и $\bar{x}_2$. В качестве $\bar{\bar{x}}_2$ берется вычисленное ранее значение $\tilde{x}_1$ в точке $t + h_0$. По значениям $\bar{x}_2$ и $\bar{\bar{x}}_2$ находится погрешность δ_2 и снова сравнивается с ε. Если $\delta_2 \geqslant \varepsilon$, то шаг снова делится пополам и т. д.

Если $\delta_k < \varepsilon$, то вычисление очередной точки интегральной кривой закончено. В качестве x для следующего этапа берется $\bar{x}_k$.

е) Для выбора исходной величины шага для следующего этапа производится сравнение δ_k с величиной $\varepsilon^* < \varepsilon$. Если $\delta_k < \varepsilon$, но $\delta_k > \varepsilon^*$, то в качестве исходной величины шага для следующего этапа берется $\frac{h_0}{2^{k-1}}$.

Если же $\delta_k < \varepsilon$ и $\delta_k \leqslant \varepsilon^*$, то на предыдущем этапе заданная точность ε выдержана с некоторым (в зависимости от величины ε^*) запасом, поэтому можно ожидать, что на следующем этапе заданная точность ε будет обеспечена при большем шаге интегрирования. В этом случае в качестве исходной величины шага для следующего этапа берется $2 \cdot \frac{h_0}{2^{k-1}}$. Если, например, уже $\delta_1 < \varepsilon$ и $\delta_1 \leqslant \varepsilon^*$, то в качестве x для следующего этапа берется $\bar{\bar{x}}_1$, а в качестве

исходной величины следующего шага берется $2h_0$. (В данной программе взято $\varepsilon^* = \frac{\varepsilon}{10}$.)

Из формул (VII. 2) видно, что для осуществления одного шага по методу Рунге — Кутта необходимо иметь четыре группы ячеек оперативной памяти:

1) для хранения вектора f;

2) для хранения вектора y — аргумента вектор-функции f, который последовательно принимает значения:

$$y_1 = x(t), \quad y_2 = x(t) + \frac{1}{2}k_1,$$

$$y_3 = x(t) + \frac{1}{2}k_2, \quad y_4 = x(t) + k_3;$$

в этой же группе ячеек получается результат $x(t+h)$;

3) для хранения вектора добавок m, который последовательно принимает значения

$$m_j = m_{j-1} + g_j k_j \quad (j = 1, 2, 3, 4),$$

где

$$m_0 = 0, \quad g_1 = g_4 = \frac{1}{6}, \quad g_2 = g_3 = \frac{1}{3}.$$

4) для хранения исходного значения x для реализации каждого очередного шага по формулам (VII. 2).

Обозначим эти четыре группы ячеек буквами F, Y, M и X соответственно.

Значения k_j по мере их получения сразу же добавляются с соответствующими коэффициентами g_j ($1/6$, $1/3$, $1/3$, $1/6$) и l_j ($1/2$, $1/2$, 1) к m_{j-1} и x для получения m_j и y_j, поэтому специальной группы ячеек для их хранения не требуется.

Для реализации автоматического выбора шага, кроме указанных выше четырех групп ячеек, необходимы еще две группы:

Z — для хранения результата двойного шага $\bar{\bar{x}}$, и

U — для хранения исходного для этапа значения x, который используется после каждого измельчения шага.

Первоначально заданные значения h_0 и t_0 не сохраняются — в программе имеется лишь по одной ячейке для хранения текущих значений h и t.

Логическая схема

Для более четкого выделения функционально различных частей программы приведем сначала ее сокращенную логическую схему

$$q\,\overline{\Phi_1}\,\overline{\vee}\,K_1.$$

Оператор q при первом обращении к программе передает управление оператору Φ_1, а в дальнейшем — оператору K_1. Вход в программу всегда осуществляется через оператор q.

Оператор Φ_1 формирует в программе команды обращения к нестандартным операторам H_1, H_2 и H_3 (см. ниже), пересылает ε в оперативную ячейку памяти, вычисляет ε^* и формирует константы, зависящие от порядка n решаемой системы. Оператор Φ_1 предназначен для однократного действия; в дальнейшей работе программы ячейки памяти, занятые командами данного оператора, используются для других целей.

Обобщенный оператор K_1 реализует алгоритм осуществления одного шага интегрирования по методу Рунге — Кутта и алгоритм автоматического выбора величины шага, дополненный функцией управления всеми частями программы. В программе эти два алгоритма четко разделены.

Теперь рассмотрим логическую схему части программы, реализующей один шаг по методу Рунге — Кутта

$$A_1 O(j)\,\overline{\wedge}\,A_2\,\overline{\vee}\,H_1 A_3 F(j) A_4 p_1 (j \geqslant 2)\,\overline{\wedge}\,A_5 p_2 (j \geqslant 4)\,\overline{\wedge}\,A_6 0.$$

Оператор A_1 вычисляет величину $\frac{h}{2}$ и пересылает содержимое группы ячеек Y в группу ячеек X (напомним, что в это момент $y = x$).

Оператор $O(j)$ засылает нули в группу ячеек M, восстанавливает $g_j = g_0$ и $l_j = l_0$ и ставит счетчик параметра j в начальное положение ($j = 0$).

Операторы A_2 и A_5 вычисляют коэффициенты g_j и l_j.

Нестандартный оператор H_1 осуществляет вычисление правых частей системы (в программе содержится лишь передача управления оператору H_1 — команда *3 d9*).

Оператор A_3 вычисляет текущее значение t для нахождения k_j.

Оператор $F(j)$ добавляет единицу к счетчику параметра j.

Оператор A_4 по f_j вычисляет k_j и, добавляя их с коэффициентами g_j и l_j к m_{j-1} и x, образует m_j и y_j.

Оператор A_6 вычисляет $x(t+h)=x(t)+m_4$ и результат помещает в группу ячеек Y.

Оператор $\emptyset$ (команда *3.ea*) передает управление стандартному выходу, обеспечивая возврат к соответствующей команде части программы, реализующей выбор величины шага.

Обозначим теперь вышеприведенную логическую схему обобщенным оператором K_4.

С точки зрения общей логической схемы программы оператор K_4 выполняет следующую функцию: по начальным данным t, $y=x(t)$ и шагу h вычисляет $x(t+h)$ в группе ячеек Y, причем в ячейке для хранения t в итоге получается $t+h$, а $x(t)$ сохраняется в группе ячеек X. Обращение к оператору K_4 осуществляется командами *3.ee*, *3.f5* и *3.f6*.

Теперь рассмотрим более подробную логическую схему оператора K_1.

$$A_7K_4\mathcal{Z}_1\overline{\vee A_8K_4K_4H_2p_3(\delta\geqslant\varepsilon)\,;\,\underline{A_9\omega}\,!}\,\wedge\, p_4(\delta\leqslant\varepsilon^*)\,!\,\overline{A_{10}}\vee H_3.$$

Оператор A_7 удваивает величину шага h и пересылает начальные данные из группы ячеек Y в группу ячеек U.

Оператор $\mathcal{Z}_1$ переносит результаты вычислений с удвоенным шагом из группы ячеек Y в группу ячеек Z.

Оператор A_8 уменьшает значение t на величину шага, делит шаг пополам и переносит содержимое группы ячеек U в группу ячеек Y.

Нестандартный оператор H_2 осуществляет вычисление величины погрешности δ (в программе имеется лишь команда обращения к этому оператору — команда *3.f7*).

Оператор A_9 переносит содержимое группы ячеек X в группу ячеек Z и уменьшает значение t на величину шага.

Оператор A_{10} удваивает величину шага.

Нестандартный оператор H_3 осуществляет обработку результатов очередного шага (в программе содержится лишь обращение к этому оператору — команда *3.ff*).

Отметим, что наличие операции переноса числа, совмещенной с передачей управления, дало возможность рабочие части пяти циклов переноса групп чисел вынести в одни и те же ячейки — *3.b7* и *3.b8* (см. гл. II, § 4).

Краткая характеристика программы

Программа работает в режиме плавающей запятой. Количество занятых ячеек $90+6n$ (n — число уравнений системы).

Число тактов на один этап: $132n+160$ — без изменения шага и $88n+108$ — на каждое уменьшение шага в два раза.

Используемые ячейки со стандартными константами: *2.00*, *2.05*, *2.06* — *2.08*, *2.0d*, *2.0e*, *2.10* — *2.12*, *2.19*, *2.1a*.

Используемые стандартные рабочие ячейки: *2.20* — *2.22*.

Дополнительные сведения.

1. Для решения конкретной задачи при помощи данной программы, которую будем называть программой метода, необходимо иметь в памяти три дополнительные программы:

а) «Вычисление правых частей» — нестандартный оператор H_1;

б) «Вычисление погрешности» — нестандартный оператор H_2;

в) «Обработка результатов шага» — нестандартный оператор H_3.

2. Программа расположена в ячейках памяти *3.a3* — *3.ff* и использует в качестве рабочих ячеек ячейки с *3.a6* — *6n* по *3.a2*.

3. Компоненты векторов f, y, $\tilde{x}$, m, x и $\bar{x}$, которым соответствуют группы ячеек F, Y, X, M, U и Z, расположены в ячейках памяти:

$$\langle f_k\rangle = 3.a6-6k,\quad \langle y_k\rangle = 3.a7-6k,$$
$$\langle \tilde{x}_k\rangle = 3.a8-6k,\quad \langle m_k\rangle = 3.a9-6k,$$
$$\langle x_k\rangle = 3.aa-6k \text{ и } \langle \bar{x}_k\rangle = 3.ab-6k$$
$$(k=1, 2, \ldots, n).$$

Инструкция по пользованию программой

Перед первым обращением к предлагаемой программе необходимо занести начальную величину шага h и начальные данные t, x_1, x_2, ..., x_n в ячейки, указанные во втором столбце приводимой ниже таблицы, а также задать в стандартные ячейки параметры программы (порядок системы n, вход программы «Вычисление правых частей», вход

программы «Вычисление погрешности», вход программы «Обработка результатов шага» и ε) по следующим правилам:

в ячейку *2.20* — число $(-n \cdot 2^{-10})_ф$;

в ячейку *2.21* — код $m_3\ m_2\ m_1\ 0$, где

m_3 — вход программы «Вычисление правых частей»,

m_2 — вход программы «Вычисление погрешностей»,

m_1 — вход программы «Обработка результатов шага»;

в ячейку *2.22* — величину ε в виде числа в системе плавающей запятой.

При последующих обращениях к программе метода изменение параметров программы посредством задания их в ячейки *2.20* — *2.22* невозможно.

Таблица. Расположение основных величин

		t	*3.a6*		
		h	*3.a7*		
f_1	*3.a0*	x_1	*3.a1*	x_1	*3.a5*
f_2	*3.9a*	x_2	*3.9b*	x_2	*3.9f*
f_3	*3.94*	x_3	*3.95*	x_3	*3.99*
...	...	...	...	...	...
f_k	(*3.a6*—*6k*)	x_k	(*3.a7*—*6k*)	x_k	(*3.ab* — *6k*)

Программа «Вычисление правых частей» должна вычислить значение функций f_i ($i = 1, 2, \ldots, n$) и поставить их в оперативные ячейки программы метода согласно первому столбцу таблицы, приведенной выше. Во втором столбце таблицы указано, из каких ячеек брать значения t, x_1, x_2, ..., x_n.

Выход из программы «Вычисление правых частей» должен состоять в безусловной передаче управления команде *3.d4* программы метода.

Программа «Вычисление погрешности» должна по тому или иному правилу найти величину погрешности δ в виде числа в системе плавающей запятой и поместить ее в ячейку *2.20*. Сравниваемые величины переменных $x_1, x_2, \ldots, x_n$ берутся из ячеек, указанных во втором и третьем столбцах таблицы.

Выход из программы «Вычисление погрешности» должен состоять в безусловной передаче управления команде *3.f8* программы метода.

Программа «Обработка результатов шага» использует результаты шага из ячеек, указанных во втором столбце

таблицы, и в случае необходимости помещает новые начальные данные для следующего шага в те же ячейки.

Как первое обращение к программе метода, так и возврат к ней из программы обработки результатов шага производятся безусловной передачей управления команде *3.c7*.

Программа

	0.00	*0.5c*	*0.00*	*0*		
	0.09	*2.8e*	*1.2a*	*a*		
3.a3	*3.ff*	*2.21*	*2.0e*	*f*		
3.a4	*3.f7*	*2.0e*	*2.05*	*8*		
3.a5	*0.00*	*0.00*	*3.a8*	*0*		
3.a6	*0.00*	*0.00*	*0.00*	*0*	Формирование команд обращения к нестандартным операторам	
3.a7	*0.00*	*0.00*	*0.00*	*0*		
3.a8	*3.f7*	*3.f7*	*2.21*	*f*		
3.a9	*3.d3*	*2.21*	*3.f7*	*a*		
3.aa	*3.f7*	*3.f7*	*2.05*	*9*		
3.ab	*3.d3*	*3.d3*	*2.06*	*9*		
3.ac	*2.20*	*2.20*	*3.be*	*9*		
3.ad	*3.ad*	*2.20*	*2.05*	*8*		
3.ae	*3.ae*	*2.20*	*3.ad*	*f*		
3.af	*3.af*	*3.cc*	*3.ae*	*a*		
3.b0	*3.b0*	*3.c1*	*2.20*	*a*	Формирование констант, зависящих от n	Φ_1
3.b1	*3.b1*	*3.c2*	*2.20*	*a*		
3.b2	*3.b2*	*3.c3*	*2.20*	*a*		
3.b3	*3.b3*	*3.c4*	*2.20*	*a*		
3.b4	*3.b4*	*3.c5*	*2.20*	*a*		
3.b5	*3.b5*	*3.db*	*3.ad*	*b*		
3.b6	*3.b8*	*3.b6*	*2.22*	*4*	Засылка ε	
3.b7	*3.b8*	*3.b8*	*3.bc*	*b*		
*3.b8**	*0.00*	*0.02*	*3.c6*	*0*		
3.b9	*3.d5*	*3.d4*	*3.ba*	*4*	Константы для переключателя	
3.ba	*3.d6*	*3.d4*	*3.b9*	*4*		
3.bb	*2.36*	*0.00*	*0.00*	*1*	$(6)_д$	

3.bc	0.00	0.06	0.06	1	Константы переадресации	Φ_1
3.bd	0.00	0.06	0.00	1		
3.be	0.06	0.06	0.00	1		
3.bf	0.06	0.06	0.06	1		
3.c0	3.a7	3.a8	3.a9	d	Первоначальный вид команд, зависящих от n	
3.c1	3.c9	3.a8	3.a7	4		
3.c2	3.ed	3.aa	3.a7	4		
3.c3	3.f0	3.ab	3.a7	4		
3.c4	3.f4	3.a7	3.aa	4		
3.c5	3.fa	3.ab	3.a8	4		
3.c6	3.c6	3.b6	2.19	8	Вычисление ε *	
3.c7*	3.d3	3.c7	3.f7	4		q
3.c8	3.b7	3.b8	3.c1	4	$(Y) \Rightarrow X$	A_1
3.c9	3.b7	3.b0	3.b8	7		
3.ca	3.ab	3.a7	2.0d	9	$\frac{h}{2} \Rightarrow <h>$	
3.cb	3.cc	3.cc	3.bd	b	Засылка нулей в группу ячеек M	O
3.cc	3.cd	3.a9	2.00	4		
3.cd	3.cb	3.af	3.cc	7		
3.ce	3.cc	3.cc	3.ae	b		
3.cf	3.d0	3.a8	3.ab	4	$\frac{h}{2} \Rightarrow <l_j>$	
3.d0	3.d1	3.a9	3.bb	4	$6 \Rightarrow <g_j>$	
3.d1	3.aa	2.00	2.1a	a	$j = 0$	
3.d2	3.a9	3.a9	2.0d	9	$\frac{1}{2} g_j \Rightarrow <g_j>$	A_2
3.d3*	0.00	0.18	3.a3	0	Обращение к H_1	H_1
3.d4*	3.d5	3.d4	3.ba	4	Переключатель	A_3
3.d5	3.a6	3.a6	3.ab	d	$t + \frac{h}{2} \Rightarrow <t>$	
3.d6	3.aa	3.aa	2.08	b		F

3.d7	3.da	3.da	3.bd	a	$(X) + l_j(F) \Rightarrow Y$ $(M) + g_j(F) \Rightarrow M$	A_4
3.d8	3.db	3.db	3.be	a		
3.d9	3.dd	3.dd	3.be	a		
3.da	2.20	3.a6	3.a8	9		
3.db	3.a7	3.a8	2.20	d		
3.dc	2.20	2.20	3.a9	8		
3.dd	3.a9	3.a9	2.20	d		
3.de	3.d7	3.b5	3.db	7		
3.df	3.da	3.da	3.ae	a		
3.e0	3.db	3.db	3.ad	a		
3.e1	3.dd	3.dd	3.ad	a		
3.e2	3.d2	3.aa	2.00	6		p_1
3.e3	3.e4	3.a8	3.a7	4	$h \Rightarrow <l_j>$	A_5
3.e4	3.a9	3.a9	2.0d	8	$2g_j \Rightarrow <g_j>$	
3.e5	3.d3	3.aa	2.1a	6		p_2
3.e6	3.e8	3.e7	3.c0	4	$(X) + (M) \Rightarrow Y$	A_6
3.e7*	0.00	0.00	0.00	0		
3.e8	3.e7	3.e7	3.bf	a		
3.e9	3.e7	3.b5	3.e7	7		
3.ea	2.10	2.12	3.ac	4		θ
3.eb	3.a7	3.a7	2.0d	8	$2h \Rightarrow <h>$	A_7
3.ec	3.b7	3.b8	3.c2	4	$(Y) \Rightarrow U$	
3.ed	3.b7	3.b1	3.b8	7		
3.ee	3.c8	3.ac	3.ee	4	Обращение к K_4	K_{4_2}
3.ef	3.b7	3.b8	3.c3	4	$(Y) \Rightarrow Z$	3_{1_2}
3.f0	3.b7	3.b2	3.b8	7		
3.f1	3.a6	3.a6	3.a7	c	$t - h \Rightarrow <t>$	A_8
3.f2	3.a7	3.a7	2.0d	9	$\frac{1}{2} h \Rightarrow <h>$	
3.f3	3.b7	3.b8	3.c4	4	$(U) \Rightarrow Y$	
3.f4	3.b7	3.b3	3.b8	7		

3.f5	3.c8	3.ac	3.f5	4	Обращение к K_4	K_4
3.f6	3.c8	3.ac	3.f6	4	Обращение к K_4	K_4
3.f7*	0.00	0.02	3.eb	0	Обращение к H_2	H_2
3.f8	3.fd	2.20	3.b6	7		p_3
3.f9	3.b7	3.b8	3.c5	4	$(X) \Rightarrow Z$	A_9
3.fa	3.b7	3.b4	3.b8	7		
3.fb	3.a6	3.a6	3.a7	c	$t-h \Rightarrow < t >$	
3.fc	0.00	0.00	3.f1	0		ω
3.fd	3.ff	3.c6	2.20	7		p_4
3.fe	3.a7	3.a7	2.0d	8	$2h \Rightarrow < h >$	A_{10}
3.ff*	0.00	0.00	0.00	0	Обращение к H_3	H_3

Стандартные константы, используемые в программе:

2.00	0.00	0.00	0.00	0	0
2.05	0.01	0.00	0.00	1	$(2^{-10})_ф$
2.06	0.00	0.01	0.00	1	$(2^{-20})_ф$
2.08	0.00	0.00	0.00	3	$(2^{-33})_ф$
2.0d	2.04	0.00	0.00	1	$\left(\frac{1}{2}\right)_п$
2.0e	0.00	0.00	3.ff	0	Выделитель первого адреса
2.19	2.45	0.00	0.00	1	$(10)_п$
2.1a	0.00	0.00	0.00	5	$(2^{-32})_ф$

Константы, зависящие от n, полученные в результате действия оператора Φ_1

3.ad	6n	6n	0.00	0	Константы переадресации
3.ae	0.00	6n	0.00	0	
3.af	3.cd	3.a9 — 6n	2.00	4	Константы сравнения
3.b0	3.c9	3.a8 — 6n	3.a7 — 6n	4	
3.b1	3.ed	3.aa — 6n	3.a7 — 6n	4	
3.b2	3.f0	3.ab — 6n	3.a7 — 6n	4	
3.b3	3.f4	3.a7 — 6n	3.aa — 6n	4	
3.b4	3.fa	3.ab — 6n	3.a8 — 6n	4	
3.b5	3.a7 — 6n	3.a8 — 6n	2.20	d	

§ 2. Стандартная программа для решения системы линейных алгебраических уравнений

Описание алгоритма

Приводимая ниже программа реализует метод главных элементов, состоящий в следующем.

Пусть дана система n совместных линейных алгебраических уравнений с n неизвестными:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad (i=1, 2, \ldots, n). \qquad \text{(VII. 3)}$$

Требуется путем эквивалентных преобразований привести систему к виду

$$x_i = b_i^* \qquad (i=1, 2, \ldots, n),$$

где b_i^* и будут решением системы (VII. 3). Эти преобразования будем выполнять последовательными этапами, причем для большего однообразия записи положим $a_{ij}^{(1)} = a_{ij}$ $(i=1, 2, \ldots, n;\ j=1, 2, \ldots, n)$ и $a_{i,n+1}^{(1)} = b_i$ $(i=1, 2, \ldots, n)$, а расширенную матрицу системы обозначим через $\underline{A}^{(1)}$.

Среди элементов $a_{ij}^{(1)}$ $(i=1, 2, \ldots, n;\ j=1, 2, \ldots, n)$ выберем элемент наибольший по абсолютной величине — *главный* элемент матрицы $A^{(1)}$. Пусть это будет элемент $a_{i_1j_1}^{(1)}$; тогда i_1-ю строку будем называть *главной* строкой.

Теперь перейдем от матрицы $A^{(1)}$ к матрице $A^{(2)}$, элементы которой $a_{ij}^{(2)}$ получаются из элементов матрицы $A^{(1)}$ по формулам:

$$a_{i,j}^{(2)} = a_{i,j}^{(1)} : a_{i_1,j_1}^{(1)} \qquad (j=1, 2, \ldots, n+1),$$

$$a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i,j}^{(1)} a_{i_1,j_1}^{(1)}}{a_{i_1,j_1}^{(1)}} \qquad \text{(VII. 4)}$$

$$(1 \leqslant i \leqslant n,\ i \neq i_1;\ j=1, 2, \ldots, n+1).$$

В результате этих преобразований получим $a_{j_1,j_1}^{(2)} = 0$ при $i \neq i_1$ и $a_{i_1,j_1}^{(2)} = 1$. Это первый этап обработки матрицы.

Найдем теперь наибольший по абсолютной величине элемент среди элементов $a_{ij}^{(2)}$ $(1 \leqslant i \leqslant n,\ i \neq i_1;\ j=1, 2, \ldots, n)$ — главный элемент матрицы $A^{(2)}$ — и перейдем от

матрицы $A^{(2)}$ к матрице $A^{(3)}$ путем преобразования элементов по формулам, аналогичным формулам (VII. 4). Это будет вторым этапом обработки матрицы.

Вообще на k-м этапе находим элемент $a_{i_k j_k}^{(k)}$ — наибольший по абсолютной величине среди элементов $a_{ij}^{(k)}$ ($1 \leqslant l \leqslant n$; $i \neq i_1, i_2, \ldots, i_{k-1}$; $j = 1, 2, \ldots, n$). Это главный элемент матрицы $A^{(k)}$. От матрицы $A^{(k)}$ переходим к матрице $A^{(k+1)}$, элементы которой $a_{ij}^{(k+1)}$ получаются из элементов $a_{ij}^{(k)}$ матрицы $A^{(k)}$ по формулам:

$$a_{i_k j}^{(k+1)} = a_{i_k j}^{(k)} : a_{i_k j_k}^{(k)} \quad (j = 1, 2, \ldots, n+1),$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{i_k j}^{(k)} a_{i j_k}^{(k)}}{a_{i_k j_k}^{(k)}} \quad \text{(VII. 5)}$$

($1 \leqslant n$, $i \neq i_k$; $j = 1, 2, \ldots, n+1$).

Это верно для любого $k = 1, 2, \ldots, n$. Строки с номерами $i_1, i_2, \ldots, i_{k-1}$ будем называть *особыми* строками матрицы $A^{(k)}$. Главный элемент матрицы $A^{(k)}$ ищется среди элементов всех ее строк, кроме особых. При преобразовании матрицы $A^{(k)}$ ее столбцы с номерами $j_1, j_2, \ldots, j_{k-1}$ не меняются, поэтому после n этапов обработки исходной матрицы получим матрицу $A^{(n+1)}$, у которой на месте главных элементов матриц $A^{(1)}, A^{(2)}, \ldots, A^{(n)}$ стоят единицы, а на месте остальных элементов, кроме элементов последнего столбца, стоят нули. Тем самым система (VII. 3) приведется к виду

$$x_{j_k} = a_{i_k, n+1}^{(n+1)} \quad (k = 1, 2, \ldots, n).$$

В последнем столбце расположены искомые значения неизвестных системы в некотором порядке, определяемом следующим правилом: в i_k-й строке находится значение j_k-го неизвестного, где i_k, j_k — индексы главного элемента матрицы $A^{(k)}$ ($k = 1, 2, \ldots, n$). Расположив значения неизвестных в порядке возрастания их номеров, мы найдем решение системы (VII. 3).

Логическая схема

В программе используется метод логических шкал [2]. *Указателем шкалы строк* будем называть код, эквивалентный числу $(2^{l-34})_{\Phi_1}$, где l является номером обрабатываемой строки. На k-м этапе обработки матрицы указателю шкалы главных строк будет соответствовать код, эквивалентный числу $(2^{k-34})_{\Phi_1}$, шкале особых строк будет соответствовать код, эквивалентный числу

$$(2^{i_1-34} + 2^{i_2-34} + \ldots + 2^{i_{k-1}-34})_{\Phi_1}.$$

Элементы матриц $A^{(k)}$ при $k = 2, 3, \ldots, n$ будем помещать в те же ячейки, в которых первоначально хранились соответствующие элементы матрицы $A^{(1)}$. Условимся обозначать через $\alpha(l, j)$ адрес ячейки, в которой хранится элемент $a_{ij}^{(k)}$ при $k = 1, 2, \ldots, n$.

Кроме того, для упрощения логической схемы условимся к операторам переадресации и восстановления относить команды, изменяющие состояние счетчика или указателя шкалы по данному параметру. С учетом этих замечаний логическая схема программы имеет следующий вид:

$$O_1(k=1) \dagger L_1 L_2 F_6(k) p_1(k \geqslant n+1) \dagger 10.$$

Обобщенный оператор L_1 устанавливает режим фиксированной запятой и выбирает главный элемент матрицы $A^{(k)}$.

Обобщенный оператор L_2 корректирует шкалу особых строк в соответствии с найденным главным элементом матрицы $A^{(k)}$, устанавливает режим плавающей запятой и получает элементы матрицы $A^{(k+1)}$ по формулам (VII. 5).

Обобщенный оператор L_3 производит упорядочение найденных значений неизвестных x_j и переносит их на нужное место.

Обобщенный оператор L_1 описывается следующей логической схемой:

$$\Phi_1 \mathcal{З}_1 O_2(l=1) \dagger \underline{p_2 \, O_3(j=1) \dagger p_3(l, j) \, \mathcal{З}_2 F_1(j)} \rightarrow$$

$$\rightarrow \underline{p_4(j \geqslant n+1) \dagger F_2(l) \, \underline{F_3(l) \, p_5(l \geqslant n+1) \dagger \underline{F_2^*(l) \omega}}}$$

Оператор Φ_1 устанавливает режим фиксированной запятой и формирует оператор $p_3(1,1)$.

Оператор $\mathcal{Z}_1$ засылает нуль на место главного элемента матрицы $A^{(k)}$.

Оператор p_2 проверяет, отличается ли рассматриваемая строка от особой.

Оператор $p_3(i, j)$ сравнивает очередной элемент матрицы $A^{(k)}$ с наибольшим по абсолютной величине из рассмотренных ранее элементов матрицы $A^{(k)}$. Если очередной элемент матрицы $A^{(k)}$ по абсолютной величине меньше наибольшего из ранее рассмотренных элементов, то управление передается оператору $F_1(j)$.

Оператор $\mathcal{Z}_2$ посылает элемент $a_{ij}^{(k)}$ на место главного, а также фиксирует в стандартных ячейках показания счетчиков параметров i и j, указателя шкалы обрабатываемой строки и величину $\alpha(i, j)$. После окончания работы оператора L_1 в этих ячейках будут находиться величины $a_{i_k, j_k}^{(k)}$, $(i_k \cdot 2^{-20})_\Phi$, $(j_k \cdot 2^{-20})_\Phi$, $(2^{k-34})_\Phi$ и $(\alpha(i_k, j_k) \cdot 2^{-20})_\Phi$.

Оператор $F_2(i)$ переадресует оператор $p_3(i, j)$ на единицу в случае не особой строки, а оператор $F_2^*(i)$ переадресует оператор $p_3(i, j)$ на $n+1$ в случае особой строки.

Оператор $F_3(i)$ изменяет остальные величины, зависящие от параметра i.

Обобщенный оператор L_2 описывается следующей логической схемой:

$$A_1 \Phi_2(i = i_k) \mathcal{Z}_3(i) \Phi_3 L_4(i) \rightarrow \underline{\rightarrow F_3(k)} \; \underline{\rightarrow F_4^*(i)} \; \omega_2,$$

$$\rightarrow O_4(i = 1) \vee p_6 \; L_5(i, s, i) \; F_4(i) \; p_7(i \geqslant n+1)!$$

Оператор A_1 добавляет $(2^{k-34})_\Phi$ к шкале особых строк.

Оператор $\Phi_2(i = i_k)$ формирует оператор $\mathcal{Z}_3(i_k)$.

Оператор $\mathcal{Z}_3(i)$ засылает величину $(j_k \cdot 2^{-20})_\Phi$ в рабочую ячейку с адресом $\pi + i$.

Оператор Φ_3 формирует операторы $L_4(i_k)$, $L_5(i_k, j_k, 1)$, а также константы, необходимые для их работы, и устанавливает режим плавающей запятой.

Обобщенный оператор $L_4(i)$ получает элементы i-й строки матрицы $A^{(k+1)}$ по первой из формул (VII. 5).

Оператор $O_4(i = 1)$ засылает величину $(2^{-33})_\Phi$ на место указателя шкалы обрабатываемой строки и величину $(2^{-20})_\Phi$ — на место счетчика по параметру i.

Оператор p_6 проверяет, отличается ли обрабатываемая строка от главной.

Обобщенный оператор $L(i, s, i)$ получает элементы i-й строки матрицы $A^{(k+1)}$ по второй формуле (VII. 5), где i и s являются индексами главного элемента.

Оператор $F_4^*(i)$ производит дополнительную переадресацию оператора $L(i, s, i)$ в случае главной строки.

Обобщенные операторы L_3, $L_4(i)$ и $L_5(i, s, i)$ описывают работу простейших циклов, поэтому подробную схему их работы мы приводить не будем.

Краткая характеристика программы

Программа занимает 87 ячеек памяти (77 команд и 10 констант); $n+1$ следующих за ней ячеек используются в качестве рабочих; $n(n+1)$ ячеек занимает матрица коэффициентов системы.

Число тактов для системы порядка n: максимальное — $(10n^3+24n^2+29n+7)$, минимальное — $(7n^3+21n^2+35n+7)$.

Используются стандартные константы *2.00*, *2.05* — *2.08*, *2.0e* и стандартный выход *2.10* — *2.12*.

Рабочие ячейки: *2.20* — *2.2a* и π, $\pi+1$, ..., $\pi+n$, где π — адрес последней ячейки, занятой программой.

Инструкция по пользованию программой

Описываемая здесь программа составлена как стандартная программа. Она пригодна для решения системы линейных алгебраических уравнений любого порядка, не превышающего 22, может быть поставлена на любое место в памяти и последней командой передает управление стандартному выходу.

Чтобы использовать программу, надо расположить элементы матрицы $a_{11}, a_{12}, \ldots, a_{1,n+1}, a_{21}, \ldots, a_{n,n+1}$ в последовательных ячейках памяти, начиная с ячейки $\alpha(1,1)$. Такое расположение элементов матрицы (по строкам) позволяет легко пропускать при обработке главную или особую строку и

несколько упрощает переадресации переменных команд. Некоторое неудобство создается при решении серии систем, различающихся лишь свободными членами, так как при таком расположении матрицы свободные члены стоят не в последовательных ячейках памяти. Однако поставить на нужные места вычисленные в машине или вводимые с перфоленты свободные члены не представляет труда с помощью дополнительной программы.

Значения неизвестных системы записываются в последовательные ячейки памяти $\delta+1$, $\delta+2$, ..., $\delta+n$: в ячейку $\delta+i$ записывается значение i-го неизвестного. Положение матрицы и ячеек, в которые записывается решение, выбираются в памяти произвольно перед началом работы.

Для решения конкретной системы дополнительно к программе и матрице надо ввести информацию, состоящую из трех чисел:

0.00 $n+1$ *0.00 1* (в ячейку *2.2 b*),
0.00 $\alpha(1,1)-1$ *0.00 1* (в ячейку *2.2 c*),
0.00 δ *0.00 1* (в ячейку *2.2 d*).

Эта информация указывает порядок решаемой системы, место матрицы в памяти и место, на которое надо поставить решение системы.

С помощью этой программы можно решать системы уравнений до 18-го порядка включительно, не обращаясь к магнитному барабану. Использование магнитного барабана позволяет решать системы до 22-го порядка включительно.

При этом следует отметить, что если при работе приходится пользоваться магнитным барабаном, то на барабане удобнее записывать матрицу, так как к ячейкам, в которых стоят элементы матрицы, программа обращается реже, чем к ячейкам, хранящим команды программы. Это ускоряет, следовательно, работу машины.

В том случае, когда решение системы линейных алгебраических уравнений является самостоятельной задачей, надо составить короткую ведущую программу, включающую в себя перевод исходной матрицы коэффициентов в двоичную систему счисления, обращение к программе решения системы уравнений и печать решения, переведенного в десятичную систему счисления.

Программа

	0.56	*0.56*	*0.00*	*0*		
	0.00	*0.2f*	*2.4d*	*f*		
3.00	*3.01*	*2.20*	*2.00*	*4*	$0 \Rightarrow 2.20$	O_1
3.01	*3.02*	*2.21*	*2.06*	*4*	$(2^{-20})_\Phi \Rightarrow 2.21$	
3.02	*0.00*	*0.18*	*3.03*	*0*	Режим ФЗ	Φ_1
3.03	*3.0a*	*3.4f*	*2.2c*	*b*	Формирование (*3.0a*)	
3.04	*3.05*	*2.22*	*2.00*	*4*	$0 \Rightarrow 2.22$	$З_1$
3.05	*3.06*	*2.23*	*2.08*	*4*	$(2^{-33})_\Phi \Rightarrow 2.23$	O_2
3.06	*3.07*	*2.24*	*2.06*	*4*	$(2^{-20})_\Phi \Rightarrow 2.24$	
3.07	*2.25*	*2.20*	*2.23*	*f*		p_2
3.08	*3.49*	*2.00*	*2.25*	*7*		
3.09	*3.0a*	*2.25*	*2.06*	*4*	$(2^{-20})_\Phi \Rightarrow 2.25$	O_3
*3.0a**	*0.00*	*0.00*	*0.00*	*0*	$p(\lvert a_{ij}\rvert \geqslant \lvert(2.22)\rvert)$	p_3
3.0b	*2.26*	*3.0a*	*3.56*	*f*	$(\alpha(i, j)\cdot 2^{-20})_\Phi \Rightarrow 2.26$	$З_2$
3.0c	*3.0d*	*2.27*	*2.24*	*4*	$(i\cdot 2^{-20})_\Phi \Rightarrow 2.27$	
3.0d	*3.0e*	*2.28*	*2.25*	*4*	$(j\cdot 2^{-20})_\Phi \Rightarrow 2.28$	
3.0e	*3.0f*	*3.50*	*2.26*	*b*	Формирование (*3.0f*)	
*3.0f**	*0.00*	*0.00*	*0.00*	*0*	$a_{ij}^{(k)} \Rightarrow 2.22$	
3.10	*3.11*	*2.29*	*2.23*	*4*	$(2^{i-34})_\Phi \Rightarrow 2.29$	
3.11	*2.25*	*2.25*	*2.06*	*b*	Изменение счетчика j	F_1
3.12	*3.0a*	*3.0a*	*2.06*	*b*	Переадресация (*3.0a*)	
3.13	*3.0a*	*2.25*	*2.2b*	*7*	$p_4(j \geqslant n+1)$	p_4
3.14	*3.0a*	*3.0a*	*2.06*	*b*	Переадресация (*3.0a*)	F_2
3.15	*2.23*	*2.23*	*2.23*	*b*	Сдвиг указателя строки	F_3
3.16	*2.24*	*2.24*	*2.06*	*b*	Изменение счетчика i	

3.17	3.07	2.24	2.2b	7	$p_5(i \geqslant n+1)$	p_5
3.18	2.20	2.20	2.29	b		A_1
3.19	3.1a	3.51	2.27	a	Формирование (3.1a)	Φ_2
3.1a*	0.00	0.00	0.00	0	$(j_k \cdot 2^{-20})_\Phi \Rightarrow \pi + l_k$	$\Im_3$
3.1b	2.23	2.26	2.28	a	$((\alpha(l_k, 1)-1) \cdot 2^{-20})_\Phi \Rightarrow \Rightarrow 2.23$	Φ_3
3.1c	2.24	2.2c	2.28	b	$(\alpha(1, j_k) \cdot 2^{-20})_\Phi \Rightarrow 2.24$	
3.1d	2.25	2.23	2.05	8	$((\alpha(l_k, 1)-1)(2^{-20}+$	
3.1e	2.25	2.23	2.25	b	$+2^{-10}))_\Phi \Rightarrow 2.25$	
3.1f	2.26	2.2c	2.05	8	$((\alpha(1,1)-1)(2^{-20}+$	
3.20	2.26	2.2c	2.26	b	$+2^{-10}))_\Phi \Rightarrow 2.26$	
3.21	2.27	2.2c	2.05	9	$((\alpha(1,1)-1) \cdot 2^{-30})_\Phi \Rightarrow \Rightarrow 2.27$	
3.22	2.28	2.2b	2.05	8	$((n+1)(2^{-20}+$	
3.23	2.28	2.2b	2.28	b	$+2^{-10}))_\Phi \Rightarrow 2.28$	
3.24	3.57	2.2b	2.05	9	$((n+1) \cdot 2^{-30})_\Phi \Rightarrow \pi$	
3.25	3.2a	3.52	2.25	a	Формирование (3.2a), (3.31) и (3.35)	
3.26	3.31	3.50	2.24	b		
3.27	3.35	3.53	2.26	a		
3.28	0.00	0.02	3.29	0	Режим ПЗ	
3.29	2.24	3.2a	2.28	a	Формирование константы	
3.2a*	0.00	0.00	0.00	0	$a_{ij}^{(k+1)} \Rightarrow \alpha(l_k, j)$	L_4
3.2b	3.2a	3.2a	3.4d	a	Переадресация (3.2a)	
3.2c	3.2a	3.2a	2.24	7	$p(j \geqslant n+1)$	
3.2d	3.2e	2.24	2.08	4	$(2^{-33})_\Phi \Rightarrow 2.24$	O_4
3.2e	3.2f	2.25	2.06	4	$(2^{-20})_\Phi \Rightarrow 2.25$	
3.2f	2.26	2.29	2.24	f		p_6
3.30	3.4b	2.00	2.26	7		

3.31*	0.00	0.00	0.00	0	$a_{ij_k} \Rightarrow 2.22$	L_5
3.32	3.34	3.54	2.23	b	Формирование (3.34)	
3.33	2.2a	3.34	2.2b	b	Формирование константы	
3.34*	0.00	0.00	0.00	0	$a_{ij}^{(k+1)} \Rightarrow \alpha(l, j)$	
3.35*	0.00	0.00	0.00	0		
3.36	3.34	3.34	2.06	b	Переадресация (3.34) и (3.35)	
3.37	3.35	3.35	3.4d	a		
3.38	3.34	3.34	2.2a	7	$p(j \geqslant n+1)$	
3.39	3.31	3.31	2.2b	b		F_4
3.3a	2.24	2.24	2.24	b		
3.3b	2.25	2.25	2.06	b		
3.3c	3.2f	2.25	2.2b	7	$p_1(i \geqslant n+1)$	p_7
3.3d	2.21	2.21	2.06	b	Изменение счетчика k	F_5
3.3e	3.02	2.21	2.2b	7	$p_1(k \geqslant n+1)$	p_1
3.3f	2.20	3.55	2.2d	a	Подготовка формирования (3.45)	L_3
3.40	2.20	2.20	2.27	a		
3.41	3.44	3.4e	2.07	a	Восстановление (3.44)	
3.42	2.21	3.4e	3.57	a	Формирование константы	
3.43	2.20	2.20	3.57	a	Переадресация (2.20)	
3.44*	3.45	2.20	3.58*	a	Формирование (3.45)	
3.45*	0.00	0.00	0.00	0	$a_{k,n+1}^{(n+1)} \Rightarrow \delta + j_k$	
3.46	3.44	3.44	2.07	a	Переадресация (3.44)	
3.47	3.43	3.44	2.21	7	$p(l_k \geqslant n+1)$	
3.48	0.00	0.00	2.10	0		0
3.49	3.0a	3.0a	2.2b	b	Переадресация (3.0a)	F_2^*
3.4a	0.00	0.00	3.15	0		ω_1
3.4b	3.35	3.35	2.28	a	Переадресация (3.35)	F_3

3.4c	*0.00*	*0.00*	*3.39*	*0*	ω_2
3.4d	*0.01*	*0.01*	*0.00*	*1*	Константы
3.4e	*3.45*	*2.20*	*3.57*	*a*	
3.4f	*3.11*	*0.01*	*2.22*	*7*	
3.50	*2.22*	*0.00*	*2.00*	*b*	
3.51	*3.1b*	*3.57*	*2.28*	*4*	
3.52	*0.01*	*0.01*	*2.22*	*8*	
3.53	*0.01*	*0.01*	*2.26*	*c*	
3.54	*2.26*	*0.01*	*2.22*	*9*	
3.55	*3.46*	*0.00*	*0.00*	*4*	
3.56	*0.00*	*3.ff*	*0.00*	*1*	

Стандартные константы, используемые в программе:

2.00	*0.00*	*0.00*	*0.00*	*0*	$(-0)_п$
2.05	*0.01*	*0.00*	*0.00*	*1*	$(2^{-10})_ф$
2.06	*0.00*	*0.01*	*0.00*	*1*	$(2^{-20})_ф$
2.07	*0.00*	*0.00*	*0.01*	*1*	$(2^{-30})_ф$
2.08	*0.00*	*0.00*	*0.00*	*3*	$(2^{-33})_ф$

Примечание. Переменные команды *3.0a*, *3.0f*, *3.1a*, *3.2a*, *3.31*, *3.34*, *3.35* и *3.45* формируются в процессе работы программы в соответствии с заданной информацией и индексами главного элемента, выбранного на данном этапе. На перфоленте на соответствующих местах пробиты нули. Нули пробиты также на месте команды *3.44*.

§ 3. Стандартная программа для вычисления определенных интегралов методом Симпсона

Предлагаемая программа предназначена для вычисления определенных интегралов $I=\int_{x_1}^{x_2} f(x)\,dx$ в режиме плавающей запятой, причем вычисление $f(x)$ осуществляется нестандартной подпрограммой по формулам конкретной задачи.

Обращение к стандартной программе производится тремя командами N, $N+1$ и $N+2$ основной программы задачи.

После работы программы величина интеграла помещается в ячейку *2.20* и управление передается команде $N+3$.

Команды N, $N+1$, $N+2$ должны иметь вид:

N	m_3	*2.20*	N	*4*
$N+1$	*0.00*	n_2	n_1	*0*
$N+2$	l_3	l_2	l_1	*0*

где m_3 — вход программы,
n_1 — адрес ячейки, содержащей величину x_1,
n_2 — адрес ячейки, содержащей величину x_2,
l_1 — адрес ячейки, содержащей величину шага интегрирования h,
l_2 — адрес ячейки обратной связи с программой вычисления $f(x)$ (это означает, что выход из программы вычисления $f(x)$ должен осуществляться командой вида *2.10 2.12* l_2 *4*),
l_3 — вход программы для вычисления $f(x)$.

Программа для вычисления $f(x)$ должна брать x из ячейки *2.20* и помещать вычисленную величину $f(x)$ в ячейку *2.20*.

Описание алгоритма

Вычисление интеграла $I=\int_{x_1}^{x_2} f(x)\,dx$ осуществляется по формуле Симпсона

$$I\approx\frac{2}{3}h\left[\frac{1}{2}f(x_1)+2f(x_1+h)+f(x_1+2h)+\ldots\right.$$
$$\ldots+2f(x_1+(2k-1)h)+f(x_1+2kh)+\ldots$$
$$\left.\ldots+2f(x_1+(2n-1)h)+\frac{1}{2}f(x_1+2nh)\right],$$

причем

$$x_2-(x_1+2nh)\leqslant h.$$

Логическая схема

Логическая схема программы очень проста:

$$\Phi K 0.$$

Оператор Φ формирует в программе команду обращения на вычисление $f(x)$, засылает первоначальное положение «переключателя» (см. ниже) и пересылает величины x_1, x_2 и h в оперативные ячейки программы.

Оператор K осуществляет вычисление интегралов по формуле Симпсона. Наиболее интересна в реализации этого оператора команда переключателя *3.1 d* (см. текст программы), которая при первом обращении к ней передает управление команде *3.19*, а в дальнейшем работает как двухпозиционный переключатель, передающий управление то *3.1e*, то *3.1f*; этот переключатель обеспечивает правильный выбор коэффициентов $^1/_2$, 2, 1, 2, 1 ... в формуле Симпсона.

Ввиду того, что выход из программы может быть осуществлен лишь после четного числа шагов, в программе имеется сравнение на четность шага — команда *3.20*.

Оператор 0 передает управление стандартному выходу, обеспечивая при этом выход к команде $N+3$ основной программы (см. начало настоящего параграфа).

Краткая характеристика программы

Программа работает в режиме плавающей запятой.

Количество занятых ячеек 41.

Число тактов на шаге в среднем 7 (на нечетном шаге 6, на четном 8).

Используемые ячейки со стандартными константами: *2.00*, *2.07*, *2.0d*, *2.0e*, *2.10 — 2.13*.

Используемые стандартные рабочие ячейки: *2.20*, *2.21*.

Дополнительные сведения:

1. Для пользования программой необходимо иметь в памяти программу для вычисления подынтегральной функции;

2. Программа может быть помещена в любое место запоминающего устройства по тем же правилам, которые применяются к стандартным подпрограммам.

3. Следующие непосредственно за программой две ячейки используются в качестве рабочих,

Программа

	0.27	*0.26*	*0.00*	*0*		
	0.00	*3.43*	*3.20*	*4*		
3.00	*2.20*	*2.20*	*2.0e*	*f*		
3.01	*3.02*	*3.03*	*2.20*	*b*		
*3.02**	*0.00*	*0.00*	*0.00*	*0*		
3.03	*3.0c*	*2.21*	*0.01*	*4*		
3.04	*3.1f*	*3.1d*	*3.05*	*4*	Константы переключателя	
3.05	*3.1e*	*3.1d*	*3.04*	*4*		
3.06	*3.19*	*3.1d*	*3.05*	*4*		
3.07	*0.00*	*0.00*	*3.1b*	*4*	Константы для формирования команд	
3.08	*3.1b*	*3.28*	*0.00*	*4*		
3.09	*3.27*	*0.00*	*2.00*	*d*		
3.0a	*3.02*	*3.18*	*0.00*	*4*		
3.0b	*2.16*	*0.00*	*0.00*	*1*	$(1,5)_\text{п}$	Φ
3.0c	*3.12*	*3.1d*	*3.06*	*4*		
3.0d	*2.20*	*2.21*	*2.0e*	*f*		
3.0e	*2.21*	*2.21*	*2.20*	*a*		
3.0f	*3.1b*	*2.21*	*3.07*	*b*		
3.10	*3.15*	*3.08*	*2.20*	*b*		
3.11	*3.15*	*2.20*	*3.18*	*4*		
3.12	*2.20*	*2.21*	*2.0e*	*f*		
3.13	*2.21*	*2.21*	*2.20*	*a*		
3.14	*3.15*	*3.09*	*2.21*	*a*		
*3.15**	*0.00*	*0.00*	*0.00*	*0*		
3.16	*3.18*	*3.0a*	*2.20*	*b*		
3.17	*3.02*	*3.02*	*2.13*	*a*		
*3.18**	*0.00*	*0.00*	*0.00*	*0*		
3.19	*3.15*	*2.20*	*2.0d*	*g*	$\frac{1}{2}f(x_1) \Rightarrow 3.15$	
3.1a	*3.1b*	*2.20*	*3.18*	*4*	$x \Rightarrow 2.20$	K
*3.1b**	*0.00*	*0.00*	*0.00*	*0*	$f(x) \Rightarrow 2.20$	
3.1c	*3.18*	*3.18*	*3.28*	*d*	$x+h \Rightarrow (x)$	

Жоголев Евгений Андреевич,
Росляков Геннадий Степанович,
Трифонов Николай Павлович,
Шура-Бура Михаил Романович.

Система стандартных подпрограмм

Редактор *Ю. М. Безбородов.*

Технический редактор *С. Н. Ахламов.*
Корректор *О. А. Сигал.*

Сдано в набор 1/VII 1958 г. Подписано к печати 4/X 1958 г. Бумага 84×108 1/32. Физ. печ. л. 7,25. Услови. печ. л. 11,89. Уч.-изд. л. 12,20. Тираж 8000 экз. Т-08269.
Цена книги 6 руб. 10 коп. Заказ № 3293.

Государственное издательство физико-математической литературы
Москва, В-71, Ленинский проспект, 15.

Типография № 2 им. Евг. Соколовой УПП Ленсовнархоза.
Ленинград, Измайловский пр., 29.