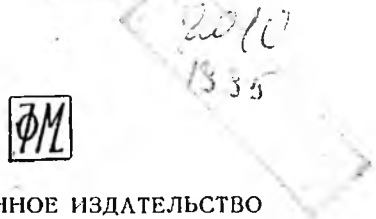
Б. В. ГНЕДЕНКО, В. С. КОРОЛЮК, Е. Л. ЮЩЕНКО

ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

ИЗДАНИЕ ВТОРОЕ
СТЕРЕОТИПНОЕ

Допущено Министерством
высшего и среднего специального образования РСФСР
в качестве учебного пособия
для высших учебных заведений



ГОСУДАРСТВЕННОЕ ИЗДАТЕЛЬСТВО
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА 1963

518
Г 56

АННОТАЦИЯ

Книга «Элементы программирования» может служить руководством по программированию для цифровых автоматических машин. В ней нашли отражение исследования в области автоматизации программирования, решения логических задач на цифровых автоматических машинах, а также операторный метод, предложенный А. А. Ляпуновым.

Книга предназначена для студентов университетов, втузов и может быть полезна для работников научно-исследовательских институтов, занимающихся вопросами программирования.

Книга может служить учебным пособием при подготовке кадров программистов.

Борис Владимирович Гнеденко, Владимир Семенович Королюк, Екатерина Логвиновна Ющенко.

ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

М., Физматгиз, 1963 г., 348 стр. с илл.

Редактор *Соловьева Л. А.*

Техн. редактор *Мурашова И. Я.* Корректор *Андрианова Л. Е.*

Печать с матриц. Подписано к печати 1/XII 1962 г. Бумага 60×90¹/₁₆. Физ. печ. л. 21,75. Услови. печ. л. 21,75. Уч.-изд. л. 17,34. Тираж 50 000 экз. Цена книги 62 коп. Заказ № 1166.

Государственное издательство физико-математической литературы.
Москва, В-71, Ленинский проспект, 15.

Типография № 2 им. Евг. Соколовой УЦБ и ПП Ленсовнархоза.
Ленинград, Измайловский пр., 29.

ОГЛАВЛЕНИЕ

ОТ АВТОРОВ

Настоящая книга написана на основании опыта работы авторов на универсальных цифровых автоматических машинах, чтения курсов лекций по программированию, проведения соответствующих семинаров, руководства практикой студентов, а также участия в проектировании новых вычислительных машин. Отправным пунктом для нашей работы был курс лекций проф. А. А. Ляпунова, который он читал в Московском университете. Содержание нашей книги ясно из оглавления, и на нем нет необходимости останавливаться.

Мы будем искренне благодарны всем читателям, которые найдут возможным прислать нам свои пожелания, замечания и указания на замеченные недостатки.

Гнеденко Б. В., Королюк В. С., Ющенко Е. Л.

ВВЕДЕНИЕ

Прошло лишь немногим более десяти лет с тех пор, как была построена первая автоматическая электронная цифровая машина с программным управлением, но уже можно говорить с полной определенностью, что ее появление знаменовало собой решительный скачок по пути прогресса для большого числа научных и технических проблем нашего времени.

Впервые перед наукой открылась возможность производить колоссальные по масштабам вычислительные работы в исключительно короткие сроки. Производство вычислений диктуется практической необходимостью, и потребности в счете возрастают с каждым годом.

Дело в том, что любые научные и технические разработки, любое строительство — корабля, плотины, турбины, самолета, ракеты, атомного реактора — нуждаются в предварительных, зачастую очень сложных расчетах. Порой вся их сложность состоит лишь в том, что для получения окончательного результата необходимо выполнить миллионы, а то и миллиарды арифметических операций. Для примера укажем на решение систем большого числа линейных алгебраических уравнений. Принципиальная сторона вопроса здесь достаточно хорошо выяснена уже давно, и единственная трудность, которая возникает, — это необходимость осуществления большого числа сложений, вычитаний, умножений и делений. Для систем указанного вида, содержащих 200—300 неизвестных с соответствующим числом уравнений, получение ответа достигается лишь после производства нескольких десятков миллионов элементарных арифметических действий. Даже лучшему вычислителю, в руках которого находятся простейшие вычислительные средства — арифмометры, таблицы и пр., — требуются годы, а то и десятилетия напряженного труда для решения одной-единственной задачи столь простого типа. Конечно, иногда задача допускает разбиение на части, каждая из которых может быть решена независимо одна от другой. В таком случае можно значительно ускорить вычислительный процесс, поручив выполнение независимых групп операций различным вычислителям. Однако

зачастую такое разбиение невозможно и вся задача требует последовательного выполнения операций.

Многие задачи, формальное решение которых еще совсем недавно казалось исключительно далеким от возможности практического использования из-за вычислительных трудностей, в настоящее время в связи с появлением новой быстродействующей вычислительной техники удалось довести до численных расчетов, а тем самым и до непосредственных применений. В инженерной практике вычислительные трудности приводили к тому, что часто инженеры ограничивались весьма приближенными прикидками, а иногда и совсем отказывались от расчетов даже в тех случаях, когда имелась вполне удовлетворительная теория соответствующего явления. Это обстоятельство приводило и приводит, в частности, к тому, что в практике мирятся с неоправданно высокими запасами прочности; в особо же ответственных случаях прибегают к дорогостоящим экспериментам и моделированию. Применение быстродействующих вычислительных машин открывает перед современной техникой невиданные возможности. Достаточно сказать, что лучшие образцы машин сейчас способны производить десятки тысяч умножений многозначных чисел в секунду. Иными словами, машина в секунду производит столько вычислений, сколько не способен произвести даже опытный вычислитель за целый месяц работы. Это обстоятельство не может не оказать глубокого влияния на весь комплекс технических, экономических и естественных наук, стимулируя широкое использование вычислительных методов. Внедрение этих методов позволит выбирать оптимальные варианты решения инженерных задач и тем самым принесет огромный экономический эффект.

Как ни велико значение новой вычислительной техники для производства вычислительных работ, быть может, еще большее значение имеет то, что быстродействующие цифровые вычислительные машины оказалось возможным приспособить к решению задач логического характера. В качестве примера такого рода можно привести удачные попытки осуществления автоматического перевода с одного языка на другой, проводимые в СССР, США и других странах. Оказалось, что на машину удается переложить некоторые функции интеллектуальной деятельности и тем самым освободить человека от монотонных мыслительных операций, совершаемых по определенным правилам. Особенно большие перспективы открываются в связи с использованием вычислительных машин для автоматического управления производственными процессами. Это направление привлекает большое число ученых и инженеров. Чтобы характер этого рода приложений был достаточно отчетливо выяснен, мы

приведем несколько примеров, заимствованных из технической литературы.

Фирма «Сокони мобил ойл компани» использовала электронную вычислительную машину для выбора наивыгоднейшего режима перегонки нефти. Исходными данными при этом являются стоимость на рынке нефтепродуктов, рабочей силы, различных материалов и ряд других факторов. Для составления программы были предварительно математически описаны процессы каталитического крекинга, алкилирования и др. Программа, моделирующая процесс перегонки, включает в себя 6000 команд и чисел и осуществляется машиной за 9 минут, из которых 6 минут затрачиваются на ввод в машину программы.

Фирма «Вест Пенн электрик» создала электронную вычислительную машину для выбора наиболее экономичного режима работы энергосистемы мощностью 1300 *Мвт*, объединяющей десять тепловых электростанций и одну гидростанцию. Общая протяженность линий электропередач 2500 *км* при напряжениях в 66 и 132 *кв*. Машина используется для расчета оптимального режима работы системы при учете потерь в сетях, стоимости топлива на отдельных станциях, коэффициентов полезного действия генераторов, работающих для покрытия нагрузки, и учета других факторов, определяющих стоимость энергии. Полученное решение используется для распределения нагрузок между станциями.

В настоящее время в периодической печати появилось значительное число работ, касающихся применения быстродействующих вычислительных машин к автоматическому управлению металлорежущими станками, доменным процессом, к задачам государственного планирования, для библиографических целей и т. д. Ряд такого рода интересных и важных применений был освещен в докладах на сессии Академии наук СССР, посвященной научным проблемам автоматизации производства [1].

Несомненно, что в указанных направлениях исследовательская работа еще только начинается и даже самое ближайшее будущее принесет необозримые успехи. Но для этого необходима упорная и систематическая работа по изучению и математическому описанию процессов, подлежащих автоматизации. До некоторой степени исследования этого рода представляют собой новую область математической деятельности, которая требует специальной подготовки и специального направления мысли. Несомненно, что уже теперь нужно учесть это при организации университетского образования.

До последнего времени человечество огромную долю своих творческих усилий затрачивало на то, чтобы использовать силы природы для освобождения человека от однообразного утоми-

тельного физического труда путем использования разнообразных машин. Одновременно, но в сравнительно меньшей степени, человечество стремилось и к облегчению умственного труда, к тому, чтобы переложить на те или иные приспособления часть загрузки интеллекта. На протяжении столетий на этом пути удалось сделать немало. Чтобы в этом убедиться, достаточно вспомнить об изобретении книгопечатания, фотографии, звукозаписи, счетных машин и приспособлений и многого другого. Каждое из указанных изобретений перекладывало на машины то, что раньше способен был осуществлять лишь разум человека — накапливать и передавать другим опыт и знания, запоминать образы и звуки, производить вычисления и т. д. Теперь мы стоим на пороге новой эпохи, когда силы природы будут использованы также для увеличения интеллектуальной мощи человечества и для освобождения человека от утомительной однообразной умственной работы. Это даст возможность сосредоточить творческие возможности на нерешенных проблемах, на развитии новых направлений в науке. Электронные вычислительные машины являются важным шагом на этом пути; первые попытки использования их в указанном направлении уже сделаны и оказались исключительно удачными. Дальнейшие возможности представляются нам буквально неисчерпаемыми.

Революционизирующее воздействие быстродействующих цифровых машин сказывается сейчас как на изменении инженерных устройств, так и на установлении новых связей между научными дисциплинами, отстоявшими друг от друга весьма далеко в самом недавнем прошлом, а также и в том, что они послужили мощным толчком для бурного развития новых научных дисциплин. В первую очередь в связи со сказанным нужно упомянуть кибернетику, которую можно определить как науку о способах восприятия, хранения, переработки и использования информации в машинах и в живых организмах. Моделирование некоторых физиологических и мыслительных процессов может оказать существенную помощь как физиологии, так и технике, например при построении управляющих машин. Кибернетика призвана объединить усилия математиков, биологов, физиков, техников, психологов, экономистов и лингвистов и способствовать взаимному развитию соответствующих дисциплин.

Само собой разумеется, что быстродействующие вычислительные машины оказали существенное влияние и на некоторые разделы математики. Прежде всего, возник вопрос о препарировании математических задач для постановки их на машину. Речь идет не только о том, чтобы свести интересующую нас задачу к решению того или иного уравнения или к уже готовой формуле, в которую нужно лишь подставить значения аргументов и произвести необходимые операции. Для того чтобы ма-

шина могла без вмешательства человека пройти весь путь от начальных условий задачи до выдачи численных результатов и прекращения вычислений, необходимо предварительно разложить процесс решения задачи на элементарные операции и тем самым создать условия, при которых она может автоматически в необходимом порядке, шаг за шагом, проделать все требуемые операции. Здесь очень важно подчеркнуть слово автоматически, так как всякое вмешательство человека в огромной степени задерживает производство вычислений. Действительно, если машине на производство каждой операции необходимы лишь десятитысячные и даже стотысячные доли секунды, то человеку на изменение хода вычислений нужны уже минуты. Таким образом, необходимо, чтобы машина не только производила сложение и умножение, но чтобы в ней автоматически происходил переход от одной операции к другой, «запоминание» результатов промежуточных вычислений, выбор чисел из «памяти», изменение характера вычислений, выдача окончательных результатов, своевременное прекращение работы машины. Это осуществляется с помощью устройства программного управления. Новая математическая дисциплина, позволяющая производить необходимое разбиение задачи на элементарные операции, т. е. составлять программу, по которой должна работать машина, получила наименование теории программирования.

Наша цель — изложить в настоящей книге основные идеи этой дисциплины на базе рассмотрения как совершенно элементарных, так и более сложных примеров. В значительной части эти примеры заимствованы нами из практики и встречались нам при решении конкретных задач на быстродействующих вычислительных машинах. Помимо изложения основ программирования, необходимых для каждого программиста, мы считаем необходимым познакомить читателей с новыми идеями программирования, которые разрабатываются в настоящее время. В частности, мы уделяем значительное внимание операторному методу программирования, предложенному А. А. Ляпуновым.

Следует отметить, что в основе вычислительных процессов на машине всегда лежит какой-нибудь метод приближенного решения — замена интеграла через интегральную сумму, замена дифференциального уравнения на уравнение в конечных разностях и т. д. Весь арсенал привычных приближенных методов анализа теперь пущен в ход. При этом обнаружилось, что далеко не каждый из этих методов приспособлен к особенностям автоматического счета. Для применения ранее разработанных методов приближенных вычислений к работе на машине требуется их усовершенствование и развитие. Нужно вспомнить также старые, давно забытые приемы приближенных вычислений;

может случиться, что некоторые из них окажутся удачными. Конечно, необходимо также создание новых приближенных методов; эти исследования нужно форсировать хотя бы потому, что привычные подходы к ряду важных задач приводят к исключительно трудоемким вычислениям даже для современных машин. К таким задачам относятся, в частности, многомерные задачи математической физики. Известно, например, что задачи, связанные с расчетом гетерогенных атомных котлов, при приведении их к уравнениям в конечных разностях в ряде случаев требуют рассмотрения сеток, содержащих миллионы узлов. Общее число элементарных операций, которые при этом должны быть осуществлены, достигает десятков миллиардов. При разработке приближенных методов анализа возникли новые задачи, с которыми ранее почти не приходилось встречаться; в качестве примера можно указать на вопрос исследования устойчивости счета.

В настоящее время еще не установилась окончательная терминология для наименования современных вычислительных машин с программным управлением. Первоначально их называли электронными вычислительными машинами, подчеркивая этим то, что основными элементами этих устройств являлись электронные приборы. Далее, поскольку на смену электронным элементам теперь приходят другие, кажется естественным указывать в первую очередь на то, что эти машины являются не моделирующими устройствами, а выдают цифровые результаты; было внесено предложение называть такие машины автоматическими быстродействующими цифровыми вычислительными машинами (*АБЦВМ*, см. А. А. Ляпунов и Г. А. Шестопал [2]). Мы предлагаем их назвать короче — цифровая автоматическая машина (*ЦАМ*), поскольку представление о быстродействии машины сильно меняется. Так 30 000—100 000 операций в секунду, воспринимаемые сейчас как почти рекордная скорость машины, через несколько лет, когда будут построены машины с сотнями тысяч и даже миллионами операций в секунду, будут казаться не быстродействием, а нормальной скоростью машины. Поэтому мы сохраняем лишь два основных качества в наименовании машины: то, что она цифровая, и то, что она автоматически действует.

ГЛАВА I

ПРИНЦИПЫ УСТРОЙСТВА ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

В настоящей главе рассматриваются арифметические и логические принципы устройства цифровых вычислительных машин. С техническими особенностями осуществления этих принципов и связанными с ними идеями конструирования цифровых автоматических машин (*ЦАМ*) читатель может познакомиться по имеющейся литературе.

Вместе с тем следует отметить, что в настоящее время еще не создана математическая теория конструирования *ЦАМ*, в связи с чем одной из центральных задач современной вычислительной математики и техники является развитие теории синтеза электронных вычислительных и управляющих схем.

§ 1. Выбор системы счисления

В настоящее время для изображения числа употребляется весьма совершенный *позиционный* принцип записи, согласно которому один и тот же числовой символ (цифра) имеет различные значения в зависимости от места, которое он занимает. Такая система записи чисел основана на том, что некоторое число единиц, например c, объединяется в одну единицу второго разряда; объединение c единиц второго разряда составляет единицу третьего разряда, и т. д. Число c носит название *основания счисления*.

В качестве основания счисления можно взять любое целое число, большее единицы. В системе счисления с основанием c любое число a записывается в виде

$$a = \pm \sum_{k=-\infty}^{p-1} \alpha_k c^k, \tag{1}$$

где величины α_k принимают значения 0, 1, 2, ..., $c-1$, а p — некоторое целое число, зависящее от c и от a. Если ввести

обозначение $\beta_n = \alpha_{p-n}$, то число a может быть записано в виде, несколько отличном от (1):

$$a = \pm c^p \sum_{n=1}^{\infty} \beta_n c^{-n}.$$

Число p называется *порядком* числа a (понятно, что значение p зависит от того, какую систему счисления мы употребляем). Порядок числа определяет место запятой в записи числа, а именно, запятая должна стоять перед β_{p+1}. Множитель $\sum_{n=1}^{\infty} \beta_n c^{-n}$ всегда заключен в полуинтервале [0,1]; он носит название *мантиссы числа a.*

В повседневной жизни наиболее употребительна десятичная позиционная система счисления, в которой за основание принято число десять. Любое число в этой системе счисления, как это хорошо известно, может быть записано с помощью расположенных в соответствующем порядке десяти числовых знаков: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Например, число 429,538 является сокращенной записью выражения

$$4 \cdot 10^2 + 2 \cdot 10^1 + 9 \cdot 10^0 + 5 \cdot 10^{-1} + 3 \cdot 10^{-2} + 8 \cdot 10^{-3}.$$

Подобные же сокращенные записи мы будем впоследствии употреблять для изображения чисел, записанных в позиционных системах счисления с другим, недесятичным основанием. Так, например, в восьмеричной системе счисления запись 321,57 означает, что рассматривается число

$$3 \cdot 8^2 + 2 \cdot 8^1 + 1 \cdot 8^0 + 5 \cdot 8^{-1} + 7 \cdot 8^{-2}.$$

В ряде как теоретических, так и практических задач некоторые системы счисления, отличные от десятичной, представляют известные преимущества. Так, например, двоичная система счисления используется в большинстве современных *ЦАМ* как в силу простоты технического осуществления операций над двоичными числами, так и в силу широко открывающейся при этом возможности использовать методы математической логики, в частности, исчисления высказываний. В практике работы *ЦАМ* применяются также и другие системы счисления — восьмеричная, шестнадцатеричная и некоторые другие.

Отметим, что двоичная система счисления для изображения одного и того же диапазона чисел требует меньшего числа элементов машины для их записи, чем десятичная. Действительно, количество чисел, имеющих n разрядов, в системе счисления с основанием c равно $M = c^n$. Необходимое для представления этих чисел число элементов пропорционально $N_c = c \cdot n$. Зафи-

кируем число M и найдем то c, при котором N_c достигает минимума. Из первого написанного нами равенства находим, что

$$n = \frac{\ln M}{\ln c}.$$

Подставив это значение в выражение для N_c, находим:

$$N_c = \frac{c \ln M}{\ln c}.$$

Легко найти, что минимум этого выражения достигается при $c = e = 2{,}71828\ldots$ С рассматриваемой точки зрения самой выгодной системой счисления является троичная. Для изображения всех чисел от 1 до 10^6 в десятичной системе счисления требуется 60 элементов, в двоичной — 40, а в троичной — 38. Однако увеличение числа элементов для записи чисел в двоичной системе по сравнению с троичной невелико. Если число элементов, необходимое для записи в двоичной системе, обозначить N_2, а для записи в троичной — N_3, то

$$\frac{N_2}{N_3} = \frac{2 \ln 3}{3 \ln 2} = \frac{2 \lg_{10} 3}{3 \lg_{10} 2} \approx 1{,}056.$$

Троичная система счисления не получила широкого применения в цифровых машинах в связи с трудностями конструирования достаточно надежных быстродействующих элементов с тремя устойчивыми состояниями. В то же время имеется большое число физических приборов, которые могут в определенных схемах соединений обладать двумя устойчивыми различными состояниями, — конденсаторы, электронные лампы кристаллические триоды и др. Схемы существующих элементов, из которых составляются *ЦАМ*, по своему существу являются двоичными. Конденсатор может быть заряжен и не заряжен, лампа может проводить или не проводить ток. К тому же двоичная система счисления предпочтительнее перед другими системами счисления благодаря исключительной простоте выполнения арифметических операций над двоичными числами.

Некоторым неудобством двоичной системы счисления в *ЦАМ* (как, впрочем, и всякой другой системы счисления, отличной от десятичной) является необходимость перевода исходных данных из десятичной в двоичную систему при вводе их в машину и обратного перевода из двоичной системы в десятичную при выводе результатов вычислений. Эта необходимость обусловлена тем, что десятичная система счисления является общепринятой. Однако в подавляющем большинстве задач объем вычислительных операций превосходит количество входных и выходных данных, которые только и подлежат

переводу из одной системы счисления в другую, и поэтому указанный недостаток является совершенно несущественным. К тому же этот перевод зачастую производится автоматически.

Так как мы не предполагаем у читателя привычки к системам счисления, отличным от десятичной, то для ориентировки приводим небольшую табличку записи чисел в различных системах (см. таблицу 1). При этом отметим, что поскольку в последней графе мы приводим запись чисел в шестнадцатеричной системе и в этой системе счисления должны быть обозначения для шестнадцати цифр, то мы используем для обозначения нуля и первых девяти единиц обычные знаки, а для обозначения десяти, одиннадцати, двенадцати, тринадцати, четырнадцати и пятнадцати — соответственно знаки $\overline{0}$, $\overline{1}$, $\overline{2}$, $\overline{3}$, $\overline{4}$, $\overline{5}$.

Таблица 1. Запись чисел в различных системах

Системы счисления					
десятичная	двоичная	троичная	пятеричная	восьмеричная	шестнадцатеричная
1	1	1	1	1	1
2	10	2	2	2	2
3	11	10	3	3	3
4	100	11	4	4	4
5	101	12	10	5	5
6	110	20	11	6	6
7	111	21	12	7	7
8	1000	22	13	10	8
9	1001	100	14	11	9
10	1010	101	20	12	$\overline{0}$
11	1011	102	21	13	$\overline{1}$
12	1100	110	22	14	$\overline{2}$
13	1101	111	23	15	$\overline{3}$
14	1110	112	24	16	$\overline{4}$
15	1111	120	30	17	$\overline{5}$
16	10000	121	31	20	10
17	10001	122	32	21	11
0,5	0,1	0,11...	0,22...	0,4	0,8
0,3	0,01001...	0,02200...	0,122...	0,23146...	0,4$\overline{2}\overline{2}\overline{2}$...
0,(3)	0,0101...	0,1	0,1313...	0,2525...	0,55...

При подготовке задач нередко пользуются восьмеричной и шестнадцатеричной системами счисления. Переход от этих систем к двоичной осуществляется весьма просто. Действительно,

каждый разряд восьмеричной системы преобразуется в некоторое трехзначное двоичное число; точно так же каждый разряд шестнадцатеричной системы преобразуется в четырехзначное двоичное число. Таким образом, например, число 1175, записанное в восьмеричной системе, в двоичной системе принимает такой вид: 1001111101. Число 528, записанное в шестнадцатеричной системе, в двоичной имеет такую запись: 10111001000. Для перехода от двоичной записи числа к восьмеричной нужно разбить двоичную запись на группы по три цифры справа налево и каждую группу заменить восьмеричным числом, пользуясь таблицей 1. Для перехода от двоичной записи к шестнадцатеричной нужно двоичную запись разбить справа налево на группы по четыре цифры и каждую группу заменить соответствующей шестнадцатеричной цифрой. Например, число 11101111001, записанное в двоичной системе, в восьмеричной записи имеет вид 3571. То же число для перевода в шестнадцатеричную запись нужно предварительно записать группами по четыре цифры: 111 0111 1001; по таблице 1 находим, что в шестнадцатеричной записи это число будет выглядеть так: 779.

Нет необходимости подробно останавливаться на том, что простота указанных преобразований объясняется тем, что числа восемь и шестнадцать являются целыми степенями двойки ($8 = 2^3$; $16 = 2^4$).

Для того чтобы лучше освоить двоичную систему счисления, мы вкратце изложим на примерах правила выполнения арифметических операций над числами, представленными в двоичной записи. Эти правила ничем не отличаются от тех, которые излагаются в школьном курсе арифметики, только в двоичной системе таблицы сложения и умножения особенно просты.

Таблица двоичного сложения	*Таблица двоичного умножения*
$0+0=0$	$0\times 0=0$
$0+1=1$	$0\times 1=0$
$1+0=1$	$1\times 0=0$
$1+1=10$	$1\times 1=1$

Приведем по одному примеру на каждое арифметическое действие над двоичными числами.

Двоичное сложение	*Двоичное вычитание*
1010011,111	1100101,101
$+$ 11001,110	$-$ 10101,111
1101101,101	1001111,110

Двоичное умножение

```
   11001,01
 ×    11,01
 ----------
   1100101
  1100101
 1100101
----------
1010010,0001
```

Двоичное деление

```
10100110011 | 1011
- 1011       ------
  ------     1111001
  10011
- 1011
  ------
  10001
- 1011
  ------
  01100
   1011
  ------
- 0001011
    1011
```

§ 2. Основные логические операции

Применение в электронных вычислительных машинах двоичной системы счисления дает возможность использовать аппарат математической логики, в частности исчисление высказываний, при анализе и построении функциональных схем машин.

В математической логике под *высказыванием* понимается любое предложение, относительно которого имеет смысл говорить об его истинности или ложности, например: «*Сегодня четвертое число*», «*человек бессмертен*», «*дважды два — четыре*». Предложения, которые могут быть одновременно истинными и ложными, а также лишь частично истинными, в математической логике не рассматриваются. При оценке высказываний мы будем принимать во внимание лишь их истинность или ложность, никак не учитывая их конкретного содержания.

Высказывания мы будем обозначать заглавными буквами латинского алфавита. Если A истинно, то будем писать $A = 1$, если A ложно, то будем писать $A = 0$. Если высказывание каким-то образом зависит от некоторого параметра, то его значение истинности представляет собой функцию, способную принимать лишь два значения: 0 или 1. Высказывание всегда истинное определяет постоянную функцию, равную 1; высказыванию всегда ложному соответствует функция, которая тождественно равна 0. В приведенных выше примерах первое высказывание верно только в течение двенадцати дней в году, второе ложно всегда, третье всегда истинно.

Переменная величина, которая может принимать лишь два значения, называется *двоичной переменной*. Согласно введенному определению всякое высказывание является двоичной переменной.

В современных *ЦАМ* действия над числами (записанными в двоичной системе), передача двоичных кодов из одного блока

машины в другой, управление этой передачей осуществляются посредством устройств, которые способны находиться только в двух состояниях (этим состояниям приписываются значения 0 и 1). Таким образом, работа управляющих схем устройства передачи информации в *ЦАМ* представима некоторыми функциями от двоичных переменных, которые сами также принимают лишь два значения. Такие функции называются *двоичными функциями.*

Математическая логика изучает вопросы представления и преобразования двоичных функций от двоичных аргументов посредством некоторых логических операций, называемых логическими связями. Из простых высказываний при помощи логических связей могут быть составлены сложные высказывания, принимающие значения *истинно* (1) или *ложно* (0) в зависимости от значений составляющих простых высказываний. Так как высказывания можно рассматривать как двоичные переменные, то логические связи между высказываниями можно представить как операции над двоичными переменными величинами. Определим основные логические операции.

1. Отрицание. *Отрицание* высказывания A обозначается символом $\overline{A}$ (читается: «*не* A»). Значение истинности высказывания $\overline{A}$ определяется следующей таблицей:

Отрицание

$$\overline{1}=0$$
$$\overline{0}=1.$$

Таким образом, отрицанием высказывания A является сложное высказывание $\overline{A}$, которое ложно, когда A истинно, и истинно, когда A ложно.

2. Логическое умножение. Логическое умножение высказываний A и B обозначается символом $A \wedge B$ (читается: «A и B»). Значение истинности логического произведения $A \wedge B$ определяется в зависимости от значений истинности высказываний A и B по следующей таблице:

Логическое умножение

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

Операция логического умножения называется *конъюнкцией*, а символ $\wedge$ — *знаком конъюнкции.*

Конъюнкция $A \wedge B$ двух высказываний представляет собой сложное высказывание, которое истинно тогда и только тогда, когда истинны составляющие его высказывания A и B.

3. Логическое сложение. Логическое сложение двух высказываний A и B обозначается символом $A \vee B$ (читается: «*A или B*»). Значение истинности логической суммы $A \vee B$ определяется следующей зависимостью от значений истинности составляющих высказываний A и B:

Логическое сложение

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

Операция логического сложения называется *дизъюнкцией*, а символ $\vee$ — *знаком дизъюнкции*. Дизъюнкция двух высказываний A и B является сложным высказыванием, которое ложно тогда и только тогда, когда оба слагаемых A и B ложны.

4. Равнозначность. Равнозначность выражается символом $A \sim B$ (читается: «*A равнозначно B*»). Значение истинности высказывания $A \sim B$ определяется следующей таблицей:

Равнозначность

$$0 \sim 0 = 1$$
$$0 \sim 1 = 0$$
$$1 \sim 0 = 0$$
$$1 \sim 1 = 1$$

Равнозначность $A \sim B$ двух высказываний A и B истинна, когда оба высказывания A и B имеют одинаковое значение истинности, и ложна в противном случае.

5. Сложение по модулю 2. В электронных цифровых машинах часто используется операция над двумя двоичными переменными, которая определяется как отрицание равнозначности и обозначается символом $A \approx B$ (можно читать: «*A неравнозначно B*»). Значение истинности сложного высказывания $A \approx B$ определяется по таблице поразрядного сложения по модулю 2:

$$0 \approx 0 = 0$$
$$0 \approx 1 = 1$$
$$1 \approx 0 = 1$$
$$1 \approx 1 = 0$$

Операция сложения по модулю 2 выражается через операции равнозначности и отрицания по формуле

$$A \approx B = \overline{A \sim B}.$$

Приведенные выше операции логического умножения, логического сложения, операция равнозначности и сложение по мо-

дулю 2 *являются* коммутативными и ассоциативными, так что

$$A \,0\, B = B \,0\, A,$$

$$(A \,0\, B) \,0\, C = A \,0\, (B \,0\, C),$$

где 0 — знак некоторой операции.

Кроме того, справедлив распределительный закон для логического умножения по отношению к логическому сложению и для логического сложения по отношению к логическому умножению, т. е. имеют место формулы

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C),$$

$$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C).$$

Свойства коммутативности и ассоциативности бинарных операций и распределительный закон для логического умножения и сложения проверяются путем подстановки вместо переменных высказываний A, B и C значений 0 или 1 в различных комбинациях и применения таблиц, определяющих эти операции.

Логические связи, перечисленные выше, не являются независимыми. Например, между конъюнкциями, дизъюнкциями и отрицаниями существуют такие связи:

$$\overline{A \wedge B} = \bar{A} \vee \bar{B},$$

$$\overline{A \vee B} = \bar{A} \wedge \bar{B}.$$

Равнозначность «$\sim$» и сложение по модулю 2 (неравнозначность) «$\not\sim$» можно представить при помощи связей «—» (*не*), «$\wedge$» (*и*), «$\vee$» (*или*). Предоставляем читателю убедиться путем непосредственной проверки в справедливости следующих эквивалентностей:

$$A \sim B = (A \vee \bar{B}) \wedge (\bar{A} \vee B),$$

$$A \not\sim B = (A \wedge \bar{B}) \vee (\bar{A} \wedge B).$$

В исчислении высказываний доказывается, что любое сложное высказывание может быть построено из простых при помощи операций отрицания, логического умножения и логического сложения. Более того, существует так называемая *нормальная форма* сложных высказываний. Основная теорема теории высказываний утверждает:

Каждое сложное высказывание можно привести к нормальной форме, состоящей из некоторой конъюнкции дизъюнкций, в которой каждый дизъюнктивный член является либо основным простым высказыванием, либо его отрицанием.

Примером сложного высказывания в нормальной форме может служить выражение

$$(A \vee \bar{B} \vee C) \wedge (\bar{A} \vee \bar{B} \vee C).$$

Ограничиваясь приведенными сведениями из математической логики, заметим, что знакомство с основными понятиями и положениями математической логики необходимо как для составления принципиальных схем *ЦАМ*, так и для работы на *ЦАМ*.

§ 3. Элементарные электронные схемы

Логические связи двоичных переменных реализуются в *ЦАМ* электронными схемами, основными элементами которых служат триоды и диоды в виде электронных ламп или полупроводниковых элементов. Двоичная переменная величина в машине реализуется в виде потенциала. Цифра 1 изображается сигналом высокого напряжения, цифра 0 — сигналом низкого напряжения.

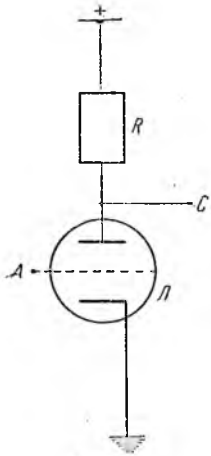


Рис. 1.

Электронные схемы логических связей обеспечивают выдачу на выходе схемы сигнала, зависящего от входных сигналов согласно таблице соответствующей логической связи.

Входные и выходные сигналы могут быть реализованы в электронных схемах наличием либо положительных, либо отрицательных потенциалов или импульсов. При рассмотрении схем логических связей высокий положительный потенциал принят за наличие сигнала, а низкий — за его отсутствие.

Рассмотрим простейшие электронные схемы, реализующие основные логические операции *«не»*, *«или»* и *«и»*.

1. Схема отрицания. Инвертор. Схема отрицания (рис. 1) состоит из электронной лампы (триода) *Л* и сопротивления *R*. Если сигнал на входе *A* отсутствует, т. е. на сетку лампы подано низкое напряжение, то сопротивление лампы будет велико и, значит, потенциал выхода *C* будет высоким, т. е. на выходе *C* будет сигнал. При поступлении сигнала на вход *A* напряжение на сетке лампы повышается, сопротивление ее резко падает и, значит, понижается потенциал выхода *C*, т. е. сигнал на выходе *C* отсутствует. Таким образом, приведенная схема осуществляет логическую операцию отрицания

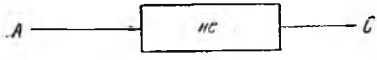


Рис. 2.

$$C = \bar{A}.$$

Электронный прибор, реализующий логическую операцию отрицания, называется *инвертором*. Функциональная схема инвертора изображена на рис. 2.

2. Схема совпадения. Схема совпадения (рис. 3) осуществляет операцию логического произведения. Эта схема представляет собой группу параллельно включенных диодов (на рис. 3 выпрямители d_1, d_2 и d_3), соединеных последовательно с сопротивлением R. Диоды обладают свойством односторонней проводимости, которое обусловлено физической природой вещества выпрямителя (наиболее распространены германиевые, селеновые и купроксные выпрямители). Направление тока, который могут пропускать выпрямители, показано стрелками. Если на входах A, B и C имеется сигнал, т. е. подано высокое напряжение, то все выпрямители заперты, не пропускают тока и, следовательно, потенциал выхода D высокий, т. е. на выходе возникает сигнал. Если хотя бы на одном входе будет отсутствовать сигнал, т. е. на этом входе будет низкое напряжение, то соответствующий выпрямитель будет пропускать ток, который вызовет падение напряжения на сопротивлении R, и на выходе D будет низкий потенциал, т. е. не будет сигнала.

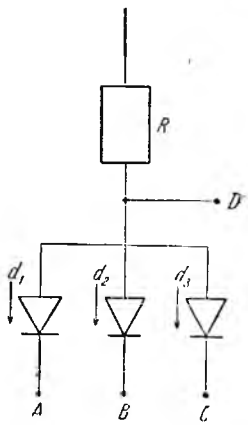


Рис. 3.

Схема совпадения осуществляет операцию логического произведения

$$D = A \wedge B \wedge C.$$

Функциональное обозначение схемы совпадения приведено на рис. 4.

3. Схема разделения. Схема разделения (рис. 5) реализует логическое сложение. Эта схема состоит

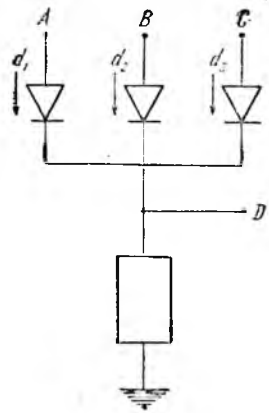


Рис. 5.

Рис. 4.

также из группы выпрямителей d_1, d_2, d_3, пропускающих ток в направлении, обозначенном стрелками. Если на входах этой схемы A, B и C нет сигнала, т. е. напряжение на входах — низкое, то будет низким также и потенциал выхода D, т. е. на выходе сигнал будет отсутствовать. При подаче сигнала хотя бы на один из входов по крайней мере через один из выпрямителей

будет проходить ток и, следовательно, потенциал выхода D окажется высоким, на выходе будет сигнал. Таким образом, схема, приведенная на рис. 5, осуществляет операцию логического сложения

$$D = A \vee B \vee C.$$

На рис. 6 изображено функциональное обозначение схемы разделения.

Сложные управляющие схемы определяются двоичными функциями от двоичных переменных и могут быть построены из рассмотренных схем инвертора, схем совпадения и схем разделения. Например, логическая операция равнозначности $C = A \sim B$ представляется формулой (см. стр. 21):

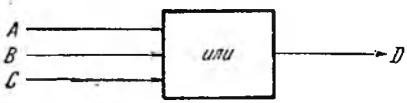


Рис. 6.

$$A \sim B = (A \vee \bar{B}) \wedge (\bar{A} \vee B).$$

Согласно этой формуле легко строится схема, осуществляющая операцию равнозначности (рис. 7).

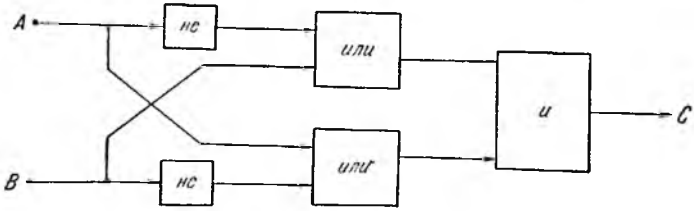


Рис. 7.

4. Схема двойного вентиля. Схема двойного вентиля, представляющая собой многополюсник с четырьмя входами и

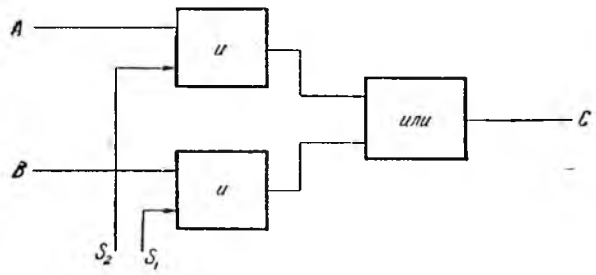


Рис. 8.

одним выходом, приведена на рис. 8. На два входа A и B поступают сигналы, которые должны пройти на выход C. На

входы S_1 и S_2 подаются управляющие сигналы. В зависимости от того, на какой из управляющих входов, S_1 или S_2, подан сигнал, на выход C пройдет либо сигнал A, либо сигнал B.

5. Избирательная схема (дешифратор). Избирательная схема (дешифратор) строится из n входов и 2^n выходов. Каждой комбинации сигналов на входах соответствует одна и только одна комбинация сигналов на выходах. На рис. 9

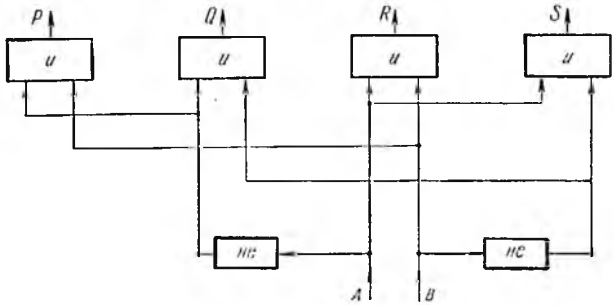


Рис. 9.

изображена функциональная схема дешифратора на два входа A и B и четыре выхода P, Q, R, S. При каждой комбинации входных сигналов выходной сигнал появляется лишь на одном выходе. Для значений $A = 0$, $B = 0$ выходной сигнал появится на Q. При $A = 0$, $B = 1$ появится сигнал на выходе P. Значениям $A = 1$, $B = 0$ соответствует сигнал на выходе S и значениям $A = 1$, $B = 1$ соответствует выходной сигнал на R.

6. Одноразрядный двоичный сумматор на два входа. Для того чтобы произвести сложение двух чисел, нужно иметь возможность определить сумму двух одноразрядных чисел и учесть перенос (предыдущего) разряда. При этом сложение трех цифр — двух слагаемых и цифры переноса — можно осуществлять либо последовательно, складывая по две цифры (сумма будет получена двумя шагами), либо одновременно, складывая все три цифры. В соответствии с этим могут быть построены одноразрядные сумматоры на два входа и на три входа.

Для того чтобы построить схему одноразрядного двоичного сумматора на два входа из логических элементов *«не»*, *«или»*, *«и»*, составим таблицу сложения двух одноразрядных двоичных чисел A и B с указанием переноса для каждой возможной комбинации значений слагаемых (см. таблицу 2).

Таблица 2. Таблица сложения одноразрядных двоичных чисел

A	B	C	$П$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Логические формулы, определяющие сумму C и перенос $П$ как функции слагаемых A и B, могут быть найдены на основании этой таблицы либо просто подбором, либо с использованием основной теоремы исчисления высказываний о представлении сложных высказываний в нормальной форме.

Для представления суммы C и переноса $П$ в нормальной форме как функций слагаемых A и B построим таблицу всех возможных дизъюнктивных членов из A и B (см. таблицу 3).

Таблица 3. Таблица дизъюнктивных членов из A и B

A	B	$A \vee B$	$A \vee \overline{B}$	$\overline{A} \vee B$	$\overline{A} \vee \overline{B}$
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Согласно таблице 2 сумма C принимает значение 0, когда оба слагаемых равны 0 или когда слагаемые A и B равны 1. По таблице 3 находим соответствующие дизъюнктивные члены $A \vee B$ и $\overline{A} \vee \overline{B}$, принимающие значение нуль при тех же значениях слагаемых A и B. Поэтому сумма C в нормальной форме представляется формулой

$$C = (A \vee B) \wedge (\overline{A} \vee \overline{B}).$$

Аналогично находим нормальную форму для переноса $П$:

$$П = (A \vee B) \wedge (A \vee \overline{B}) \wedge (\overline{A} \vee B).$$

Уже по этим формулам могут быть построены функциональные схемы суммы C и переноса $П$. Однако полученные схемы не будут самыми простыми. Поэтому возникает задача преобразования схем к простейшему виду (одна из основных задач синтеза схем). В данном случае полученные формулы могут быть

легко упрощены. Так, для C второй дизъюнктивный член преобразуем по формуле

$$\overline{A} \vee \overline{B} = \overline{A \wedge B},$$

тогда сумма C представляется формулой

$$C = (A \vee B) \wedge (\overline{A \wedge B}).$$

Преобразование формулы для переноса $П$ осуществим следующим образом: так как сумма всех дизъюнктивных членов, очевидно, равна тождественно единице, то, значит,

$$\overline{П} = \overline{A} \vee \overline{B}$$

и, следовательно,

$$П = A \wedge B.$$

Функциональная схема сумматора на два входа, осуществляющая сложение двух одноразрядных двоичных чисел и определяющая перенос, приведена на рис. 10.

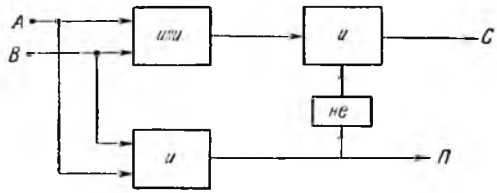


Рис. 10.

7. Двоичный сумматор на три входа. Сложение двух одноразрядных двоичных чисел и перенос из предыдущего разряда можно осуществить одновременно. При этом схема сумматора будет иметь три входа: два слагаемых A и B и перенос из предыдущего разряда C. Сумма S и перенос $П$ при этом определяются по таблице 4.

Таблица 4. Сложение одноразрядных двоичных чисел A и B и перенос из предыдущего разряда C

A	B	C	S	$П$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Как и в предыдущем случае, составим таблицу всех возможных дизъюнктивных членов из A, B и C.

Таблица 5. Дизъюнктивные члены из A, B и C

A	B	C	$A \vee B \vee C$	$A \vee B \vee \bar{C}$	$A \vee \bar{B} \vee C$	$A \vee \bar{B} \vee \bar{C}$	$\bar{A} \vee B \vee C$	$\bar{A} \vee B \vee \bar{C}$	$\bar{A} \vee \bar{B} \vee C$	$\bar{A} \vee \bar{B} \vee \bar{C}$
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

Согласно таблице 4, сумма S принимает значение 0 при следующих комбинациях значений A, B и C:

000; 011; 101; 110.

Выпишем дизъюнктивные члены, принимающие значение 0 при этих комбинациях значений A, B и C:

$$A \vee B \vee C, \quad A \vee \bar{B} \vee \bar{C}, \quad \bar{A} \vee B \vee \bar{C}, \quad \bar{A} \vee \bar{B} \vee C.$$

Следовательно,

$$S = (A \vee B \vee C) \wedge (A \vee \bar{B} \vee \bar{C}) \wedge (\bar{A} \vee B \vee \bar{C}) \wedge (\bar{A} \vee \bar{B} \vee C).$$

Аналогично находим формулу для переноса $П$:

$$П = (A \vee B \vee C) \wedge (A \vee B \vee \bar{C}) \wedge (A \vee \bar{B} \vee C) \wedge (\bar{A} \vee B \vee C).$$

Предоставляем читателю упростить эти формулы и построить соответствующую функциональную схему двоичного сумматора на три входа.

8. Цифровой элемент. Триггерная схема. Одним из основных элементов схем *ЦАМ* является *цифровой элемент*. Цифровой элемент обладает следующими свойствами:

1) может находиться в одном из двух дискретных устойчивых состояний;

2) может влиять на другие устройства, в частности на другие цифровые элементы, и воспринимать влияние этих элементов. Основой цифрового элемента является *триггерная схема* (рис. 11). Она строится из двух электронных ламп-триодов $Л_1$ и $Л_2$. Анод первого триода $Л_1$ через сопротивление R_1 соединяется с сеткой второго триода $Л_2$, а анод второго триода $Л_2$ соединен с сеткой первого через сопротивление R_2. Основное

свойство триггерной схемы заключается в том, что один из ее триодов в устойчивом состоянии открыт, а второй — закрыт; при отпирании второго триода запирается первый. Существует несколько способов управления триггерной схемой. Рассмотрим два основных типа подачи входных сигналов.

9. Триггерная схема как счетчик. На рис. 11 лампа, проводящая в данный момент ток, заштрихована. При подаче на объединенные сетки обоих триодов сигнала в виде отрицательного импульса $Л_1$ запирается. Потенциал анода $Л_1$ резко возра-

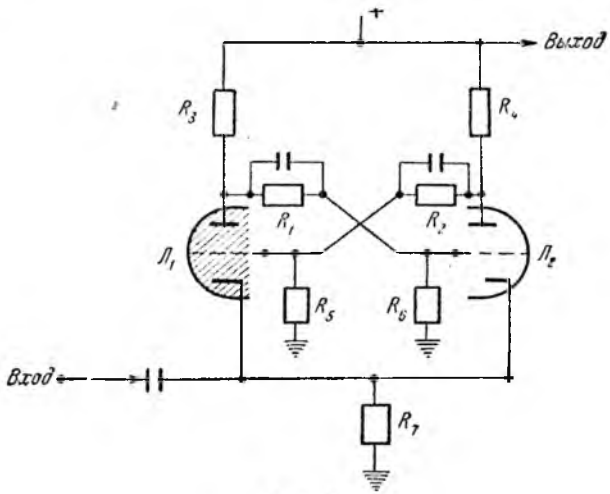
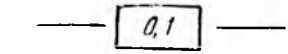


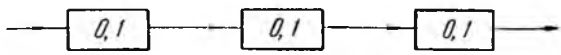
Рис. 11.

стает. Это приводит к возрастанию потенциала на сетке лампы $Л_2$. Ток через $Л_2$ увеличивается. При этом потенциал анода $Л_2$ резко падает, что влечет за собой падение потенциала на сетке лампы $Л_1$, которая будет поэтому заперта до прихода следующего сигнала. Следующий импульс запрет лампу $Л_2$, а $Л_1$ откроется, и т. д. При подаче каждого нового сигнала схема будет переходить из одного устойчивого состояния в другое.

Триггерная схема рассмотренного типа работает как счетчик по модулю 2. Аноды ламп могут быть при этом использованы в качестве выходов схемы. Анод открытой лампы имеет низкий потенциал, анод запертой лампы имеет высокий потенциал. Функциональное обозначение счетчика по модулю 2 имеет такой вид:



Последовательное соединение триггерных схем-счетчиков дает многоразрядные двоичные счетчики. Например, схема



осуществляет счет по модулю 8, т. е. от 0 до 7 (в двоичной записи 111).

10. Триггерная схема как запоминающий регистр. При втором способе управления триггерной схемой сигналы подаются на два раздельных входа: на сетку лампы $Л_1$ и на сетку лампы $Л_2$ (рис. 12).

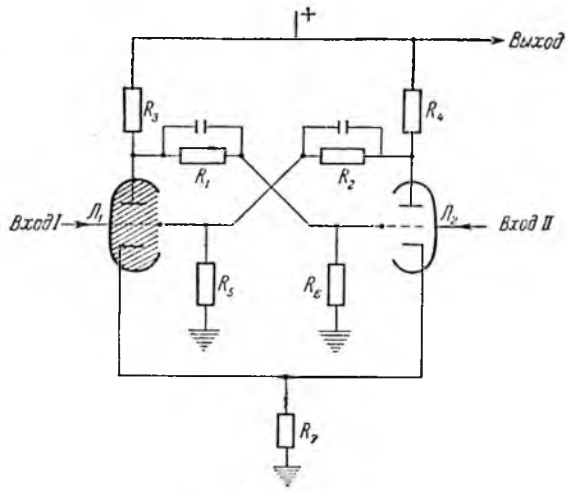
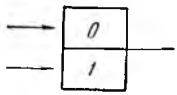


Рис. 12.

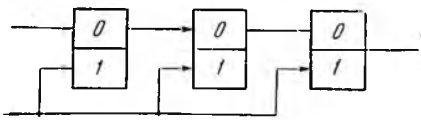
При подаче сигнала на вход *I* лампа $Л_1$ запирается, а лампа $Л_2$ отпирается, вследствие чего потенциал на аноде $Л_2$ скачком падает. Однако при повторном поступлении сигналов на сетку $Л_1$ никаких изменений в схеме происходить не будет, т. е. поступающие сигналы будут только подтверждать имеющееся состояние схемы. Состояние схемы изменится только в том случае, если сигнал будет подан на вход *II*, т. е. на сетку $Л_2$. В этом случае $Л_2$ запирается и потенциал на ее аноде скачком увеличивается, $Л_1$ отпирается.

Если подавать поочередно на вход *I* код, который нужно запомнить, а на вход *II* — гасящие импульсы и принимать наличие сигнала за 1 и его отсутствие за 0, то схема будет запоминать каждый раз, что именно подано: 1 или 0.

Схема, изображенная на рис. 12, «запоминает» полярность сигнала, поданного на один вход. Функциональное обозначение схемы запоминания следующее:



Соединением триггерных схем, являющихся запоминающими регистрами, строятся регистры для запоминания многоразрядных двоичных чисел. Например, схема



запоминает трехразрядные двоичные числа от 000 до 111, т. е. от 0 до 7. На один вход регистра подается двоичный код трехразрядного числа. На второй вход поступают импульсы сдвига, осуществляющие перенос цифр в соответствующие элементы регистра.

§ 4. Представление чисел в машине

Каждое число a, представленное в двоичной системе счисления в виде

$$a = \pm\, 10^{p} \sum_{k=1}^{n} a_k \cdot 10^{-k}, \qquad (2)$$

где a_k — либо 0, либо 1 (напомним, что в двоичной системе счисления символом «10» обозначается число два), в машине изображается определенным набором двоичных цифр. Для изображения знаков чисел также используются двоичные цифры: обычно знак плюс кодируется цифрой 0, знак минус — цифрой 1. Это удобно тем, что знаки произведения и частного определяются сложением по модулю 2 кодов знаков сомножителей делимого и делителя:

$$0+0=0; \quad 0+1=1; \quad 1+0=1; \quad 1+1=0.$$

Как легко понять из предыдущего, такое сложение реализуется в машине одноразрядным сумматором.

В зависимости от конструктивных особенностей цифровых машин применяются две формы представления чисел:

1) *естественная форма* — в машинах с фиксированной запятой,

2) *полулогарифмическая* или *нормальная форма* — в машинах с плавающей запятой.

1. Естественная форма представления чисел. При естественной форме изображения чисел в представлении (2) показатель p фиксирован для всех чисел. Чаще всего $p = 0$, т. е. все числа, которые могут быть изображены в машине, меньше единицы по абсолютной величине. Случай $p > 0$ неудобен тем, что показатель произведения двух таких чисел может оказаться больше, чем p. В случае $p < 0$ при умножении происходит большая потеря знаков. В дальнейшем будем предполагать, что в машинах с фиксированной запятой $p = 0$.

Каждое число a в машинах с фиксированной запятой изображается набором $a_0a_1a_2 \ldots a_n$, где a_0 — двоичная цифра, изображающая знак числа; $a_1a_2 \ldots a_n$ — цифры числа a в представлении (2). Каждому набору $a_0a_1a_2 \ldots a_n$ отнесем число

$$(a)=a_0,\ a_1a_2 \ldots a_n$$

и назовем его *кодом* числа a.

Положительное число

$$a = +\sum_{k=1}^{n} a_k \cdot 2^{-k}$$

определяется набором двоичных цифр

$$0a_1a_2 \ldots a_n,$$

которому соответствует код, называемый прямым

$$(a) = 0,\ a_1a_2 \ldots a_n,$$

так что для положительных чисел код совпадает с двоичной записью самого числа.

Для отрицательных чисел применяются три способа изображения, или, как говорят, три кода.

1. *Прямой код.* Пусть отрицательное число имеет такую двоичную запись:

$$a = -0,\ a_1a_2 \ldots a_n.$$

Тогда ему ставится в соответствие следующий код:

$$[a]_{\text{пр}}=1,\ a_1a_2 \ldots a_n=1+|a|=1-a.$$

Нуль имеет два представления в прямом коде:

$$[+0]_{\text{пр}}=0,\ 00 \ldots 0 \quad \text{и} \quad [-0]_{\text{пр}}=1,\ 00 \ldots 0.$$

Первое представление будем называть *положительным нулем*, второе — *отрицательным нулем*.

В прямом коде могут быть представлены все числа отрезка $[-(1-2^{-n}),\ 1-2^{-n}]$.

2. *Дополнительный код.* Отрицательное двоичное число $a = -0, a_1a_2 \ldots a_n$ кодируется в таком виде:

$$[a]_{\text{доп}} = 1, a'_1a'_2 \ldots a'_n,$$

где $a'_1, a'_2, \ldots, a'_n$ — цифры числа $a' = 0, a'_1a'_2 \ldots a'_n$, дополняющего $|a|$ до единицы:

$$|a| + a' = 1.$$

Следовательно,

$$[a]_{\text{доп}} = 1 + a' = 10 - |a|.$$

(«10» — запись числа два в двоичной системе счисления). Например, при $a = -0{,}101101$

$$[a]_{\text{доп}} = 1{,}010011.$$

В дополнительном коде нуль имеет только одно изображение:

$$[0]_{\text{доп}} = 0{,}00 \ldots 0.$$

Код 1,00 ... 0 изображает число -1:

$$[-1]_{\text{доп}} = 1, 00 \ldots 0.$$

В дополнительном коде представимы числа отрезка $[-1, 1-2^{-n}]$.

3. *Обратный код.* Отрицательное двоичное число $a = -0, a_1a_2 \ldots a_n$ в обратном коде изображается в таком виде:

$$[a]_{\text{обр}} = 1, \bar{a}_1\bar{a}_2 \ldots \bar{a}_n,$$

где $\bar{a}_k = 1$, если $a_k = 0$, и $\bar{a}_k = 0$, если $a_k = 1$ $(k = 1, 2, \ldots, n)$, т. е. обратный код двоичного числа получается из прямого кода заменой в цифровых разрядах единицы нулями и нулей единицами, так что

$$[a]_{\text{обр}} + |a| = 1, 11 \ldots 1 = 10 - (10^{-n}),$$

откуда

$$[a]_{\text{обр}} = 10 - (10^{-n}) + a,$$

где $(10^{-n}) = \underbrace{0{,}00 \ldots 01}_{n \text{ разрядов}}$ есть код двоичного числа 10^{-n}.

В обратном коде нуль имеет два представления:

$$[+0]_{\text{обр}} = 0, 00 \ldots 0; \quad [-0]_{\text{обр}} = 1, 11 \ldots 1.$$

В обратном коде, как и в прямом коде, можно представлять числа отрезка $[1 - (1-2^{-n}), 1-2^{-n}]$.

Для положительных чисел будем считать, что дополнительный и обратный коды совпадают с прямым кодом. Представление

чисел прямым кодом удобно использовать при операциях умножения и деления. Для операций сложения и вычитания удобны дополнительный и обратный коды.

В машинах с фиксированной запятой все разряды чисел с точки зрения выполнения операций однотипны. Это дает возможность весьма просто реализовать выполнение операций в таких машинах. Зато из-за узкого диапазона представимых чисел при решении задач на машинах с фиксированной запятой приходится подбирать масштабные коэффициенты, обеспечивающие нахождение числовых данных задачи и результатов вычислений в заданном диапазоне, а это часто сопряжено со значительными трудностями.

2. Нормальная форма представления чисел. В машинах с плавающей запятой в представлении чисел (2) порядок числа p — переменный. Следовательно, набор цифр, представляющий число $a = \pm 2^p \sum_{k=1}^{n} a_k 2^{-k}$, должен содержать не только изображение знака и цифр числа, но и изображение порядка числа p; один разряд выделяется для изображения знака порядка.

Пусть для изображения порядка выделен $r+1$ разряд и для изображения цифровой части числа $n+1$ разряд, тогда набор, изображающий число в нормальной форме, имеет такой вид:

$$p_0 p_1 \ldots p_r a_0 a_1 \ldots a_n.$$

Здесь p_0, p_1, p_2, ..., p_r — изображение порядка (p_0 — знак порядка), a_0, a_1, ..., a_n — изображение цифровой части числа a (a_0 — ее знак). Например, при $r=3$, $n=6$ набор двоичных цифр

0101	0101110
порядок	цифровая часть

изображает число с порядком $p=+5$ и цифровой частью $q=+0{,}101110$, т. е. изображает двоичное число $a = 10^5 \cdot 0{,}101110$.

Двоичное число $a = 10^p \cdot q$ называется *нормализованным*, если в цифровой части числа $q = a_0, a_1 a_2 \ldots a_n$ первая цифра $a_1 = 1$, т. е. если $\frac{1}{10} \leqslant |q| < 1$ (здесь «10» — двоичное число два).

Наименьшее двоичное нормализованное число, представимое в машине с плавающей запятой, в которой для изображения порядка отведено $r+1$ разрядов, есть $a = 10^{-(10^r-1)} \cdot 0{,}1 = 10^{-10^r}$, следовательно, числа $a < 10^{-10^r}$ уже не могут быть представлены в машине в нормализованном виде. Эти числа обычно принимаются за число 0. Иногда указателем нуля считается наличие порядка $-2^r + 1$, независимо от значения цифровой части.

Порядки принимают целые (десятичные) значения от $-(2^r-1)$ до 2^r-1. Например, при $r=5$ порядки могут принимать целые значения от -31 до $+31$, и соответственно диапазон чисел, которые могут быть изображены с этими порядками, состоит из чисел от 2^{-31} до 2^{+31}.

Применение нормальной формы для представления чисел дает возможность получить достаточно широкий диапазон представимых в машине чисел как весьма малых по абсолютной величине, так и больших, причем все нормализованные числа можно представить с примерно одинаковой относительной точностью.

Недостатком нормальной формы представления чисел является существенное усложнение выполнения операций по сравнению с операциями над числами в естественной форме, а тем самым серьезное усложнение конструкции узлов машины.

§ 5. Операции над числами в цифровых машинах

1. Сложение и вычитание в машинах с фиксированной запятой. Числа поступают на сумматор в дополнительном или обратном коде. Будем предполагать, что в результате операции получается число, по абсолютной величине не превосходящее единицы.

Рассмотрим сложение чисел в дополнительном коде. Пусть заданы два числа a и b, и предположим, что $|a+b|<1$.

Если $a\geqslant 0$, $b\geqslant 0$, то $a+b\geqslant 0$ и

$$[a]_{\text{доп}}+[b]_{\text{доп}}=a+b=[a+b]_{\text{доп}}.$$

Если $a<0$, $b\geqslant 0$, но $a+b\geqslant 0$, то

$$[a]_{\text{доп}}+[b]_{\text{доп}}=10+a+b=10+[a+b]_{\text{доп}};$$

для получения дополнительного кода суммы нужно отбросить единицу переноса, которая получается в знаковом разряде (в разряде единиц кода).

Если $a<0$, $b\geqslant 0$, но $a+b<0$, то

$$[a]_{\text{доп}}+[b]_{\text{доп}}=10+a+b=[a+b]_{\text{доп}}.$$

Если $a<0$ и $b<0$, то

$$[a]_{\text{доп}}+[b]_{\text{доп}}=10+a+10+b=10+[a+b]_{\text{доп}};$$

в этом случае возникает единица переноса в разряде единиц кода, которую нужно отбросить.

Итак сложение чисел в дополнительном коде может быть реализовано на сумматоре, в котором коды чисел складываются как обычные числа и отбрасывается единица переноса в знаковом разряде (если она возникает).

При этом знаковый разряд рассматривается как разряд целых единиц.

Пример 1.

$x =$	0,1101	$[x]_{доп} = 0,1101$	$+\ 0,1101$
$y =$	0,0001	$[y]_{доп} = 0,0001$	$+\ 0,0001$
$x + y =$	0,1110	$[x + y]_{доп} = 0,1110$	$\longleftarrow\ 0,1110$

Пример 2.

$x =$	$-0,0001$	$[x]_{доп} = 1,1111$	$+\ 1,1111$
$y =$	$+0,1101$	$[y]_{доп} = 0,1101$	$+\ 0,1101$
$x + y =$	0,1100	$[x + y]_{доп} = 0,1100$	$\longleftarrow\ 10,1100$

Пример 3.

$x =$	$-0,1101$	$[x]_{доп} = 1,0011$	$+\ 1,0011$
$y =$	0,0001	$[y]_{доп} = 0,0001$	$+\ 0,0001$
$x + y =$	$-0,1100$	$[x + y]_{доп} = 1,0100$	$\longleftarrow\ 1,0100$

Пример 4.

$x =$	$-0,1101$	$[x]_{доп} = 1,0011$	$+\ 1,0011$
$y =$	$-0,0001$	$[y]_{доп} = 1,1111$	$+\ 1,1111$
$x + y =$	$-0,1110$	$[x + y]_{доп} = 1,0010$	$\longleftarrow\ 11,0010$

Рассмотрим сложение чисел, заданных в обратном коде: если $a \geqslant 0$ и $b \geqslant 0$, то

$$[a]_{обр} + [b]_{обр} = a + b = [a + b]_{обр};$$

если $a < 0$, $b \geqslant 0$ и $a + b \geqslant 0$, то

$$[a]_{обр} + [b]_{обр} = 10 - (10^{-n}) + a + b = 10 - (10^{-n}) + [a + b]_{обр}.$$

В этом случае нужно отбросить единицу переноса в старшем разряде (знаковом разряде) и прибавить единицу в младшем разряде, что можно осуществить циклическим переносом из старшего разряда в младший.

Если $a < 0$, $b \geqslant 0$ и $a + b < 0$, то

$$[a]_{обр} + [b]_{обр} = 10 - (10^{-n}) + a + b = [a + b]_{обр};$$

если $a < 0$, $b < 0$, то $a + b < 0$ и

$$[a]_{обр} + [b]_{обр} = 10 - (10^{-n}) + [a + b]_{обр},$$

т. е. нужно осуществить циклический перенос единицы, возникающей в знаковом разряде, в младший разряд.

Итак, сложение чисел в обратном коде осуществляется сумматором с циклическим переносом из старшего разряда в младший. При этом коды складываются, как обычные числа, а знаковый разряд является старшим разрядом целых единиц.

Пример 1.

$x =$	0,1101	$[x]_{обр} = 0,1101$	$+\ 0,1101$
$y =$	0,0001	$[y]_{обр} = 0,0001$	$+\ 0,0001$
$x + y =$	0,1110	$[x + y]_{обр} = 0,0111$	$\longleftarrow\ 0,1110$

Пример 2.

$$\begin{array}{lll} x = -0{,}0001 & [x]_{\text{обр}} = 1{,}1110 & +\ 1{,}1110 \\ y = \ \ 0{,}1101 & [y]_{\text{обр}} = 0{,}1101 & +\ 0{,}1101 \\ \hline & & 10{,}1011 \\ & & \text{циклический перенос} \\ x + y = \ \ 0{,}1100 & [x + y]_{\text{обр}} = 0{,}1100 \longleftarrow & 0{,}1100 \end{array}$$

Пример 3.

$$\begin{array}{lll} x = -0{,}1101 & [x]_{\text{обр}} = 1{,}0010 & +\ 1{,}0010 \\ y = \ \ 0{,}0001 & [y]_{\text{обр}} = 0{,}0001 & +\ 0{,}0001 \\ \hline x + y = -0{,}1100 & [x + y]_{\text{обр}} = 1{,}0011 \longleftarrow & 1{,}0011 \end{array}$$

Пример 4.

$$\begin{array}{lll} x = -0{,}1101 & [x]_{\text{обр}} = 1{,}0010 & +\ 1{,}0010 \\ y = -0{,}0001 & [y]_{\text{обр}} = 1{,}1110 & +\ 1{,}1110 \\ \hline & & 11{,}0000 \\ & & \text{циклический перенос} \\ [x + y] = -0{,}1110 & [x + y]_{\text{обр}} = 1{,}0001 \longleftarrow & 1{,}0001 \end{array}$$

Вычитание чисел сводится к алгебраическому сложению этих чисел. При этом меняется знак у второго слагаемого на противоположный.

2. Сложение и вычитание в машинах с плавающей запятой. В машинах с плавающей запятой действия осуществляются над числами в нормальной форме. При сложении двоичных чисел, заданных в нормальной форме $a = 10^p \alpha$ и $b = 10^q \beta$, сначала выравниваются порядки. Если $p > q$, т. е. $p = q + k$ $(k > 0)$, то слагаемое b заменяется ненормализованным числом $b_1 = 10^p \beta_1$, где β_1 образовано сдвигом числа β на k разрядов вправо *). Затем производится сложение цифровых частей α и β_1 по правилам сложения двоичных чисел в машинах с фиксированной запятой. После сложения цифровых частей может оказаться, что сумма не является нормализованным числом При $|\alpha + \beta_1| > 1$ появляется нарушение нормализации влево; при $|\alpha + \beta_1| < \frac{1}{10}$ результат получается с нарушением нормализации вправо.

Таким образом, после сложения цифровых частей требуется произвести нормализацию результата.

Признаком нарушения нормализации вправо (т. е. получения цифровой части суммы, меньшей по абсолютной величине 0,1) является наличие двух одинаковых цифр 0,0 (для положительных чисел) или 1,1 (для отрицательных чисел

*) Число b_1 совпадает с числом b с точностью до 10^{p-n} (все цифры здесь и дальше, пока не будет особо оговорено, даны в двоичной системе счисления).

в обратном или дополнительном кодах) в двух соседних разрядах: в разряде знака и в старшем цифровом разряде.

Пример. $a = 10^{10} \cdot 0{,}110110$ $\quad b = 10^{10} \cdot (-0{,}101011)$

	порядок	цифровая часть
обратный код числа a	0 010	0,110110
обратный код числа b	0 010	1,010100
		10,001010
		циклический перенос
код суммы	0 010	0,001011

После нормализации (сдвига влево на два разряда) получим цифровую часть суммы 0,101100. Сумма равна $a + b = 10^{10} \cdot 10^{-10} \cdot 0{,}101100 = 10^{0} \cdot 0{,}101100$.

Для определения нарушения нормализации влево (т. е. получения цифровой части суммы, бо́льшей по абсолютной величине, чем единица) применяется так называемый *модифицированный код,* при котором знаки чисел изображаются двумя разрядами: положительные числа представляются в виде $00, a_1 a_2 \ldots a_n$, отрицательные — модифицированным дополнительным кодом

$$[a]_{\text{доп. м}} = 100 + a$$

или модифицированным обратным кодом

$$[a]_{\text{обр. м}} = 100 - (10^{-n}) + a.$$

Числа, по абсолютной величине меньшие единицы, в модифицированном коде имеют в знаковых разрядах либо два нуля (00, ...), либо две единицы (11, ...). Сложение чисел в модифицированном дополнительном коде производится так же, как и в обычном дополнительном коде, только отбрасывается единица переноса, возникающая в старшем знаковом разряде. Сложение чисел в модифицированном обратном коде производится так же, как в обычном обратном коде, только циклический перенос осуществляется из старшего знакового разряда в младший цифровой разряд.

Признаком нарушения нормализации влево при сложении мантисс двух чисел в модифицированном коде будет появление в знаковых разрядах разных цифр 10 или 01.

Пример 1. $a = 10^{10} \cdot 0{,}110110$, $\quad b = 10^{10} \cdot 0{,}010110$.

Сумма цифровых частей в модифицированном коде

$$\begin{array}{r} 00{,}110110 \\ +\,00{,}010110 \\ \hline 01{,}001100 \end{array}$$

Сдвиг на один разряд вправо дает нормализованную цифровую часть суммы. Получаем:

$$a + b = 10^{11} \cdot 0{,}100110.$$

Пример 2. $a = 10^{10}(-0{,}111011)$, $b = 10^{10}(-0{,}010110)$.
Сумма цифровых частей в модифицированном дополнительном коде

$$\begin{array}{r} 11{,}000101 \\ 11{,}101010 \\ \hline 10{,}101111 \end{array}.$$

После сдвига на один разряд вправо и занесения в старший разряд знака единицы получим нормализованную сумму цифровых частей в модифицированном дополнительном коде: 11,010111. Следовательно, $a + b = 10^{11}(-0{,}1010001)$.

Вычитание чисел в нормальной форме сводится к алгебраическому сложению их обратных или дополнительных кодов, так же как и в машинах с фиксированной запятой.

3. Умножение и деление. Умножение чисел в машинах обычно осуществляется в прямом коде. Умножение цифровых частей чисел, заданных в двоичной системе счисления, состоит в последовательном сдвиге влево множимого на количество разрядов, равное номерам значащих цифр множителя, и сложении полученных частей произведений. Знак произведения получается сложением знаков сомножителей на одноразрядном двоичном сумматоре по следующим правилам:

$$0+0=0; \quad 1+0=1; \quad 0+1=1; \quad 1+1=0.$$

Пример.

$$\begin{array}{r} 0{,}110101 \\ \times\ 0{,}101101 \\ \hline 110101 \\ 110101 \\ 110101 \\ 110101 \\ \hline 0{,}100101010001 \end{array}$$

Умножение чисел, заданных в нормальной форме, проводится в три этапа:

1. Сложение цифр, изображающих знаки сомножителей, в результате чего получается знак произведения.

2. Алгебраическое сложение порядков сомножителей, которое дает порядок произведения.

3. Умножение цифровых частей сомножителей, что дает цифровую часть произведения.

В случае необходимости производится нормализация результата.

4. Деление чисел в машинах с фиксированной запятой. Деление чисел в двоичной системе осуществляется последовательным

определением цифр частного путем вычитания делителя. Если при вычитании делителя разность положительна, то соответствующая цифра частного равна единице; если разность отрицательна, то соответствующая цифра частного равна нулю, а предыдущий остаток сдвигается на один разряд влево и т. д. Возвращение к предыдущему остатку происходит путем прибавления делителя к отрицательному остатку. Этот цикл операций повторяется n раз (n — количество разрядов числа).

Пример. Делимое $a = 0{,}100101$, делитель $b = 0{,}110101$.

```
 0,1001010 | 0,110101
-  110101  | 0,101100
 ---------
  0010101
-  110101
 ---------
  -001011
 ---------
  101010
-  110101
 ---------
  0011111
-  110101
 ---------
  001001
-  110101
 ---------
  -100011
 ---------
  010010
-  110101
 ---------
  -10001
 ---------
  100100
```

Частное 0,101100,
остаток 0,0000001.

При определении второй, пятой и шестой цифр частного разность получилась отрицательной. При этом сохраняются соответственно первый, четвертый и пятый остатки, сдвинутые на один разряд влево.

Деление чисел в нормальной форме производится в три этапа:

1. Определение знака частного сложением знаков делимого и делителя на одноразрядном сумматоре.

2. Вычитание порядков с учетом знаков порядков. В результате получается порядок частного.

3. Деление цифровой части делимого на цифровую часть делителя, что дает цифровую часть частного.

В случае необходимости производится нормализация результата.

5. **Поразрядные операции.** Если операции над цифрами каждого разряда чисел производятся независимо друг от друга, то такие операции называются поразрядными. Рассмотрим некоторые примеры поразрядных операций.

1. Поразрядное дополнение. Операция преобразует набор $a_0a_1a_2 \dots a_n$ в набор $a'_0a'_1 \dots a'_n$, где $a'_k = 1 - a_k$ ($k = 0, 1, 2, \dots, n$). В машинах с фиксированной запятой такая операция превращает число a, заданное обратным кодом, в число $-a$, заданное тем же кодом, так как $[a]_{\text{обр}} + |a| = 1{,}1 \dots 1 = 10 - (2^{-n})$.

Таким образом, при задании чисел в машине с фиксированной запятой в обратном коде операция изменения знака является операцией поразрядного дополнения.

2. Поразрядное сложение. Поразрядное сложение является сложением цифр по модулю 2:

$$0+0=0;\ 0+1=1+0=1;\ 1+1=0.$$

С помощью операции поразрядного сложения можно осуществить поразрядное дополнение числа, сложив его с числом, имеющим во всех разрядах цифру 1.

3. Поразрядное логическое сложение. Логическое сложение двоичных цифр определяется таблицей, приведенной на стр. 20.

Пользуясь этой операцией, можно получить из числа, заданного прямым кодом, его отрицательный модуль путем сложения этого числа с кодом 1,00 ... 0.

4. Поразрядное логическое умножение. Логическое умножение двоичных цифр определяется таблицей, приведенной на стр. 19). Эта операция позволяет из набора $a_0a_1 \dots a_n$ выбрать любые его разряды. Для этого нужно заданный набор умножить на набор, в котором в соответствующих разрядах стоят 1, а в остальных разрядах —0. Таким образом выясняется знак числа, абсолютная величина порядка числа или цифровой его части, код операции или код какого-либо адреса. Эта операция может быть использована для выделения той или другой части кода.

5. Сдвиг. Операция сдвига заключается в смещении цифр кода и на фиксированное число разрядов влево или вправо. Сдвиг кода $a_0a_1 \dots a_n$ на s разрядов влево заменяет его на код $a_{s+1}a_{s+2} \dots a_n \underbrace{0 \dots 0}_{s}$. Сдвиг на s разрядов вправо дает код

$$\underbrace{0 \dots 0}_{s}\ a_0a_1 \dots a_{n-s}.$$

В машинах с плавающей запятой существует операция сдвига цифровой части числа. Операция сдвига в машинах с фиксированной запятой эквивалентна умножению или делению без округления на 2^k.

6. **Операция сравнения.** Эта операция выясняет, какое из двух чисел a и b больше (меньше). Для сравнения двух чисел a и b производится вычитание $a - b$, и по знаку разности

определяется, какое из этих чисел больше. Случай равенства a и b воспринимается по разности в зависимости от того, какой код применяется для представления чисел. При дополнительном коде нуль имеет один знак плюс и, следовательно, знак равенства воспринимается как неравенство $a > b$. При обратном коде вычитание равных чисел дает отрицательный нуль

$$[-0]_{\text{обр}} = 1,11\ldots1$$

и, следовательно, знак равенства воспринимается как неравенство $a < b$.

7. Точность производства операций и округление результата. Точность выполнения операций над числами в машине определяется видом операции и способом ее реализации.

1. Операция сдвига. Выполнение этой операции соответствует умножению на 2^k. При допустимом сдвиге влево не происходит потери значащих цифр. Такой сдвиг есть точная операция умножения на 2^k, где k — положительное число. При сдвиге вправо на p разрядов последние p значащих цифр теряются. В результате операции сдвига вправо по сравнению с точным делением на 2^p появляется ошибка, меньшая единицы последнего разряда.

Наибольшая ошибка, равная двоичному числу

$$10^{-n}(1-10^{-p}),$$

произойдет при наличии единиц во всех p отбрасываемых разрядах. В соответствии с этим в машинах с плавающей запятой при нормализации числа с цифровой частью, большей 1 по абсолютной величине, совершается ошибка, не превосходящая единицы последнего разряда цифровой части числа.

2. Сложение и вычитание. В машинах с фиксированной запятой операции сложения и вычитания являются точными. При сложении и вычитании чисел в машинах с плавающей запятой возникают сдвиги для выравнивания порядков и нормализации результатов, которые создают ошибки. Выравнивание порядков дает ошибку, меньшую единицы последнего знака, однако после нормализации влево ошибка может превратиться даже в единицу первого знака. Действительно, пусть $n=6$, $a = 10^{11}\cdot 0,100000$, $b = 10^{10}\cdot 0,111111$. При вычитании выравниваются порядки: $b_1 = 10^{11}\cdot 0,011111$ и $a-b = 10^{11}\cdot 0,000001 = 10^{-10}\cdot 0,100000$. Точное значение разности есть $10^{-10}\cdot 0,01$.

Такое положение может возникнуть при условии, если порядки отличаются на единицу. Такой рост ошибки исключается при наличии в сумматоре дополнительного разряда, т. е. если число цифровых разрядов сумматора на единицу больше, чем цифровых разрядов в ячейках ЗУ (запоминающего устройства). В самом деле, если порядки чисел отличаются больше чем на

единицу, то при нормализации результата может происходить сдвиг влево лишь на один разряд, что дает ошибку, меньшую единицы последнего разряда. Нормализация результата больше чем на один разряд может произойти только при условии, что порядки слагаемых отличаются не больше чем на единицу. Но в этом случае наличие дополнительного разряда в сумматоре обеспечивает сохранение всех цифр и, следовательно, точность результата.

3. Умножение и деление. При умножении и делении чисел в машине ошибка не превосходит единицы младшего разряда.

Для уменьшения ошибок при производстве операций над числами в машине вводятся операции с округлением, для чего используется дополнительный разряд на сумматоре.

При производстве операций над числами с округлением ошибка, как правило, не превосходит половины единицы младшего разряда. Кроме того, при выполнении последующих операций с округлением возможна компенсация ошибок.

§ 6. Перевод из одной системы счисления в другую

При преобразовании чисел из десятичной системы в двоичную, а также из двоичной — в десятичную в качестве промежуточного этапа применяется запись в двоично-десятичной системе. В двоично-десятичной системе каждая цифра десятичной записи задается в двоичной системе. Цифры 0, 1, 2, 3, ..., 9 записываются в виде двоичных четырехзначных чисел **0000, 0001, 0010, ..., 1001.** Двоично-десятичная система менее экономна, чем двоичная, так как четыре двоичных разряда используются всего лишь для записи 10 цифр (вместо 16 возможных); запись числа в двоично-десятичной системе на 20% длиннее чисто двоичной его записи. Например, число 637 в двоично-десятичной системе имеет вид **0110 0011 0111** (12 цифр), а в двоичной записи **1001111101** (10 цифр).

Чтобы найти двоичную запись числа $a=\sum_{k=1}^{n} a_k \cdot 10^{m-k}$, каждая цифра которого a_k задана в виде четверки двоичных цифр: $a_k=\alpha_{k1}\alpha_{k2}\alpha_{k3}\alpha_{k4}$ $(k=1, 2, \ldots, n)$, нужно последовательно выделять четверки цифр числа a (т. е. коэффициенты a_k) и производить вычисление многочлена $\sum_{k=1}^{m} a_k t^{m-k}$ при $t=10$ (в двоичной системе $t=1010$).

Рассмотрим теперь преобразование двоичного числа в десятичное. Пусть $a=0, a_1a_2 \ldots a_n$ — двоичная запись числа ($a_k=0$

или 1, $k = 1, 2, \ldots, n$). Цифры для десятичной записи

$$a = \sum_{k=1}^{m} \alpha_k 10^{-k}$$

находим последовательно по такой схеме:

1) вычисляем целую часть числа $10a$

$$\alpha_1 = [10a];$$

2) вычисляем дробную часть числа $10a$

$$b_1 = \{10a\} = \sum_{k=2}^{m} \alpha_k 10^{-k};$$

3) вычисляем целую часть числа $10\,b_1$

$$\alpha_2 = [10b_1]$$

и т. д. Итак, последовательность цифр $\alpha_1, \alpha_2, \ldots, \alpha_m$ десятичной записи числа a находится по следующей схеме:

$$\alpha_k = [10b_k], \quad b_k = \{10b_{k-1}\}, \quad b_0 = a \quad (k = 1, 2, \ldots, m).$$

Для преобразования двоичного числа, заданного в нормальной форме $a = 10^p q$, в десятичную систему счисления цифровая часть преобразуется в десятичную систему по предыдущему правилу и определяется порядок числа s в десятичной системе из условия $10^s > a \geqslant 10^{s-1}$.

ГЛАВА II

ПРИНЦИПЫ ПРОГРАММНОГО УПРАВЛЕНИЯ НА ЦИФРОВЫХ АВТОМАТИЧЕСКИХ МАШИНАХ

Всякая вычислительная машина с точки зрения математика представляет собой устройство, способное выполнять некоторый набор арифметических и логических операций, хранить и выдавать сведения о результатах этих операций.

В зависимости от того, может ли машина выполнять лишь определенную последовательность операций, обеспечивающую решение некоторого специального круга задач, или эти последовательности операций, а значит, и задачи, решаемые на машине, могут быть самыми различными, цифровые электронные машины разделяются на два типа: специализированные и универсальные.

В этой главе мы изложим основы программного (автоматического) решения задач на универсальных электронных машинах.

§ 1. Принципиальная схема *ЦАМ*

1. Общие принципы построения. Как уже отмечалось, электронные вычислительные машины универсального типа предназначены для решения различных вычислительных и логических задач, требующих большого количества действий и высокой точности.

Общие принципы построения современных универсальных *ЦАМ* в некотором смысле аналогичны принципам конструирования ранее известных цифровых машин (электрические клавишные машины и др.). Принципиально новым в электронных счетных машинах, помимо применения современных средств электроники, является применение схем, обеспечивающих запоминание результатов вычислений, полную автоматизацию всего вычислительного процесса и исключительно большую его скорость.

После выбора численного метода алгоритм решения задачи для реализации на *ЦАМ* должен быть расчленен на последовательность вычислительных операций, выполняемых машиной,

которые мы в дальнейшем будем называть элементарными операциями.

Для обеспечения больших скоростей вычислений, разумеется, недостаточно огромного быстродействия выполнения только собственно вычислительных операций, так как последние составляют лишь одну сторону вычислительного процесса, другой неотъемлемой частью которого является указание необходимого порядка работы — программы вычислений. Для полной автоматизации вычислительного процесса на машине должна быть реализована возможность быстродействующего сообщения ей этих программных указаний. Такое автоматическое выполнение всех расчетов по заданной программе реализуется с помощью устройства управления, которое в связи с этим еще называют устройством программного управления.

Универсальная вычислительная машина состоит из ряда устройств — блоков, каждый из которых имеет специальное назначение. Основные блоки машины следующие:

1) устройство ввода данных,
2) запоминающее устройство (*ЗУ*),
3) арифметическое устройство (*АУ*),
4) устройство (автоматического и ручного) управления (*УУ*),
5) устройство вывода результатов.

Блок-схема *ЦАМ* приведена на рис. 13; стрелки указывают на связь между отдельными устройствами, состоящую в возможности передачи закодированной информации (хранящейся или вырабатываемой в блоке) из одного устройства в другое.

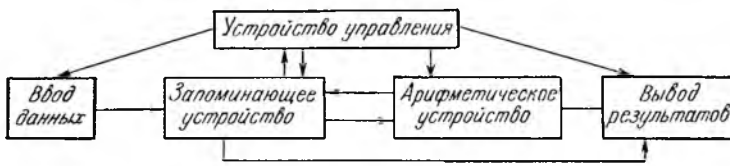


Рис. 13.

Устройство ввода предназначается для передачи в *ЗУ* закодированной информации, необходимой для решения задачи. Чаще всего работа этого устройства осуществляется автоматически с помощью перфорированных карт или лент.

Устройства *ЗУ*, *АУ*, *УУ* предназначены для непосредственного выполнения вычислительных операций.

Запоминающее устройство служит для хранения вводимых в машину данных и результатов промежуточных и окончательных вычислений.

Арифметическое устройство современной вычислительной машины представляет собой объединение электронных вычисли-

тельных схем, выполняющих с колоссальной скоростью различные операции над поступающими в него кодами. При этом каждой машине присущ свой набор операций, выполняемых *АУ* над кодами. Для выполнения каждой отдельной операции *АУ* настраивается (включается) специальным образом с помощью коммутатора операций.

Управляющее устройство осуществляет последовательность вычислений в соответствии с заданной программой.

Устройство вывода предназначается для выдачи из машины результатов счета, являющихся согласно программе окончательными для данной задачи. Чаще всего оно реализуется в виде печатающего устройства, с помощью которого искомые результаты наносятся в определенной последовательности, например, на ленту.

2. Запоминающее устройство. *ЗУ* состоит из ряда занумерованных ячеек, обычно имеющих одинаковое число разрядов. Разряды ячеек в свою очередь также занумерованы. В каждом разряде может храниться 0 или 1. Совокупность нулей и единиц, хранящихся в ячейке, называется *кодом*, хранящимся в ней.

Обычно одна и та же ячейка *ЗУ* может служить как для хранения кодов чисел (с выбранной фиксацией запятой), так и для хранения управляющих кодов (так называемых приказов или команд), посредством которых выполняются элементарные операции в машине и определяется порядок их следования. Для удобства номерами команд или чисел называют номера соответствующих ячеек.

Как указано на блок-схеме, *ЗУ* связано с устройствами ввода, вывода, с арифметическим и управляющим устройствами. Исходная информация решаемой задачи (числовые данные и команды программы) поступает через устройство ввода (в закодированном виде) в предусмотренные программой ячейки *ЗУ*. *В* устройство управления из *ЗУ* поступают коды управляющих инструкций — команды программы, под воздействием которых извлекаются коды из *ЗУ* и отсылаются в *АУ* для выполнения над ними операций; из *АУ* поступают в *ЗУ* результаты вычислений. Согласно программе результаты вычислений из *ЗУ* поступают в выводное устройство.

Запоминающее устройство обычно состоит из двух частей: внутренней и внешней. *Внутренняя память* отличается от внешней большей скоростью работы, т. е. из нее извлекаются или помещаются в нее с большей скоростью коды; она предназначена для автоматического использования непосредственно *АУ*. Ячейки внутренней памяти имеются двух видов: *пассивные* (постоянные) и *активные* (переменные, оперативные). Как из ячеек пассивной, так и из ячеек активной памяти посредством *УУ*

(в соответствии с программой) могут извлекаться (считываться) коды и отсылаться в *АУ* для выполнения над ними операций; содержащиеся в этих ячейках коды при этом сохраняются. Однако результаты вычислений могут отсылаться только в ячейки активной памяти, вытесняя при этом (стирая) прежде хранившиеся там коды. Это обстоятельство учитывается при составлении программ.

Емкость внутренней части запоминающего устройства (количество отдельных кодов, которое может одновременно храниться в нем) в значительной мере определяет возможности машины. Однако объем внутренней памяти ограничивается как чисто конструктивными соображениями, так и необходимостью уменьшения времени, которое требуется для выборки кодов из памяти. Поэтому в современных *ЦАМ* обычно имеется большого размера *внешняя память* (иногда нескольких видов), конструктивно реализуемая более простыми средствами. Чаще всего это — «магнитная» память, на магнитных барабанах, магнитных лентах или магнитной проволоке.

Внешняя память непосредственно с *АУ* не связана; коды, находящиеся в ней, согласно программе поступают группами во внутреннюю активную память, а также из внутренней памяти могут передаваться группами во внешнюю. Время выборки кодов из внешней памяти относительно велико, однако объем внешней памяти (количество кодов) обычно значительно превосходит объем внутренней памяти.

3. Устройство программного управления. Устройство программного управления машины имеет регистр, в котором помещаются последовательно один за другим управляющие коды — команды выполняемой программы.

Совокупность команд, необходимых для решения данной задачи, занумерованных согласно их размещению во внутренней памяти, вместе с соответствующим образом занумерованными исходными данными (включая постоянные, встречающиеся в решении задачи) представляет собой *программу задачи.*

В программе отмечается начальная команда, с которой должна быть начата работа программы.

Помимо команд вычислительного назначения, т. е. команд, связанных с выработкой посредством *АУ* кодов-значений функций от соответствующих кодов-аргументов, для автоматического решения задачи на машине необходимы специальные *управляющие* команды, посредством которых происходит обращение к внешним устройствам (ввода, вывода, внешней памяти), а также реализующие предусмотренную программой последовательность выполнения операций.

Действие устройства управления в каждый данный момент времени определяется тем, какая команда находится в его

регистре, и завершается засылкой соответствующего кода команды из *ЗУ* в этот регистр. Тем самым машина автоматически переходит к выполнению следующей команды.

В целях контроля над выполнением программы (или ее части) по отдельным командам, помимо автоматического управления, в машинах устанавливается система *ручного управления*. С помощью ручного управления нажатием пусковой кнопки выполняется также начальная команда, т. е. происходит запуск машины на автоматическую работу.

В зависимости от конструкции машины разделяются на машины с *последовательным* (или *естественным*) и *принудительным* порядком выполнения команд.

В машинах первого типа после выполнения k-й команды (т. е. команды, хранящейся в k-й ячейке *ЗУ*) выполняется $(k+1)$-я команда (или, как говорят, передается управление $(k+1)$-й команде) и т. д. Исключение составляют так называемые *команды передачи управления*, т. е. команды, с помощью которых в зависимости от результатов предшествующих вычислений в соответствующих местах программы выясняется и реализуется необходимость продолжения расчетов по тем или иным частям программы.

При этом заметим, что последовательное перечисление элементарных вычислительных операций, необходимых для решения задачи, привело бы к программам огромной длины, само составление которых занимало бы времени не меньше, чем выполнение всех расчетов при ручном счете. Кроме того, такая программа чрезмерно загрузила бы ячейки *ЗУ*. Поэтому задачей программиста является составление таких программ, по которым при сравнительно небольшом числе команд может быть выполнено большое количество отдельных вычислений (за счет многократного использования одних и тех же команд, что осуществляется с помощью операций передач управления).

В машине с последовательным порядком выполнения команд каждая команда вычислительного назначения должна содержать шифр выполняемой операции и номера ячеек, над кодами которых должна быть выполнена эта операция и в которые должен быть отослан результат вычислений. Для составления команд не обязательно знать число, над которым производится операция, достаточно в команде указать номер ячейки — *адрес*, в которую оно было помещено в процессе ввода или в результате выполнения какой-либо предыдущей операции. Таким образом, не зная заранее результатов предшествующих вычислений, можно составлять команды для выполнения операций над ними, указывая в адресах номера ячеек, в которые эти результаты отсылаются для хранения.

Поскольку в адресах команд указываются не числа, а лишь их номера, число двоичных разрядов, необходимых для кодировки команд, может быть небольшим при небольшой емкости внутреннего *ЗУ*. Действительно, для кодировки каждого адреса достаточно ограничиться числом разрядов n, если число ячеек внутреннего *ЗУ* $M \leqslant 2^n$.

Число адресов, которые могут быть даны в одной команде, определяет *адресность* машины.

В машинах с принудительным порядком выполнения команд в каждой команде указывается адрес ячейки, хранящей следующую команду.

Из сказанного ясно, что одноадресные машины должны работать по принципу последовательного выполнения команд. Кроме того, очевидно, что одноадресные машины должны иметь в *АУ* один или несколько регистров для хранения промежуточных результатов. Арифметические действия при этом расчленяются на несколько этапов таким образом, чтобы каждая элементарная операция машины относилась только к одной из величин. В связи с этим в одноадресных машинах список элементарных операций должен содержать, например, такие операции, как «взятие числа из ячейки α и отсылка его в регистр *АУ* с предварительной очисткой этого регистра»:

$$P_1\alpha;$$

«прибавление числа, содержащегося в ячейке α, к содержимому регистра *АУ*»:

$$P_2\alpha;$$

«взятие числа из регистра *АУ* и отсылка его в ячейку α»:

$$P_3\alpha,$$

и т. д. Так, например, для выполнения следующей операции: «сложить число, хранящееся в ячейке α, с числом, хранящимся в ячейке β, и поместить результат в ячейку γ», в программу для одноадресной машины следует поместить три команды:

$$P_1\alpha;$$
$$P_2\beta;$$
$$P_3\gamma.$$

В трехадресной машине с естественным порядком выполнения команд для этой же цели понадобится только одна команда: $P\alpha\beta\gamma$. Вместе с тем для сложения чисел, хранящихся в ячейках $\alpha+1$, $\alpha+2$, ..., $\alpha+n$, с помещением результата в ячейку γ в одноадресной машине потребуется $n+1$ команда, а в трехадресной — $n-1$ команда. В среднем же программы для трех-

адресных машин примерно вдвое короче соответствующих программ для машин одноадресных.

Программа, составленная для определенной машины (с определенной адресностью, способом выполнения операций и их набором), без особых затруднений может быть переработана в программу для любой другой универсальной машины. Поэтому мы в дальнейшем будем излагать основы программирования для трехадресных машин с последовательным выполнением операций. При этом мы не станем ограничиваться каким-либо определенным набором выполнимых на машине элементарных операций.

Запись команд для трехадресных машин может быть представлена в таком достаточно удобном виде:

Код операции	I адрес	II адрес	III адрес

Так, в трехадресной машине *Стрела*, работающей на 43 разрядах, для кодировки знака операции отводится 6 разрядов и на каждый из адресов — 12 разрядов; один разряд — контрольный знак *)

I адрес	II адрес	III адрес	Контрольный знак	Код операции
0—11	12—23	24—35	36	37—42

Рассмотрим, что происходит в трехадресной машине с последовательным порядком выполнения команд после поступления в *УУ* из ячейки *ЗУ* с номером *n* кода, представляющего собой команду вычислительного характера.

Работа, вызванная выполнением этой команды, может быть разделена на четыре такта.

Первый такт. Код операции из регистра *УУ* передается на коммутатор операций и настраивает *АУ* на выполнение соответствующей этому коду операции.

Второй такт. Содержимое первого адреса команды (*IA*) передается на коммутатор (внутреннего) *ЗУ*; открывается ячейка *ЗУ*, номер которой указан в *IA*, в результате чего содержимое этой ячейки считывается на соответствующий регистр *АУ*.

Третий такт. Содержимое ячейки, номер которой указан в *IIA* (во втором адресе), считывается на соответствующий регистр *АУ*.

Четвертый такт. *АУ* вырабатывает результат операции; содержимое *IIIA* регистра *УУ* передается на коммутатор *ЗУ*;

*) Контрольный знак используется при проверке программы.

открывается ячейка, номер которой указан в *IIIA* выполняемой команды, и выработанный в *АУ* код поступает в эту ячейку (вытесняя прежде хранившийся в ней код); на регистр *УУ* поступает содержимое следующей по номеру команды — код ячейки с номером $n + 1$.

§ 2. Элементарные операции, выполняемые *ЦАМ*

Как указывалось, для решения тех или иных математических или логических задач конструктивно в машинах предусматривается возможность выполнения определенного набора элементарных операций, кодируемых в виде отдельных команд. В зависимости от адресности и других особенностей машины можно указать те или иные необходимые наборы элементарных операций; однако ограничение конструкций машин такими наборами приводит к значительному удлинению программ и усложнению процесса программирования, а тем самым к загромождению памяти машины и усложнению процесса ввода. Поэтому обычно выбирается достаточно широкий набор элементарных операций, включающий минимум необходимых.

Здесь мы рассмотрим элементарные операции для машины с трехадресным кодом команд. Элементарные операции, выполняемые машиной, в зависимости от назначения можно разбить на несколько групп:

1) основные арифметические операции;
2) дополнительные операции вычислительного назначения;
3) логические операции;
4) операции обращения к внешним устройствам;
5) операции над командами (команды переадресации);
6) операции передачи управления.

1. Основные арифметические операции. К основным арифметическим операциям относятся операции сложения, вычитания умножения и деления. При этом команды имеют запись:

1. Сложение (+). Команда

+	α	β	γ

означает «сложить число, номер которого α указан в первом адресе (*IA*), с числом, номер которого β указан во втором адресе (*IIA*), и результат направить в ячейку *ЗУ*, номер которой γ указан в третьем адресе (*IIIA*)».

2. Вычитание (—). Команда

—	α	β	γ

означает «вычесть из числа, номер которого α указан в *IA*, число, номер которого β указан в *IIA*, результат направить в ячейку *ЗУ*, номер которой γ указан в *IIIA*».

В машинах с плавающей запятой перед сложением и вычитанием чисел выравниваются их порядки, результат операции нормализуется и округляется.

3. Умножение (×). Команда

×	α	β	γ

означает «умножить число, номер которого α указан в *IA*, на число, номер которого β указан в *IIA*, и результат направить в ячейку *ЗУ*, номер которой γ указан в *IIIA*».

4. Деление (:). Команда

:	α	β	γ

означает «разделить число, номер которого α указан в *IA*, на число, номер которого β указан в *IIA*, и результат направить в ячейку *ЗУ*, номер которой γ указан в *IIIA*».

В операциях умножения и деления результаты вычислений округляются, а в машинах с плавающей запятой еще и нормализуются. В некоторых машинах предусматривается выполнение операции умножения без округления результата, умножения с выводом удвоенного количества разрядов, а также деления с выводом остатка.

После выполнения любой команды рассматриваемой группы выполняется следующая по номеру команда программы, т. е. управление передается (посредством *УУ*) к следующей по номеру команде.

2. Дополнительные операции вычислительного назначения. Такими операциями могут быть, например:

1. Разность модулей (|—|). Команда

|—|	α	β	γ

означает «из модуля числа, номер которого α указан в *IA*, вычесть модуль числа, номер которого β указан в *IIA*, результат поместить в ячейку, номер которой γ указан в *IIIA*».

В машинах с плавающей запятой результат данной операции нормализуется.

2. Минимум двух чисел (min). Команда

min	α	β	γ

означает «из двух чисел, номера которых α и β указаны в I*A* и II*A*, выбрать минимальное и поместить в ячейку, номер которой γ указан в III*A*».

3. Минимум модулей двух чисел (|min|). Команда

|min|	α	β	γ

означает «из двух чисел, номера которых α и β указаны в I*A* и II*A*, выбрать число с меньшим модулем и поместить его модуль в ячейку, номер которой γ указан в III*A*».

Аналогично последним двум командам могут быть предусмотрены следующие:

4. Максимум двух чисел (max). Команда имеет такой вид

max	α	β	γ

5. Максимум модулей двух чисел (|max|). Общий вид команды такой:

|max|	α	β	γ

6. Извлечение квадратного корня ($\sqrt{}$). Команда

$\sqrt{}$	α		β

означает «из числа, номер которого α указан в I*A*, извлечь квадратный корень и результат поместить в ячейку β, номер которой указан в III*A*». Второй адрес команды в данной операции не используется.

Аналогично операции 6 машина может выполнять такие операции, как:

7. Образование модуля числа (mod). Команда имеет такой вид:

mod	α		β

8. Выделение дробной части числа ({ })

{ }	α		β

9. Выделение целой части числа ([ ]).

[ ]	α		β

Последние две команды, естественно, имеют смысл для машины с плавающей запятой.

10. Сдвиг по числу ($\rightarrow q$). Команда

$\rightarrow q$	α	β	γ

означает «разряды числа, номер которого β указан в II*A*, сдвинуть на число разрядов, равное порядку числа, номер которого α указан в I*A*, вправо или влево, в зависимости от знака порядка числа α, результат поместить в ячейку γ, номер которой указан в III*A*». Операция имеет смысл для машин с плавающей запятой.

11. Сдвиг по адресу вправо ($\rightarrow$)

$\rightarrow$	α	β	γ

12. Сдвиг по адресу влево ($\leftarrow$)

$\leftarrow$	α	β	γ

Команды 11 и 12 означают то же, что и в операции 10, но сдвиг производится вправо на число разрядов, равное числу α, помещенному в 1*A* в операции 11, и влево — в операции 12. Здесь как исключение первый адрес играет роль числа, а не роль адреса.

В некоторых машинах предусматривается использование свободных адресов команд. Например, можно объединить команды 8 и 9, предусмотрев, чтобы в ячейке, номер которой указан в II*A*, образовалась целая часть числа, а в ячейке, номер которой указан в III*A*, — его дробная часть.

13. Циклическое сложение ($+Ц$)

$+Ц$	α	β	γ

Операция 13 отличается от операции обычного сложения тем, что в случае появления переноса в старшем разряде последний передается в младший разряд результата. Операция 13 используется обычно для целей контроля правильности вычислений и контроля ввода программ.

Как и в случае команд, относящихся к группе I, после выполнения каждой из приведенных команд управление передается к следующей по номеру команде.

3. Логические (поразрядные) операции. Приведем несколько примеров логических операций, наличие которых в *ЦАМ* значительно упрощает решение математических задач, а также позволяет весьма просто программировать решение логических задач.

1. Логическое (поразрядное) умножение ($\times Л$). Команда

$\times Л$	α	β	γ

означает «код числа, номер которого α указан в $1A$, логически умножить на код числа, номер которого β указан в $2A$, и результат поместить в $3A$». Другими словами, в каждый разряд ячейки γ заносится 0 или 1, в зависимости от наличия 0 или его отсутствия в соответственных разрядах чисел α и β, а именно:

α	β	γ
0	0	0
0	1	0
1	0	0
1	1	1

Операция $\times Л$ может быть использована для выделения некоторых разрядов данного кода, например для выделения одного из адресов команды. Для этого данное число следует умножить логически на число, у которого на местах выделяемых разрядов стоят единицы, а на местах остальных разрядов поставлены нули.

2. Логическое (поразрядное) сложение ($+Л$)

$+Л$	α	β	γ

Здесь значение разрядов числа в ячейке γ определяется из таблицы

α	β	γ
0	0	0
0	1	1
1	0	1
1	1	1

Операция $+Л$ может быть использована для формирования кода из подготовленных его частей, например для формирования команды из адресов, подготовленных с помощью операции $\times Л$.

3. Проверка совпадения $\sim$

$\sim$	α	β	γ

Каждый разряд кода в III*А* определяется поразрядно, в зависимости от значений в соответствующих разрядах I*А* и II*А*, по правилу:

α	β	γ
0	0	1
0	1	0
1	0	0
1	1	1

Таким образом, если коды разрядов в ячейках α и β совпадают, то в соответствующем разряде ячейки γ появляется 1, и в противном случае 0.

Операция $\sim$ удобна для выяснения равенства двух чисел (совпадения кодов), так как только при полном совпадении кодов во всех разрядах III*А* появятся единицы, что и будет служить признаком совпадения кодов.

4. Логическое отрицание $(-Л)$

$-Л$	α		γ

В каждый разряд числа, расположенного в ячейке γ, заносится 0, если в соответствующем разряде числа, расположенного

в ячейке α, имеется единица, и 1 — в противном случае. Второй адрес команды не используется.

После выполнения каждой из приведенных команд этой группы управление передается следующей по номеру команде.

4. Операции обращения к внешним устройствам. К операции обращения к внешним устройствам относятся операции ввода, записи, считки кодов, печати и останова.

Обычно различные пересылки кодов являются групповыми операциями и производятся в порядке возрастания номеров ячеек; при этом кодируется начальный и конечный номера входящих в операцию ячеек или начальный номер группы ячеек и их количество. Понятно, что время, необходимое для выполнения команд данной группы, зависит от количества передаваемых чисел.

1. Ввод кодов (*Вв*)

Вв	n	$\alpha+1$	

Содержание команды *Вв* следующее: «ввести в n ячеек $\alpha+1$, $\alpha+2$, ..., $\alpha+n$ оперативного запоминающего устройства соответствующие коды из вводного устройства (например с перфокарт)».

Обращение к внешней памяти может быть двух видов: пересылка кодов из ячеек внутреннего *ЗУ* во внешнее — «запись» и, наоборот, пересылка кодов из внешнего *ЗУ* в соответствующие ячейки внутреннего *ЗУ* — «считка». В обоих случаях кодирование операции производится в две команды.

2. Запись (передача кодов из внутреннего *ЗУ* во внешнее) (*Ва, Вб*)

Ва	n	α	γ
Вб	$З$	β	

В первой команде в IA указывается, например, номер n барабана или ленты внешнего *ЗУ*, куда производится запись, в IIA — номер α ячейки барабана или ленты, с которой можно начинать запись, а в $IIIA$ — номер ячейки γ внутреннего *ЗУ*, с которой начинается операция.

Во второй команде в IA указывается вид операции — «запись», в IIA — номер числа во внешней памяти β, до которого можно производить запись, $IIIA$ второй команды не используется.

3. Считка (передача кодов из внешнего *ЗУ* во внутреннее) (*Ва*, *Вб*)

Ва	*n*	α	γ
Вб	*c*	β	

Значение адресов то же, что и в предыдущем случае, кроме *IA* второй команды, где указывается вид операции «считка».

Искомые результаты расчетов перед печатью или другими способами вывода должны переводиться из двоичной системы в десятеричную. В некоторых машинах эта операция выполняется при обращении к соответствующей команде *).

4. Печать (*П*). Команда

П		*n*	α

означает «напечатать десятеричные коды группы чисел, содержащих (*n* + 1) число, начиная с номера α, указанного в *IIA*».

В зависимости от наличия в машине нескольких выводных устройств в наборе операций необходимо иметь соответствующее число команд вида 4. Для того чтобы различать такие команды, можно использовать свободный первый адрес.

5. Останов (*ост.*)

ост.			

После выполнения этой команды вычисления прекращаются.

После выполнения команд этой группы, кроме команды останова, управление также передается следующей по номеру команде программы.

5. Команды переадресации или операции над командами. Кодирование команд в виде набора цифр значительно расширяет возможности при программировании задач, т. е. позволяет в необходимых случаях в процессе вычислений на машине производить преобразование команд с помощью специальных операций.

Однако, как мы видели, например, в машинах с плавающей запятой перед операциями сложения и вычитания производится

*) В машинах, в схемах которых не предусмотрен перевод чисел в десятеричную систему, в программах задач перед обращением к выводному устройству предусматривается обращение к специальной стандартной подпрограмме перевода из двоичной в десятеричную систему.

выравнивание порядков чисел и результат операции нормализуется. В этих же машинах разряды, отведенные для кодирования порядка числа, обычно используются для кодирования вида операции команды. А так как над командами выравнивания порядков и нормализацию результата производить не нужно, удобно иметь в наборе элементарных операций, выполняемых машиной, специальные команды для операций над командами. Поскольку при рассмотрении содержимого ячейки как команды последнее разбивается на части — код операции и адреса и изменения команд связаны с изменением одной или нескольких из этих частей, — команды изменения команд получили название команд переадресации.

1. Сложение команд ($\oplus$)

$\oplus$	K_1	α	K_2

2. Вычитание команд ($\ominus$)

$\ominus$	K_1	α	K_2

Содержание 1-й команды следующее: прибавить, не выравнивая предварительно порядки, к числу, номер которого K_1 указан в I*A*, число, номер которого α указан в II*A*; результат, не нормализуя, переслать в ячейку, номер которой K_2 указан в III*A*. Вторая команда аналогична первой.

Понятно, что операция, выполнимая командой типа команды 2, может быть выполнена командой типа команды 1, если в ячейку α будет помещено соответствующее отрицательное число.

Увеличение I*A*, II*A*, III*A* на 1, 2, 3, ... соответствует прибавлению к числу, представляющему команду, определенного числа единиц в соответствующих разрядах (например, для машины *Киев* эти числа соответственно равны 2^{-16}, 2^{-28}, 2^{-40} и т. д.). Такие числа будем в дальнейшем обозначать через «1» I*A*, «2» I*A*, «1» I*A* + «2» II*A* и т. д. или же записывать в виде таблицы

—	1	—	—
—	2	—	—
—	1	2	—

и т. д.

Как и в случае предыдущих групп команд после выполнения команд 1 и 2 управление передается следующей по номеру команде программы.

6. Операции передачи управления. Для расчетов на автоматических машинах представляют интерес задачи, решение которых может быть расчленено на полностью или частично повторяемые вычислительные этапы и, следовательно, в которых одна команда или группа команд может быть использована многократно, т. е. задачи, для которых могут быть составлены программы, число команд которых во много раз меньше числа необходимых для их решения элементарных операций.

Как правило, почти все сложные математические (и логические) задачи обладают свойством повторяемости тех или иных расчетных этапов в зависимости от результатов расчета. Однако для программирования таких задач введенных ранее (в более или менее полном составе) команд групп **1—5** явно недостаточно, так как после выполнения каждой такой команды выполняется следующая по номеру команда программы. Этот список должен быть пополнен командами, позволяющими нарушать обычный порядок вычислений и передавать управление команде, не расположенной непосредственно за только что выполненной, или передавать управление одной из двух (или нескольких) команд для выбора того или иного продолжения процесса счета в зависимости от определенных *признаков* — условий, заранее сформированных или вытекающих из результатов вычислений. Таким образом, необходимо иметь возможность проверить выполнение соответствующих условий — *признаков* и в зависимости от этого осуществить передачу управления одному или другому участку программы. Поскольку в качестве такого рода признаков в различных машинах принимаются те или иные свойства результатов вычислений, разные машины могут иметь различные, так называемые команды *условной передачи управления* (одного или нескольких видов).

Заметим при этом, что каждая конкретная программа в значительной степени определяется не только избранным методом решения задачи, но и особенностями данной машины, в первую очередь ее адресностью и списком операций, среди которых существенная роль принадлежит командам передачи управления.

Приведем несколько команд, осуществляющих передачу управления в различных машинах.

1. Передача управления в зависимости от сравнения двух чисел с учетом знаков ($\leqslant$)

$\leqslant$	α	β	k

Это значит — сравнить число, находящееся в ячейке α, с числом, находящимся в ячейке β, и, если первое из них больше второго, перейти к выполнению следующей команды программы, а если первое меньше или равно второму, перейти к выполнению команды, номер которой k указан в третьем адресе.

Таким образом, в качестве признака, служащего критерием для выбора между передачей управления команде, номер которой указан в $III A$ данной команды, и следующей по номеру команде, здесь принят знак разности чисел, расположенных в ячейках α и β*).

2. Передача управления в зависимости от сравнения модулей двух чисел ($|\leqslant|$)

$\|\leqslant\|$	α	β	k

Эта операция аналогична 1, с той только разницей, что сравниваются не сами числа, а их модули.

3. Передача управления в зависимости от равенства двух чисел ($=$)

$=$	α	β	k

Другими словами, сравнить число, находящееся в ячейке α, с числом, находящимся в ячейке β, и если эти числа равны, передать управление команде, указанной в третьем адресе, в случае неравенства перейти к выполнению следующей команды.

4. Передача управления в зависимости от неравенства двух чисел ($\neq$)

$\neq$	α	β	k

Эта операция аналогична операции 3, различие состоит только в том, что управление в случае неравенства чисел, расположенных в ячейках α и β, передается команде, указанной по третьему адресу, в противном случае — следующей по порядку команде. Для передачи управления употребляются также команды, использующие другие соотношения между числами ($<$, $>$ и др.).

Заметим, что если необходимы ответвления к двум различным командам k_1 и k_2, каждую из которых нельзя помещать

*) При этом всюду мы будем считать, что в результате вычитания равных между собой чисел получается «—0» *(отрицательный нуль)*

$$a - a = -0.$$

непосредственно после выполняемой команды, это можно осуществить следующим образом:

$\leqslant$	α	β	k_1
$\leqslant$			k_2

Такие разветвления могут иметь место, например, при обращении к двум разным стандартным подпрограммам в зависимости от результатов вычислений (см. гл. III, § 6).

Команды 1—4 могут быть также использованы для осуществления обязательной передачи управления (безусловного перехода) команде, указанной по третьему адресу*). Для этого достаточно в командах 1, 2 и 3 в IA и IIA указать одно и то же число, удобнее всего при этом использовать *нулевую* ячейку**). Тогда команды безусловной передачи управления принимают следующий вид:

$\leqslant$			k
$\lvert\leqslant\rvert$			k
$=$			k

Таким образом, в командах безусловного перехода, по существу, остаются неиспользованными разряды IA и IIA. В машинах, в которых команды безусловного перехода кодируются особо, имеется возможность использовать разряды IA и IIA.

Если же в качестве операции безусловного перехода использовать операцию 4, то в один из адресов (IA или IIA) следует поместить ячейку с числом a, заведомо отличным от $+0$ (например, ячейку, содержащую некоторую отличную от нуля константу).

5. Передача управления по признаку (операция условного перехода). В некоторых машинах (например, *Стрела, Урал*) при выполнении арифметических,

*) Можно показать, что для программирования любых задач наличие в списке команд одной из команд группы 6 наряду с необходимым набором команд групп 1—5 является достаточным.

**) При этом предполагаем, что *нулевая* ячейка содержит код «+0».

логических и некоторых других операций в зависимости от результатов вычислений в специальном запоминающем устройстве (триггере) вырабатывается *признак*, который сохраняется до следующей команды и используется, если команда является командой передачи управления. Таким признаком может служить, например, знак минус в результате выполнения сложения или вычитания, получение результата, большего единицы (в машине с плавающей запятой), при выполнении операции деления, образование единиц во всех разрядах результата при выполнении логических операций и пр.

В таких машинах существует команда условной передачи управления по признаку (*УПП*)

УПП	k_1	k_2	

Это означает — передать управление команде, указанной по первому адресу, если в результате выполнения предыдущей операции в триггере выявился признак, и команде, указанной во втором адресе, в противном случае. Третий адрес в данной операции не используется.

Если в первом и во втором адресах поместить номер одной и той же команды k, то команда *УПП* будет осуществлять операцию безусловной передачи управления команде k

УПП	k	k	

Таким образом, для использования операции передачи управления по признаку в виде операции условного перехода необходимо предусмотреть выработку соответствующего признака непосредственно в предыдущей команде, если же эта операция выполняется как операция безусловного перехода, то безразлично, вырабатывается признак в предыдущей команде или нет.

Операция условного перехода удобна тем, что в необходимых случаях она позволяет в местах разветвлений в зависимости от результатов предыдущих вычислений (что определяется существованием признака в предыдущей команде) переходить к выполнению необходимых команд.

6. Передача управления по знаку числа. В некоторых задачах выяснение необходимости разветвлений в нескольких местах программы может быть связано с тем или иным знаком одного и того же числа.

В таком случае при использовании операции передачи управления вида *УПП* перед каждым из разветвлений необходимо в команде, предшествующей операции *УПП*, повторить форми-

рование команды, используя знак этого числа α (например, повторяя каждый раз команду: сложить число α с 0, если в качестве признака принят знак «—» в результате операции сложения).

Учитывая это, удобно вместо *УПП* включить в список операций следующую операцию передачи управления по числу (*УПЧ*):

УПЧ	α	k_1	k_2

Содержание этой команды такое: «если число, номер которого α указан в $1A$, имеет знак «+», управление передается команде, указанной во втором адресе; если этот знак «—», то управление передается команде, номер которой указан в третьем адресе».

Операцию безусловного перехода к команде k в этом случае можно осуществить операцией *УПЧ*, в которой в первом адресе помещен код «+0» (использована нулевая ячейка):

УПЧ		k	

или помещая во второй и третий адреса этой команды номер ячейки k:

УПЧ	α	k	k

,

которой должно быть передано управление. Содержание первого адреса в последнем случае роли не играет.

ГЛАВА III

ЭЛЕМЕНТАРНОЕ ПРОГРАММИРОВАНИЕ

Решению задач на *ЦАМ* предшествует подготовительная работа по выбору численного метода и его обоснованию, заканчивающаяся составлением расчетных формул и составлением рабочей программы с учетом всех особенностей конкретной машины — собственно задача программирования.

Обе части этой работы взаимно связаны: составление компактных программ требует зачастую пересмотра как расчетных формул, так и избранного метода решения; с другой стороны, необходимость решения на *ЦАМ* задач, нередко требующих для своего завершения выполнения сотен тысяч и даже миллионов элементарных операций, приводит к необходимости выработки рациональных способов программирования.

В этой главе мы коснемся элементов непосредственного программирования.

§ 1. Непосредственное программирование

1. Ввод программы и вывод результатов. Для выполнения расчетов по заданной программе последняя должна быть введена в соответствующие места памяти машины. При этом в связи с наличием двух видов внутренней памяти — оперативной и пассивной — в машинах имеется два способа ввода, в зависимости от конкретной реализации которых составленная программа кодируется на перфокарты, перфоленты, осуществляется штеккерным набором и т. п. Ввод во внешнюю память машины производится чаще всего посредством внутренней памяти с помощью команд вида

Ва	*n*	α	γ
Вб	*З*	β	

реализующих групповую пересылку заранее введенных кодов из ячеек γ, $\gamma + 1$, ... внутреннего запоминающего устройства в n-ю зону внешнего запоминающего устройства от α до β.

Ввод в оперативное запоминающее устройство обычно осуществляется автоматически посредством *УУ* и специальной подпрограммы ввода, которая в простейшем случае может состоять, например, из одной команды

Вв	$\alpha + 1$	n	

реализующей ввод в ячейки $\alpha + 1$, $\alpha + 2$, ..., $\alpha + n$ оперативного запоминающего устройства соответствующих кодов, например, с перфокарт.

Команда ввода в свою очередь может быть введена в память машины с помощью вводного устройства с перфокарты (в таком случае соответствующая перфокарта помещается первой) или с пульта управления вручную, после чего для решения задачи производится запуск машины на автоматическую работу.

Возможен также и ручной ввод программы в оперативное *ЗУ*, при котором коды набираются на пульте управления и вводятся в соответствующие ячейки *ЗУ*. Однако ручной ввод сопряжен с большими затратами времени, и обычно им пользуются лишь для внесения надлежащих исправлений при отладке программы.

Пассивная память, как уже отмечалось, обычно состоит из двух частей: неизменяемых кодов, часто встречающихся при решении задач (так называемых констант и стандартных подпрограмм), и изменяемой части, которая может быть быстро набрана, содержимое ячеек которой для каждой отдельной задачи может подготавливаться заранее вне рабочего времени машины (обычно в другом помещении).

Поскольку машины работают с двоичными числовыми кодами, то *числовые* данные задач (но не команды программы и не вспомогательные коды!) перед вводом их в машину должны переводиться из десятичной системы в двоичную, а результаты расчетов перед их выводом должны переводиться из двоичной в десятичную систему. В некоторых машинах эти операции предусмотрены в схеме машины; в других производятся с помощью ручного перевода числовых данных или с помощью специальных подпрограмм перевода из двоичной системы в десятичную и наоборот. В последнем случае перед выводом каждого числа в программе предусматривается переход на подпрограмму перевода. При вводе и выводе чисел каждая десятичная цифра задается в двоичной системе (см. главу I, § 4).

Выдача результатов на ленту, например, в виде цифровых знаков предусматривается в определенной последовательности, обеспечивающей простоту их расшифровки.

В дальнейшем вопроса перевода из десятичной системы счисления в двоичную и наоборот, а также вопроса ввода программы в машину мы касаться не будем ввиду значительной зависимости этих операций от особенностей той или иной конкретной машины.

2. Программирование по формулам. Простейшими для программирования, естественно, являются задачи, в которых вычисления проводятся по формулам, распадающимся на элементарные операции, осуществляемые отдельными командами машины. Программирование таких задач (или таких частей задач) не вызывает особых затруднений и сводится к наиболее рациональному выбору последовательности элементарных операций и размещению относящихся к задаче данных в памяти машины, т. е. к размещению исходных данных, вспомогательных констант, промежуточных и окончательных результатов, а также к размещению команд программы в ячейках памяти машины.

Для удобства при составлении программ обычно пользуются буквенно-числовыми обозначениями — условными адресами. Так, например, при нумерации команд программы можно пользоваться следующими обозначениями: $N+1$, $N+2$, ...; при нумерации исходных данных и вспомогательных констант — следующими символами:

$$\alpha_1,\ \alpha_2,\ \ldots \quad \text{или} \quad \alpha+1,\ \alpha+2,\ \ldots$$
$$\gamma_1\ \gamma_2,\ \ldots \quad \text{или} \quad \gamma+1,\ \gamma+2,\ \ldots$$

В последнем случае подчеркивается размещение кодов в последовательно занумерованных ячейках, что в ряде случаев особенно существенно. Ячейки, в которых хранятся промежуточные результаты вычислений, — «рабочие ячейки» — будем обозначать через $\omega+1$, $\omega+2$, ...; $r+1$, $r+2$, ...; или ω_1, ω_2, ...; r_1, r_2, ..., если последовательное размещение этих величин в ячейках *ЗУ* несущественно.

Введение таких обозначений позволяет при программировании составлять программы до размещения соответствующего материала по конкретным ячейкам памяти, когда число ячеек, необходимое для размещения команд, промежуточных или окончательных результатов и пр., еще неизвестно, а также неизвестны номера свободных ячеек *ЗУ*. После окончательного составления программы в буквенном виде нужно каждой из букв N, α, γ, ω, r придать числовое значение и тем самым решить вопрос о размещении программы в конкретных ячейках.

При программировании простейших задач по заданным формулам, прежде всего, необходимо выбрать определенный поря-

док действия, т. е. разложить вычисления на последовательность элементарных операций. При этом следует, как и в случае обычных вычислений, стремиться к возможному уменьшению числа элементарных операций, пользуясь для этого тождественными преобразованиями формул. Уменьшение числа элементарных операций при решении задачи уменьшает число команд программы, а тем самым сокращает время вычислений и разгружает *ЗУ*.

Вместе с тем порядок действий необходимо выбирать с учетом того, что вычисления на машине производятся с округлениями, и поэтому нужно стремиться к уменьшению ошибок округления.

Заметим, что последовательность выполняемых операций может быть выбрана несколькими способами благодаря возможности перестановки элементарных операций; при этом количество рабочих ячеек может меняться.

Для реализации элементарных операций необходимо иметь в соответствующих ячейках *ЗУ* числа, входящие в эти операции. Для экономии ячеек *ЗУ* результаты промежуточных вычислений следует помещать в ячейки *ЗУ*, содержащие ранее использованные результаты, которые не понадобятся в дальнейшем.

Заметим, что, используя элементарные операции, выполняемые на машине, можно строить программы для вычисления величин, не задаваемых аналитическими выражениями.

Пример 1. Вычисление дробно-линейной функции

$$y = \frac{ax + b}{cx + d}. \tag{1}$$

Вычисление y будем проводить при помощи такой последовательности элементарных операций:

$$\begin{gathered} 1)\ A_1 = ax; \quad 2)\ A = A_1 + b; \quad 3)\ B_1 = cx; \\ 4)\ B = B_1 + d; \quad 5)\ y = \frac{A}{B}. \end{gathered} \tag{2}$$

Для этого поместим в запоминающем устройстве числа a, b, c, d, x в ячейки

$$\alpha_1,\ \alpha_2,\ \alpha_3,\ \alpha_4,\ \alpha_5$$

соответственно.

Каждая из элементарных операций (2) выполняется отдельной командой; при этом результаты промежуточных вычислений будем помещать в рабочие ячейки (в ячейки активной части запоминающего устройства) — ω_1 и ω_2.

Таким образом, вычисление по формулам (2) проводится такой последовательностью команд:

$N+1$	$\times$	α_1	α_5	ω_1	$A_1 = ax$
$N+2$	$+$	ω_1	α_2	ω_1	$A = A_1 + b$
$N+3$	$\times$	α_3	α_5	ω_2	$B_1 = cx$
$N+4$	$+$	ω_2	α_4	ω_2	$B = B_1 + d$
$N+5$	:	ω_1	ω_2	ω_1	$y = \frac{A}{B}$

Заметим, что порядок выполнения команд определяется выбранной последовательностью операций (2). Для окончания расчетов на машине необходимо предусмотреть автоматический останов, который осуществляется командой

ост.			

В нашем случае для запоминания A мы использовали ячейку, в которой содержался результат A_1, а для запоминания B — ячейку, в которой содержалось B_1 (для этих величин требуются разные ячейки, так как они участвуют в вычислении y). Искомый результат может быть помещен в любую из ячеек, в которых находились A и B. В соответствии со сказанным в приведенной программе используются две рабочие ячейки: ω_1 и ω_2.

Для автоматического ввода команд программы в оперативную память машины примем за начальную команду программы команду ввода

Вв	N	$N+6$	

Если результат вычислений — получаемое в ячейке ω_1 значение y — нужно выдать на устройстве вывода, перед командой останова следует поместить команду печати

П			ω_1

Таким образом, программа для вычисления y по формуле (1) может быть записана в таком виде:

Программа 1

Числа

Номер ячейки	Начальное заполнение	Окончательное заполнение
α_1	a	
α_2	b	
α_3	c	
α_4	d	
α_5	x	
ω_1		y
ω_2		

Команды

Номер команды	Код операции	ЦА	ПА	ПЦА	Результат операции
$N+1$	$\times$	α_1	α_5	ω_1	ax
$N+2$	$+$	ω_1	α_2	ω_1	$ax+b$
$N+3$	$\times$	α_3	α_5	ω_2	cx
$N+4$	$+$	ω_2	α_4	ω_2	$cx+d$
$N+5$	:	ω_1	ω_2	ω_1	$\frac{ax+b}{cx+d}=y$
$N+6$	$П$			ω_1	
$N+7$	*ост.*				

При этом подразумевается, что перед началом работы программы соответствующие числовые коды введены в ячейки $\alpha_1, \alpha_2, \ldots, \alpha_5$.

Для работы данной программы эти ячейки могут быть как ячейками постоянной, так и оперативной памяти. Положим $\alpha_1 = N + 8$, $\alpha_2 = N + 9$, ..., $\alpha_5 = N + 12$ и примем за начальную команду — команду ввода

N	*Вв*	12	$N+1$

Тогда наша программа запишется в таком виде:

Программа 1 — *а*

N	*B*	12	$N+1$		
$N+1$	×	$N+8$	$N+12$	ω_1	Коды команд
$N+2$	+	ω_1	$N+9$	ω_1	
$N+3$	×	$N+10$	$N+12$	ω_2	
$N+4$	+	ω_2	$N+11$	ω_2	
$N+5$	:	ω_1	ω_2	ω_1	
$N+6$	*П*			ω_1	
$N+7$	*ост.*				
$N+8$	*a*				Числовые коды
$N+9$	*b*				
$N+10$	*c*				
$N+11$	*d*				
$N+12$	*x*				

В дальнейшем мы будем пользоваться записью программ в первом из приведенных видов, опуская команду ввода. При такой записи начальной командой программы, если это не оговорено особо, считается первая команда таблицы команд программы.

Пример 2. Действия над комплексными числами. Каждое комплексное число $a + ib$ в *ЗУ* можно зафиксировать, используя две ячейки, например α и $\alpha + 1$, содержащие соответственно вещественную и мнимую части числа.

Для выполнения арифметических операций над комплексными числами необходимо вычислить вещественную и мнимую части результата. Рассмотрим для примера деление комплексных чисел

$$(a+bi):(c+di)=x+yi, \tag{3}$$

где

$$x=\frac{ac+bd}{c^2+d^2}; \qquad y=\frac{bc-ad}{c^2+d^2}. \tag{4}$$

Будем вычислять x и y, используя такую последовательность элементарных операций:

$$1)\ A=c^2;\quad 2)\ B=d^2;\quad 3)\ C=A+B;\quad 4)\ A_1=ac;$$
$$5)\ A_2=bd;\quad 6)\ A_3=A_1+A_2;\quad 7)\ x=\frac{A_3}{C};\quad 8)\ B_1=bc;$$
$$9)\ B_2=ad;\ 10)\ B_3=B_1-B_2;\quad 11)\ y=\frac{B_3}{C}. \tag{5}$$

Пусть комплексные числа $a+ib$ и $c+id$ содержатся соответственно в ячейках $ЗУ$

$$\alpha_1,\ \alpha_1+1 \quad \text{и} \quad \alpha_2,\ \alpha_2+1.$$

В данной программе используются четыре рабочие ячейки: ω_1, ω_1+1, ω_2, ω_3; в первых двух получаем искомый результат.

Программа 2

Числа			Команды				
α_1	a	$N+1$	$\times$	α_2	α_2	ω_1	c^2
α_1+1	b	$N+2$	$\times$	α_2+1	α_2+1	ω_1+1	d^2
α_2	c	$N+3$	$+$	ω_1	ω_1+1	ω_1+1	c^2+d^2
α_2+1	d	$N+4$	$\times$	α_1	α_2	ω_1	ac
ω_1		x $N+5$	$\times$	α_1+1	α_2+1	ω_2	bd
ω_1+1		y $N+6$	$+$	ω_1	ω_2	ω_1	$ac+bd$
ω_2		$N+7$	:	ω_1	ω_1+1	ω_1	x
ω_3		$N+8$	$\times$	α_1+1	α_2	ω_2	bc
		$N+9$	$\times$	α_1	α_2+1	ω_3	ad
		$N+10$	$-$	ω_2	ω_3	ω_2	$bc-ad$
		$N+11$	:	ω_2	ω_1+1	ω_1+1	y
		$N+12$	*ост.*				

Пример 3. Вычисление значений многочлена одного переменного

$$f(x) = a_0x^n + a_1x^{n-1} + \ldots + a_n. \qquad (6)$$

Рассмотрим для примера два способа вычисления значений многочлена. При этом убедимся, что в зависимости от выбора последовательности действий может потребоваться различное количество элементарных операций; в зависимости от этого получим различное число команд и оперативных ячеек.

1) Вычислим сначала все степени x^k $(k = 1, 2, \ldots, n)$, затем все попарные произведения $a_{n-k}x^k$ степеней x^k на соответствующие коэффициенты a_{n-k} и, наконец, сумму этих произведений при помощи такой последовательности элементарных операций:

$$1)\ A_1 = x^2;\quad 2)\ A_2 = x^3;\quad \ldots;\quad n-1)\ A_{n-1} = x^n;$$
$$n)\ A_n = a_0x^n;\quad n+1)\ A_{n+1} = a_1x^{n-1};\quad \ldots;\quad 2n-1)\ A_{2n-1} = a_{n-1}x;$$
$$2n)\ A_{2n} = a_0x^n + a_1x^{n-1};\quad \ldots;$$
$$3n-1)\ A_{3n-1} = a_0x^n + a_1x^{n-1} + \ldots + a_n = f(x).$$

При этом для вычисления понадобится $3n - 1$ команд.

2) Представим предварительно многочлен $f(x)$ в виде

$$f(x) = \{[(a_0x + a_1)x + \ldots + a_{n-2}]x + a_{n-1}\}x + a_n. \qquad (7)$$

Будем производить вычисления по этой формуле (схема Горнера). Последовательность элементарных операций в этом случае будет такая:

$$1)\ A_1 = a_0x;\quad 2)\ A_2 = A_1 + a_1;\quad 3)\ A_3 = A_2x; \qquad (8)$$
$$4)\ A_4 = A_3 + a_2;\quad \ldots;\quad 2n)\ A_{2n} = A_{2n-1} + a_n.$$

Заметим, что вычисления по формуле (7), помимо сокращения числа выполняемых элементарных операций ($2n$ вместо $3n - 1$), а значит, числа команд программы имеют и другое преимущество: вычисления по формуле (7) дают бóльшую точность.

В силу приведенных соображений ограничимся составлением программы для второго из приведенных способов вычисления $f(x)$.

Пусть величины

$$a_0,\ a_1,\ a_2,\ \ldots,\ a_n,\ x$$

хранятся в ячейках

$$\alpha_0,\ \alpha_1,\ \alpha_2,\ \ldots,\ \alpha_n,\ \alpha_{n+1}$$

соответственно. Согласно формулам (8) для хранения промежуточных результатов можно ограничиться одной рабочей ячейкой ЗУ ω, в которой и будет получен искомый результат. Действи-

тельно, для вычисления каждой из величин A_i используется только одно число A_{i-1}, которое в дальнейших вычислениях уже не встречается.

Программа 3

	Числа			Команды				
α_0	a_0		$N+1$	×	α_0	α_{n+1}	ω	a_0x
α_1	a_1		$N+2$	+	ω	α_1	ω	(a_0x+a_1)
α_2	a_2		$N+3$	×	ω	α_{n+1}	ω	$(\quad)x$
α_3	a_3		$N+4$	+	ω	α_2	ω	$(\quad)x+a_2$
.			.	.				
.			.	.				
.			.	.				
α_n	a_n		$N+2n-1$	×	ω	α_{n+1}	ω	$\{\quad\}x$
α_{n+1}	x		$N+2n$	+	ω	α_n	ω	$\{\quad\}x+a_n$
ω		$f(x)$	$N+2n+1$	ост.				

Пример 4. Выделение целой и дробной части числа

$$E(x),\ \{x\}.$$

Предположим, что в машине с плавающей запятой наибольший возможный порядок p_0 числа x не превосходит число цифр цифровой части числа. Пусть в этом случае число представляется в виде

$$x = 2^p \cdot 0,\ x_1x_2 \ldots x_n,$$

где n — число двоичных разрядов цифровой части числа.

Тогда

$$E(x) = \begin{cases} 0, & \text{если } p \leqslant 0, \\ x_1x_2 \ldots x_p, & \text{если } p > 0. \end{cases}$$

Таким образом, выделение целой части числа в машине с плавающей запятой сводится к выделению первых p разрядов цифровой части числа при $p > 0$. При сделанном предположении о возможном наибольшем порядке числа выделение $E(x)$ и $\{x\}$ можно осуществить по формулам

$$E(x) = (x + 2^{n-1}) - 2^{n-1};$$
$$\{x\} = x - E(x).$$

Действительно, при сложении и вычитании числа приводятся к одинаковому порядку, а затем результат операций нормализуется.

Имеем

$$x + 2^{n-1} = 2^p \cdot 0,\ x_1x_2 \ldots x_n + 2^{n-1} =$$
$$= 2^n \cdot 0,\underbrace{0 \ldots 0}_{n-p}x_1x_2 \ldots x_p + 2^n \cdot 0,1 = 2^n \cdot 0,10 \ldots x_1x_2 \ldots x_p.$$

Далее,

$$(x + 2^{n-1}) - 2^{n-1} = 2^n \cdot 0,10 \ldots 0x_1 \ldots x_p - 2^n \cdot 0,1 =$$
$$= 2^n \cdot 0,0 \ldots 0x_1 \ldots x_p = 2^p \cdot 0,x_1 \ldots x_p\underbrace{0 \ldots 0}_{n-p}.$$

В соответствии с предложенными формулами поместим в ячейки $ЗУ$ α_1 и α_2 величины x и 2^{n-1}.

Определяемые величины $E(x)$ и $\{x\}$ поместим в ячейки $ЗУ$ ω_1 и ω_2.

Программа 4. Вычисление $E(x)$.

	Числа			Команды			
α_1	x		$N+1$	$+$	α_1	α_2	ω_1
α_2	2^{n-1}		$N+2$	$-$	ω_1	α_2	ω_1
ω_1		$E(x)$	$N+3$	*ост.*			

Программа 5. Вычисление $\{x\}$.

	Числа			Команды			
α_1	x		$N+1$	$+$	α_1	α_2	ω_1
α_2	2^{n-1}		$N+2$	$-$	ω_1	α_2	ω_1
ω_1		$E(x)$	$N+3$	$-$	α_1	ω_1	ω_2
ω_2		$\{x\}$	$N+4$	*ост.*			

Если для вычислений требуется только дробная часть числа $\{x\}$, то последняя может быть образована в той же ячейке ω_1.

Выводы

1. Для обеспечения точности расчетные формулы для вычислений должны выбираться так, чтобы ошибки округления, возникающие в результате вычислений, были по возможности меньшими.

2. *Для сокращения* времени расчетов порядок действий при программировании по формулам следует выбрать так, чтобы количество элементарных операций, т. е. количество команд и используемых рабочих ячеек, было по возможности меньшим.

3. Для разгрузки памяти результаты промежуточных вычислений можно помещать в ячейки *ЗУ*, содержащие ранее использованные и не участвующие в дальнейшем счете результаты.

Упражнения

В предлагаемых упражнениях заданными в *ЗУ* считаются величины, входящие в правую часть приведенных формул.

1. Составить программу для вычисления

$$y = \frac{ax^2 + bx + c}{a_1x^2 + b_1x + c_1}$$

по данным a, b, c, a_1, b_1, c_1, x_n, ограничиваясь десятью командами (включая команду останова) и двумя оперативными ячейками.

2. Составить программу для вычисления

$$(x + iy) = (a + ib)(c + id).$$

3. Составить программу для вычисления

$$(x + iy) = (a + ib)^4,$$

ограничиваясь десятью командами (включая команду останова) и тремя оперативными ячейками.

4. Составить программу для выделения первых k разрядов числа в машине с фиксированной запятой.

5. Составить программу для выделения целой и дробной части числа в машине с запятой, фиксированной после k-го разряда.

§ 2. Разветвляющиеся процессы

Во многих случаях, в зависимости от значений исходных данных или промежуточных результатов, необходимо продолжить вычисления по той или иной вычислительной схеме. Вопрос о выяснении направления дальнейших вычислений может быть всегда сведен к проверке выполнения некоторого логического условия, выражающего определенное свойство входящих в расчеты величин.

Подобные вычислительные схемы естественно называть *разветвляющимися*, а каждое из направлений вычислений — *ветвью вычислений*.

Разветвляющимся вычислительным процессам соответствуют *разветвляющиеся* программы.

Программирование разветвляющихся процессов связано с размещением в памяти машины команд, осуществляющих вычисления по всем ветвям разветвления, и команд, осуществляющих передачу управления.

Заметим, что если вычислительный процесс разветвляется на три или более ветвей, то соответствующее условие в свою очередь может быть расчленено на ряд условий, каждое из которых отделяет одну ветвь от остальных. Поэтому вопрос программирования разветвляющихся процессов может быть сведен к построению разветвляющихся программ для случая двух ветвей.

Каждое заданное условие можно рассматривать как переменное высказывание, принимающее значение *истинно* при своем осуществлении и *ложно* в противном случае. Двум возможным значениям переменных высказываний можно сопоставить соответствующие значения логических переменных, принимающих два значения: 1, если условие истинно (осуществлено), или 0 в противном случае.

Для хранения значений соответствующих логических переменных в машинах вводится специальный одноразрядный регистр (машины *Стрела, Урал*) или используется один, обычно знаковый, разряд ячеек памяти (машина *Киев*). В последнем случае содержание остальных разрядов ячейки, используемой для размещения логической переменной, безразлично. Другими словами, если ячейка α предназначена для хранения значения логической переменной, то это последнее подбирается так, чтобы значению условия *истинно* соответствовал положительный знак содержащегося в ячейке α числа и отрицательный в противном случае.

В этом смысле *Стрелу* или *Урал*, у которых после выполнения каждой арифметической или логической операции в зависимости от результатов вырабатывается *признак* — значение логической переменной, который хранится до выполнения следующей команды, можно рассматривать как машины, имеющие один-единственный регистр для хранения значений логических переменных. Поэтому для программной реализации узла разветвления для таких машин требуется по меньшей мере две команды, непосредственно следующие одна за другой, первая из которых окончательно формирует значение логической переменной (вырабатывает соответствующий признак по принятой терминологии), а вторая представляет собой команду условного перехода в зависимости от значения логической переменной.

В зависимости от наличия в наборе элементарных операций тех или иных операций условной передачи управления условия, по выполнении которых выясняется ход дальнейших вычислений по разветвляющимся программам, должны быть преобразованы к соответствующему виду.

Приведем логические условия, проверку реализации которых осуществляют операции условной передачи управления, рассмотренные в § 2 главы II.

1. Операция условной передачи управления в зависимости от сравнения двух чисел с учетом знаков ($\leqslant$)

$\leqslant$	α	β	k

реализует проверку следующего логического условия:

$$P_1 = \begin{cases} 1, & \text{если } (\alpha) \leqslant (\beta), \\ 0, & \text{если } (\alpha) > (\beta), \end{cases}$$

и передачу управления команде k, указанной в ША, если логическое условие выполнено (P_1 равно 1), и следующей по порядку команде в противном случае. Здесь (α) означает содержимое ячейки α.

2. Операция условной передачи управления в зависимости от сравнения модулей двух чисел ($|\leqslant|$)

$\lvert\leqslant\rvert$	α	β	k

осуществляет проверку логического условия

$$P_2 = \begin{cases} 1, & \text{если } |(\alpha)| \leqslant |(\beta)|, \\ 0, & \text{если } |(\alpha)| > |(\beta)|, \end{cases}$$

и передачу управления команде k, указанной в ША, в первом случае и следующей за операцией *УП* команде в противном случае.

3. Операция условной передачи управления в зависимости от равенства двух чисел ($=$)

$=$	α	β	k

осуществляет проверку логического условия

$$P_3 = \begin{cases} 1, & \text{если } (\alpha) = (\beta), \\ 0, & \text{если } (\alpha) \neq (\beta), \end{cases}$$

и передачу управления команде k, если $P_3 = 1$, и следующей команде в противном случае.

4. Операция условной передачи управления в зависимости от неравенства двух чисел ($\neq$)

$\neq$	α	β	k

осуществляет проверку логического условия

$$P_4 = \begin{cases} 1, & \text{если } (\alpha) \neq (\beta), \\ 0, & \text{если } (\alpha) = (\beta), \end{cases}$$

и передачу управления команде k, если $P_4 = 1$, и следующей команде в противном случае.

5. Операция условной передачи управления по признаку (*УПП*)

УПП	k_1	k_2	

осуществляет проверку логического условия

$$P_5 = \begin{cases} 1, & \text{если } \omega = 1, \\ 0, & \text{если } \omega = 0, \end{cases}$$

и передает управление команде k_1, указанной в IA, если в предыдущей команде выработан признак ω, и команде k_2, указанной во IIA, в противном случае.

6. Операция условной передачи управления по знаку числа (*УПЧ*)

УПЧ	α	k_1	k_2

осуществляет проверку логического условия

$$P_6 = \begin{cases} 1, & \text{если } (\alpha) \leqslant -0, \\ 0, & \text{если } (\alpha) > -0, \end{cases}$$

и передает управление команде k_1, если $P_5 = 1$, и команде k_2, если $P_6 = 0$.

Пример 1. Образование модуля числа. В некоторых машинах операция взятия модуля числа в наборе элементарных операций отсутствует *).

Построим программу для получения модуля числа x, находящегося в ячейке α, с помещением искомого результата в ячейку ω. Кроме того, потребуем, чтобы, независимо от знака x, после получения $|x|$ управление было передано одной и той же команде. Последнее требование необходимо, если $|x|$ является некоторым промежуточным результатом.

В зависимости от знака числа x в ячейку ω должно быть передано:

а) содержимое ячейки α, если данное число x положительное;

*) Предполагается также, что разряд знака числа не участвует в операции логического умножения.

б) содержимое ячейки α с обратным знаком, если оно отрицательное.

Построим программу для машины с операцией условной передачи управления по сравнению $\leqslant$.

В первом случае (а) программа образования модуля числа x состоит из команд

k	$+$		α	β
$k+1$	$\leqslant$			p

и во втором

n	$-$		α	β
$n+1$	$\leqslant$			p

Операции безусловных передач $(k+1)$-я и $(n+1)$-я пишутся для того, чтобы после операции образования модуля числа перейти к выполнению команды p.

Для выяснения направления разветвления может быть использовано логическое условие

$$P=\begin{cases} 0, & \text{если } 0>(\alpha), \\ 1, & \text{если } 0\leqslant(\alpha). \end{cases}$$

Проверку выполнения этого условия и соответствующие передачи управления выполняет команда

$\leqslant$		α	k

которой следует сообщить номер $n-1$. Получаем программу

$n-1$	$\leqslant$		α	k
n	$-$		α	ω
$n+1$	$\leqslant$			p
k	$+$		α	ω
$k+1$	$\leqslant$			p
p				

Если положить $p = k + 1$, то последняя команда может быть отброшена. Кроме того, можно положить $k = n + 2$.

Программа 6

Числа

α	x
ω	$\lvert x\rvert$

Команды

$n-1$	$\leqslant$		α	$n+2$
n	$-$		α	ω
$n+1$	$\leqslant$			$n+3$
$n+2$	$+$		α	ω
$n+3$				

Пример 2. Решение квадратного уравнения

$$ax^2 + bx + c = 0.$$

Для определения корней уравнения воспользуемся формулой

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

В зависимости от знака разности $D = b^2 - 4ac$ корни уравнения будут либо вещественными, либо мнимыми. В первом случае результатом вычислений будем считать выдачу на печать корней уравнения, а во втором — выдачу на печать вещественной и мнимой частей комплексных корней. Наличие вещественных корней условимся отмечать тем, что перед выдачей корней уравнения будем печатать условный знак «+0».

В ячейках α_1, α_2, α_3, β_1 расположим соответственно величины a, b, c, 4. Независимо от вида корней для вычислений понадобятся величины: $-b$, $D = b^2 - 4ac$, $2a$, которые будем помещать соответственно в ячейки ω_1, ω_2, ω_3.

Общая часть вычислений, предшествующих разветвлению, выполняется командами

n	$\times$	α_2	α_2	ω_1	b^2
$n+1$	$\times$	α_1	α_3	ω_2	ac
$n+2$	$\times$	ω_2	β_1	ω_2	$4ac$
$n+3$	$-$	ω_1	ω_2	ω_2	D
$n+4$	$-$		α_2	ω_1	$-b$
$n+5$	$+$	α_1	α_1	ω_3	$2a$

При $D > 0$ следует выполнить команды

k	П				Печать «+ 0»
$k+1$	$\sqrt{}$	ω_2		ω_2	$\sqrt{D}$
$k+2$	+	ω_1	ω_2	ω_4	$-b+\sqrt{D}$
$k+3$	—	ω_1	ω_2	ω_2	$-b-\sqrt{D}$
$k+4$	:	ω_4	ω_3	ω_1	x_1
$k+5$	:	ω_2	ω_3	ω_2	x_2
$k+6$	П			ω_1	
$k+7$	П			ω_2	
$k+8$	*ост.*				

В случае $D < 0$ выполняются следующие команды:

m	—		ω_2	ω_2	$-D$
$m+1$	$\sqrt{}$	ω_2		ω_2	$\sqrt{-D}$
$m+2$	:	ω_1	ω_3	ω_1	$-b/2a$
$m+3$	:	ω_2	ω_3	ω_2	$\sqrt{D}/2a$
$m+4$	П			ω_1	
$m+5$	П			ω_2	
$m+6$	*ост.*				

Здесь m-я команда — изменение знака величины D (в данном случае отрицательной) для возможности извлечения из нее корня в следующей команде.

Проверка условия

$$P = \begin{cases} 0, & \text{если } D < 0, \\ 1, & \text{если } D \geqslant 0, \end{cases}$$

и передача управления соответствующим ветвям вычислений может быть выполнена командой

$k-1$	$\leqslant$	ω_2		m

Поскольку последние четыре команды каждой из ветвей совпадают, они могут быть внесены в общую часть программы, реализуемую после вычислений по той или другой ветви разветвления. С этой целью после $k+4$-й команды следует поместить команду безусловного перехода к команде $m+3$

$k+5$	$\leqslant$			$m+3$

Положим $k-1=n+6$; $m=k+6=n+13$ и получим программу.

Программа 7

Числа

α_1	a	
α_2	b	
α_3	c	
β	4	
ω_1		результат
ω_2		
ω_3		рабочие
ω_4		

Команды

n	$\times$	α_2	α_2	ω_1	b^2
$n+1$	$\times$	α_1	α_3	ω_2	ac
$n+2$	$\times$	ω_2	β	ω_2	$4ac$
$n+3$	$-$	ω_1	ω_2	ω_2	D
$n+4$	$-$		α_2	ω_1	$-b$
$n+5$	$+$	α_1	α_1	ω_3	$2a$
$n+6$	$\leqslant$	ω_2		$n+13$	
$n+7$	П				
$n+8$	$\sqrt{}$	ω_2		ω_2	$\sqrt{D}$
$n+9$	$+$	ω_1	ω_2	ω_4	$-b+\sqrt{D}$
$n+10$	$-$	ω_1	ω_2	ω_2	$-b-\sqrt{D}$
$n+11$	:	ω_4	ω_3	ω_1	x_1
$n+12$	$\leqslant$			$n+16$	
$n+13$	$-$		ω_2	ω_2	$-D$

$n+14$	$\sqrt{}$	ω_2		ω_2	$\sqrt{-D}$
$n+15$	:	ω_1	ω_3	ω_1	$-b/2a$
$n+16$	:	ω_2	ω_3	ω_2	$x_2\left(\frac{\sqrt{-D}}{2a}\right)$
$n+17$	П			ω_1	
$n+18$	П			ω_2	
$n+19$	*ост.*				

В случае наличия операции условного перехода по числу *УПЧ* разветвление программы может быть осуществлено командой

$n+6$	*УПЧ*	ω_2	$n+7$	$n+13$

где в ячейке *ЗУ* ω_2 содержится величина D. Поменяв в этой команде II и III адреса местами, мы можем расположить непосредственно после операции *УПЧ* вторую из ветвей разветвления. Общее число команд в программе сохраняется прежним.

В случае наличия операции условной передачи управления по признаку для выработки последнего понадобится дополнительная команда, например

(*)	+		ω_2	ω_2

если признак вырабатывается в результате появления знака минус при выполнении операции сложения. Последнюю необходимо поместить непосредственно перед операцией *УПП*. Однако, заметив, что величина D появляется в результате выполнения команды $n+3$ и что команды $n+3$ и $n+5$ могут быть переставлены местами, можно избежать пополнения программы дополнительной командой (*). При этом подразумевается, что признак вырабатывается при появлении знака минус в результате выполнения операции вычитания.

Пример 3. Вычисление значений функции

$$z=\begin{cases} xy & \text{при } x^2+y^2\leqslant 1 \quad \text{(I ветвь);} \\ \sqrt{x}+\dfrac{x+y}{x-y} & \text{при } x^2+y^2>1 \text{ и } x\leqslant 0 \quad \text{(II ветвь);} \\ \sqrt{2x}+\dfrac{2x+y}{2x-y} & \text{при } x^2+y^2>1 \text{ и } x>0 \quad \text{(III ветвь).} \end{cases} \tag{9}$$

В связи со строением нашей функции вычислительный процесс разветвляется на три ветви.

Разместим величины x, y, 1 в ячейках α, β, γ.

Вычисления по первой ветви выполняются командами

n	$\times$	α	β	ω	z
$n+1$	*ост.*				

Вычисления по второй ветви могут быть выполнены командами

k	$+$	α	β	ω	$x+y$
$k+1$	$-$	α	β	ω_1	$x-y$
$k+2$	:	ω	ω_1	ω	$\frac{x+y}{x-y}$
$k+3$	$\sqrt{}$	α		ω_1	$\sqrt{x}$
$k+4$	$+$	ω	ω_1	ω	z
$k+5$	*ост.*				

Вычисления по третьей ветви могут быть выполнены при помощи команд k, $k+1$, ..., $k+5$, если предварительно в ячейку α вместо x будет помещена величина $2x$:

p	$+$	α	α	α	$2x$
$p+1$	$+$	α	β	ω	$2x+y$
$p+2$	$-$	α	β	ω_1	$2x-y$
$p+3$	:	ω	ω_1	ω	$\frac{2x+y}{2x-y}$
$p+4$	$\sqrt{}$	α		ω_1	$\sqrt{2x}$
$p+5$	$+$	ω	ω_1	ω	z
$p+6$	*ост.*				

Переход к вычислению I ветви соответствует выполнению следующего условия:

$$P_1 = \begin{cases} 1 \text{ при } R \leqslant 1, \\ 0 \text{ при } R > 1, \end{cases}$$

где $R = x^2 + y^2$.

Вычислим предварительно величину R:

N	$\times$	α	α	ω	x^2
$N+1$	$\times$	β	β	ω_1	y^2
$N+2$	$+$	ω	ω_1	ω	R

Тогда проверку выполнения условия P_1 и соответствующую передачу управления можно выполнить командой

$N+3$	$\leqslant$	ω	γ	n

Переход к вычислению II ветви соответствует выполнению условия

$$P_2 = \begin{cases} 1 \text{ при } x \leqslant 0, \\ 0 \text{ при } x > 0, \end{cases}$$

если $R > 1$ и $x \leqslant 0$, т. е. при выполнении условия $\overline{P}_1 \wedge P_2$, где $\wedge$ — знак логического умножения (P_1 — отрицание P_1). Проверку выполнения этого условия и соответствующие передачи управления осуществляет расположенная непосредственно после команды $N + 3$ команда

$N+4$	$\leqslant$	α		k

после которой следует поместить программу вычислений по III ветви.

Таким образом, если положить

$$N + 5 = p,$$

то задача может быть решена совокупностью команд

$$N,\ N+1,\ \ldots,\ N+4,\ N+5=p,\ N+6,\ \ldots,\ N+11,$$
$$n,\ n+1,\ k,\ k+1,\ \ldots,\ k+5 \qquad \text{(A)}$$

или

$$N,\ N+1,\ \ldots,\ N+4,\ N+5,\ \ldots,\ N+11,\ k,$$
$$k+1,\ \ldots,\ k+5,\ n,\ n+1. \qquad \text{(B)}$$

Программа может быть сокращена за счет совпадающих по содержанию команд, встречающихся во II и III ветвях. Поместим в (А) вместо группы команд $N+6, \ldots, N+11$ команду безусловного перехода

$N+6$	$\leqslant$			k

или вместо группы команд $k, k+1, \ldots, k+5$ команду безусловного перехода

k	$\leqslant$			$N+6$

В этом случае результат будет тот же самый, если опустить команду k, а в команде $N+4$ в III адресе вместо k поставить $N+6$. Тогда наша программа будет иметь следующий вид:

Программа 8

Числа

α	x
β	y
γ	1
ω	
ω_1	

z

Команды

N	$\times$	α	α	ω
$N+1$	$\times$	β	β	ω_1
$N+2$	$+$	ω	ω_1	ω
$N+3$	$\leqslant$	ω	γ	$N+12$
$N+4$	$\leqslant$	α		$N+6$
$N+5$	$+$	α	α	α
$N+6$	$+$	α	β	ω
$N+7$	$-$	α	β	ω_1
$N+8$	:	ω	ω_1	ω
$N+9$	$\sqrt{}$	α		ω_1
$N+10$	$+$	ω	ω_1	ω
$N+11$	*ост.*			
$N+12$	$\times$	α	β	ω
$N+13$	*ост.*			

Если результат вычислений z по формулам (9) является промежуточным, то независимо от разветвления управление в конце программы должно быть передано одной и той же команде, например $N + 13$. В этом случае вместо $(N + 11)$-й команды останова следует поместить команду безусловного перехода к команде $N + 13$, начиная с которой размещается следующая часть программы.

Выводы

При построении разветвляющихся программ необходимо знать следующее.

1. Размещение в оперативной памяти машины команд, осуществляющих вычисления по всем ветвям разветвления, упрощается, если использовать буквенно-числовые обозначения при нумерации ячеек *ЗУ*.

2. С целью экономии ячеек *ЗУ* при построении программ для отдельных ветвей следует стремиться к такому распределению материала в *ЗУ*, которое позволяет выделить максимально возможное число команд в общие для ветвей части программы.

3. Условия разветвления следует преобразовать так, чтобы их проверка легко осуществлялась с помощью имеющихся в наличии команд условной передачи управления.

4. В случае процессов, разветвляющихся более чем на две ветви, следует подобрать такую последовательность условий, при которой произойдет постепенное отделение одной из ветвей от всех остальных.

Упражнения

1. Составить программу для вычисления интеграла

$$\int_a^b x^\alpha\,dx = \begin{cases} \dfrac{1}{\alpha+1}\,x^{\alpha+1}\Big|_a^b, & \text{если } \alpha \neq -1, \\ \ln x\Big|_a^b, & \text{если } \alpha = -1. \end{cases}$$

2. Составить программу для вычисления значений функции

$$y = \begin{cases} 2x^2 + \sqrt{x} & \text{при } x > 0, \\ 3x^3 + \sqrt{-x} & \text{при } x \leqslant 0. \end{cases}$$

3. Изменить программу упр. 2, организовав печать всех результатов одной командой.

§ 3. Циклические процессы

Как указывалось выше, одним из условий эффективного использования машины с программным управлением является возможность выполнения большого числа операций с помощью небольшого числа команд. Осуществление этого принципа основано на том, что всякий алгоритм, как правило, может быть

представлен в виде ряда повторяющихся серий операций. Такие вычислительные процессы называют *циклическими*, а повторяющиеся в них участки — *циклами*.

Общее число циклов, необходимых для расчетов, может либо указываться заранее, либо определяться в процессе вычислений. В первом случае, когда число циклов заранее известно, может быть составлена развернутая программа, в которой каждому циклу будет соответствовать свой участок. Однако такое решение задачи на *ЦАМ* потребовало бы большого времени для составления команд программы и привело бы к чрезвычайному загромождению памяти машины.

Для циклических процессов с заранее неизвестным числом циклов развернутая программа вообще не может быть составлена.

При надлежащем распределении исходных данных и результатов промежуточных вычислений в *ЗУ* для циклических процессов могут быть составлены так называемые циклические программы, состоящие из команд, необходимых для выполнения одного из циклов, и команд, имеющих вспомогательное значение.

Группа команд, которая при выполнении вычислений повторяется последовательно некоторое число раз, называется *циклической программой* или *циклом*.

Для построения циклической программы в каждом отдельном случае подбирается подходящим образом логическое условие, при выполнении которого может быть определено окончание циклического процесса, и программа дополняется командами, осуществляющими его проверку и передачу управления началу нового цикла или последующим командам в зависимости от исхода этой проверки. Кроме того, в программу цикла должны быть включены команды, подготавливающие для перехода от цикла к циклу надлежащее заполнение *ЗУ*.

При выборе логических условий обычно пользуются изменяемой от цикла к циклу величиной, встречающейся в процессе вычислений, либо специально введенной переменной. При этом следует также учитывать наличие в наборе элементарных операций, выполнимых машиной, тех или иных команд условной передачи управления.

1. Итерационные циклы. В простейшем случае циклический процесс сводится к применению к исходным данным

$$x_1^0,\ x_2^0,\ \ldots,\ x_n^0$$

некоторых формул

$$x_1^1 = f_1(x_1^0,\ \ldots,\ x_n^0);\ \ldots;\quad x_n^1 = f_n(x_1^0,\ \ldots,\ x_n^0), \tag{10}$$

результаты вычислений по которым $x_1^1,\ x_2^1,\ \ldots,\ x_n^1$ в свою очередь

принимаются за исходные данные для вычислений по тем же формулам

$$x_1^2 = f_1(x_1^1, \ldots, x_n^1); \quad \ldots; \quad x_n^2 = f_n(x_1^1, \ldots, x_n^1)$$

и т. д. Процесс повторяется некоторое число раз, известное или определяемое в процессе вычислений. Получение $x_1^{k+1}, \ldots, x_n^{k+1}$ по $x_1^k, \ldots, x_n^k$ представляет собой отдельный цикл вычислений. Такие циклические процессы будем называть *итерационными*.

Для построения программы по данной вычислительной схеме выделим для исходных данных $x_1^0, \ldots, x_n^0$ специальные ячейки, и после вычислений по формулам (10) будем помещать в них величины $x_1^1, \ldots, x_n^1$. Тогда группа команд, осуществляющих вычисления $x_1^1, \ldots, x_n^1$ по $x_1^0, \ldots, x_n^0$, может быть использована для вычислений $x_1^2, \ldots, x_n^2$ по $x_1^1, \ldots, x_n^1$; $x_1^3, \ldots, x_3^n$ по $x_1^2, \ldots, x_n^2$ и т. д.

Построение циклической программы заканчивается введением в программу команд, обеспечивающих передачу управления начальной команде и дальнейшим вычислениям (необходимое число раз) или передачу управления на команду останова.

Экономия в командах получается, если удается в процессе вычислений величины $x_1^{k+1}, \ldots, x_n^{k+1}$ помещать в ячейки, в которых содержались величины $x_1^k, \ldots, x_n^k$. Рассмотрим пример.

Пример 1. Вычисление и печатание таблицы квадратов членов арифметической прогрессии. Для составления таблицы квадратов членов арифметической прогрессии

$$a, \ a+c, \ a+2c, \ \ldots$$

от a до $a + cN$, где a — первый член прогрессии, c — разность прогрессии, примем следующий порядок вычислений. Предполагая заданным первый член прогрессии a, возводим его в квадрат и печатаем результат; подготавливаем следующий член прогрессии, для чего к числу a прибавляем разность прогрессии c; полученное число $a + c$ возводим в квадрат и печатаем; прибавляем к числу $a + c$ снова разность прогрессии и т. д. Тем самым вычисление квадратов членов арифметической прогрессии распадается на циклы: образование квадрата данного члена арифметической прогрессии, печать этого квадрата, образование следующего члена прогрессии.

Для программирования первого цикла необходимо иметь в *ЗУ* числа a и c, которые мы поместим соответственно в ячейки *ЗУ* α_1 и α_2. Вычисляемый следующий член арифметической прогрессии

$a + c$ помещается в ячейку α_1 (вытесняя «предыдущий» член прогрессии). Квадрат члена поместим в оперативную ячейку ω.

Первый цикл осуществляется последовательностью команд

					I цикл	II цикл
k	×	α_1	α_1	ω	a^2	$(a+c)^2 \ldots$
$k+1$	П			ω		
$k+2$	+	α_1	α_2	α_1	$a+c$	$a+2c \ldots$

Остается отметить, что содержимое ячеек *ЗУ* для следующего цикла подготовлено первым циклом, а значит, те же самые команды могут осуществить выполнение второго и последующих циклов.

В данном примере общее число циклов заранее известно и равно N.

Для завершения построения циклической программы необходимо выбрать логическое условие, по которому может быть определено окончание циклического процесса.

а) Пусть в машине имеется операция передачи управления по сравнению «$\leqslant$» или по сравнению модулей «$|\leqslant|$».

Из условия ясно, что переход от данного цикла к другому должен осуществляться до тех пор, пока число $a + rc$ (r — номер цикла), получаемое в ячейке α_1, не превзойдет числа $a + Nc$. Поместим последнее в ячейку *ЗУ* α_3.

Положим

$$P = \begin{cases} 0 \text{ при } a + rc > a + Nc, \\ 1 \text{ при } a + rc \leqslant a + Nc. \end{cases}$$

Проверку выполнения условия P и передачу управления можно осуществить командами

$\leqslant$	α_1	α_3	k

или

$\|\leqslant\|$	α_1	α_3	k

,

которые передают управление следующему циклу до тех пор, пока в ячейке α_1 не появится число $a + (N+1)c$; после этого расчеты нужно закончить, т. е. нужно поместить команду останова.

Программа 9

Числа

α_1	a	
α_2	c	
ω		$(a+rc)^2$
α_3	$a+Nc$	

Команды

k	$\times$	α_1	α_1	ω
$k+1$	Π			ω
$k+2$	$+$	α_1	α_2	α_1
$k+3$	$\leqslant$	α_1	α_3	k
$k+4$	*ост.*			

Рассмотрим подробнее работу программы. Работа начинается с k-й команды:

Начало первого цикла

1) вычисление квадрата a^2 и засылка в ячейку ω,
2) печать a^2,
3) вычисление следующего члена арифметической прогрессии $a+c$ и засылка в ячейку α_1,
4) сравнение числа $a+c$ и числа $a+cN$, и если $a+c \leqslant a+cN$, управление передается команде k.

Начало второго цикла

5) вычисление квадрата $(a+c)^2$ и засылка в ячейку ω,
6) печать $(a+c)^2$,
7) вычисление следующего члена арифметической прогрессии $a+2c$ и засылка в ячейку α_1,
8) сравнение числа $a+2c$ и числа $a+cN$, и если $a+2c \leqslant a+cN$, управление передается команде k.

И так до тех пор, пока в ячейке *ЗУ* α_1 не появится число $a+c(N+1)$.

Начало N-го цикла

$4N-3$) вычисление $(a+cN)^2$ и засылка в ячейку ω,
$4N-2$) печать $(a+cN)^2$,
$4N-1$) вычисление $a+c(N+1)$ и засылка в ячейку α_1,
$4N$) сравнение $a+c(N+1)$ с $a+cN$, и так как $a+c(N+1)>a+cN$, переход к выполнению команды $k+4$, т. е.
$4N+1$) останов.

Таким образом, с помощью пяти команд будет выполнено $4N+1$ элементарных операций. Притом для вычислений,

независимо от величины числа N, используются только четыре ячейки $ЗУ$.

Выяснение необходимости вычислений следующего цикла или прекращения расчетов можно производить в начале каждого цикла. В этом случае, когда в α_1 появится число $a + c(N + 1)$, передача управления должна производиться на останов.

В соответствии с этим в ячейке $ЗУ$ α_3 помещается число $a + c(N + 1)$ и передача управления на останов осуществляется командой

$k-1$	$\leqslant$	α_3	α_1	$k+4$

которая помещается в начале цикла.

Для перехода к следующему циклу цикл пополняется так называемой «командой обязательного перехода»

$k+3$	$\leqslant$			$k-1$

передающей управление $(k-1)$-й команде. До тех пор, пока в ячейке α_1 находится число, меньшее числа $a + c(N + 1)$ (ячейка α_3), циклы выполняются последовательно один за другим; после появления в ячейке α_1 числа $a + c(N + 1)$ первая команда передает управление команде

ост.			

Наша программа тогда будет записана в следующем виде:

Числа

α_1	a	
α_2	c	
α_3	$a+(N+1)c$	
ω		$(a+rc)^2$

Команды

$k-1$	$\leqslant$	α_3	α_1	$k+4$
k	$\times$	α_1	α_1	ω
$k+1$	Π			ω
$k+2$	$+$	α_1	α_2	α_1
$k+3$	$\leqslant$			$k-1$
$k+4$	*ост.*			

В последнем случае программа имеет одной командой больше. Однако такое размещение команд в отдельных случаях может оказаться необходимым. Так, проверка выполнения логического условия должна производиться в начале цикла, когда при не-

которых значениях параметров циклический участок вычислений вообще может опускаться — число циклов при этих значениях параметров оказывается равным нулю.

б) Машина имеет операцию передачи управления по неравенству $\neq$.

Циклический процесс должен продолжаться до тех пор, пока в ячейке α_1 не появится число, равное числу $a + (N + 1)c$, которое поместим в ячейку α_3.

Положим

$$P = \begin{cases} 0 \text{ при } a + rc = a + (N+1)c, \\ 1 \text{ при } a + rc \neq a + (N+1)c. \end{cases}$$

Проверку выполнения условия P и передачу управления в этом случае можно осуществить командой

$k+3$	$\neq$	α_1	α_3	k

в) Случай передачи управления по равенству $=$.

Воспользуемся ячейкой α_3, в которой, как и в предыдущем случае, поместим число $a + (N + 1)c$. Положим

$$P = \begin{cases} 0 \text{ при } a + rc \neq a + (N+1)c, \\ 1 \text{ при } a + rc = a + (N+1)c. \end{cases}$$

Теперь для выполнения циклического процесса программу из команд k, $k + 1$, $k + 2$ дополним командами

$k+3$	$=$	α_1	α_3	$k+5$
$k+4$	$=$			k
$k+5$	*ост.*			

где команда $k + 4$ является операцией безусловной передачи управления.

г) Случай передачи управления по числу — *УПЧ*.

Циклический процесс должен продолжаться до тех пор, пока

$$s = (a + rc) - (a + Nc) \leqslant -0\ ^*).$$

Положим

$$P = \begin{cases} 0 \text{ при } s > -0, \\ 1 \text{ при } s \leqslant -0. \end{cases}$$

---

*) Предположим, что при вычитании равных кодов в результате получим код «—0».

Введем в программу из команд k, $k+1$, $k+2$ команду для вычисления величины s, которую поместим в ячейку ω:

$k+3$	—	α_1	α_3	ω

Тогда для передачи управления началу цикла может быть использована команда

$k+4$	*УПЧ*	ω	$k+5$	k

д) Случай передачи управления по команде — *УПП*.

Если в результате операции вычитания признак вырабатывается при наличии отрицательного знака разности, то выполнение циклической программы с командой *УПП* ничем не будет отличаться от предыдущего случая. Здесь только важно, чтобы команда, в результате выполнения которой появляется необходимый признак, непосредственно предшествовала команде передачи управления.

2. Итерационные вычислительные процессы. Показательным примером алгоритмов, распадающихся на циклы, является решение уравнения

$$x = f(x) \tag{11}$$

методом итераций.

Исходя из начального значения x_0, последовательно вычисляются

$$x_1 = f(x_0), \quad x_2 = f(x_1), \quad \ldots, \quad x_n = f(x_{n-1}) \tag{12}$$

и т. д. Процесс вычислений продолжается до тех пор, пока два соседних значения x_n и x_{n-1} не совпадут с точностью до ε, т. е. пока $|x_n - x_{n-1}|$ не станет меньше ε, где ε заранее задано. В данном случае общее число циклов, необходимых для решения уравнения (11), заранее неизвестно. Оно зависит от величины ε и условий сходимости процесса. Рассмотрим пример.

Пример 2. Извлечение корня

$$y = \sqrt{x}. \tag{13}$$

Нахождение $y = \sqrt{x}$ сводится к решению уравнения

$$f(y) = y^2 - x = 0. \tag{14}$$

Для решения этого уравнения применим метод касательных:

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)};$$

Вообще, при программировании итерационных процессов более целесообразно вести вычисления по формулам

$$\left.\begin{aligned} \delta_{n+1} &= f(x_n) - x_n; \\ x_{n+1} &= x_n + \delta_{n+1}; \end{aligned}\right\} \qquad (16')$$

y_0 может быть выбрано, вообще говоря, произвольным, но это в то же время зависит от конструктивных особенностей машины. Например, для машины с фиксированной запятой, оперирующей с числами, по модулю меньшими единицы, можно положить y_0 равным 0,75. При этом легко показать, что все итерации, получаемые по формулам (16), для любого $|x| < 1$ не превосходят по модулю единицы. Для машины с плавающей запятой для ускорения сходимости y_0 можно выбрать в зависимости от величины x.

3. Вычисление элементарных функций.

Пример 3. Вычисление показательной функции e^x:

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots \qquad (17)$$

Для цикличности вычислений суммы ряда (17) воспользуемся рекуррентными соотношениями между членами ряда u_n и частными суммами s_n:

$$\left.\begin{aligned} u_{n+1} &= u_n \frac{x}{n+1}; \\ s_{n+1} &= s_n + u_{n+1}. \end{aligned}\right\} \qquad (18)$$

Отдельный цикл вычислений состоит в вычислении по данным

$$n,\ u_n,\ s_n,$$

которые мы поместим в ячейки $ЗУ$

$$\alpha_1,\ \alpha_2,\ \alpha_3,$$

величин $n+1$, u_{n+1}, s_{n+1}, которые помещаются в тех же ячейках $ЗУ$ α_1, α_2, α_3. Для выполнения вычислений по формулам (18) числа x и 1 поместим в ячейках $ЗУ$ α_4 и α_5. Вычисления прекращаются, лишь только вычисленный член ряда u_{n+1} окажется меньше заданного числа ε, которое помещается в ячейке $ЗУ$ α_6.

Для составления программы остается заметить, что исходные значения переменных величин n, u_n, s_n должны быть следующие:

$$n = 0; \quad u_0 = 1; \quad s_0 = 1.$$

Программа 11

Числа			Команды				
α_1	0	n	$k+1$	+	α_1	α_5	α_1	$n+1$
α_2	1	u_n	$k+2$	×	α_2	α_4	α_2	$u_n x$
α_3	1	s_n	$k+3$	:	α_2	α_1	α_2	u_{n+1}
α_4	x		$k+4$	+	α_3	α_2	α_3	s_{n+1}
α_5	1		$k+5$	$\|\leqslant\|$	α_6	α_2	$k+1$	
α_6	ε		$k+6$	*ост.*				

Пример 4. Вычисление тригонометрических функций. Рассмотрим для примера вычисление функции $\sin x$ при помощи степенного ряда

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

Как и в предыдущем случае, воспользуемся соотношениями между членами ряда и частными суммами:

$$\left.\begin{aligned} u_{n+1} &= -x^2 \frac{u_n}{a_{n+1}}, \quad \text{где} \quad a_n = 2n(2n+1), \\ s_{n+1} &= s_n + u_{n+1} \quad (n = 1, 2, \ldots). \end{aligned}\right\} \tag{19}$$

Для цикличности вычислений по формулам (19) остается составить рекуррентные соотношения для a_n. Обозначим $b_n = 2n$; $c_n = 2n + 1$; тогда

$$b_{n+1} = c_n + 1; \; c_{n+1} = b_{n+1} + 1; \; a_{n+1} = b_{n+1} \cdot c_{n+1}. \tag{20}$$

И следовательно, каждый цикл вычислений по формулам (19) и (20) состоит в вычислении по данным u_n, s_n, c_n, которые мы поместим в ячейках *ЗУ* α_1, α_2, α_3, величин u_{n+1}, s_{n+1}, c_{n+1}, которые помещаются в тех же ячейках *ЗУ* α_1, α_2, α_3.

Числа $-x^2$, 1, b_n поместим в ячейках *ЗУ* α_4, α_5, α_7.

Для числа a_n можно использовать ту же ячейку α_7, так как значение b_n для подсчета следующего цикла не нужно.

Исходное значение переменных величин следующее:

$$u_0 = x; \quad s_0 = x; \quad c_0 = 1.$$

Как и в предыдущем примере, вычисления прекращаются при условии, что вычисленный член ряда u_{n+1} меньше заданного числа ε, которое поместим в ячейку *ЗУ* α_6.

Программа 12

Числа

α_1	x	u_n
α_2	x	s_n
α_3	1	c_n
α_4	$-x^2$	
α_5	1	
α_6	c	
α_7		b_n

Команды

$k+1$	+	α_3	α_5	α_7	b_1
$k+2$	+	α_7	α_5	α_3	c_1
$k+3$	×	α_7	α_3	α_7	a_1
$k+4$	×	α_1	α_4	α_1	$-x^2u_0$
$k+5$	:	α_1	α_7	α_1	u_1
$k+6$	+	α_2	α_1	α_2	s_1
$k+7$	$\|\leqslant\|$	α_6	α_1	$k+1$	
$k+8$	*ост.*				

Для вычисления $\operatorname{tg} x$ воспользуемся разложением его в непрерывную дробь. При условии, что $|x| < \frac{\pi}{4}$, $\operatorname{tg} x$ можно вычислить с точностью до 10^{-10} посредством формулы

$$\operatorname{tg} x = \frac{x}{y}, \tag{21}$$

где

$$y = 1 - \cfrac{x^2}{3 - \cfrac{x^2}{5 - \cfrac{x^2}{7 - \cfrac{x^2}{9 - \cfrac{x^2}{11 - \cfrac{x^2}{13 - \frac{x^2}{15}}}}}}} \tag{22}$$

По схеме, аналогичной схеме Горнера, формула (22) запишется в виде

$$y \cong (1 - x^2 : \{3 - x^2 : [\ldots - x^2 : \{11 - x^2 : (13 - x^2 : [15])\} \ldots] \ldots\}). \tag{23}$$

Вычисления y по формуле (23) состоят в последовательном вычислении скобок u_{2k-1} по формуле

$$u_{2k-1} = (2k-1) - \frac{x^2}{u_{2k+1}} \qquad (k = 7,\ 6,\ \ldots,\ 1) \tag{24}$$

и представляет собой циклический процесс, причем $y \cong u_i$. Отдельный цикл состоит в вычислении по данным

$$(2k+1), \quad u_{2k+1},$$

которые помещаются в ячейках *ЗУ*

$$\alpha_1, \ \alpha_2,$$

величин $(2k-1), u_{2k-1}$, которые помещаются в те же ячейки *ЗУ* α_1, α_2.

Для вычисления y используются числа x^2 и 2, которые поместим в ячейках *ЗУ* α_3 и α_4.

Исходное значение переменных величин следующее:

$$2k+1 = 2 \cdot 7 + 1 = 15;$$
$$u_{2k+1} = u_{15} = 15.$$

Процесс вычислений y по формуле (24) заканчивается после вычисления u_1; при этом в ячейке *ЗУ* α_1 будет содержаться число 1.

В соответствии с этим переход от одного цикла к другому до получения величины y можно осуществить при помощи следующей команды условного перехода:

$\leqslant$	α_4	α_1	$k+1$

после которой следует вычисление $\operatorname{tg} x$ по формуле (21) и прекращение расчетов, т. е. команды

:	α_5	α_2	α_1
ост.			

где в ячейке *ЗУ* α_5 содержится x.

Программа 13

Числа

α_1	15	$2k+1$
α_2	15	u_{2k+1}
α_3	x^2	
α_4	2	
α_5	x	

Команды

$k+1$	—	α_1	α_4	α_1	$2k-1$
$k+2$	:	α_3	α_2	α_2	$\frac{x^2}{2k-1}$
$k+3$	—	α_1	α_2	α_2	u_{2k-1}
$k+4$	$\leqslant$	α_4	α_1	$k+1$	
$k+5$	:	α_5	α_2	α_1	$\operatorname{tg} x$
$k+6$	*ост.*				

Программа 13 представляет собой простейший пример соединения программы циклического процесса (4 команды) и программы вычислений по формуле (1 команда).

Пример 5. Приведение аргумента. Во многих случаях для вычисления тех или иных значений функций удобно пользоваться формулами приведения аргумента. Это бывает необходимо, например, при вычислениях тригонометрических функций для больших значений аргумента в машинах с фиксированной запятой. Приведение аргумента для случая функций, вычисляемых при помощи их разложения в ряды, очевидно, сократит число отдельных циклов вычислений.

Рассмотрим приведение аргумента для функции $\sin x$ к интервалу $|x| \leqslant \frac{\pi}{2}$.

Обозначим

$$\psi(x) = \left| \left\{ \frac{x}{2\pi} - \frac{1}{4} \right\} 2\pi - \pi \right| - \frac{\pi}{2}. \tag{25}$$

Очевидно, при произвольном x

$$\sin x = \sin \psi(x) \quad \text{и} \quad |\psi(x)| \leqslant \frac{\pi}{2}.$$

Таким образом, вычисление $\sin x$ разбивается на два этапа:

1) вычисление $\psi(x)$ по формуле (25),

2) вычисление $\sin \psi(x)$.

Второй из этапов — вычисление $\sin \psi(x)$ — можно осуществить по программе 12 вычисления синуса, если содержание *ЗУ* будет предварительно надлежащим образом подготовлено.

С целью сохранения вида команд, осуществляющих вычисление синуса по программе 12, занесем в ячейки *ЗУ*

$$\alpha_3,\ \alpha_5,\ \alpha_6$$

величины

$$1,\ 1,\ \varepsilon.$$

Для подготовки содержания ячеек α_1, α_2, α_4 используем процесс вычисления $\psi(x)$.

Для вычисления $\psi(x)$ по формуле (25) поместим величины

$$x,\ 2\pi,\ \frac{1}{4},\ \pi,\ \frac{\pi}{2}$$

в ячейки *ЗУ*

$$\alpha_1,\ \alpha_2,\ \alpha_4,\ \alpha_7,\ \alpha_8$$

соответственно. Будем предполагать, что среди элементарных операций, выполняемых машиной, имеется операция взятия дробной части числа $\{\ \}$ (см. программу 5) и взятия модуля (см. программу 6).

Тогда вычисления по формуле (25) осуществляются командами

$k+1$	:	α_1	α_2	α_1	$\frac{x}{2\pi}$
$k+2$	—	α_1	α_4	α_1	$\frac{x}{2\pi}-\frac{1}{4}$
$k+3$	$\{\ \}$	α_1		α_1	$\left\{\frac{x}{2\pi}-\frac{1}{4}\right\}$
$k+4$	$\times$	α_1	α_2	α_1	$\{\ \}2\pi$
$k+5$	—	α_1	α_7	α_1	$\{\ \}2\pi-\pi$
$k+6$	$\vert\ \vert$	α_1		α_1	$\vert\{\ \}2\pi-\pi\vert$
$k+7$	—	α_1	α_8	α_1	$\vert\ \vert-\frac{\pi}{2}=\psi(x)$

Для вычисления $\sin\psi(x)$ остается подготовить содержание ячеек *ЗУ* α_2 и α_4, что можно осуществить командами

$k+8$	+	α_1		α_2
$k+9$	$\times$	α_1	α_1	α_4
$k+10$	—		α_4	α_4

Программа 14

$$\sin x=\sin\psi(x);\quad \psi(x)=\left|\left\{\frac{x}{2\pi}-\frac{1}{4}\right\}2\pi-\pi\right|-\frac{\pi}{2}$$

Числа

α_1	x	u_n
α_2	2π	s_n
α_3	1	c_n
α_4	$\frac{1}{4}$	$-\psi^2(x)$
α_5	1	
α_6	ε	
α_7	π	b_n
α_8	$\frac{\pi}{2}$	

Команды

$k+1$	:	α_1	α_2	α_1
$k+2$	—	α_1	α_4	α_1
$k+3$	$\{\ \}$	α_1		α_1
$k+4$	$\times$	α_1	α_2	α_1
$k+5$	—	α_1	α_7	α_1
$k+6$	$\vert\ \vert$	α_1		α_1
$k+7$	—	α_1	α_8	α_1
$k+8$	+	α_1		α_2

$k+9$	$\times$	α_1	α_1	α_4
$k+10$	$-$		α_4	α_4
$k+11$	$+$	α_3	α_5	α_7
$k+12$	$+$	α_7	α_5	α_3
$k+13$	$\cdot\times$	α_7	α_3	α_7
$k+14$	$\times$	α_1	α_4	α_1
$k+15$	:	α_1	α_7	α_1
$k+16$	$+$	α_2	α_1	α_2
$k+17$	$\lvert\lessdot\rvert$	α_6	α_1	$k+11$
$k+18$	*ост.*			

Программа 14 представляет собой пример соединения двух программ: программы вычислений по формуле (25) (10 команд) и циклической программы вычисления синуса (8 команд).

В некоторых случаях вычисление значений функции может быть сокращено, если использовать соответствующие рекуррентные соотношения. Рассмотрим примеры.

Пример 6. Вычисление и печать таблицы значений функции e^{kh}, $k = 1, 2, \ldots, N$.

Это можно осуществить по формуле

$$e^{kh} = e^{(k-1)h} \cdot e^{h}. \tag{26}$$

Пусть величина e^h известна и находится в ячейке $ЗУ\,\beta_1$, вычисляемое значение e^{kh} будет помещаться в ячейке $ЗУ\,\beta_2$.

Отдельный цикл вычислений состоит в умножении по формуле (26) и печати числа, находящегося в ячейке $ЗУ\,\beta_2$, содержащей вначале $e^0 = 1$, и выполняется командами

k	$\times$	β_1	β_2	β_2
$k+1$	П			β_2

Число циклов в данном случае равно N.

Однако окончательное (N-е) значение ни одной из встречающихся величин заранее не известно. Поэтому для перехода

от одного цикла к другому и прекращения расчетов необходимо воспользоваться известным числом N повторений цикла, которое мы поместим в ячейку $ЗУ\ \beta_3$.

Введем в рассмотрение вспомогательную «целочисленную» переменную *) i, изменяющую свое значение при переходе от цикла к циклу на «единицу» и имеющую начальное значение $i = 0$. Поместим эту переменную в ячейку β_4. Цикл вычислений дополним командой, которую поместим в начале цикла:

$k-1$	$+$	β_4	β_5	β_4

где в ячейке β_5 содержится «единица». Тогда ячейка β_4 в любой момент времени содержит число единиц, равное числу выполнений цикла. Такие ячейки принято называть *счетчиками повторений цикла.*

Теперь логическое условие, например,

$$P\{i \neq N\}$$

может быть использовано для определения окончания цикла, а команда

$k+2$	$\neq$	β_3	β_4	$k-1$

находящаяся в конце цикла, будет осуществлять проверку этого условия и соответствующую передачу управления.

Программа 15

	Числа		Команды				
β_1	e^h		$k-1$	$+$	β_4	β_5	β_4
β_2	1	e^{kh}	k	$\times$	β_1	β_2	β_2
β_3	N		$k+1$	П			β_2
β_4	0	i	$k+2$	$\neq$	β_3	β_4	$k-1$
β_5	1		$k+3$	*ост.*			

*) При этом в качестве «единицы» может быть принята, например, единица младшего разряда.

Вместо логического условия $P\{i \neq N\}$ может быть использовано логическое условие

$$P\{i \geqslant N\}.$$

Команда

$<$	β_3	β_4	$k+4$

находящаяся после $(k-1)$-й команды, выполняет переход от одного цикла к другому и передачу управления на останов при окончании расчетов. При этом цикл заканчивается операцией безусловного перехода к началу нового цикла, т. е. командой

$\leqslant$			$k-1$

Программа 15_1

Числа

β_1	e^h		$k-1$
β_2	1	e^{kh}	k
β_3	N		$k+1$
β_4	0	k	$k+2$
β_5	1		$k+3$
			$k+4$

Команды

$+$	β_4	β_5	β_4
$<$	β_3	β_4	$k+4$
$\times$	β_1	β_2	β_2
П			β_2
$\leqslant$			$k-1$
ост.			

Команда $k-1$ выполняет счет числа циклов, т. е. работу счетчика (ячейка *ЗУ* β_1).

Команду $k-1$ можно поместить и после команды k, но в этом случае для выполнения N циклов в ячейку *ЗУ* β_3 следует поместить число $N-1$, или, сохранив прежнее содержание ячейки β_3, к началу первого цикла в счетчик β_4 следует занести единицу.

Команда безусловного перехода $k+3$ может быть опущена, если команду k поместить в конце цикла, переставив в нем местами первый и второй адреса. Теперь команда «$<$» будет передавать управление команде $k-1$ до тех пор, пока в ячейке *ЗУ* β_1 не появится число N, после чего будет выполнен останов. Всего будет выполнено N циклов.

Программа 15_2

Числа				Команды			
β_1	e^h		$k+1$	+	β_4	β_5	β_4
β_2	1	e^{kh}	$k+2$	×	β_1	β_2	β_2
β_3	N		$k+3$	П			β_2
β_4	0	k	$k+4$	<	β_4	β_3	$k+1$
β_5	1		$k+5$	*ост.*			

Пусть теперь величина e^h не дана, а должна вычисляться по данному h, которое поместим, имея в виду вычисление e^h по программе 11, в ячейку α_4.

Вычисление и печать таблицы значений функции e^{kh}, $k = 1, 2, \ldots, N$, в этом случае распадается на два этапа:

1) вычисление e^h по программе 11; значение e^h, согласно этой программе, будет получено в ячейке α_3;

2) вычисление e^{kh} по программе 15, в которой ячейка *ЗУ* β_1 должна быть заменена ячейкой *ЗУ* α_3 и третий адрес в $(k+4)$-й команде должен измениться согласно с новым номером начальной команды. Кроме того, в качестве единицы для счетчика можно использовать вместо ячейки *ЗУ* β_5, ячейку α_5. Программа будет иметь следующий вид:

Программа 16

Числа				Команды			
α_1	0		$k+1$	+	α_1	α_5	α_1
α_2	1		$k+2$	×	α_2	α_4	α_2
α_3	1	e^h	$k+3$	:	α_2	α_1	α_2
α_4	h		$k+4$	+	α_3	α_2	α_3
α_5	1		$k+5$	$\lvert\cdot\rvert$	α_6	α_2	$k+1$
α_6	ε		$k+6$	×	α_3	β_2	β_2
β_2	1	e^{kh}	$k+7$	П			β_2
β_3	N		$k+8$	+	β_4	α_5	β_4
β_4	0		$k+9$	<	β_4	β_3	$k+6$
			$k+10$	*ост.*			

Программа 16 представляет собой соединение двух циклических программ — программы 11 (5 команд) и программы 15 (5 команд). Как видим из данного примера, при соединении программ, содержащих команды условного и безусловного перехода, третьи адреса последних могут изменяться.

Вычисление и печать таблицы значений функций $\cos(a + kh)$, $\sin(a + kh)$ $(k = 1, 2, \ldots, N)$ можно получить, используя формулы:

$$\left.\begin{aligned}\cos(a+kh) &= \cos h \cos[a+(k-1)h] - \sin h \sin[a+(k-1)h],\\ \sin(a+kh) &= \sin h \cos[a+(k-1)h] + \cos h \sin[a+(k-1)h].\end{aligned}\right\} \quad (27)$$

Пусть величины $\cos h$ и $\sin h$ известны и находятся в ячейках $ЗУ$ β_1 и β_2. Вычисляемое значение $\cos(a + kh)$ и $\sin(a + kh)$ будем помещать в ячейки $ЗУ$ β_3 и β_4.

Отдельный цикл состоит в вычислении по формулам (27) и печатании чисел β_3 и β_4 и осуществляется командами:

$k+1$	×	β_2	β_3	ω_1	$\sin h \cos(a+(k-1)h)$
$k+2$	×	β_1	β_3	β_3	$\cos h \cos(a+(k-1)h)$
$k+3$	×	β_1	β_4	ω_2	$\cos h \sin(a+(k-1)h)$
$k+4$	×	β_2	β_4	β_4	$\sin h \sin(a+(k-1)h)$
$k+5$	—	β_3	β_4	β_3	$\cos(a+kh)$
$k+6$	П			β_3	
$k+7$	+	ω_1	ω_2	β_4	$\sin(a+kh)$
$k+8$	П			β_4	

Как и в предыдущем примере, число циклов заранее известно и равно N, но окончательные значения переменных, встречающихся в вычислениях, заранее не известны. Поместим в ячейку $ЗУ$ β_5 число $N - 1$ и используем ее для перехода от одного цикла к другому и определения окончания расчетов. С этой целью введем счетчик числа циклов в ячейке $ЗУ$ β_6, начальное содержание которой будет 0, и в ячейку $ЗУ$ β_7 поместим «единицу» для счетчика. Отдельный цикл вычислений, таким образом, пополняется командами

+	β_6	β_7	β_6
$\leqslant$	β_6	β_5	$k+1$

которые мы поместим в конце цикла.

Исходное содержание ячеек β_3 и β_4 должно быть $\cos a$ и $\sin a$. Предположим также, что эти величины известны.

Программа 17

Числа

β_1	$\cos h$	
β_2	$\sin h$	
β_3	$\cos a$	$\cos(a+kh)$
β_4	$\sin a$	$\sin(a+kh)$
β_5	$N-1$	
β_6	0	k
β_7	1	
ω_1		
ω_2		

Команды

$k+1$	$\times$	β_2	β_3	ω_1
$k+2$	$\times$	β_1	β_3	β_3
$k+3$	$\times$	β_1	β_4	ω_2
$k+4$	$\times$	β_2	β_4	β_4
$k+5$	$-$	β_3	β_4	β_3
$k+6$	П			β_3
$k+7$	$+$	ω_1	ω_2	β_4
$k+8$	П			β_4
$k+9$	$+$	β_6	β_7	β_6
$k+10$	$\leqslant$	β_6	β_5	$k+1$
$k+11$	*ост.*			

Выводы

1. Для циклических вычислительных процессов нужно составлять программы, состоящие из команд, необходимых для выполнения одного (первого) цикла вычислений, и некоторых вспомогательных команд.

2. Вспомогательными в циклической программе являются команды, подготавливающие надлежащее заполнение *ЗУ* для перехода от цикла к циклу, и команды, обеспечивающие необходимое число повторений циклов.

3. При построении циклических программ вычисляемые значения переменных, подготавливаемые для выполнения $(k+1)$-го цикла, нужно помещать в ячейки, в которых находились их k-е значения.

4. Для построения логического условия, определяющего окончание циклического процесса, можно воспользоваться либо специально введенной переменной, либо переменной величиной, встречающейся в вычислениях.

В первом случае удобнее всего ввести «целочисленную» переменную k (k — номер выполняемого цикла), так называемый счетчик.

При этом необходимо иметь:

а) счетчик числа циклов — ячейку, изменяющую свое состояние от цикла к циклу по определенному закону;

б) условную единицу для счетчика — ячейку, с помощью которой происходит изменение счетчика;

в) ячейку, содержащую предельное заполнение счетчика;

г) команду, выполняющую работу счетчика (входящую в цикл);

д) команду, выполняющую проверку заполнения счетчика для выяснения окончания процесса и передачи управления начальной команде или следующей за циклом команде (также входящую в цикл);

е) помимо ячеек а) — в) и команд г) — д), необходимо к моменту работы циклической программы обеспечить «начальное» заполнение счетчика и всех изменяемых от цикла к циклу величин соответственно их значениям перед прохождением первого цикла.

Переменной величиной, входящей в вычисления, можно воспользоваться для определения окончания циклического процесса, если известно ее окончательное значение или условие, которому последнее должно удовлетворять.

5. Если допустим набор значений входящих в вычисления параметров, при котором циклический участок программы должен быть опущен, то команды, осуществляющие проверку логического условия, определяющего окончание циклического процесса, надо располагать в начале цикла. В таком случае цикл в конце дополняется операцией безусловной передачи управления первой команде.

Упражнения

1. Составить программу для вычисления и печати таблицы квадратов нечетных чисел натурального ряда.

2. Составить программу извлечения кубического корня $y = \sqrt[3]{x}$, используя итерационный процесс

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

3. Составить программу вычисления обратной величины для машины, у которой нет операции деления.

4. Составить программу вычисления $\ln x$ при помощи ряда.

5. Составить программу извлечения кубического корня $y = \sqrt[3]{x}$, используя итерационный процесс, приведенный в упражнении 2, для машины, у которой нет операции деления.

6. Составить программу вычисления $\ln(1 + x)$, $|x| < 1$ при помощи ряда.

7. Составить программу вычисления e^x, используя представление

$$e^x = \frac{y+x}{y-x}, \text{ где } y = 2 + \cfrac{x^2}{6 + \cfrac{x^2}{10 + \cfrac{x^2}{14 + \cfrac{x^2}{18 + \frac{x^2}{22}}}}},$$

для $0 < |x| \leqslant 1$.

8. Составить программу вычисления e^x, $x > 0$, используя соотношение

$$e^x = e^{E(x)} e^{\{x\}}$$

и имея в *ЗУ* величину e.

9. Составить программу вычисления и печати $\sin(a + kh)$ и $\cos(a + kh)$ для $k = 0, 1, \ldots, N$ в случае, когда $\cos h$ и $\sin h$ заранее не заданы. Использовать программу вычисления $\sin h$, формулу

$$\cos h = \sqrt{1 - \sin^2 h}$$

и программу 10. Считать, что в число элементарных операций, выполняемых машиной, входит операция извлечения квадратного корня.

10. Составить программу вычисления и печати $\sin(a + kh)$ и $\cos(a + kh)$, $k = 0, \ldots, N$, в случае, когда $\cos h$ и $\sin h$ заранее не заданы. Использовать программу 16 и программу вычисления синуса для вычисления $\sin x$ и $\cos x = \sin\left(\frac{\pi}{2} - x\right)$.

11. Составить программу вычисления $\cos x$ с помощью ряда

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

12. Составить программу решения уравнения

$$\frac{1}{4} x^3 + x - 1{,}2502 = 0$$

методом итераций.

13. Составить программу для вычисления скалярного произведения двух векторов.

14. Построить управление циклическим процессом в программе 15 с помощью операций передачи управления «=», «УПЧ», «УПП».

§ 4. Циклические процессы, зависящие от параметров

В предыдущем параграфе рассматривались циклические процессы, в которых при переходе от цикла к циклу изменялось лишь содержание числовых ячеек *ЗУ*. Здесь мы рассмотрим циклические процессы, в которых при переходе от цикла к циклу изменяются также адреса чисел, участвующих в вычислениях.

Пример 1. Вычисление таблицы значений квадратов членов арифметической прогрессии с запоминанием во внешней памяти. Пусть требуется вычислить квадраты членов арифметической

прогрессии от нулевого до N-го включительно и запомнить их во внешней памяти. Выделим оперативные ячейки

$$\omega,\ \omega+1,\ \ldots,\ \omega+N, \tag{28}$$

в которые поместим последовательно вычисляемые квадраты. После окончания вычислений всех квадратов произведем пересылку кодов из ячеек (28) внутреннего *ЗУ* на места с номерами $r, r+1, r+2, \ldots, r+N$ внешнего *ЗУ* на p-й участок.

Вычисления естественно разбиваются на $N+1$ цикл: в n-м цикле вычисляется и запоминается квадрат n-го члена прогрессии $a+(n-1)c$.

Воспользуемся программой 9, которую несколько видоизменим: опустим вторую команду — печать вычисленных квадратов чисел, а первой команде придадим вид, обеспечивающий запоминание $(a+(n-1)c)^2$ в ячейке $\omega+N$, т. е.

$k+1$	$\times$	α_1	α_1	$\omega+N$

Для того чтобы вычисленная величина $(a+(n-1)c)^2$ на n-м цикле не была вытеснена получаемым результатом на $(n+1)$-м цикле, дополним программу командами

$k-N-1$	$+$	$\omega+1$		ω
$k-N$	$+$	$\omega+2$		$\omega+1$
⋮				
k	$+$	$\omega+N$		$\omega+N-1$

Введем для приведенной группы команд $k-N-1 \div k$ символическую команду группового сдвига «*ГС*»:

k	*ГС*	$\omega+1$	N	ω

выполняющую сдвиг содержимого ячеек $\omega+1, \ldots, \omega+N$ на одну ячейку влево. Начальное заполнение ячеек $\omega, \omega+1, \ldots, \omega+N$ безразлично. В результате N сдвигов величины окажутся соответственно в ячейках (28).

Для пересылки полученных результатов во внешнее $ЗУ$ введем в программу команды

$k+4$	Ba	p	$\omega+1$	$r+1$
$k+5$	$B\bar{0}$	$З$	$\omega+N$	

Получим программу 18.

Программа 18

Числа

α_1	a	
α_2	c	
α_3	$a+Nc$	
ω		a^2
$\omega+1$		$(a+c)^2$
⋮		
$\omega+N$		$(a+Nc)^2$

Команды

k	$ГС$	$\omega+1$	N	ω
$k+1$	$\times$	α_1	α_1	$\omega+N$
$k+2$	$+$	α_1	α_2	α_1
$k+3$	$\leqslant$	α_1	α_3	k
$k+4$	Ba	p	$\omega+1$	$r+1$
$k+5$	$B\bar{0}$	$З$	$\omega+N$	
$k+6$	$ост.$			

Команды в нашей программе выполняют следующие функции: команда $k+2$ подготавливает следующий член арифметической прогрессии; команда $k+1$ в каждом цикле вычисляет искомую величину квадрата в одной и той же «стандартной» ячейке $\omega+N$; k обеспечивает запоминание полученного результата — его «высылку» из стандартной ячейки. Эти команды могут быть переставлены местами при надлежащем их изменении и при подборе соответствующего начального заполнения ячейки α_1. То же построение программ возможно при вычислении таблиц любых других функций для последовательно вычисленных значений аргументов.

Пример 2. Вычисление значений многочлена. Пусть

$$f(x)=\{\ldots[(a_0x+a_1)x+a_2]x+\ldots+a_{n-1}\}x+a_n. \quad (29)$$

Рассмотрим программу вычисления многочлена по схеме Горнера, приведенную в § 1. Все команды этой программы (про-

грамма 3) с нечетными номерами, за исключением первого, одинаковы и имеют вид

$N+2k+1$	$\times$	ω	β	ω	$k=0, 1, \ldots, n.$

Вычисления по формуле (29) можно разбить на ряд циклов, в каждом из которых происходит умножение x на соответствующую скобку и прибавление соответствующего коэффициента a_i. Нулевая скобка равна a_0, в связи с чем для обеспечения цикличности работы вычисляемое значение скобки будем помещать в ячейку, содержащую a_0. Тогда вычисления на $(k+1)$-м цикле могут быть выполнены теми же командами, что и на k-м, если коэффициент a_{k+1} будет помещен в ячейку, в которой находился коэффициент a_k.

В связи с этим данные будем размещать следующим образом. Коэффициенты многочлена $a_0, a_1, \ldots, a_n$ разместим в рабочих ячейках $\alpha, \alpha+1, \ldots, \alpha+n$, а величину x поместим в ячейку β. В первом цикле программа вычислений будет иметь вид

$N+1$	$\times$	α	β	α
$N+2$	$+$	α	$\alpha+1$	α

Для обеспечения цикличности введем команды

$+$	$\alpha+2$		$\alpha+1$
$\cdot$	$\cdot$		$\cdot$
$\cdot$	$\cdot$		$\cdot$
$\cdot$	$\cdot$		$\cdot$
$+$	$\alpha+n$		$\alpha+n-1$

или команду

$ГС$	$\alpha+2$	$n-1$	$\alpha+1$

выполняющую сдвиг коэффициентов на одну ячейку влево и тем самым подготовку содержания *ЗУ* к выполнению следующего цикла.

Для выяснения окончания расчетов воспользуемся тем, что число циклов заранее известно и равно числу n (его мы поместим в ячейку $ЗУ$ β_1). Для отсчета числа выполненных циклов введем счетчик — ячейку β_2, в которую сначала поместим нуль, а в ячейку $ЗУ$ β_3 поместим единицу для счетчика. Программа цикла дополнится командами

N	$+$	β_2	β_3	β_2
$N+4$	$\leqslant$	β_2	β_1	N

первую из которых поместим в начале цикла, а вторую — в конце.

Получим программу.

Программа 19

Числа		Команды				
α	a_0	N	$+$	β_2	β_3	β_2
$\alpha+1$	a_1	$N+1$	$\times$	α	β	α
⋮		$N+2$	$+$	α	$\alpha+1$	α
$\alpha+n$	a_n	$N+3$	$ГС$	$\alpha+2$	$n-1$	$\alpha+1$
β	x	$N+4$	$\leqslant$	β_2	β_1	N
β_1	n	$N+5$	*ост.*			
β_2	0					
β_3	1					

В отличие от программы, где команда $ГС$ (или соответствующая группа команд переноса) выполняла функцию запоминания последовательности результатов, получаемых в стандартной ячейке, здесь эта команда (или соответствующая группа команд) производит засылку данных (коэффициентов) из последовательности в стандартную ячейку $\alpha+1$.

Команду $ГС$ при надлежащем ее изменении можно переставить с командами $N \div N+2$.

Программы приведенных примеров обладают тем недостатком, что в них имеет место непроизводительный перенос последовательности величин в каждом цикле, на что расходуется машинное время и ячейки *ЗУ*. С увеличением сложности задач фактическая перестановка величин в ячейках *ЗУ* может оказаться весьма затруднительной. Вместе с тем при построении программ для громоздких задач приходится считаться также и с ограниченностью памяти машины.

При наличии в наборе элементарных операций команды переадресации такая перестановка величин для построения циклических программ не обязательна.

Вместо перестановки величин, расположенных в определенной последовательности, достаточно к началу нового цикла изменить адреса команд, по которым производится выборка величин из памяти или засылка результатов в некоторую последовательность ячеек.

Таким образом, задав в машину начальный вид циклической программы, последнюю дополняют командами, подготавливающими их состояние к повторному выполнению цикла, — машина не только выполняет команды, но и подготавливает их для последующего выполнения.

Важное значение при этом приобретает вопрос распределения памяти.

Величины, участвующие в вычислениях, нужно распределить так, чтобы вычислительный процесс на отдельных этапах — циклах мог быть выполнен по программам, отличающимся между собой только тем, что один или несколько адресов каждой из этих программ изменяется в зависимости от номера цикла по закону

$$N = \alpha + ip,$$

где i — номер цикла, α — начальное значение адреса (может быть разным для различных адресов), p — некоторая константа, одинаковая для всех переменных адресов. Такие процессы принято называть *циклическими процессами, зависящими от параметра i.*

Теперь отдельный цикл вычислений в отличие от итерационного цикла должен быть дополнен командами переадресации, изменяющими соответственным образом переменные адреса программы. При этом, если число циклов заранее известно, для выяснения окончания расчетов может быть использована переменная команда цикла.

После окончания циклического процесса в случае, когда по данной программе возможны вычисления в будущем, необходимо предусмотреть восстановление исходного состояния

команд, т. е. приведение переменных команд к состоянию, исходному для первого цикла.

Пример 3. Вычисление таблицы квадратов чисел натурального ряда с запоминанием во внешней памяти. Вернемся к первому разделу этого параграфа. Для вычисления квадрата числа n используем программу 18, которую несколько видоизменим: опустим команду k (или соответствующую группу команд пересылки), а $(k+1)$-й команде придадим вид, обеспечивающий запоминание вычисленного квадрата n^2 в соответствующей ячейке $\omega + n$, т. е. вид

$k+1$	$\times$	α_1	α_1	$\omega + n$

Третий адрес команды $k+1$ — переменный, он зависит от номера выполняемого цикла, в нашем случае совпадающего с числом n. В связи с этим дополним цикл вычислений командой, подготавливающей $(k+1)$-ю команду для выполнения следующего цикла:

$\oplus$	$k+1$	β	$k+1$

которую поместим после команды $k+1$ перед командой $\leqslant$ (до или после команды «+», подготавливающей следующее число натурального ряда, — безразлично); здесь в ячейке *ЗУ* β помещается код, соответствующий «единице» III адреса, т. е. число

β				1

Для пересылки полученных в оперативных ячейках *ЗУ* $\omega+1$, $\omega+2$, ..., $\omega+N$ результатов во внешнее *ЗУ* введем в программу команды

Ва	p	$\omega+1$	$r+1$
Вб	*З*	$\omega+N$	

Получим программу.

Программа 20

Числа

α_1	1	n	
α_2	1		
α_3	N		
$\omega+1$		1^2	
⋮			
$\omega+N$		N^2	
β	¢1э III A		

Команды

$k+1$	$\times$	α_1	α_1	$\omega+1$
$k+2$	$+$	α_1	α_2	α_1
$k+3$	$\oplus$	$k+1$	β	$k+1$
$k+4$	$\leqslant$	α_1	α_3	$k+1$
$k+5$	*Ва*	p	$\omega+1$	$r+1$
$k+6$	*Вб*	*З*	$\omega+N$	

Для проверки окончания вычислений может быть также использована переменная $(k+1)$-я команда, вид которой в начале последнего (N-го) цикла известен:

$k+1$	$\times$	α_1	α_1	$\omega+N$

Поместим теперь в ячейку α_3 код, соответствующий команде $(k+1)$, а $(k+4)$-я команда заменяется командой

$k+4$	$\leqslant$	$k+1$	α_3	$k+1$

Получим программу.

Программа 20_1

Числа

α_1	1				
α_2	1				
$\omega+1$		1^2			
⋮					
$\omega+N$		N^2			
β				1	
α_3	$\times$	α_1	α_1	$\omega+N$	

Команды

$k+1$	$\times$	α_1	α_1	$\omega+1$
$k+2$	$+$	α_1	α_2	α_1
$k+3$	$\oplus$	$k+1$	β	$k+1$
$k+4$	$\leqslant$	$k+1$	α_3	$k+1$
$k+5$	*Ва*	p	$\omega+1$	$r+1$
$k+6$	*Вб*	3	$\omega+N$	
$k+7$	*ост.*			

Пример 4. Вычисление значений многочлена с применением команды переадресации. Построим программу для примера 2 этого параграфа, воспользовавшись командой переадресации. Расположим теперь коэффициенты многочлена

$$a_0,\ a_1,\ \ldots,\ a_n \text{ и } x$$

в ячейках $ЗУ$

$$\omega,\ \alpha+1,\ \alpha+2,\ \ldots,\ \alpha+n,\ \alpha$$

соответственно; тогда команда с номером $N+2k$ программы 19 запишется так:

$N+2k$	$+$	ω	$\alpha+k$	ω

а первая команда примет вид ее нечетных команд.

Вычисления значений многочлена, таким образом, можно разбить на циклы, состоящие из команд

$N+2k-1$	$\times$	ω	α	ω
$N+2k$	$+$	ω	$\alpha+k$	ω

Команда $N+2k$ имеет переменный второй адрес, который при переходе от цикла к циклу должен быть увеличен на единицу. Для построения циклической программы дополним команды $N+1$, $N+2$

$N+1$	$\times$	ω	α	ω
$N+2$	$+$	ω	$\alpha+1$	ω

командой переадресации

$N+3$	$\oplus$	$N+2$	β_4	$N+2$

где в ячейку β_4 помещено вспомогательное число

β_4			1	

Для окончания расчетов воспользуемся тем же счетчиком, что и в предыдущей программе, чем программа цикла будет завершена.

Однако для возможности повторных вычислений по данной программе ее следует пополнить командой (не входящей в цикл), восстанавливающей исходное состояние переменной $(N+2)$-й команды.

Это можно сделать двумя различными способами.

1. Восстановление переменной команды программы с помощью команды переадресации. Для восстановления к исходному виду $(N+2)$-й команды выполним после окончания циклического процесса команду

$N+5$	$\ominus$	$N+2$	β_5	$N+2$

где в ячейку $ЗУ$ β_5 помещено вспомогательное число

β_5			n	

Получим программу.

Программа 21

Числа

α_1	x	
ω	a_0	$f(x)$
$\alpha+1$	a_1	
$\alpha+2$	a_2	
$\vdots$		
$\alpha+n$	a_n	

β_1			1	
β_5			n	

β_2	n	
β_3	0	k
β_4	1	

Команды

N	$+$	β_3	β_4	β_3
$N+1$	$\times$	ω	α_1	ω
$N+2$	$+$	ω	$\alpha+1$	ω
$N+3$	$\oplus$	$N+2$	β_1	$N+2$
$N+4$	$\leqslant$	β_3	β_2	N
$N+5$	$\ominus$	$N+2$	β_5	$N+2$
$N+6$	*ост.*			

Число используемых в программе ячеек может быть уменьшено, если в качестве единицы для счетчика — ячейки β_4 — принять единицу второго адреса — ячейку β_1, и в качестве константы для сравнения β_2 — соответственно константу β_5.

Программа 21_1

Числа

α_1	x				
ω	a_0	$f(x)$			
$\alpha+1$	a_1				
.	.				
.	.				
.	.				
$\alpha+n$	a_n				
β_1			1		
β_2			n		
β_3	0				

Команды

N	+	β_3	β_1	β_3
$N+1$	$\times$	ω	α_1	ω
$N+2$	+	ω	$\alpha+1$	ω
$N+3$	$\oplus$	$N+2$	β_1	$N+2$
$N+4$	$\leqslant$	β_3	β_2	N
$N+5$	$\odot$	$N+2$	β_2	$N+2$
$N+6$	*ост.*			

2. Восстановление переменной команды с помощью засылки в ее адрес соответствующего кода.

Поместим в ячейку *ЗУ* β вспомогательное число

β	+	ω	$\alpha+1$	ω

и дополним программу из команд N, $N+1$, ..., $N+4$ так называемой командой засылки:

$\oplus$		β	$N+2$

В результате выполнения этой команды программа, содержащая переменную команду, приобретает исходный вид, пригодный для повторных вычислений по ней. При этом, разумеется, перед началом повторных вычислений надлежащим образом должны быть подготовлены также и числа.

Программа 22

Числа

α_1	x	
ω	a_0	$f(x)$
$\alpha+1$	a_1	
$\alpha+n$	a_n	

β_1			1	
β_2			n	
β_3	0			
β	+	ω	$\alpha+1$	ω

Команды

N	+	β_3	β_1	β_3
$N+1$	$\times$	ω	α_1	ω
$N+2$	+	ω	$\alpha+1$	ω
$N+3$	$\oplus$	$N+2$	β_1	$N+2$
$N+4$	$\leqslant$	β_3	β_2	N
$N+5$	$\oplus$		β	$N+2$
$N+6$	*ост.*			

Как в первом, так и во втором случае команда восстановления может быть размещена перед командами цикла. В первом случае в связи с этим надо надлежаще изменять соответствующую переменную команду. Во втором — безразлично, какой код помещен в эту ячейку. Таким образом, по количеству вводимой информации второй способ восстановления переменных команд более рациональный. Кроме этого, в случае возможных сбоев в работе машины второй способ оказывается также эффективнее: внесенная константа на место искаженного кода команды восстанавливает его к истинному виду, тогда как по первому способу появившийся сбой в переменной команде командой восстановления не будет устранен.

Программа 22 может быть сокращена, если воспользоваться тем обстоятельством, что окончание расчетов должно произойти после того, когда команда $N+2$ примет вид

γ	+	ω	$\alpha+n$	ω

Таким образом, из программы 22 можно выбросить команду N (и ячейку β_3), а для выяснения окончания расчетов и передачи управления началу нового цикла использовать сравнение переменной команды с постоянной, соответствующей коду γ.

Программа 22_1

Числа

α_1	x	
ω	a_0	$f(x)$
$\alpha+1$	a_1	
$\alpha+n$	a_n	

β_1			1	
β_2			n	
γ	+	ω	$\alpha+n$	ω

Команды

$N+1$	$\times$	ω	α_1	ω
$N+2$	+	ω	$\alpha+1$	ω
$N+3$	$\oplus$	$N+2$	β_1	$N+2$
$N+4$	$<$	$N+2$	γ	$N+1$
$N+5$	$\ominus$	$N+2$	β_2	$N+2$
$N+6$	*ост.*			

Поставленная задача может быть решена иначе путем выделения так называемой стандартной ячейки. А именно, выделим стандартную ячейку *ЗУ* δ, в которую в начале каждого i-го цикла вычислений будем последовательно заносить коэффициенты a_i, $i = 1, 2, \ldots, n$, с помощью команды переноса

(*)	+		$\alpha+i$	δ

(начальное состояние ячейки δ безразлично).

Первый адрес этой команды — переменный, в связи с чем программу цикла дополним командой переадресации

(**)	$\oplus$	(*)	β_1	(*)

где в ячейке *ЗУ* β_1 помещается «1» *ПА*. Начальное состояние команды (*) должно быть следующим:

(*)	+		$\alpha+1$	δ

Команду (*) поместить после команды (**); в последнем случае в ячейке $ЗУ$ $\alpha + 1$ вначале должно быть помещено число a_1, а в первом адресе команды ( ) — номер $\alpha + 2$.

Программа 22_2

Числа

α	x	
$\alpha+1$	a_1	
⋮		
$\alpha+n$	a_n	
ω	a_0	$f(x)$
δ		рабочая

β_1			1	
β_2	+		$\alpha+n$	δ
β_3	+		$\alpha+1$	δ

Команды

N	+		$\alpha+1$	δ
$N+1$	×	ω	α	ω
$N+2$	+	ω	δ	ω
$N+3$	⊕	N	β_1	N
$N+4$	≤	N	β_2	N
$N+5$	⊖		β_3	N
$N+6$	*ост.*			

Здесь для выяснения окончания циклического процесса нами использована переменная команда программы. Последняя программа имеет одной командой больше; однако во многих случаях такое осуществление программы для циклического процесса, зависящего от параметра, бывает удобным и приводит к значительному сокращению программы. Следующий пример приведен с целью иллюстрации данного положения.

Пример 5. Вычисление суммы

$$s = \sum_{i=1}^{n} \frac{x_i^3 + a}{x_i \sqrt{x_i}\,(x_i - a)}.$$

Вычисления естественно распадаются на n циклов: в k-м цикле ($k = 1, 2, \ldots, n$) вычисляется k-е слагаемое

$$A_k = \frac{x_k^3 + a}{x_k \sqrt{x_k}\,(x_k - a)}$$

и прибавляется к сумме предыдущих слагаемых s_k. Очевидно, команды, осуществляющие вычисления, в каждом из циклов будут зависеть от номера цикла.

Для построения циклической программы разместим данные следующим образом:

№ ячейки	$\alpha+1$	$\alpha+2$	. . .	$\alpha+n$	β
Ее содержимое	x_1	x_2		x_n	a

и выделим оперативную ячейку ω для накопления суммы, в которую вначале поместим 0.

Программа вычислений на k-м цикле будет следующая:

$N+1$	$\times$	$\alpha+k$	$\alpha+k$	ω_1	x_k^2	I, II
$N+2$	$\times$	ω_1	$\alpha+k$	ω_1	x_k^3	II
$N+3$	$+$	ω_1	β	ω_1	x_k^3+a	
$N+4$	$\sqrt{}$	$\alpha+k$		ω_2	$\sqrt{x_k}$	I
$N+5$	$\times$	ω_2	$\alpha+k$	ω_2	$x_k\sqrt{x_k}$	II
$N+6$	$-$	$\alpha+k$	β	ω_3	x_k-a	I
$N+7$	$\times$	ω_2	ω_3	ω_2	$x_k\sqrt{x_k}(x_k-a)$	
$N+8$	:	ω_1	ω_2	ω_1	A_k	
$N+9$	$+$	ω	ω_1	ω	s_k	

Здесь справа в таблице указаны адреса команд, зависящих от параметра — номера цикла.

В связи с наличием переменных адресов программа цикла должна быть пополнена соответствующими командами переадресации. При этом легко заметить, что число констант переадресации может быть уменьшено, если, воспользовавшись коммутативностью операции умножения, I и IIA команд $N+2$ и $N+5$ переставить местами.

Для выяснения окончания расчетов воспользуемся одной из переменных команд, например $(N+2)$-й, для чего в ячейку $ЗУ$ γ поместим соответствующую вспомогательную константу

γ	$\times$	ω_1	$\alpha+n$	ω_1

Программа будет иметь вид:

Программа 23

Числа

$\alpha+1$	x_1	
$\alpha+2$	x_2	
⋮		
$\alpha+n$	x_n	
β	a	
ω	0	s
ω_1		рабочие
ω_2		
ω_3		

γ	×	$\alpha+n$	ω_1	ω_1
γ_1		1	1	
γ_2		1		

Команды

$N+1$	×	$\alpha+1$	$\alpha+1$	ω_1
$N+2$	×	$\alpha+1$	ω_1	ω_1
$N+3$	+	ω_1	β	ω_1
$N+4$	$\sqrt{}$	$\alpha+1$		ω_2
$N+5$	×	$\alpha+1$	ω_2	ω_2
$N+6$	−	$\alpha+1$	β	ω_3
$N+7$	×	ω_2	ω_3	ω_2
$N+8$	:	ω_1	ω_2	ω_1
$N+9$	+	ω	ω_1	ω
$N+10$	⊕	$N+1$	γ_1	$N+1$
$N+11$	⊕	$N+2$	γ_2	$N+2$
$N+12$	⊕	$N+4$	γ_2	$N+4$
$N+13$	⊕	$N+5$	γ_2	$N+5$
$N+14$	⊕	$N+6$	γ_2	$N+6$
$N+15$	⩽	$N+2$	γ_2	$N+1$
$N+16$	*ост.*			

Если по данной программе возможны повторные вычисления, например в том случае, когда данная группа команд является промежуточным этапом вычислений внутри некоторого другого цикла, то она должна быть дополнена соответствующими пятью командами восстановления, а совокупность чисел — надлежащими константами восстановления.

Воспользуемся теперь приемом засылки в стандартную ячейку. С этой целью выделим стандартную ячейку *ЗУ* (оперативную) δ и дополним цикл командой переноса

N	+		$\alpha+1$	δ

Теперь команды $N+1$, $N+2$, $N+4$, $N+5$, $N+6$ становятся не зависящими от параметра и вместо пяти команд переадресации в программу следует включить команду

$N+10$	$\oplus$	N	γ_2	N

Вместе с тем отпадает надобность в константе переадресации. Получим программу.

Программа 23_1

Числа

$\alpha+1$	x_1	
$\alpha+2$	x_2	
⋮		
$\alpha+n$	x_n	
β	a	
ω	0	s
ω_1		рабочие
ω_2		
ω_3		
γ_1		1
γ_2	+	$\alpha+n$ $\quad$ $\bar{0}$

Команды

N	+		$\alpha+1$	$\bar{0}$
$N+1$	$\times$	$\bar{0}$	$\bar{0}$	ω_1
$N+2$	$\times$	$\bar{0}$	ω_1	ω_1
$N+3$	+	ω_1	β	ω_1
$N+4$	$\surd$	$\bar{0}$		ω_2
$N+5$	$\times$	$\bar{0}$	ω_2	ω_2
$N+6$	—	$\bar{0}$	β	ω_3
$N+7$	$\times$	ω_2	ω_3	ω_2
$N+8$	:	ω_1	ω_2	ω_1
$N+9$	+	ω	ω_1	ω
$N+10$	$\oplus$	N	γ_1	N
$N+11$	$\leqslant$	N	γ_2	N
$N+12$	*ост.*			

Экономия в ячейках получается еще большей, если имеется необходимость в восстановлении переменных команд. Заметим, что в данной программе за счет изменения порядка вычислений может быть сэкономлена одна рабочая ячейка.

Пример 6. Вычисление интеграла

$$I=\int_0^1 y\,dx, \quad \text{где} \quad y=\frac{\sqrt{1+x^2}-\sqrt{1-x^2}}{1+x}. \tag{30}$$

По формуле Симпсона

$$\int_a^b y\,dx = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + \ldots + 2y_{2n-2} + 4y_{2n-1} + y_{2n}) \tag{31}$$

для заданного числа делений $2n$; $y_i = y(x_i)$, $x_i = x_0 + ih$,

$$h = \frac{b-a}{2n}.$$

Поскольку вычисления y_i для всех значений $i = 0, 1, 2, \ldots, 2n$ могут быть выполнены с помощью одной и той же группы команд, представляется естественным предварительное вычисление этих величин с запоминанием, после которого остаются вычисления по формуле (31). Однако в таком случае понадобится $2n + 1$ ячеек $ЗУ$ для запоминания величин y_i.

Программа 24

Числа

α	x_i	
β	1	
γ	h	
ω_1		рабочие
ω_2		
$\omega+1$		y_i

Команды

$k+1$	$\times$	α	α	ω_1	x_i^2
$k+2$	$+$	β	ω_1	ω_2	$1+x_i^2$
$k+3$	$\surd$	ω_2		ω_2	$\sqrt{1+x_i^2}$
$k+4$	$-$	β	ω_1	ω_1	$1-x_i^2$
$k+5$	$\surd$	ω_1		ω_1	$\sqrt{1-x_i^2}$
$k+6$	$-$	ω_2	ω_1	ω_1	$\sqrt{1+x_i^2}-\sqrt{1-x_i^2}$
$k+7$	$+$	β	α	ω_2	$1+x_i$
$k+8$	:	ω_1	ω_2	$\omega+1$	y_i
$k+9$	$+$	α	γ	α	x_{i+1}

(A)

Число ячеек, необходимых для запоминания величин y_i, может быть значительно сокращено, если мы примем следующую схему вычислений:

$$\left.\begin{aligned} \Delta s_i &= \frac{h}{3}(y_{2i} + 4y_{2i+1} + y_{2i+2}),\\ s_{i+1} &= s_i + \Delta s_i;\quad i = 0,\ 1,\ \ldots,\ n-1;\quad s_0 = 0,\\ I &= s_n. \end{aligned}\right\} \tag{32}$$

Теперь для перехода от вычислений величин y_{2i}, y_{2i+1}, y_{2i+2} к вычислению s_{i+1} необходимо удерживать в памяти лишь величины s_i, y_{2i}, y_{2i+1}, y_{2i+2}. Вычисление по схеме (32) можно разбить на ряд циклов: на i-м цикле вычисляется величина Δs_i и прибавляется к s_i. Общее число циклов равно n.

Вычисление величин y_{2i}, y_{2i+1},, y_{2i+2}, необходимых для подсчета Δs_i, в свою очередь можно выполнить с помощью циклической программы из трех циклов. Действительно, выделим для этих величин последовательность ячеек $\omega + 1$, $\omega + 2$, $\omega + 3$. Тогда программа (A) для вычисления величины y_{2i} может быть использована для вычисления y_{2i+1} и y_{2i+2}, если над командой, содержащей адрес $\omega + 1$, произвести соответствующую переадресацию. В связи с этим дополним программу (A) командой переадресации

$k+10$	$+$	$k+8$	δ	$k+8$

где

δ_1				$\cdot 1$

Команды

$k+11$	$\leqslant$	$k+8$	δ_2	$k+1$
$k+12$	$+$		δ_3	$k+8$

где

δ_2	:	ω_1	ω_2	$\omega+3$
δ_3	:	ω_1	ω_2	$\omega+1$

выполняют трехкратное повторение цикла и восстановление переменной команды для повторных вычислений на следующем шаге. Поскольку на следующем шаге вновь понадобится величина y_{2i+1}, введем в программу команду

$k+13$	$-$	α	γ	α

восстанавливающую необходимое значение аргумента. Для выяснения момента прекращения расчетов воспользуемся известным значением x на последнем цикле $x = 1$.

Получим программу. 24_1.

Программа 24_1

Числа

α	x_0			
β	1			
γ	h			
ω_1				
ω_2				
$\omega+1$		y_{2i}		
$\omega+2$		y_{2i+1}		
$\omega+3$		y_{2i+2}		
δ_1				1
δ_2	:	ω_1	ω_2	$\omega+3$
δ_3	:	ω_1	ω_2	$\omega+1$
β_1	4			
β_2	$h/3$			
β_3	0			

Команды

$k+1$	$\times$	α	α	ω_1
$k+2$	$+$	β	ω_1	ω_2
$k+3$	$\vee$	ω_2		ω_2
$k+4$	$-$	β	ω_1	ω_1
$k+5$	$\vee$	ω_1		ω_1
$k+6$	$-$	ω_2	ω_1	ω_1
$k+7$	$+$	β	α	ω_2
$k+8$	:	ω_1	ω_2	$\omega+1$
$k+9$	$+$	α	γ	α
$k+10$	$\oplus$	$k+8$	δ_1	$k+8$
$k+11$	$\leqslant$	$k+8$	δ_2	$k+1$
$k+12$	$\oplus$		δ_3	$k+8$
$k+13$	$-$	α	γ	α
$k+14$	$\times$	$\omega+2$	β_1	ω_1
$k+15$	$+$	ω_1	$\omega+1$	ω_1
$k+16$	$+$	ω_1	$\omega+3$	ω_1
$k+17$	$\times$	ω_1	β_2	ω_1
$k+18$	$+$	β_3	ω_1	β_3
$k+19$	$\leqslant$	α	β	$k+1$
$k+20$	Π			β_3
$k+21$	*ост.*			

Недостатком приведенной программы является то, что величина y_{2i+2}, использованная на i-м шаге (как содержимое ячейки $\omega+3$) и встречающаяся в расчетах на $(i+1)$-м шаге (как содержимое ячейки $\omega+1$), каждый раз повторно вычисляется для всех $x_i = x_0 + ih$, на что расходуется рабочее время машины. Кроме того, наличие переменной $(k+8)$-й команды влечет за собой работу команд $k+10$ переадресации и $k+12$ восстановления и также требует соответствующего заполнения ячеек δ_1, δ_3.

В связи с изложенным примем следующий прием построения программы, так называемый прием «циркуляции величин в стандартных ячейках».

Поместим вычисляемое значение y в ячейку $\omega+4$ и введем в программу команды переноса

$k+10$	+	$\omega+2$		$\omega+1$
$k+11$	+	$\omega+3$		$\omega+2$
$k+12$	+	$\omega+4$		$\omega+3$

или

ГП	$\omega+2$	3	$\omega+1$

и команды

$k+13$	—		δ	δ
$k+14$	$\leqslant$		δ	$k+1$

где в ячейку δ вначале помещается код «—0», обеспечивающий двукратное повторение цикла. Теперь отпадает надобность в командах переадресации и восстановления, а также в команде $k+13$ и высвобождаются ячейки, использованные в качестве вспомогательных констант. Получаем программу 24_2.

Программа 24_2

Числа

α	x_0	
β	1	
γ	h	
ω_1		
ω_2		
$\omega+1$		y_{2i}
$\omega+2$		y_{2i+1}
$\omega+3$	y_0	y_{2i+2}
δ	-0	
β_1	4	
β_2	$\frac{n}{3}$	
β_3	0	

Команды

$k+1$	$\times$	α	α	ω_1
$k+2$	$+$	β	ω_1	ω_2
$k+3$	$\vdots$	ω_2		ω_2
$k+4$	$-$	β	ω_1	ω_1
$k+5$	$\vdots$	ω_1		ω_1
$k+6$	$-$	ω_2	ω_1	ω_1
$k+7$	$+$	β	α	ω_2
$k+8$	$:$	ω_1	ω_2	$\omega+4$
$k+9$	$+$	α	γ	α
$k+10$	$+$	$\omega+2$		$\omega+1$
$k+11$	$+$	$\omega+3$		$\omega+2$
$k+12$	$+$	$\omega+4$		$\omega+3$
$k+13$	$-$		δ	δ
$k+14$	$\leqslant$		δ	$k+1$
$k+15$	$\times$	$\omega+2$	β_1	ω_1
$k+16$	$+$	ω_1	$\omega+1$	ω_1
$k+17$	$+$	ω_1	$\omega+3$	ω_1
$k+18$	$\times$	ω_1	β_2	ω_1
$k+19$	$+$	β_3	ω_1	β_3
$k+20$	$\leqslant$	α	β	$k+1$
$k+21$	$П$			β_3
$k+22$	*ост.*			

По данной программе величина y_0, необходимая для вычислений на первом шаге, предварительно вычисляется и помещается в ячейку $\omega+3$. Во избежание этих вычислений можно

изменить команды, управляющие повторением внутреннего цикла, таким образом, чтобы внутренний цикл на первом шаге выполнялся три раза, а на последующих шагах — два раза.

Приведем одно из возможных построений такого рода.

Поместим в ячейку δ величину «3» и вместо команд $k+13$, $k+14$ дополним программу командами

$k+13$	—	δ	β	δ
$k+14$	$\leqslant$		δ	$k+1$
$k+15$	+		δ_2	δ

где в ячейку δ_2 помещено число «2».

Команда $k+15$ восстанавливает содержимое ячейки δ к виду, необходимому для выполнения вычислений, последующих за первым циклом. В таком случае начальное заполнение ячейки $\omega+3$ несущественно.

Пример 7. Сдвиг последовательности чисел $a_1, a_2, \ldots, a_n$. Числа $a_1, a_2, \ldots, a_n$ (в машине с фиксированной запятой) сдвинуть влево на такое число разрядов (одно и то же для всех чисел), чтобы по крайней мере одно из них стало по модулю больше или равно $\frac{1}{2}$.

Решение поставленной задачи распадается на ряд циклов, в каждом из которых должно быть проверено условие

$$P\left(a_i < \tfrac{1}{2};\ 1 \leqslant i \leqslant n\right) = P_1\left(a_1 < \tfrac{1}{2}\right) \wedge \ \ldots \ \wedge P_n\left(a_n < \tfrac{1}{2}\right), \quad (33)$$

и если это условие выполнено, группа чисел сдвигается на один разряд влево. При невыполнении условия (33) расчеты прекращаются.

В данном примере число циклов заранее не известно. Проверка условия, определяющего число циклов, должна производиться в начале цикла, так как возможен случай, когда число циклов окажется равным нулю. Как мы имели в программе 9, в таком случае цикл должен быть дополнен в конце командой безусловной передачи управления его первой команде.

Поместим в ячейки $ЗУ\ \alpha_1, \alpha_2, \ldots, \alpha_n$ величины $a_1, \ldots, a_n$ соответственно, в ячейку β — число $\frac{1}{2}$. Тогда программа решения поставленной задачи представлена программой 25.

Программа 25.

Числа		Команды				
α_1	a_1	$N+1$	$\|\leqslant\|$	β	α_1	$N+2n+2$
α_2	a_2	$N+2$	$\|\leqslant\|$	β	α_2	$N+2n+2$
.		.				
.		.				
.		.				
α_n	a_n	$N+n$	$\|\leqslant\|$	β	α_n	$N+2n+2$
β	$\frac{1}{2}$	$N+n+1$	.	α_1	β	α_1
		$N+n+2$	.	α_2	β	α_2
		.				
		.				
		.				
		$N+2n$	.	α_n	β	α_n
		$N+2n+1$	$\|\leqslant\|$			$N+1$
		$N+2n+2$	*ост.*			

Команды $N+1 \div N+n$ осуществляют проверку выполнения сложного условия (33); команды $N+n+1 \div N+2n$ выполняют сдвиг группы ячеек $\alpha+1, \ldots, \alpha+n$ влево на один разряд. Из математической логики известно соотношение

$$\overline{P_1 \wedge P_2 \wedge \ldots \wedge P_n} = \overline{P}_1 \vee \overline{P}_2 \vee \ldots \vee \overline{P}_n,$$

иллюстрирующее то обстоятельство, что невыполнение сложного условия $P = \wedge P_i$ возможно лишь при условии невыполнения одного из элементарных условий $P_1, P_2, \ldots, P_n$.

Это наглядно иллюстрируется группой команд $N+1, \ldots, N+n$. Передача управления на команду $N+2n+2$ возможна лишь при невыполнении по крайней мере одного из элементарных условий $P_1, \ldots, P_n$.

Если теперь величины $a_1, a_2, \ldots, a_n$ разместить в последовательность ячеек, номера которых образуют арифметическую прогрессию, например в последовательность $\alpha+1, \alpha+2, \ldots, \alpha+n$, то для каждой из групп команд $N+1 \div N+n$ и $N+n+1 \div N+2n$ могут быть построены циклические программы с одинаковым числом циклов, равным n.

Программа 25_1

Числа

$\alpha+1$	a_1			
$\alpha+2$	a_2			
⋮				
$\alpha+n$	a_n			
β	$\frac{1}{2}$			
γ	0			
γ_1	1			
γ_2	n			
δ_1			1	
δ_2		1		1
δ_3	$\|\leqslant\|$	β	$\alpha+1$	$N+13$
δ_4	⋮	$\alpha+1$	β	$\alpha+1$

Команды

N	+			γ
$N+1$	+	γ	γ_1	γ
$N+2$	$\|\leqslant\|$	β	$\alpha+1$	$N+13$
$N+3$	$\oplus$	$N+2$	δ_1	$N+2$
$N+4$	$\|\leqslant\|$	γ	γ_2	$N+1$
$N+5$	+	—	—	γ
$N+6$	+	γ	γ_1	γ
$N+7$	⋮	$\alpha+1$	β	$\alpha+1$
$N+8$	$\oplus$	$N+7$	δ_2	$N+7$
$N+9$	$\|\leqslant\|$	γ	γ_2	$N+6$
$N+10$	$\oplus$	—	δ_3	$N+2$
$N+11$	$\oplus$	—	δ_4	$N+7$
$N+12$	$\|\leqslant\|$			N
$N+13$	*ост.*			

Преимущество последней программы состоит в том, что она содержит небольшое количество команд, которые не зависят от количества рассматриваемых чисел в группе. Зависимым от этого числа оказывается лишь содержимое ячейки γ_2, содержащей константу для сравнения счетчика γ. Однако в отличие от предыдущих программ данная программа имеет переменные команды, в связи с чем для вычислений ее команды должны быть помещены в оперативную память.

Пример 8. Проверка выполнения неравенств $a_0>0$, $a_1>0$ ..., $a_n>0$. Пусть дана последовательность величин a_i ($i=0, 1, 2, \ldots, n$) как функций параметров k_1 и k_2 вида

$$a_i=\alpha_{i0}+\alpha_{i1}k_1+\alpha_{i2}k_2+\beta_ik_1k_2+\gamma_ik_1^2+\delta_ik_2^2. \tag{34}$$

Проверить выполнение неравенств

$$a_0>0,\quad a_1>0,\quad \ldots,\quad a_n>0. \tag{35}$$

При этом требуется в случае, когда условие (35) не выполняется, напечатать условный знак, и если оно выполнено, передать управление некоторой команде.

Расположим данные параметры в ячейку *ЗУ*:

№ ячейки	Содержание
$r+i$	α_{i0}
$s+i$	α_{i1}
$t+i$	α_{i2}
$p+i$	β_i
$q+i$	γ_i
$v+i$	δ_i

Проверка указанного условия (35), естественно, разбивается на ряд циклов: в i-м цикле, $i=0, 1, \ldots, n$, определяется знак коэффициента уравнения a_i. Если $a_i > 0$ и $i < n$ (проверка необходимого признака еще не закончена), управление передается следующему циклу. Если $a_i \leqslant 0$ (что означает невыполнение указанного условия), печатается условный знак и вычисления прекращаются. Наконец, если $a_i > 0$, $i = n$, управление передается согласно условию команде N.

Цикл, естественно, дополняется соответствующими командами переадресации, обеспечивающими надлежащее состояние памяти при переходе от цикла к циклу. Для сокращения числа команд в многочленах (34) предварительно сделаны возможные вынесения за скобки, а именно, вычисления коэффициентов a_i ведутся по формулам

$$a_i = \alpha_{i0} + k_1(\alpha_{i1} + \beta_i k_2 + \gamma_i k_1) + k_2(\alpha_{i2} + \delta_i k_2), \quad i = 0, 1, \ldots, n.$$

Программа 26

Числа

$r+i$	α_{i0}
$s+i$	α_{i1}
$t+i$	α_{i2}
$p+i$	β_i
$q+i$	γ_i

Команды

$k+1$	$\times$	p	ε_2	ω_1
$k+2$	$+$	s	ω_1	ω_1
$k+3$	$\times$	q	ε_1	ω_2
$k+4$	$+$	ω_1	ω_2	ω_1
$k+5$	$\times$	ω_1	ε_1	ω_1

Числа

$v+i$	δ_i			
ε_1	k_1			
ε_2	k_2			
η	условный знак			
Δ	«1» 1Δ			
ξ	$\times$	$p+n$	ε_2	ω_1

Команды

$k+6$	$+$	r	ω_1	ω_1
$k+7$	$\times$	v	ε_2	ω_2
$k+8$	$+$	t	ω_2	ω_2
$k+9$	$\times$	ω_2	ε_2	ω_2
$k+10$	$+$	ω_1	ω_2	ω_1
$k+11$	$\oplus$	$k+1$	Δ	$k+1$
$k+12$	$\oplus$	$k+2$	Δ	$k+2$
$k+13$	$\oplus$	$k+3$	Δ	$k+3$
$k+14$	$\oplus$	$k+6$	Δ	$k+6$
$k+15$	$\oplus$	$k+7$	Δ	$k+7$
$k+16$	$\oplus$	$k+8$	Δ	$k+8$
$k+17$	$\leqslant$	ω_1	—	$k+20$
$k+18$	$\leqslant$	$k+1$	ξ	$k+1$
$k+19$	$\leqslant$	—	—	N
$k+20$	Π			η
$k+21$	*ост.*			

Здесь мы имеем пример циклического процесса, заканчивающегося по сложному логическому условию: либо на i-м цикле, $i=0, 1, \ldots, n$, по признаку — коэффициент $a_i \leqslant 0$, либо на n-м цикле по счетчику, если $a_i > 0$ для всех $i=0, 1, \ldots, n$. При этом в зависимости от способа окончания циклического процесса управление передается разным командам.

Выводы

1. Цикличность программ вычислительных процессов, зависящих от параметров, достигается благодаря упорядоченному размещению в ячейках *ЗУ* зависящих от параметров величин и применению команд переадресации.

2. Программы циклических процессов содержат команды, подготавливающие в каждом цикле заполнение чисел *ЗУ* и переменных команд для выполнения следующего цикла.

3. Для построения логических условий, определяющих окончание параметрических циклических процессов, могут быть использованы переменные команды программы.

4. Для повторного применения изменяемой циклической программы необходимо предусматривать восстановление переменных команд. В таком случае в программу вводятся (не входящие в цикл) команды восстановления, которые осуществляются: а) с помощью операции переадресации, б) с помощью засылки в адрес переменной команды соответствующего кода. Впрочем, восстановление переменных команд может осуществляться путем операции повторного ввода всей программы из внешнего во внутреннее *ЗУ*.

5. В результате применения операции засылки зависящих от параметра величин в стандартные ячейки вычислительная часть программы может быть сделана неизменной; при этом во многих случаях уменьшается число переменных команд программы, чем достигается экономия в ячейках *ЗУ*.

Упражнения

1. Составить циклическую программу для вычисления суммы

$$s=\sum_{i=1}^{n}\frac{x_i+a}{x_i-b}.$$

2. Составить программу для вычисления значений многочлена с передачей управления от цикла к циклу командой «=», или *«УПЧ»*, или *«УПП»*.

3. Составить программу для вычисления интеграла

$$I=\int_a^b y\,dx, \quad \text{где} \quad y=\frac{x\sin x}{1+x^2},$$

по формуле Симпсона для различных видов условных передач управления.

4. Составить программу для рассмотренного в тексте примера 7 (программа 25_1), используя для окончания внутренних циклов заранее известный вид переменных команд.

5. Составить программу для рассмотренного в тексте примера 7 в случае, когда число сдвигов учитывается (с запоминанием масштаба) для машины с условной передачей управления по числу; для машин с операцией условной передачи управления по неравенству.

§ 5. Блок-схемное программирование и передача управления на подпрограммы

При составлении программ для сложных задач удобно пользоваться разбиением программы на так называемые *блоки*.

Блоком называется такой участок программы, к которому передача управления от других блоков возможна лишь к его

первой команде и передача управления от которого другим блокам возможна лишь от его последней команды. Внутри блока допускаются передачи управления как условные, так и безусловные, не связанные с выходом на другие блоки.

По блокам вначале составляется блок-схема программы, на которой для наглядности стрелками указываются возможные переходы между блоками, а затем последовательно составляются программы для всех блоков. Удобно также в местах разветвлений на каждой из стрелок указывать условие, при котором происходит передача управления к данному блоку.

При блок-схемном программировании можно достичь существенного сокращения программ за счет выделения групп команд, повторяющихся в процессе решения задачи, в отдельные блоки. Такие повторяющиеся группы команд принято называть подпрограммами. Наличие специальных команд передачи управления на подпрограммы облегчает их повторное использование при программировании.

Если данная группа команд представляет интерес с точки зрения решения многих задач, то такие тщательно проверенные подпрограммы удобно хранить в виде библиотеки на специальных перфокартах или перфолентах или иметь в памяти в виде стандартных (быстронаборных) блоков во внутренней части *ЗУ* или в специально отведенных местах внешней памяти. При составлении новой программы библиотечные подпрограммы могут включаться в нее как часть, не требующая повторной проверки, что весьма сокращает время ввода и проверки программы.

Пусть после команды k основной программы требуется перейти к выполнению некоторой подпрограммы, которая занимает m команд и задана в условных адресах 1, 2, ..., m. Мы можем просто включить подпрограмму в основную программу, сделав ее первую команду $(k+1)$-й командой основной программы, вторую — $(k+2)$-й и т. д. При этом должны быть соответственно изменены, а именно, увеличены на величину k, те адреса «этой» подпрограммы, в которых указаны номера команд (а не чисел) — адреса команд передач управления (условных и безусловных) и команд переадресации. Такое включение подпрограммы в основную программу требует, во-первых, преобразования команд подпрограммы и, во-вторых, при необходимости повторных вычислений по подпрограмме приводит к повторению всех команд подпрограммы, соответственно удлиняя программу.

Для тех же целей можно поступить иначе. Разместим команды подпрограммы в фиксированных (свободных) ячейках *ЗУ* от $n+1$ до $n+m$ и забронируем еще одну ячейку с номером $n+m+1$, в которую поместим так называемую команду возврата к основной программе. В ячейке $k+2$ основной про-

граммы поместим команду безусловного перехода к подпрограмме

БП			$n+1$

а в $(k+1)$-й ячейке — команду, подготавливающую $(n+m+1)$-ю команду для надлежащего возврата к $(k+3)$-й команде основной программы, например команду

⊕	γ		$n+m+1$

где γ — номер ячейки, содержащей код

БП			$k+3$

Приведем пример программы с неоднократно повторяемыми обращениями к подпрограммам.

Пусть по группе команд Q производятся вычисления некоторой функции от аргумента, содержащегося в ячейке α, и при решении задачи встречаются вычисления значений этой функции от двух различных аргументов, помещаемых в процессе вычислений в ячейки β и γ.

Рассмотрим детально блок-схему такой программы с выделением группы команд Q в подпрограмму.

Обозначим группу команд, производящих вычисления до первого обращения к подпрограмме Q, через A, группу команд, производящих вычисления между первым и вторым обращением к подпрограмме, через B и последующие вычисления после второго обращения к подпрограмме через C: схема вычислений, таким образом, имеет вид

$$AQBQC. \tag{36}$$

Пусть каждая из групп состоит:

A — из p команд;
B — из r команд;
C — из s команд;
Q — из m команд.

При выполнении вычислений без выделения подпрограммы Q программа будет состоять из $p+r+s+2m$ команд. (Возможные условные и безусловные передачи управления, не связанные с переходом на подпрограмму, которые могут встречаться

внутри любой из указанных групп команд, в данном рассмотрении мы оставляем в стороне.)

Для автоматического выполнения вычислений на машине с выделением подпрограммы Q требуются дополнительные команды:

1) команды, выполняющие перед обращением к подпрограмме засылку аргумента, находящегося соответственно в ячейках β и γ, в ячейку α;

2) команды, выполняющие переход к подпрограмме;

3) команды, выполняющие возврат от подпрограммы к дальнейшим вычислениям по программе (к группам команд B — после первого обращения к подпрограмме и к C — после второго).

Выделяя особо эти команды, обозначим через a_1 и b_1 команды, осуществляющие засылку аргумента из ячеек β и γ в специально отведенную для аргумента подпрограммы ячейку α; a_2 и b_2 — команды, обеспечивающие надлежащий возврат из подпрограммы к основной программе; a_3 и b_3 — команды безусловного перехода к подпрограмме.

Кроме того, в подпрограмму должна быть введена переменная команда q — команда «возврата» к основной программе, подготавливаемая командами a_2 и b_2.

Схема программы будет иметь вид

$$Aa_1a_2a_3QqBb_1b_2b_3QqC. \tag{37}$$

Число команд в программе, согласно (37), будет равно $p + r + s + m + 7$.

Дополнительно для подготовки команды возврата понадобятся две константы, коды которых соответствуют командам безусловного перехода к первой команде группы B или C; мы поместим их в ячейки γ_1 и γ_2. На блок-схеме (рис. 14) стрелками указан порядок выполнения команд; взаимодействие соответствующих команд и команды «возврата» от подпрограммы к основной программе указано пунктиром. Выделенные команды в схеме приведены в развернутом виде в одном из возможных вариантов.

В ячейках γ_1 и γ_2 помещены коды:

γ_1	*БП*			$p+4$
γ_2	*БП*			$p+r+7$

В принятом размещении команд в памяти подпрограмма Q помещается в ячейках $N + 1$, $N + 2$, ..., $N + m$, где $N \geqslant p + r + s + 6$.

Для удобства можно все подпрограммы заканчивать командой безусловного перехода к одной и той же специально выделенной в оперативной памяти («регистре возврата») ячейке и при переходе на подпрограммы предварительно засылать в ее IIIA номер команды основной программы, которому должно

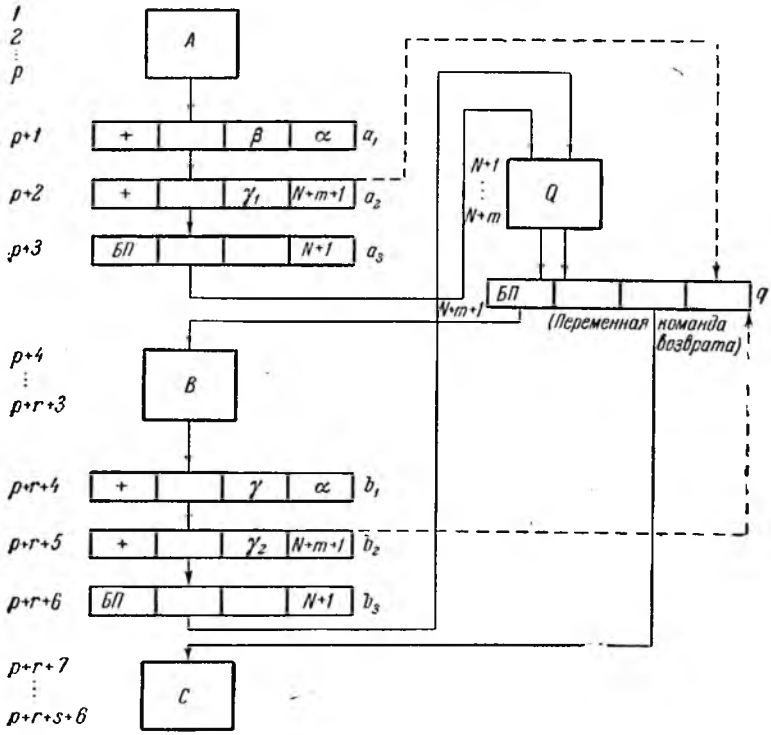


Рис. 14.

быть передано управление после выполнения подпрограммы. В регистре возврата при этом помещается код операции безусловного перехода.

Среди элементарных операций, выполняемых машиной, в связи с изложенным удобно иметь следующие операции:

1) безусловный переход на подпрограммы *БПП*

БПП	α	k_1	k_2

2) безусловный переход по регистру возврата *БПРВ*

БПРВ			

Первая из этих операций выполняет безусловный переход к операции, номер которой k_1 указан в II*A* (номер первой команды подпрограммы), и засылает в регистр возврата номер команды k_2, указанный в III*A* (номер основной программы), которой

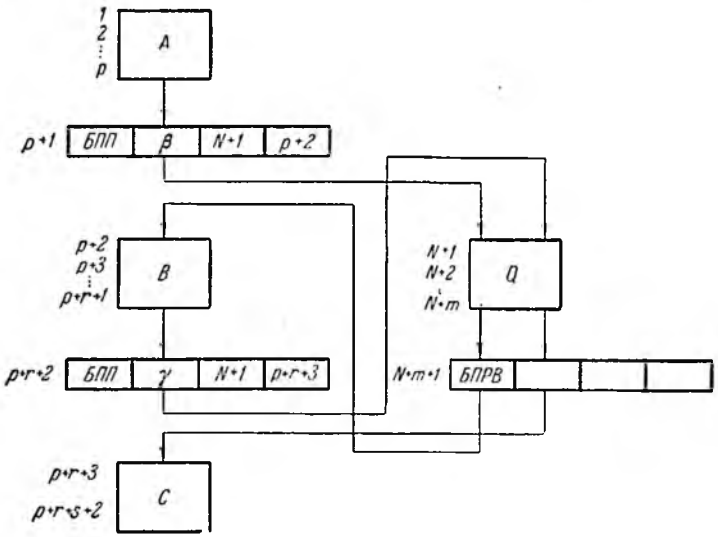


Рис. 15.

должно быть передано управление после выполнения подпрограммы. Кроме того, при выполнении операции *БПП* можно потребовать использования свободного адреса I: содержимое ячейки, номер которой указан в I*A*, пересылается в «стандартную» ячейку (например, № 1), предназначаемую для хранения «аргумента» всех подпрограмм. Команда *БПРВ* передает управление команде, номер которой содержится в данный момент на *РВ* (рис. 15). Число команд в программе равно $p + r + s +$, $+ m + 3$.

Среди элементарных операций, выполняемых машиной, в связи с передачами на подпрограммы удобно также иметь команды:

3) условный переход на подпрограмму по «признаку»

УППП	α	k_1	k_2

Данная операция (а) выполняет передачу управления команде, номер которой указан в IIA, если в предыдущей команде был выработан «признак», (б) засылает в регистр возврата номер команды, указанной в $IIIA$ (номер команды, которой должно быть передано управление после выполнения подпрограммы), (в) передает управление следующей по номеру команде, если такой признак не был выработан; кроме того, содержимое ячейки α при передаче управления по IIA пересылается в стандартную ячейку, предназначенную для хранения аргументов подпрограммы;

4) условный переход на подпрограмму по числу (*УППЧ*)

УППЧ	α	k_1	k_2

Содержание команды *УППЧ* следующее: если в ячейке, номер которой α указан в IA, содержится положительное число, управление передается команде, номер которой k_1 указан в IIA (первой команде подпрограммы), содержимое $IIIA$ засылается в регистр возврата (для надлежащего возврата из подпрограммы), а если это число отрицательное, управление передается следующей по номеру команде программы. Если в процессе выполнения подпрограммы встречается в свою очередь переход на другую подпрограмму, то для таких многоступенчатых переходов потребовался бы соответствующий «многоступенчатый» регистр возврата. Во избежание возникающих при этом технических трудностей можно ограничиться единым регистром возврата для всех стандартных подпрограмм и для подпрограмм, не содержащих в себе переходов на другие подпрограммы (подподпрограммы), а для подпрограмм с подподпрограммами пользоваться приведенной в примере на стр. 143 схемой (рис. 14).

Однако, как мы видели, в местах перехода на промежуточные подпрограммы в таком случае в программу включаются три специальные команды, для каждой из которых, кроме того, требуется константа.

В связи с этим в наборе элементарных операций, выполняемых машиной, желательно иметь команду вида

5) переход на подпрограмму (*ПП*)

n	*ПП*	α	k_1	k_2

Здесь команда *ПП*, находящаяся в ячейке n, передает управление команде, номер которой k_1 указан в IIA; засылает в $IIIA$

команды k_2, указанной в $IIIA$, номер $n+1$ следующей команды. Кроме того, содержимое ячейки, номер которой указан в IA, засылается в стандартную ячейку, предназначаемую для хранения аргументов стандартной программы. При этом ячейка с номером k_1 содержит первую команду подпрограммы и ячейка с номером k_2 содержит код команды безусловного перехода и является последней командой подпрограммы.

Заметим здесь же, что в связи с обращением к подпрограммам при программировании следует учитывать заполнение *ЗУ* для основной программы. Например, некоторые ячейки *ЗУ* должны быть специально подготовлены для работы подпрограммы (пересылка аргументов в соответствующие ячейки и пр.); некоторые ячейки могут быть использованы в подпрограмме как оперативные, а значит, в них не должны храниться необходимые для дальнейшего величины в основной программе, и т. п.

Обычно для этих целей во всех подпрограммах одни и те же ячейки используются как оперативные, в одних и тех же ячейках помещаются аргументы, и т. п.

Команды подпрограммы, которые в процессе счета изменяются, должны быть восстановлены для возможности повторного их использования. Поэтому подпрограммы, включающие переменные команды, начинаются или заканчиваются командами восстановления.

§ 6. Формирование содержания запоминающего устройства

В предыдущих параграфах при составлении программ содержание *ЗУ* определялось лишь из условия обеспечения цикличности процессов счета.

Вместе с тем распределение чисел и команд программы по ячейкам *ЗУ* желательно производить с учетом обеспечения возможно более быстрой и надежной работы программы, включая ввод программы в машину. При этом следует учитывать взаимосвязь между отдельными частями *ЗУ*.

Поскольку в построенных до настоящего времени машинах ввод программ — исходных чисел и команд — не является быстродействующим, большого внимания заслуживает работа по сокращению объема программ.

В некоторых машинах имеется возможность ввода соответствующих частей программ в постоянную часть внутреннего *ЗУ* и во внешнее *ЗУ* вне рабочего времени машины. Ячейки постоянной памяти, существующие в виде быстронаборных блоков, а также блоки внешнего *ЗУ* в виде бобин с магнитной лентой и т. п. подготавливаются к каждой программе заранее (в отдельном помещении), до постановки программы на машину. Обычно в этих частях *ЗУ* имеются специально отведенные участки, в ко-

торых хранятся типовые константы и стандартные подпрограммы, и по необходимости в *ЗУ* вводится тот или иной блок.

Наличие таких стандартных тщательно проверенных программ, помимо ускорения процесса ввода и упрощения программирования, повышает надежность работы машины.

Библиотеку подпрограмм в процессе практического использования машины целесообразно совершенствовать и пополнять.

При распределении стандартных программ между внешним и внутренним *ЗУ* учитывается, с одной стороны, тот факт, что обращение к внешнему *ЗУ* является операцией сравнительно медленной, а следовательно, использование подпрограмм, расположенных в нем, удлиняет время расчетов, а с другой, — то обстоятельство, что внешнее *ЗУ* технически более просто выполнимо. Как уже отмечалось во введении, для выполнения расчетов программа или та ее часть, в которую должны входить команды, должна находиться во внутреннем (быстродействующем) *ЗУ*. Поэтому стандартные подпрограммы, содержащие переменные команды, удобнее помещать во внешнем *ЗУ*. При выполнении расчетов по таким подпрограммам последние будут помещены в оперативное *ЗУ*, что проще всего сделать с помощью операции «считка», выполняющей передачу соответствующей подпрограммы из внешнего в оперативное *ЗУ*.

Оперативные части даже небольших программ, полностью размещающихся во внутреннем *ЗУ* (если таковые имеются), следует предварительно вводить во внешнее *ЗУ*.

Действительно, при работе на *ЦАМ* возможны различного рода сбои (выключение тока в цепи и пр.), в результате которых содержимое оперативной части *ЗУ* может искажаться. Наличие исходного вида программы во внешнем *ЗУ* позволяет с помощью операции «считка» восстанавливать программу в оперативном *ЗУ* без обращения к вводным устройствам. Точно так же при вводе и отладке программы в последней могут быть обнаружены ошибки в тот момент, когда некоторые ее переменные команды изменили свой вид. «Прочитав» программу из внешнего *ЗУ* и внеся в нее надлежащие исправления, следует исправленную программу «записать» во внешнее устройство. Это облегчит дальнейшую корректировку программы и работу по ней.

Вместе с тем для небольших программ, размещающихся в оперативной памяти, нет надобности в обращении в процессе работы к внешнему *ЗУ:* программа переносится полностью во внутреннее *ЗУ*, после чего начинаются по ней расчеты.

Если для размещения программы объем внутренней части *ЗУ* оказывается недостаточным, то обращение к внешнему *ЗУ* в процессе работы программы становится неизбежным. В таком случае выгоднее помещать во внешнюю память те части

программы, которые в процессе расчетов используются сравнительно редко, ибо, как мы уже отмечали, операции, связанные с обращением к внешнему *ЗУ*, являются сравнительно медленными. При этом массив перфокарт или перфолент, представляющий собой закодированную программу, разбивается на соответствующие группы кодов, которые вводятся одна за другой в оперативное *ЗУ*, а из него — во внешнее *ЗУ*, на специально отведенные места. Программой предусматривается в нужные моменты вызов («считка») в оперативное *ЗУ* тех групп кодов, которые нужны для работы.

Разбивка программы на такие группы сопряжена с известными трудностями.

Действительно, нельзя выполнить передачу управления команде, не содержащейся в данный момент в оперативном *ЗУ*. Поэтому каждая группа команд программы, размещаемая во внутреннем *ЗУ*, должна обладать следующим свойством: каждой команде передачи управления на команду какой-либо другой группы должна предшествовать «считка» команд этой группы. При этом учитывается, что после операции «считка» ячейки, в которые производилась «считка», уже содержат вновь прочитанные коды.

Использование внешней памяти желательно также и для такой программы, которая изменяется в процессе работы и в то же время многократно используется, например для программы циклических процессов, зависящих от параметров. Наличие исходного вида такой программы во внешней памяти позволяет восстанавливать программу для повторных вычислений с помощью операции «считка». Объем программы при этом может сократиться, так как высвобождаются ячейки для хранения констант, необходимых для этих команд восстановления. Время, занимаемое на машине непосредственными расчетами, может при этом несколько увеличиться (операция «считка» может занять больше времени, чем выполнение команд восстановления), однако получается экономия за счет сокращения времени на ввод программы, так как сокращается объем вводимой информации о программе.

Формирование содержания *ЗУ* зависит существенным образом от того, какое место занимает данная программа в общей схеме решения задачи. Для различных видов программ содержание *ЗУ* формируется различно.

Для программы, которая является вспомогательной при решении данной задачи, т. е. подпрограммой к основной программе, при формировании содержания *ЗУ* необходимо учитывать, что некоторые данные для нее подготавливаются основной программой и, следовательно, содержатся в оперативных ячейках *ЗУ*.

Если подпрограмма используется многократно, то необходимо предусмотреть восстановление содержания *ЗУ* к исходному виду, включая и переменные команды.

Здесь мы ограничимся рассмотрением формирования содержания *ЗУ* лишь для некоторых из приведенных выше программ.

Пример 1. Программа вычисления таблицы квадратов членов арифметической прогрессии с выводом на печать (см. § 3, программа 9). Содержание *ЗУ* для программы 9 состоит из двух постоянных чисел c и $a + cN$, которые расположены в ячейках *ЗУ* α_2 и α_3; из двух рабочих ячеек *ЗУ* α_1 и ω, в которых находятся члены арифметической прогрессии и их квадраты, и из команд, которые можно разместить в любом месте *ЗУ*.

Если первый член арифметической прогрессии a находится в ячейке *ЗУ* β, то в начале программы должна быть команда переноса a из ячейки β в рабочую ячейку α_1:

$k-1$	+	β	—	α_1

Пример 2. Извлечение квадратного корня (см. § 3, программа 10). Программа вычисления квадратного корня является обычно стандартной программой и помещается в пассивных ячейках *ЗУ*. Числа y_0 и x должны быть в рабочих ячейках *ЗУ*. Впрочем, например, для машины с фиксированной запятой y_0 можно выбрать не зависящим от x и, следовательно, поместить в пассивной ячейке *ЗУ* β, предусмотрев в программе команду пересылки y_0 из этой ячейки в нужную рабочую ячейку *ЗУ*.

Пример 3. Вычисление значений многочлена (см. § 4, программа 21). Естественно предположить, что коэффициенты многочлена и величина аргумента x получены в результате счета по основной программе и, следовательно, располагаются в рабочих ячейках *ЗУ*. Нуль в ячейке β_3, как уже упоминалось выше, лучше всего образовать, введя в начало программы команду очистки ячейки β_3

$N-1$	+	—	—	β_3

Числа 1 (β_4) и 1 во II адресе (β_1), как правило, имеются во всех машинах в пассивной памяти. Число n единиц во II адресе и число n — постоянные для данной программы — могут помещаться в пассивных ячейках *ЗУ* либо формироваться основной программой в активных ячейках *ЗУ*. Команды программы расположены в активных ячейках *ЗУ*, так как программа имеет переменные команды.

Если программа помещается во внешнюю память, то для выполнения расчетов по ней необходимы соответствующие команды, обеспечивающие перенос программы во внутреннее *ЗУ* («считку»), в связи с чем нумерация команд, вообще говоря, может измениться. В соответствии с новой нумерацией команд во внутреннем *ЗУ* должны быть изменены команды, содержащие в своих адресах номера команд. Такой командой в программе 21 являются 4-я и 6-я команды, I и III адреса которых должны быть изменены в соответствии с новым номером переменной команды.

Пример 4. Формирование содержания *ЗУ* для программы 12 (§ 3). Программа вычисления $\sin x$ также является типичной стандартной программой. Значение аргумента x подготавливается основной программой, например, в активной ячейке *ЗУ* α_1. Числа x в ячейке α_2 и $-x^2$ в ячейке α_4 должны формироваться в данной программе, т. е. программа 12 должна быть дополнена командами

$k-2$	+	α_1	—	α_2
$k-1$	×	α_1	α_1	α_4
k	—	—	α_4	α_4

Исходное размещение чисел в ячейках *ЗУ* упрощается:

α_1	x
α_2	—
α_3	1
α_4	—
α_5	1
α_6	ε

Первоначальное заполнение ячеек α_2 и α_4 безразлично.

Пример 5. Объединение программ вычисления $\sin x$ и $\cos x$ (§ 3, см. программу 12 и упражнение 10).

Программы вычисления $\sin x$ и $\cos x$ отличаются друг от друга лишь начальным содержанием числовых ячеек в *ЗУ*. Поэтому счет $\sin x$ и $\cos x$ можно вести по одной и той же программе, нужно только предусмотреть соответствующее формирование числовых ячеек в *ЗУ*. Сравним начальное содержание числовых ячеек в *ЗУ* для программ $\sin x$ и $\cos x$:

Исходное содержание чисел в *ЗУ* для программы $\sin x$

α_0	x	
α_1	x	u_n
α_2	x	s_n
α_3	1	c_n
α_4	$-x^2$	
α_5	1	
α_6	ε	

Исходное содержание чисел в *ЗУ* для программы $\cos x$

α_0	x	
α_1	1	u_n
α_2	1	s_n
α_3	0	c_n
α_4	$-x^2$	
α_5	1	
α_6	ε	

Программы $\sin x$ и $\cos x$ отличаются исходным заполнением ячеек α_1, α_2 и α_3. Формирование чисел x и 1 для программы $\sin x$ выполняется следующими командами:

+	α_0	—	α_1
+	α_0	—	α_2
+	α_5	—	α_3

Для программы $\cos x$ числа в ячейках α_1, α_2 и α_3 образуются командами

+	α_5	—	α_1
+	α_5	—	α_2
+	—	—	α_3

Теперь можно составить единую программу для счета $\sin x$ и $\cos x$:

Программа 27

Числа

α_0	x	
α_1	—	u_n
α_2	—	s_n
α_3	—	c_n
α_4	—	$-x^2$
α_5	1	
α_6	ε	
α_7	—	

Команды

$N+1$	+	α_0	—	α_1	начало счета $\sin x$
$N+2$	+	α_0	—	α_2	
$N+3$	+	α_5	—	α_3	
$N+4$	$\lvert\leqslant\rvert$	—	—	$N+8$	
$N+5$	+	α_5	—	α_1	начало счета $\cos x$
$N+6$	+	α_5	—	α_2	
$N+7$	+	—	—	α_3	
$N+8$	$\times$	α_0	α_0	α_4	
$N+9$	—	—	α_4	α_4	
$N+10$	+	α_3	α_5	α_7	
$N+11$	+	α_7	α_5	α_3	
$N+12$	$\times$	α_7	α_3	α_7	
$N+13$	$\times$	α_1	α_4	α_1	
$N+14$	:	α_1	α_7	α_1	
$N+15$	+	α_2	α_1	α_2	
$N+16$	$\lvert\leqslant\rvert$	α_6	α_1	$N+10$	
$N+17$	*ост.*				

Выводы

1. При формировании содержания *ЗУ* необходимо учитывать наличие различных видов памяти в машине и место данной программы в общей схеме решения задачи.

2. Для многократного использования программы в процессе вычислений необходимо предусмотреть восстановление содержания *ЗУ*, включая и переменные команды.

3. *Восстановление* содержания *ЗУ* может быть осуществлено двумя способами: 1) перенесением исходного содержания *ЗУ* из одного вида памяти в другой, 2) восстановлением содержания *ЗУ* при помощи элементарных операций (очистка ячеек, сложение, вычитание и т. п.).

4. При использовании подпрограмм необходимое содержание *ЗУ* должно приводиться в соответствие с основной программой.

5. При формировании содержания *ЗУ* желательно максимально использовать пассивную память, если она существует.

6. При разбивке программы на части, находящиеся во внешней памяти, необходимо надлежащим образом согласовывать передачи управления из одной группы команд в другую.

Упражнения

1. Проанализировать формирование содержания *ЗУ* для программ:
а) деления (§ 2, программа 8),
б) вычисления $\operatorname{tg} x$ (§ 3, программа 13),
в) вычисления многочлена (§ 4, программы 21, 22_1, 22).

§ 7. Групповые операции

При решении задач на *ЦАМ* огромное значение имеют вопросы контроля и в первую очередь вопросы контроля правильности программ. Если ошибка в коде числа обусловливает неточность в решении некоторых вариантов данной задачи, то ошибка в одном из разрядов кода какой-либо команды может привести в негодность всю программу. Вместе с тем опыт работы на *ЦАМ* показывает, что даже тщательно проверенные программы, как правило, могут содержать ошибки, не выявленные в процессе их составления, кодировки и ввода в память машины и обнаруживаемые лишь при проведении по ним специально подготовленных контрольных расчетов.

Если для решения одной и той же задачи согласно избранному методу могут быть предложены две различные программы, то с точки зрения удобства ввода в память и быстроты отладки предпочтение должно быть отдано той из программ, которая занимает меньшее число ячеек *ЗУ*, так сказать, содержит информацию о задаче в более компактном виде. Предпочтение такой программе должно быть отдано даже в том случае, когда непосредственные вычисления по ней потребуют несколько большей затраты времени. С другой стороны, более предпочтительной может оказаться программа с большим числом используемых ячеек *ЗУ*, если ее команды не изменяются в процессе работы, что более удобно в смысле ввода и контроля.

В связи с этим большого внимания заслуживает выбор наиболее удобного набора элементарных операций, выполняемых машиной, и их рационального кодирования в виде команд.

На примерах циклических программ, зависящих от параметров, мы видели, что экономия в командах программы может быть достигнута за счет включения в набор элементарных операций машины, операции переадресации и соответствующего упорядоченного размещения величин в *ЗУ*.

В общем случае циклические процессы, как правило, являются сложными (многократными) и содержат параметры, от которых может зависеть ряд величин, встречающихся в цикле. В связи с этим общее число команд обслуживающего назначения, не относящихся к командам вычислительного характера, значительно возрастает. Приемы программирования, позволяющие иногда сокращать такого рода программы, были приведены в § 3. Можно пойти дальше и поставить вопрос о сокращении циклических программ, но уже не за счет удачного программирования, а за счет введения в машину некоторых специальных операций, называемых *групповыми*, при помощи усложнений в ее схеме.

Заметим здесь же, что введение групповых операций, как правило, расширяет возможности построения неизменяемых программ.

Поскольку программы для параметрических циклических процессов могут быть построены таким образом, чтобы все переменные адреса при переходе от цикла к циклу изменялись на одну и ту же величину p — шаг переадресации, напрашивается идея проводить изменения адресов вне устройства памяти машины, путем включения соответствующего приспособления в устройство управления.

Назовем параметр, характеризующий номер выполняемого цикла, индексом (или константой) модификации.

Необходимой информацией для цикла, зависящего от параметра, являются:

1) перечень операций (команд) цикла в исходном виде;

2) указание переменных адресов (признаков изменяемости адресов);

3) информация об индексе модификации (иными словами, изменяемости).

Информацию об индексе модификации составляют следующие четыре величины:

а) α — исходное значение индекса;

б) p — шаг изменения *) индекса;

*) Не исключается, что шаг p является в свою очередь переменной величиной, зависящей от номера цикла, например вычисляемой по некоторому закону внутри цикла.

в) α_k — конечное значение индекса;

г) α_t — текущее значение индекса.

Величина α_k — конечного значения индекса модификации — используется для выяснения окончания циклического процесса; величина α_t — текущего значения индекса — необходима для реализации многократных циклических процессов, зависящих от внешних и внутренних параметров.

Переменные адреса $a_{(t)}$ должны формироваться по закону

$$\begin{aligned} \alpha_t &= \alpha_{t-1} + p; \\ a_{(t)} &= a_{(0)} + \alpha_t, \end{aligned} \tag{38}$$

где a_0 — исходное заполнение переменного адреса.

Информация 1 может быть задана, как обычно, с помощью соответствующих команд счета. Для выделения переменных адресов, т. е. для выяснения необходимости формирования адреса согласно (38), в каждом из адресов следует выделить специальный разряд, код «единица» в котором будет признаком зависимости адреса от параметра, — «признак групповой операции» (адрес подлежит изменению лишь при наличии признака групповой операции). Указанный разряд является дополнительным к разрядам, на которых кодируются адреса ячеек *ЗУ*.

Для модификации переменных адресов согласно (38) (при наличии в них признака групповой операции) в машине должны предусматриваться специальные устройства. Это может быть реализовано, например, следующим образом.

Для хранения текущего значения α_t индекса модификации предусматривается специальный регистр модификации адресов. Выполнение операции

$$a_{(t)} = a_0 + \alpha_t$$

может осуществляться либо на основном арифметическом устройстве, либо, в силу простоты этой операции (все коды — размера одного адреса), на специальном сумматоре адреса. На этом же устройстве может осуществляться операция образования последующего значения индекса модификации:

$$\alpha_t = \alpha_{t-1} + p,$$

с засылкой получаемого результата в регистр модификации.

Для хранения α_k — конечного значения индекса модификации — предусматривается специальный регистр циклов; кроме того, в схему машины включается устройство совпадения, с помощью которого для выяснения окончания работы цикла сравниваются величины α_t и α_k на совпадение (содержимое регистра модификации адресов и регистра циклов).

Величины α, p, α_k, характеризующие параметр, поместим в адресах одной ячейки

β		α	p	α_k

Пополним набор элементарных операций машины следующими операциями:

1) операция *НГО* — начало групповой операции

НГО	γ	δ	β

2) операция *КГО* — конец групповой операции

КГО	β	k_1	k_2

Пусть по операции *НГО* выполняется следующее:

а) содержимое регистра модификации адресов, хранящее текущее значение индекса модификации на предыдущем цикле, переносится в ячейку γ (в один из ее адресов, например первый); если внешний (по отношению к данному) цикл не зависит от параметра, вместо γ в $1A$ команды *НГО* ставится нуль; если же зависимость от параметра имеет место, то ячейка γ будет сохранять необходимую информацию, которая аналогичным способом будет использована при повторном обращении к внешнему циклу;

б) содержимое ячейки δ (ее первого адреса) пересылается на регистр модификации адресов ($1A$ δ хранит величину α_l);

в) содержимое III адреса ячейки β засылается на регистр циклов (конечное значение индекса α_k).

Пусть по операции *КГО*:

а) к содержимому регистра модификации адресов (α_{l-1}) прибавляется содержимое второго адреса ячейки β — шага переадресации индекса:

$$\alpha_l = \alpha_{l-1} + p;$$

б) сравнивается содержимое регистра циклов и регистра модификации адресов; при их совпадении передается управление команде, адрес которой k_1 указан во II адресе команды *КГО*,

в противном случае — команде k_2, адрес которой указан в ПА команды *КГО*.

Все переменные адреса отмечаются признаком групповой операции, т. е. в отведенном под этот признак разряде кодируется единица.

При выполнении команд, адреса которых имеют признак групповой операции, последние модифицируются, т. е. увеличиваются на величину, содержащуюся в регистре модификации адресов (вид команд в памяти при этом сохраняется неизменным).

Таким образом, при наличии групповых операций приведенного вида программы становятся более компактными и могут быть полностью неизменяемыми. Рассмотрим примеры.

Пример 1. Вычислить сумму

$$s=\sum_{j=1}^{m}\sum_{i=1}^{n}\frac{a_ix_j+b_iy_j}{c_ix_j+d_iy_j}.$$

Расположим данные, зависящие от индексов i и j, в соответствующие последовательности ячеек. Тогда вычисление суммы s может быть сведено к двойному циклическому процессу: вычисление каждого из слагаемых внутренней суммы при фиксированном j можно выполнить по циклической программе (внутренний цикл) — на i-м цикле вычисляется i-е слагаемое суммы и прибавляется к сумме предыдущих слагаемых; программа внутреннего цикла пополняется необходимыми командами переадресации. Зависящие от j величины x_j и y_j перед каждым обращением к внутреннему циклу будем засылать в стандартные ячейки ω_1 и ω_2. Дальнейшее построение программы не требует пояснений.

В данном примере перед каждым обращением к программе внутреннего цикла (команды $N+5 \div N+17$) требуется восстановление команд, зависящих от параметра i (команды $N+5$, $N+6$, $N+8$, $N+9$): последовательности ячеек с номерами, зависящими от номера цикла параметра i, используемые во внутреннем цикле, фиксированы

$$\alpha+1, \ldots, \alpha+n;$$
$$\beta+1, \ldots, \beta+n;$$
$$\gamma+1, \ldots, \gamma+n;$$
$$\delta+1, \ldots, \delta+n.$$

В связи с этим программа дополнена командами восстановления $N+18 \div N+21$.

Программа 28

Числа

$\alpha+1$	a_1
$\alpha+2$	a_2
⋮	
$\alpha+n$	a_n
$\beta+1$	b_1
$\beta+2$	b_2
⋮	
$\beta+n$	b_n
$\gamma+1$	c_1
$\gamma+2$	c_2
⋮	
$\gamma+n$	c_n
$\delta+1$	d_1
$\delta+2$	d_2
⋮	
$\delta+n$	d_n
$\tau+1$	x_1
⋮	
$\tau+m$	x_m
$\rho+1$	y_1
⋮	

Команды

$N+1$	$+$		$\tau+1$	ω_1
$N+2$	$+$		$\rho+1$	ω_2
$N+3$	$\oplus$	$N+1$	Δ_1	$N+1$
$N+4$	$\oplus$	$N+2$	Δ_1	$N+2$
$N+5$	$\times$	ω_1	$\alpha+1$	ω_3
$N+6$	$\times$	ω_2	$\beta+1$	ω_4
$N+7$	$+$	ω_3	ω_4	ω_3
$N+8$	$\times$	ω_1	$\gamma+1$	ω_4
$N+9$	$\times$	ω_2	$\delta+1$	ω_5
$N+10$	$+$	ω_4	ω_5	ω_4
$N+11$	:	ω_3	ω_4	ω_3
$N+12$	$+$	ω_3	ω	ω
$N+13$	$\oplus$	$N+5$	Δ_1	$N+5$
$N+14$	$\oplus$	$N+6$	Δ_1	$N+6$
$N+15$	$\oplus$	$N+8$	Δ_1	$N+8$
$N+16$	$\oplus$	$N+9$	Δ_1	$N+9$
$N+17$	$\leqslant$	$N+5$	Δ_4	$N+5$
$N+18$	$\ominus$	$N+5$	Δ_3	$N+5$
$N+19$	$\ominus$	$N+6$	Δ_3	$N+6$
$N+20$	$\ominus$	$N+8$	Δ_3	$N+8$
$N+21$	$\ominus$	$N+9$	Δ_3	$N+9$

$\rho+m$	y_m
ω_1	x_j
ω_2	y_l
ω	0 s

Δ_1			1	
Δ_2	+		$\tau+m$	ω_1
Δ_3			n	
Δ_4	×	ω_1	$\alpha+n$	ω_3

$N+22$	$<$	$N+1$	Δ_2	$N+1$
$N+23$	*ост.*			

Используя групповые операции, получим программу.

Программа 29

Числа

$\alpha+1$	a_1
⋮	
$\alpha+n$	a_n
$\beta+1$	b_1
⋮	
$\beta+n$	b_n
$\gamma+1$	c_1
⋮	
$\gamma+n$	c_n
$\delta+1$	d_1
⋮	
$\delta+n$	d_n

Команды

$N+1$	*ИГО*	—	Δ_2	Δ_2
$N+2$	+	—	$1\|\tau+1$	ω_1
$N+3$	+	—	$1\|\rho+1$	ω_2
$N+4$	*КГО*	Δ_2	$N+15$	$N+5$
$N+5$	*ИГО*	Δ_2	Δ_1	Δ_1
$N+6$	×	$1\|\alpha+1$	ω_1	ω_3
$N+7$	×	$1\|\beta+1$	ω_2	ω_4
$N+8$	+	ω_3	ω_4	ω_3
$N+9$	×	ω_1	$1\|\gamma+1$	ω_4
$N+10$	×	ω_2	$1\|\delta+1$	ω_5
$N+11$	+	ω_4	ω_5	ω_4
$N+12$	:	ω_3	ω_4	ω_3

Числа

$\tau+1$	x_1			
⋮				
$\tau+m$	x_m			
$\rho+1$	y_1			
⋮				
$\rho+m$	y_m			
ω_1		x_j		
ω_2		y_j		
S	0			
Δ_1	—	—	1	n
Δ_2	—	—	1	m

Команды

$N+13$	+	ω_3	S	S
$N+14$	*КГО*	Δ_1	$N+1$	$N+6$
$N+15$	*ост.*			

Здесь и в дальнейшем признак групповой операции в адресе указывается наличием единицы за вертикальной чертой.

Проследим за работой программы. По команде $N+1$ в регистр модификации адресов засылается содержимое первого адреса ячейки Δ_2, которая вначале в этом адресе содержит пуль. Кроме того, в регистр циклов засылается содержимое третьего адреса ячейки Δ_2 — величина m.

Поскольку при первом выполнении команд $N+2$, $N+3$ регистр модификации адресов содержит 0, эти команды будут выполнены в том виде, в котором они записаны в программе, несмотря на наличие признаков групповой операции во вторых адресах этих команд.

По команде $N+4$ к содержимому регистра модификации адресов будет прибавлено содержимое второго адреса ячейки Δ_2 — шаг переадресации, — равное единице, т. е. регистр модификации адресов будет содержать уже единицу.

Поскольку содержимое регистра модификации и регистра циклов не совпадает, команда $N+4$ на этот раз передаст управление команде $N+5$.

По команде $N+5$ содержимое регистра модификации (единица) будет переслано в $1A$ ячейки Δ_2; содержимые I и III адресов ячейки Δ_1 (пуль и число n) будут пересланы на регистр

модификации и *регистр* циклов соответственно. Команды $N+6 \div N+13$ при этом снова будут выполнены в том виде, как они записаны в программе.

По команде $N+14$ содержимое регистра модификации увеличится на единицу (на содержимое IIA ячейки Δ_1), и так как этот регистр и регистр циклов будут содержать различные коды, управление будет передано команде $N+6$.

Теперь при выполнении команд $N+6 \div N+13$ произойдет изменение адресов, имеющих признак групповой операции (первые адреса команд $N+6$, $N+7$ и вторые адреса команд $N+9$, $N+10$), а именно эти адреса будут увеличены на содержимое регистра модификации, равное в этом случае единице.

Команда $N+14$ снова увеличит содержимое регистра модификации на единицу и передаст управление команде $N+6$. Теперь адреса переменных команд будут увеличены на 2 и так до тех пор, пока по команде $N+14$ в результате прибавления единицы (содержимого IIA ячейки Δ_1) в регистре модификации не появится число n (заметим, что это означает, что в предыдущем цикле переменные адреса были увеличены на число $n-1$, т. е., например, I адрес команды $N+6$ был равен

$$(\alpha+1)+(n-1)=\alpha+n,$$

т. е. что выполнен последний цикл программы); теперь содержимое регистров модификации и циклов равны и управление передается команде $N+1$.

По команде $N+1$ в регистр модификации будет прислано содержимое IA ячейки Δ_2, теперь равное единице, и в регистр циклов — число m (содержимое $IIIA$ ячейки Δ_2). В результате команды $N+2$ и $N+3$ выполнят засылку x_2 и y_2 (ячейки $\tau+2$ и $\rho+2$) в стандартные ячейки ω_1 и ω_2. По команде $N+4$ содержимое регистра модификации увеличится на единицу (и станет равным 2) и управление будет передано к внутреннему циклу на команду $N+5$. Так будет продолжаться до тех пор, пока в IA ячейки Λ_2 (в результате выполнения команды $N+4$) не окажется число $m-1$. Это число по команде $N+1$ будет передано на регистр модификации и после выполнения команд $N+2$, $N+3$ (со вторыми адресами, равными $\tau+m$, $\rho+m$) команда $N+4$ увеличит содержимое регистра модификации на единицу и передаст управление на команду $N+15$ — останов (поскольку регистры модификации и циклов будут содержать равные коды).

Пример 2. Вычислить произведение

$$\pi=\sum_{i=1}^{n}\frac{a_ix+b_iy}{c_ix+d_iy}\sum_{i=n+1}^{2n}\frac{a_ix+b_iy}{c_ix+d_iy}\cdots\sum_{i=(m-1)n+1}^{mn}\frac{a_ix+b_iy}{c_ix+d_iy}.$$

Расположим, как и в предыдущем примере, данные, зависящие от параметра i, в последовательности ячеек. Вычисление также сводится к двойному циклическому процессу: во внутреннем цикле вычисляется следующий член произведения; во внешнем — полученный результат умножается на произведение предыдущих сомножителей.

В отличие от предыдущего примера здесь команды внутреннего цикла при каждом последующем обращении к нему не восстанавливаются: последовательности ячеек с номерами, зависящими от номера цикла параметра i, используемые во внутреннем цикле, являются одна продолжением другой.

Программа 30

Числа

$\alpha+1$	a_1
$\alpha+2$	a_2
⋮	
$\alpha+n$	a_n
$\alpha+n+1$	a_{n+1}
$\alpha+n+2$	a_{n+2}
⋮	
$\alpha+2n$	a_{2n}
$\alpha+2n+1$	a_{2n+1}
⋮	
$\alpha+3n$	a_{3n}
⋮	
$\alpha+mn$	a_{mn}
$\beta+1$	b_1
⋮	

Команды

N	$+$	$-$	$-$	ω_1
$N+1$	$\times$	$\alpha+1$	α_1	ω_3
$N+2$	$\times$	$\beta+1$	α_2	ω_4
$N+3$	$+$	ω_3	ω_4	ω_3
$N+4$	$\times$	$\gamma+1$	α_1	ω_4
$N+5$	$\times$	$\delta+1$	α_2	ω_5
$N+6$	$+$	ω_4	ω_5	ω_4
$N+7$	:	ω_3	ω_4	ω_3
$N+8$	$+$	ω_3	ω_1	ω_1
$N+9$	$\oplus$	$N+1$	Δ_1	$N+1$
$N+10$	$\oplus$	$N+2$	Δ_1	$N+2$
$N+11$	$\oplus$	$N+4$	Δ_1	$N+4$
$N+12$	$\oplus$	$N+5$	Δ_1	$N+5$
$N+13$	$\leqslant$	$N+1$	Δ_2	$N+1$
$N+14$	$\times$	ω_2	ω_1	ω_2

$\beta + mn$	b_{mn}
$\gamma + 1$	c_1
. . .	
$\gamma + mn$	c_{mn}
$\delta + 1$	d_1
. . .	
$\delta + mn$	d_{mn}

ω_1	0	Σ
ω_2	1	Π
α_1	x	
α_2	y	

Δ_1		1		
Δ_2	$\times$	$\alpha + n$	α_1	ω_3
Δ_3		n		
Δ_4	$\times$	$\alpha + mn$	α_1	ω_3

$N+15$	$\oplus$	Δ_2	Δ_3	Δ_2
$N+16$	$\leqslant$	$N+1$	Δ_4	N
$N+17$	*ост.*			

Программа этого примера на групповых операциях может быть записана в виде

Программа 30_1

Числа

Δ_1	—	—	1	n
Δ_2	—	n		n
Δ_3	—	mn		—

Команды

$N+1$	*НГО*	—	Δ_1	Δ_1
$N+2$	+	—	—	ω_1
$N+3$	$\times$	$1 \mid \alpha + 1$	α_1	ω_3
$N+4$	$\times$	$1 \mid \beta + 1$	α_2	ω_4
$N+5$	+	ω_3	ω_4	ω_3
$N+6$	$\times$	$1 \mid \gamma + 1$	α_1	ω_4

	Команды			
$N+7$	$\times$	$1\|\delta+1$	α_2	ω_5
$N+8$	$+$	ω_4	ω_5	ω_4
$N+9$	:	ω_3	ω_4	ω_3
$N+10$	$+$	ω_3	ω_1	ω_1
$N+11$	*КГО*	Δ_1	$N+12$	$N+2$
$N+12$	$\times$	ω_2	ω_1	ω_2
$N+13$	$+$	Δ_1	Δ_2	Δ_1
$N+14$	$\leqslant$	Δ_1	Δ_3	$N+1$
$N+15$	*ост.*			

Рекомендуем читателю последовательно проследить за изменением содержимого регистров модификации и циклов в процессе работы программы.

Пример 3. Перевод чисел из двоичной в десятичную систему. В схемах некоторых машин отсутствует преобразование двоичных кодов чисел в десятичные. В таких случаях в набор стандартных подпрограмм машины включается специальная подпрограмма перевода и перед выводом чисел в программах предусматривается надлежащее обращение к ней.

Рассмотрим для определенности машину с фиксированной запятой и с 40 двоичными значащими разрядами.

Пусть ячейка α является входной для подпрограммы, т. е. для преобразования двоичного кода числа A в десятичный последний должен быть помещен в ячейку α. Пусть ячейка β является выходной, т. е. в ячейке β формируется преобразованный код данного числа A.

Обозначим через a_i соответствующие десятичные знаки числа A, т. е.

$$A=\frac{a_1}{10^1}+\frac{a_2}{10^2}+\ldots$$

Для фиксации каждого из чисел a_i нужны четыре двоичных разряда, т. е. на 40 разрядах ячейки β могут быть зафиксированы 10 десятичных кодов.

Задача преобразования двоичного кода числа может рассматриваться как задача построения некоторого кода

$$B=\frac{a_1}{16^1}+\frac{a_2}{16^2}+\ldots+\frac{a_{10}}{16^{10}},$$

на каждом из четырех двоичных разрядов которого зафиксирован соответствующий десятичный код.

Программа может быть представлена в виде

Программа 31

Числа

α_1	$\frac{10}{16}$			
α	A			
β		B		
Δ	—	—	1	10

Команды

N	*НГО*	—	—	Δ	
$N+1$	+	—	—	β	
$N+2$	×	α	α_1	α	
$N+3$	$\rightarrow ч$	36	α	r	сдвиг на 36 разрядов вправо
$N+4$	$\leftarrow ч$	4	β	β	сдвиг на 4 разряда влево
$N+5$	+	r	β	β	
$N+6$	$\leftarrow ч$	36	r	r	
$N+7$	—	α	r	α	
$N+8$	$\leftarrow ч$	4	α	α	
$N+9$	*КГО*	Δ	$N+2$	k	

Здесь k — номер команды, которой передается управление после выполнения программы.

Через $\rightarrow ч$ и $\leftarrow ч$ обозначены коды операции сдвига вправо и влево на число разрядов, указанных в IA соответствующей команды.

Для циклических процессов с известным числом циклов и состоящих лишь из одной команды, удобно иметь специальную групповую операцию *ГО*-I, кодируемую в виде команды

ГО-I	α	p	d_k

В адресах команды *ГО*-I непосредственно содержится информация о параметре: в IA — начальное значение параметра индекса модификации, в IIA — шаг, в $IIIA$ — конечное значение параметра.

После выполнения команды *ГО*-I следующая за ней команда выполняется повторно с изменением адресов, имеющих признак групповой операции, до тех пор, пока параметр не

приобретет конечного значения. После этого управление передается следующей команде. Вид команды *ГО*-I в памяти остается неизменным.

Таким образом, с помощью групповой операции *ГО*-I могут быть выполнены следующие виды операций:

1) $[A] \odot a$,
2) $a \odot [A]$,
3) $[A] \odot [B]$,
4) $[A] \odot$,

где $\odot$ — любая операция из групп I — III, V; A и B — n-мерные векторы; a — скаляр (быть может, равный нулю). Случай 4) относится к команде, у которой *IIA* не используется.

Для операций 1) и 4) признак групповой операции устанавливается в *IA* и *IIIA*, для операции вида 2) — в *IIA* и *IIIA*, для операции 3) — во всех адресах команды.

Для некоторых операций, связанных с обращением к внешним устройствам и употребляемых, как правило, одновременно для групп кодов, вводятся специальные групповые операции, кодируемые в виде отдельных команд специальными кодами операций. Примеры таких операций были приведены в главе II (операции ввода, вывода, обращения к внешнему запоминающему устройству). Приведем дополнительно еще следующие групповые операции:

1. Перевод n чисел $a+1$, $a+2$, ..., $a+n$ из десятеричной системы счисления в двоичную и запись их в ячейки *ЗУ* $b+1$, $b+2$, ..., $b+n$

код I	$a+1$	n	$b+1$

2. Перевод n чисел $a+1$, $a+2$, ..., $a+n$ из двоичной в десятеричную систему счисления и запись в ячейки $b+1$, $b+2$, ..., $b+n$

код II	$a+1$	n	$b+1$

3. Перевод из зоны a внешнего запоминающего устройства n чисел и запись их в ячейки $b+1$, $b+2$, ..., $b+n$ *ЗУ*

код III	a	n	$b+1$

Групповой сдвиг — перенос кодов из ячеек $a+1, \ldots, a+n$ *ЗУ* в ячейки с номерами $b+1, \ldots, b+n$

ГС	$a+1$	n	$b+1$

Заметим, наконец, что уже для выполнения групповой операции *ГО*-I с помощью команд переадресации потребовалось бы по крайней мере четыре команды: команда операции, команда переадресации ее переменных адресов, команда, осуществляющая проверку окончания цикла и соответствующую передачу управления. Кроме этого, как правило, нужны также команды для восстановления переменных команд и констант переадресации и восстановления. Вместе с тем за счет уменьшения количества команд, выполняющих действия над заданными числами, и вспомогательных команд достигается уменьшение количества выполняемых циклов в машине, т. е. сокращается время вычислений.

Таким образом, наличие групповых операций того или иного вида в машине упрощает процесс программирования, высвобождает ячейки *ЗУ*, а тем самым сокращает объем вводимой в машину информации, а также, как правило, сокращает время выполнения операций.

Относительно программирования при помощи групповых операций сложных циклических процессов, зависящих от нескольких параметров, ограничимся следующим замечанием.

Для кодировки «признаков» зависимости адресов от тех или иных параметров можно выделить в адресах несколько разрядов и соответственно усложнить управление групповыми операциями так, чтобы переменные адреса изменялись в зависимости от наличия того или иного набора признаков в них. Однако такое решение этого вопроса ведет к значительному усложнению устройства управления.

Можно ограничиться наличием одного разряда для кодировки признака групповой операции. Действительно, произведем следующее расщепление параметров. Пусть, например, имеем двойной циклический процесс: внешний цикл зависит от параметра i, внутренний — от параметра j. Если во внутреннем цикле нет адресов, зависящих от параметра i, параметры считаем уже расщепленными. При их наличии поступаем следующим образом: все величины, входящие в I и II адреса внутреннего цикла и зависящие от параметра i, перед программой внутреннего цикла пересылаем в стандартные ячейки; адреса III внутреннего цикла, зависящие от параметра i, заменяем

соответствующими стандартными ячейками, а после программы внутреннего цикла пересылаем их из стандартных ячеек в соответствующие ячейки (зависящие от i).

Упражнения

Используя групповые операции, составить программы:

1. Вычисления значений многочлена.
2. Вычисления скалярного произведения двух векторов.
3. Умножения матрицы на вектор.
4. Умножения матриц.
5. Решения системы линейных алгебраических уравнений методом простых итераций.
6. Решения системы линейных алгебраических уравнений методом исключения неизвестных.
7. Вычисления определителей.

ГЛАВА IV

ОПЕРАТОРНОЕ ПРОГРАММИРОВАНИЕ

Анализ программ, приведенных в главе III, показывает, что при программировании разнообразных задач применяются стандартные приемы, которые можно сформулировать в виде общих правил, не связанных с конкретным содержанием отдельных задач. Вместе с тем эти правила должны формулироваться как указания для программирования определенных алгоритмических функций в виде групп команд. Обычно такие группы команд выделяются в отдельные блоки. Блок-схемное описание программ облегчает их обозримость и делает возможным осуществление программирования отдельных блоков независимо друг от друга. Таким образом, классификация блоков программ по их алгоритмическому (функциональному) назначению в общей схеме решения задачи существенно облегчает задачу программирования.

Метод программирования, основанный на использовании классификации частей программ по их функциональному назначению, был предложен А. А. Ляпуновым и получил название операторного метода программирования. В данной главе излагаются основы этого метода.

§ 1. Схемы счета

Исходным материалом для решения всякой вычислительной задачи является совокупность начальных числовых данных, — обозначим ее символом $x_{\text{нач}}$. Конечной целью при решении задачи является получение числовых значений интересующих исследователя величин, — обозначим их через $x_{\text{кон}}$.

Процесс решения вычислительной задачи представляет собой некоторый алгоритм переработки начальных данных $x_{\text{нач}}$ в конечные $x_{\text{кон}}$. Этот алгоритм может быть представлен в виде ряда более или менее самостоятельных этапов вычислений. На каждом этапе определенная последовательность элементарных операций осуществляет переработку числовых данных, полученных на предыдущем этапе.

Совокупность операций, преобразующих числовые данные x в y, принято называть *оператором счета.* Если обозначить оператор счета через **A**, то это преобразование запишется как $x\mathbf{A} = y$. Обозначение операторов будем давать в дальнейшем жирным шрифтом. Оператор счета обычно определяется формулами, которые содержат числовые данные x и указания о последовательности выполнения операций над ними. Эти указания, вообще говоря, не однозначны, но должны обеспечивать единственность результата.

Весь процесс счета состоит из последовательности выполнения операторов счета. Переход от одного этапа к другому может быть заранее определен или может определяться в процессе счета на основании проверки тех или иных свойств числовой информации. Проверка определенных свойств информации осуществляется вычислением значений логических переменных. Условия, на основании которых происходит выбор порядка вычислений, называются *логическими условиями*, а соответствующие операторы, осуществляющие такой выбор, — *логическими операторами.* Логическое условие и соответствующий ему логический оператор будем обозначать соответственно буквами P и **P**. Логические условия можно задавать таблицей, содержащей соотношения, по которым вычисляются их значения. В простейших случаях содержание логического условия записывается в скобках после его обозначения, например: $P(a \leqslant b)$ или $P(a \neq b)$ и т. п. Если проверяемое логическое условие выполнено, то его значение считается равным единице, и в этом случае после логического оператора ставится стрелка с указанием оператора, которому передается управление. Если логическое условие не выполнено, то его значение считается равным нулю и управление передается оператору, следующему за логическим оператором.

Таким образом, алгоритм решения вычислительной задачи состоит из последовательности операторов счета, преобразующих числовую информацию, и логических операторов, осуществляющих переход к выполнению следующих операторов счета в соответствии со значением логических условий.

Полная последовательность операторов, определяющая весь процесс преобразования числовых данных задачи $x_{\text{нач}}$ в $x_{\text{кон}}$, называется *схемой счета.* Последовательное применение операторов записывается в виде их произведения:

$$\mathbf{A}_1 \cdot \mathbf{A}_2 \cdot \ldots \cdot \mathbf{A}_n = \prod_{k=1}^{n} \mathbf{A}_k.$$

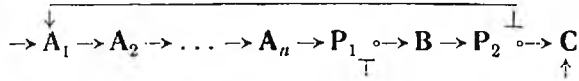
При написании схемы счета нужно знать, что оператор действует на числовую информацию, записанную от него слева,

после выполнения оператора счета действует следующий за ним справа оператор; при выполнении логического оператора определяется значение логического условия, если оно равно нулю, то выполняется следующий по порядку оператор; если логическое условие равно единице, то происходит переход к выполнению оператора, указание о котором содержит стрелка, стоящая после логического оператора. Например, порядок действия операторов в схеме счета

$$\prod_{k=1}^{n} \mathbf{A}_k \mathbf{P}_1 \overset{\mathbf{C}}{\uparrow} \mathbf{B} \mathbf{P}_2 \overset{\mathbf{A}_1}{\uparrow} \mathbf{C}\,!$$

следующий: сначала последовательно выполняются операторы счета $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$, затем вычислительный процесс разветвляется: если логическое условие P_1, которому соответствует логический оператор $\mathbf{P}_1$, равно нулю, то выполняется оператор счета $\mathbf{B}$, и если логическое условие P_2, которому соответствует логический оператор $\mathbf{P}_2$, равно единице, то выполняется снова оператор $\mathbf{A}_1$ и т. д.; если же логическое условие P_1 равно единице или если логическое условие P_2 равно нулю, то выполняется оператор счета $\mathbf{C}$ и процесс счета прекращается, что отмечается восклицательным знаком ! .

Можно отказаться от линейной записи схемы счета и переходы от одного оператора к другому указывать стрелками, причем от оператора счета исходит только по одной стрелке, от логических операторов — по две стрелки, одна с отметкой 1 (ı→), другая с отметкой 0 (∘→) в соответствии со значениями логического условия. Приведенная выше схема счета в виде граф-схемы имеет вид



Для того чтобы облегчить ориентировку в схеме счета, перед оператором, которому передается управление от логического оператора, можно ставить стрелку с номером соответствующего логического оператора, от которого происходит передача управления. Например, приведенная выше схема счета может быть представлена в форме

$$\overset{2}{\downarrow} \prod_{k=1}^{n} \mathbf{A}_k \mathbf{P}_1 \overset{\mathbf{C}}{\uparrow} \mathbf{B} \mathbf{P}_2 \overset{\mathbf{A}_1}{\uparrow} \overset{1}{\downarrow} \mathbf{C}!$$

Построение схемы счета осуществляется после выбора определенного метода решения задачи. Разбиение вычислительного процесса на операторы счета, вообще говоря, производится

неоднозначным способом. Практические навыки по рациональному построению схем счета вырабатываются в процессе работы.

В главе III построение программ осуществлялось после анализа задачи и построения схемы счета. При этом вид программы зависит от выбора порядка выполнения операций и от степени уточнения этого порядка в самой схеме счета.

В примере 1 вычисления дробно-линейной функции $y = \frac{ax+b}{cx+d}$ (§ 1 главы III) сама эта формула может служить схемой счета. Однако схема счета может быть уточнена установлением однозначного порядка выполнения операций, что и сделано при рассмотрении этого примера (см. формулы (2)).

В примере 3 вычисления значений многочлена от одного переменного $f(x) = a_0x^n + a_1x^{n-1} + \ldots + a_n$ (§ 1 главы III) приведены две схемы счета (6) и (8). Как показывает сравнение этих схем, вычисление значений многочлена от одного переменного по схеме Горнера (8) существенно сокращает число необходимых операций. Дальнейший анализ приводит к построению следующей схемы счета. Введем операторы

$$\begin{aligned} &\mathbf{B}_1 : \mathrm{A}_1 = a_0x, && \mathrm{A}_2 = \mathrm{A}_1 + a_1, \\ &\mathbf{B}_2 : \mathrm{A}_3 = \mathrm{A}_2x, && \mathrm{A}_4 = \mathrm{A}_3 + a_2, \\ &\qquad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &\mathbf{B}_n : \mathrm{A}_{2n-1} = \mathrm{A}_{2n-2}x, && \mathrm{A}_{2n} = \mathrm{A}_{2n-1} + a_n = f(x). \end{aligned}$$

Тогда схема счета (8) в операторной форме имеет вид $\prod_{k=1}^{n} \mathbf{B}_k$.

Операторы $\mathbf{B}_k$ определяются формулами

$$\mathrm{A}_{2k} = \mathrm{A}_{2k-2}x + a_k, \qquad k = 1, 2, \ldots, n.$$

Однотипность этих формул дает возможность построить рациональную программу вычисления значений многочлена от одного переменного (программа примера 2 § 4 главы III). Для обеспечения рационального программирования нужно дополнить схему счета операторами, учитывающими специфику как данной схемы, так и данной *ЦАМ*.

§ 2. Схемы программ

Схема счета позволяет осуществить решение задачи в соответствии с выбранным алгоритмом. Однако для автоматизации процесса вычислений, т. е. для выполнения процесса вычислений на *ЦАМ*, построения схемы счета недостаточно. При программировании задачи на *ЦАМ* должны быть учтены возможности и особенности автоматического управления процессом

вычислений на *ЦАМ*. Поскольку в командах, выполняющих элементарные операции, кодируются номера ячеек *ЗУ*, содержащих числа, над которыми должны производиться операции, одна и та же команда может выполнять элементарные операции над различными числами. Например, в программе 9 главы III $(k+2)$-я команда вычисляет последовательно числа натурального ряда.

Напомним, что сами команды в *ЦАМ* являются числами, над которыми также можно производить операции, поэтому, изменяя их соответствующим образом и применяя повторно, можно решать новые задачи — производить новые операции над теми же числами или те же операции над числами, содержащимися в других ячейках *ЗУ*. Например, в программе 19 § 4 $(N+2)$-я переменная команда использует для вычислений последовательно числа, содержащиеся в ячейках $\alpha+1$, $\alpha+2$, ..., $\alpha+n$.

При программировании задач на *ЦАМ* нужно также стремиться к наиболее рациональному использованию *ЗУ*. При помощи небольшого числа команд должно быть произведено большое число операций. Наконец, для осуществления вычислительного процесса на *ЦАМ* нужно предусмотреть ввод и вывод данных, перевод чисел из одной системы счисления в другую, контроль вычислений и т. п. При программировании вычислительной задачи для *ЦАМ* схема счета должна быть дополнена специальными операторами, которые называются *операторами управления*. Операторы управления осуществляют подготовку содержания *ЗУ* и обеспечивают наиболее рациональное управление вычислительным процессом на *ЦАМ*.

Схема счета, дополненная операторами управления, позволяющими представить алгоритм в виде программы (совокупности команд), называется *схемой программы* или *схемой работы машины*.

В схеме программы каждый оператор представляет собой определенную группу команд программы. Для упрощения схемы программы желательно, чтобы каждый оператор содержал максимальное число команд. В то же время совокупность команд каждого отдельного оператора обладает единым функциональным назначением. По своему функциональному назначению различаются операторы счета, логические операторы и операторы управления. В свою очередь можно выделить несколько типов операторов управления:

1) операторы переадресации,
2) операторы формирования,
3) операторы восстановления,
4) операторы засылки,
5) операторы переноса,
6) операторы циркуляции.

По мере накопления опыта программирования классификация операторов управления может совершенствоваться.

Перечисленные типы операторов позволяют строить весьма разнообразные программы.

Операторный метод программирования состоит в следующем. Вначале строится схема счета задачи, содержащая операторы счета и логические условия. Информация об операторах счета представляется в виде, обеспечивающем наиболее рациональное построение программы. Далее, в схему счета вводятся операторы управления, обеспечивающие выполнение схемы счета на *ЦАМ*. Наконец, составляется описание построенной схемы программы, т. е. задается информация о каждом операторе схемы. По заданной информации производится программирование операторов с использованием определенных правил. Отметим ряд преимуществ, свойственных операторному методу программирования.

Операторный метод программирования позволяет строить программы сложных задач по частям. При этом программирование отдельных частей программы может производиться после некоторого перерыва без предварительного изучения всей задачи и даже разными лицами. Использование операторного метода программирования облегчает отладку программы на *ЦАМ*, внесение исправлений в программу и создает условия для автоматизации самого процесса программирования.

Рассмотрим программы главы III, имеющие различные операторы управления.

В примере 3 вычисления таблицы квадратов чисел натурального ряда с запоминанием во внешней памяти (§ 4 главы III) $(k+1)$-я команда программы имеет переменный III адрес, который изменяется на единицу от цикла к циклу. Изменение III адреса $(k+1)$-й команды программы осуществляется $(k+3)$-й командой путем прибавления к $(k+1)$-й команде константы, содержащейся в ячейке β, т. е. $(k+3)$-я команда программы является командой переадресации с константой переадресации, содержащейся в ячейке β.

В примере 4 (§ 4 главы III) вычисления значений многочлена программа также содержит команду переадресации ($(N+3)$-я команда). Каждая команда переадресации может изменять одновременно один или несколько адресов одной и той же команды. Для изменения адресов различных команд программы необходимы различные команды переадресации. Так, в примере 5 (§ 4 главы III) вычисления суммы

$$s=\sum_{i=1}^{n}\frac{x_i^3+a}{x_i\sqrt{x_i}(x_i-a)}$$

программа содержит пять команд переадресации: от $(N+10)$-й до $(N+14)$-й включительно.

В рассмотренных примерах программы операторов счета содержат переменные команды, которые меняются при переходе от цикла к циклу. Если оператор содержит команды, изменяющиеся по определенному правилу при повторном его применении, то говорят, что *оператор зависит от параметра*. Зависимость оператора **A** от параметра k будем обозначать нижним индексом у оператора — $\mathbf{A}_k$ и задавать с помощью специальной *таблицы зависимости адресов от параметра*. Эта таблица содержит номера переменных команд, шаг переадресации и номера адресов, зависящих от параметра.

Оператор управления, осуществляющий переадресацию команд операторов, зависящих от параметров, в соответствии с таблицей зависимости адресов от параметра, называется *оператором переадресации* и обозначается $\mathbf{F}(pk)$, где p — шаг переадресации, т. е. число единиц, на которое переадресуется параметр в каждом цикле. Программа оператора переадресации строится по таблице зависимости адресов от параметра.

В примере 3 (§ 4 главы III) вычисления таблицы квадратов чисел натурального ряда с запоминанием оператор переадресации $\mathbf{F}(n)$ осуществляется $(k+3)$-й командой переадресации. Исходная информация для построения программы оператора переадресации $\mathbf{F}(n)$ — таблица зависимости адресов команд от параметра (сокращенно *ТЗП*) — в этом примере содержит одну строку.

Таблица 6 зависимости адресов от параметра

№ команды, зависящей от параметра	Зависимость от параметра I адреса	Зависимость от параметра II адреса	Зависимость от параметра III адреса
$k+1$	—	—	1

В примере 4 (§ 4 главы III) вычисления значений многочлена *ТЗП* также содержит одну строку

	$N+2$	—	1	—

по которой строится константа переадресации

β_1	—	—	1	—

и программа оператора переадресации $F(k)$, т. е. команда переадресации

$N+3$	$\oplus$	$N+2$	β_1	$N+2$

В примере 5 (§ 4 главы III) вычисления суммы

$$s=\sum_{i=1}^{n}\frac{x_i^3+a}{x_i\sqrt{x_i}(x_i-a)}$$

таблица зависимости адресов от параметров следующая:

ТЗП

$N+1$		1	1	—
$N+2$		1	—	—
$N+4$		1	—	—
$N+5$		1	—	—
$N+6$		1	—	—

В соответствии с этим строятся две константы переадресации

γ_1		1	1	
γ_2		1		

и программа оператора переадресации

$N+10$	$\oplus$	$N+1$	γ_1	$N+1$
$N+11$	$\oplus$	$N+2$	γ_2	$N+2$
$N+12$	$\oplus$	$N+4$	γ_2	$N+4$
$N+13$	$\oplus$	$N+5$	γ_2	$N+5$
$N+14$	$\oplus$	$N+6$	γ_2	$N+6$

Если программа задачи должна быть подготовлена для повторного применения, то необходимо предусматривать восстановление команд, измененных в процессе работы программы.

Оператор управления, подготавливающий исходное содержание *ЗУ* для повторной работы программы, называется *оператором формирования* и обозначается буквой **Ф**. Информация об операторах формирования должна содержать номера переменных ячеек *ЗУ*, требующих восстановления, и исходный вид этих ячеек.

В примере 4 (§ 4 главы III) вычисления значений многочлена для повторного применения программы должно быть предусмотрено восстановление исходного вида переменной команды $N + 2$

$N+2$	$+$	ω	$\alpha+1$	ω

что в программе 22 осуществляется командой

$N+5$	$\oplus$		α	$N+2$

где α — номер ячейки *ЗУ*, содержащей исходный вид $(N + 2)$-й команды.

В рассматриваемом примере известен заранее конечный вид переменной $(N + 2)$-й команды

$N+2$	$+$	ω	$\alpha+n+1$	ω

поэтому восстановление ее исходного вида может быть осуществлено также при помощи команды переадресации

$N+5$	$\ominus$	$N+2$	β_2	$N+2$

с константой переадресации

β_2			n	

Таким образом, оператор формирования **Ф** в рассматриваемом примере осуществляется командой $N + 5$.

В примере 5 (§ 4 главы III) оператор формирования **Ф** должен восстановить исходный вид команд $N+1$, $N+2$, $N+4$, $N+5$ и $N+6$

$N+1$	$\times$	$\alpha+1$	$\alpha+1$	ω_1
$N+2$	$\times$	$\alpha+1$	ω_1	ω_1
$N+4$	$\vee$	$\alpha+1$		ω_2
$N+5$	$\times$	$\alpha+1$	ω_2	ω_2
$N+6$	—	$\alpha+1$	β	ω_2

Поместим исходный вид этих команд в ячейки δ_1, δ_2, δ_3, δ_4 и δ_5 соответственно. Тогда оператор восстановления реализуется группой команд

$\Phi+1$	$\oplus$		δ_1	$N+1$
$\Phi+2$	$\oplus$		δ_2	$N+2$
$\Phi+3$	$\oplus$		δ_3	$N+4$
$\Phi+4$	$\oplus$		δ_4	$N+5$
$\Phi+5$	$\oplus$		δ_5	$N+6$

Оператор счета, зависящий от параметра, можно освободить от этой зависимости введением специального оператора засылки переменных данных в стандартные ячейки.

Оператором засылки называется оператор управления, осуществляющий последовательный перенос чисел в стандартные ячейки *ЗУ*. Будем обозначать его буквой **З**. Информация об операторе **З** должна состоять из пар номеров ячеек *ЗУ*. Первый номер пары определяет ячейку, из которой производится засылка, второй номер пары определяет ячейку, в которую производится засылка чисел.

В примере 4 § 4 главы III программа 22_2 содержит оператор засылки $З\{\alpha+1 \Rightarrow \delta\}$, реализуемый командой

N	$+$		$\alpha+1$	δ

В примере 5 § 4 главы III введение оператора засылки $З\{\alpha+1 \Rightarrow \delta\}$ реализуется командой

N	+		$\alpha+1$	δ

что значительно улучшает программу задачи (ср. программы 23 и 23_1).

Оператор засылки, не зависящий от параметра, будем называть *оператором переноса.*

В примере 2 § 3 главы III извлечения корня $y=\sqrt{x}$ программа содержит оператор переноса $З\{\omega_1 \Rightarrow \alpha_1\}$, реализуемый командой

$k+5$	+	ω_1		α_1

которая подготавливает содержание *ЗУ* для следующего итерационного цикла.

Специальным видом засылки является *циркуляция* последовательности величин в ячейках *ЗУ*.

Оператор циркуляции производит перестановку (циркуляцию) данных в ячейках *ЗУ*. Информацией для оператора циркуляции служит последовательность номеров ячеек *ЗУ*, в которых производится циркуляция чисел и порядок перестановки — подстановка.

В программе примера 6 (§ 4 главы III) вычисления интеграла $\int_0^1 y\,dx$ по формуле Симпсона производится циркуляция величин y_{2i}, y_{2i+1}, y_{2i+2} в ячейках $\omega+1$, $\omega+2$, $\omega+3$ при помощи группы команд

$k+10$	+	$\omega+2$		$\omega+1$
$k+11$	+	$\omega+3$		$\omega+2$
$k+12$	+	$\omega+4$		$\omega+3$

которые и определяют оператор циркуляции в данной программе.

Перейдем к анализу схем программ задач, рассмотренных в предыдущей главе.

Рассмотрим пример 1 § 3 главы III вычисления и печатания таблицы значений квадратов членов арифметической прогрессии. В этом примере схема счета в операторной форме имеет вид $\prod_{k=0}^{N} \mathbf{A}_k$, где операторы счета $\mathbf{A}_k$ определяются формулами

$$a_k^2 = b_k, \quad a_k + c = a_{k+1} \qquad (k = 0, 1, 2, \ldots, N),$$

причем $a_0 = a$. Так как числа a_k и b_k $(k = 0, 1, \ldots, N)$ в каждом цикле не запоминаются, то индекс k отмечает лишь их различные значения в каждом цикле, но не различное место в ячейках *ЗУ*.

В том случае, когда в последовательности $a_0, a_1, \ldots, a_N$ числа различны по величине, но помещаются в одну и ту же ячейку *ЗУ*, индекс в обозначении таких величин будем брать в скобки $(a_{(k)})$.

Таким образом, операторы счета $\mathbf{A}_k$ в рассматриваемом примере в соответствии с распределением ячеек *ЗУ* определяются формулами

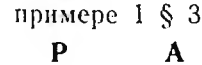
$$a_{(0)} = a; \quad a_{(k)}^2 = b_{(k)}, \quad a_{(k)} + c = a_{(k+1)} \qquad (k = 0, 1, \ldots, N).$$

Теперь очевидно, что операторы $\mathbf{A}_k$ определяются одинаковыми формулами, а следовательно, реализуются в машине одинаковыми группами команд.

Для построения схемы программы остается ввести оператор «П» — печати чисел $b_{(k)}$ и оператор **Р** — передачи управления по логическому условию,

$$P = \begin{cases} 1, & a_{(k)} \leqslant a + cN, \\ 0, & a_{(k)} > a + cN. \end{cases}$$

Схема программы в примере 1 § 3 главы III имеет вид



Это — типичная схема программы итерационных процессов и рекуррентных вычислений (см. программы 7, 8, 9, 10, 11 главы III).

В § 4 предыдущей главы рассмотрены примеры программ с операторами счета, зависящими от параметра.

В примере 2 § 4 главы III схема счета значений многочлена, как было нами установлено (стр. 172), имеет вид $\prod_{k=1}^{n} \mathbf{B}_k$; операторы $\mathbf{B}_k$ определяются формулами

$$\mathrm{A}_0 = a_0; \quad \mathrm{A}_{(2k-2)} x + a_k = \mathrm{A}_{(2k)}, \qquad k = 1, 2, \ldots, n; \quad \mathrm{A}_{(2n)} = f(x),$$

которые отличаются только числом a_k, т. е. операторы $\mathbf{B}_k$ содержат величины, зависящие от параметра.

Располагая числа $a_0, a_1, \ldots, a_n$, x в ячейках $ЗУ\,\omega$, $\alpha+1$, ..., $\alpha+n$, α соответственно, построим программы операторов $\mathbf{B}_k$ и отметим зависимость адресов команд программ от параметра:

	Команды				Зависимость адресов от параметра		
$B+1$	×	ω	α	ω			
$B+2$	+	ω	$\alpha+k$	ω		1	

Введем оператор переадресации $\mathbf{F}(k)$, осуществляющий изменение второго адреса $(B+2)$-й команды на единицу, т. е. переход от программы оператора $\mathbf{B}_k$ к программе оператора $\mathbf{B}_{k+1}$. Тогда, циклически повторяя выполнение команд счета $\mathbf{B}_k$ и подготавливая эти команды для выполнения следующего цикла, можно осуществить вычисление значений многочлена степени n небольшим числом команд. Для управления циклическим процессом вычислений введен оператор передачи управления $\mathbf{P}$ по условию, что II адрес $(B+2)$-й команды содержит код меньший, чем $\alpha+n+1$.

Для этого константа сравнения изображается кодом

β	-f	ω	$\alpha+n$	ω

и помещается в ячейку β. Логическое условие определяется соотношением

$$P=\begin{cases} 1, & (B+2)\leqslant(\beta), \\ 0, & (B+2)>(\beta). \end{cases}$$

Для восстановления переменной $(B+2)$-й команды программы введен оператор формирования $\mathbf{Ф}$, использующий исходный вид $(B+2)$-й команды:

$B+2$	+	ω	$\alpha+1$	ω

Теперь схеме счета $\prod\limits_{k=1}^{n}\mathbf{B}_k$ будет соответствовать схема программы

$$\mathbf{P}\quad\mathbf{B}$$
$$\downarrow \mathbf{B}_k\mathbf{F}(k)\,\mathbf{P}\uparrow\mathbf{Ф}!$$

Это — типичная схема программы для циклических процессов, зависящих от одного параметра (см. программы 6, 7, 8, 9 главы III).

Приведенные выше схемы программ являются основными схемами программ простейших циклических процессов. Аналогичные схемы программ могут отличаться наличием или отсутствием операторов формирования **Ф**, местом в программе операторов управления и логических операторов. Как правило, оператор формирования имеется в каждой программе и часто стоит в ее начале. Логическое условие, обеспечивающее циклический процесс, может стоять перед оператором счета, тогда в конце программы помещается команда безусловной передачи управления началу программы. Соответствующее команде безусловной передачи управления логическое условие обозначим P^0. Схема программы 6 главы III выглядит так:

$$\begin{matrix} 2 & 1 & & \mathbf{P}_1\,1 \\ \downarrow \mathbf{P}_1 & \uparrow & \mathbf{AP}_2^0 & \uparrow \downarrow ! \end{matrix}$$

Следует иметь в виду, что при перестановке операторов в схеме программы соответствующие им команды, вообще говоря, могут изменяться (см. программы 6 и 6_1 главы III).

§ 3. Программирование сложных циклических процессов

Программы многих практически важных задач не укладываются в простейшие схемы циклических процессов. В связи с этим необходимо рассмотреть программирование сложных циклических процессов. В настоящем параграфе мы рассмотрим циклические процессы с усложненными схемами программ.

Циклические процессы, рассмотренные в предыдущей главе, являются простыми (одинарными): их реализация зависит от одного логического условия. Наличие двух и более логических условий, организующих циклические процессы, усложняет схему программы.

Комбинируя простейшие схемы программ циклических процессов, можно получить различные виды двойных циклических процессов. Если результаты вычислений простого циклического процесса $\mathbf{AP}\uparrow !$ или $\mathbf{A}_k\mathbf{F}(k)\mathbf{P}\uparrow !$ являются исходными данными для нового циклического процесса с оператором счета **B**, то вычислительный процесс становится двойным циклическим процессом, например со схемой программы вида

$$\mathbf{\Phi AP}_1\uparrow\mathbf{BP}_2\uparrow ! \quad \text{или} \quad \mathbf{\Phi}(k)\,\mathbf{A}_k\mathbf{P}_1\uparrow\mathbf{BP}_2\uparrow !$$

Оператор счета B также может зависеть от какого-нибудь параметра. Циклический процесс будет также двойным, если оператор счета зависит одновременно от двух параметров. Схема программы тогда имеет, например, вид

$$\overset{\mathbf{A}}{\mathbf{\Phi}_1\mathbf{\Phi}_2(k)\,\mathbf{A}_{ki}\mathbf{F}(k)\,\mathbf{P}_1\uparrow}\;\mathbf{F}(i)\,\overset{\mathbf{\Phi}_2}{\mathbf{P}_2\uparrow}!$$

Число первичных циклов по k, вообще говоря, может зависеть от второго параметра i.

При программировании кратных циклических процессов необходимо следить за правильностью подготовки исходных данных и программы при переходе от цикла по одному параметру к циклам по другому параметру.

Возможность дальнейшего усложнения схем программ циклических процессов с кратными циклами очевидна.

1. Двойной циклический процесс без параметров.

1. Пример численного решения обыкновенного дифференциального уравнения первого порядка.

$$\frac{dy}{dx} = f(x, y)$$

на отрезке $[a, b]$ с начальным условием $y(a) = y_0$. Можно принять следующую вычислительную схему: отрезок $[a, b]$ разделим точками с шагом h. Обозначим $y_n = y(x_n)$ и проинтегрируем уравнение на интервале (x_{n-1}, x_n). Значение интеграла вычислим по формуле трапеций. Получим следующие формулы:

$$\Delta y_n = \frac{h}{2}\left[f(x_{n-1}, y_{n-1}) + f(x_n, y_n)\right],$$

$$y_n = y_{n-1} + \Delta y_n.$$

Вычисления по этим формулам будем вести методом итераций. Исходные данные $x_0 = a$, $y_0^{(0)} = y_0$; $y_1^{(0)} = y_0$,

$$y_n^{(k+1)} = y_{n-1} + \frac{h}{2}\left[f(x_{n-1}, y_{n-1}) + f(x_n, y_n^{(k)})\right].$$

Итерационный процесс заканчивается, как только $|y_n^{(k)} - y_n^{(k-1)}| < \varepsilon$. Будем предполагать, что вычисление значений функции $\frac{h}{2}f(x, y)$ ведется по подпрограмме, исходные данные для которой x и y помещаются в ячейках $ЗУ$ α_1 и α_2 соответственно, а результат $\frac{h}{2}f(x, y)$ определяется в ячейке ω_1. Это

предположение мы для сокращения записи обозначим в виде специальной символической команды

$\frac{h}{2} f(\alpha_1, \alpha_2)$			ω_1

Примем следующую схему счета. Исходные данные: $x_0 = a$, $y(0) = y_0$, $y_1^{(0)} = y_0$,

$$x_n = x_{n-1} + h,$$

$$y_n^{(k+1)} = \left[y_{n-1} + \frac{h}{2} f(x_{n-1}, y_{n-1})\right] + \frac{h}{2} f(x_n, y_n^{(k)}),$$

$$l_k = y_n^{(k+1)} - y_n^{(k)}.$$

В соответствии с этим оператор счета **А** в первичном цикле программируется по формулам

$$y_n^{(k+1)} = \varphi_{n-1} + \frac{h}{2} f(x_n, y_n^{(k)}),$$

$$l_k = y_n^{(k+1)} - y_n^{(k)}.$$

Оператор счета **В** во вторичном цикле состоит из вычислений:

$$\varphi_{n-1} = y_{n-1} + \frac{h}{2} f(x_{n-1}, y_{n-1}),$$

$$x_n = x_{n-1} + h.$$

Оператор формирования, подготавливающий исходные данные для оператора **А**, имеет вид

$$y_n^{(0)} = y_{n-1}.$$

Схема программы следующая:

$$\begin{matrix} & & \mathbf{A} & \mathbf{B} \\ \mathbf{B\Phi AP}_1 & \uparrow & \mathbf{P}_2 & \uparrow \end{matrix} !$$

Исходные данные для оператора **В** — числа x_{n-1}, y_{n-1}, h поместим в ячейках *ЗУ* α_1, α_2, α_3 соответственно. Оператор **В** выполняется командами

$B+1$	$\frac{h}{2} f(\alpha_1, \alpha_2)$			ω_1
$B+2$	$+$	ω_1	α_2	α_4
$B+3$	$+$	α_1	α_3	α_1

Исходные данные для оператора **А** — числа x_n, $y_n^{(0)} = y_{n-1}$, φ_{n-1} помещаются в ячейках α_1, α_2, α_4 соответственно.

Оператор **А** реализуется командами

$A+1$	$\frac{h}{2} f(\alpha_1, \alpha_2)$			ω_1
$A+2$	$+$	α_4	ω_1	ω_1
$A+3$	$-$	ω_1	α_2	α_5
$A+4$	$+$	ω_1		α_2

При выборе начального приближения $y_n^{(0)} = y_{n-1}$ в приведенной выше программе оператор формирования **Ф** отсутствует. Остается заметить, что для реализации логических условий P_1 и P_2 необходимы числа ε и b, которые мы поместим в ячейках α_6 и α_7.

Программа 1

Числа

α_1	x_0	x_n
α_2	y_0	$y_n^{(k)}$
α_3	h	
α_4		φ_{n-1}
α_5		l_n
α_6	ε	
α_7	b	
ω_1		$\frac{h}{2} f$

Команды

$k+1$	$\frac{h}{2} f(\alpha_1, \alpha_2)$			ω_1
$k+2$	$+$	ω_1	α_2	α_4
$k+3$	$+$	α_1	α_3	α_1
$k+4$	$\frac{h}{2} f(\alpha_1, \alpha_2)$			ω_1
$k+5$	$+$	α_4	ω_1	ω_1
$k+6$	$-$	ω_1	α_2	α_5
$k+7$	$+$	ω_1		α_2
$k+8$	$\lvert \leqslant \rvert$	α_6	α_5	$k+4$
$k+9$	П	α_2		
$k+10$	$\leqslant$	α_1	α_7	$k+1$
$k+11$	*ост.*			

Если в предыдущем примере интеграл заменяется по более точной формуле, в которой участвуют значения в трех и более точках, то схема программы дополняется оператором циркуляции, осуществляющим подготовку *ЗУ* для оператора счета **В**.

Воспользуемся, например, следующей формулой численного интегрирования (параболическая формула)

$$\int_{x_{n-2}}^{x_n} f(x)\,dx = \frac{h}{6}\,[f(x_{n-2}) + 4f(x_{n-2}) + f(x_n)].$$

Тогда схема счета для решения задачи предыдущего примера имеет вид

$$\mathbf{B}\begin{cases} \varphi_{n-1} = y_{n-1} + \dfrac{h}{6}\,[f_{n-2} + 4f_{n-1}], \\ x_n = x_{n-1} + h, \end{cases}$$

$$\mathbf{A}\begin{cases} y_n^{(k+1)} = \dfrac{h}{6} f(x_n,\ y_n^{(k)}) + \varphi_{n-1}, \\ l_k = y_n^{(k-1)} - y_n^{(k)}, \end{cases}$$

где обозначено $f_k = f(x_k,\ y_k)$.

Работу оператора **B** организуем следующим образом: будем считать, что $\frac{h}{6}f_0$, y_1 и число 4 заданы в ячейках α_8, α_9 и α_{10}. Тогда программа оператора **B** дополнится двумя командами:

$B+1$	$\frac{h}{6} f(\alpha_1, \alpha_2)$			ω_1
$B+2$	$\times$	ω_1	α_{10}	α_4
$B+3$	$+$	α_4	α_8	α_4
$B+4$	$+$	α_4	α_9	α_4
$B+5$	$+$	α_1	α_3	α_1

Для повторения вычислений в следующей $(n+1)$-й точке значение f_{n-1} необходимо поместить в ячейку α_8 на место f_{n-2}. Оператор сдвига **C** (простейший случай циркуляции) реализуется командой

$C+1$	$+$	ω_1		α_8

и помещается после оператора **B**.

Схема программы имеет вид

$$\mathbf{BCAP_1} \overset{\mathbf{A}}{\uparrow} \mathbf{P_2} \overset{\mathbf{B}}{\uparrow} !$$

2. Пример определения вещественных корней уравнения

$$f(x)=0, \tag{1}$$

где $f(x)$ — непрерывная функция на заданном интервале (a, b), алгебраическая или трансцендентная.

Предположим, что уравнение $f(x)=0$ не имеет вещественных корней четной кратности. Кроме того, предположим, что искомые корни располагаются друг от друга на расстоянии, превосходящем величину $d=\frac{1}{2^p}$ (p — целое).

Применим следующий метод проб, обеспечивающий последовательное определение корней уравнения $f(x)=0$ с заданной точностью $\Delta=\frac{1}{2^n}$, начиная с наименьшего и кончая наибольшим на интервале (a, b).

Разобьем интервал решеткой точек, находящихся друг от друга на расстоянии $\Delta=\frac{1}{2^N}$, где целое число $N \leqslant N_0$, N_0 — число двоичных разрядов, отведенных для фиксации мантисс чисел. Для удобства будем считать, что точки a и b принадлежат узлам этой решетки.

Множество всех узлов решетки в зависимости от знака $f(x)$ в каждом из них разбивается однозначно на два подмножества; если x^* является корнем уравнения (1), то при подсчетах на цифровой машине получаем или $f(x^*)=+0$ или $f(x)=-0$.

Условимся считать корнями уравнения «правые» узлы каждого из элементарных участков решетки, концы которых принадлежат разным подмножествам узлов.

Обозначим искомые корни уравнения (1) через x_i^* ($i=1, 2, \ldots, s$; $x_{i-1}^*<x_i^*$). За нулевую пробу x_i^0 для корня x_i^* принимаем:

$$x_i^{(0)}=\begin{cases} a & \text{для } i=1, \\ x_{i-1}^* & \text{для } i\neq 1. \end{cases} \tag{*}$$

Каждую последующую пробу $x_i^{(k+1)}$ для корня x_i^* выбираем по правилу

$$x_i^{(k+1)}=x_i^{(k)}+\Delta x_k,$$

где

$$\Delta x_0=\frac{1}{2^p}$$

и при $k>0$

$$\Delta x_k=\begin{cases}\Delta x_{k-1}, & \text{если } \operatorname{sign} f(x_i^{(k-1)})=\operatorname{sign} f(x_i^{(k)}),\\ -\frac{\Delta x_{k-1}}{2}, & \text{если } \operatorname{sign} f(x_i^{(k-1)})\neq\operatorname{sign} f(x_i^{(k)}) \text{ и } \Delta x_k>2^{-N}.\end{cases}$$

Взятие новых проб прекращается после получения пробы $x_i^{(k)}$, для которой

$$\operatorname{sign} f(x_i^{(k-1)})\neq\operatorname{sign} f(x_i^{(k)})$$

и

$$\Delta x_{k-1}=2^{-N}.$$

Легко убедиться в том, что эта проба и является корнем x_i^*, если четность чисел p и N одинакова.

Действительно, все шаги, совершаемые при переходе от k-й пробы к $(k+1)$-й вправо, будут вида $\frac{1}{2^l}$, где l имеет ту же четность, что и p, а все шаги влево — вида $-\frac{1}{2^l}$, где l имеет противоположную четность по отношению к p. Таким образом, последний шаг размера $\left|\frac{1}{2^N}\right|$ будет совершен также вправо, если p имеет одинаковую четность с N.

Полученное значение x_i^* принимается за нулевую пробу для корня x_{i+1}^* согласно с (*) и т. д. Расчеты прекращаются при получении значения x, удовлетворяющего неравенству

$$x>b+\Delta x_0.$$

При этом ясно, что максимальная разность между истинным значением корня и вычисленным может составлять $\frac{1}{2^N}$, если считать, что процесс вычисления $f(x)$ таков, что вычисляемое значение $f(x)$ всегда дает правильный знак $f(x)$.

Согласно сформулированному алгоритму имеем схему счета:

1. $x_i^{(k+1)}=x_i^{(k)}+\Delta x_k$ $(x_i^0=a;\ \Delta x_0=\Delta)$; (A)
2. $y_{k+1}=f(x_i^{(k+1)})$; (B)
3. $s_{k+1}=\operatorname{sign} y_{k+1}$; (C)
4. $\Delta x_{k+1}=\begin{cases}\Delta x_k, & \text{если } s_k=s_{k+1}; \quad (D_1)\\ -\frac{\Delta x_k}{2}, & \text{если } s_k\neq s_{k+1}. \quad (D_2)\end{cases}$

5. Если $\Delta x_k > 2^{-N}$, взятие проб прекращается, принимается $x_i^* = x_i^{(k)}$ и совершается переход к вычислению следующего корня.

6. Если $x_i^{k+1} > b + \Delta$, расчеты прекращаются. Проанализируем встречающиеся в задаче логические условия.

1) Условие окончания процесса вычислений P_1:

$$P_1 = P_1\{x_i^{(k+1)} \geqslant b + \Delta\}$$

зависит от $x_i^{(k+1)}$, в связи с чем оператор $\mathbf{P}_1$, выполняющий его проверку и надлежащие передачи управления, поместим после оператора счета **(A)**, изменяющего значение x.

2) Условие перехода к вычислению Δx_{k+1} по формуле (D_1) или (D_2):

$$P_2 = P_2\{s_k = s_{k-1}\}.$$

Условие P_2 — совпадение знаков функции в двух последовательных пробах эквивалентно следующему, более удобному для вычисления условию

$$P_2 \sim P_2' = P_2'\{f(x_i^{(k+1)}) f(x_i^{(k)}) \geqslant +0\}.$$

Для проверки условия P_2' нужна величина

$$s = f(x_i^{(k+1)}) f(x_i^{(k)}), \quad (C')$$

для которой одновременно нужны значения функции в данной $(k+1)$-й и в предыдущей k-й точке.

В связи с этим в схему программы следует включить оператор засылки $\mathbf{З}_1$, запоминающий значение функции в «данной» пробе в качестве ее значения в «предыдущей» пробе для следующего цикла.

Поскольку это запоминание должно иметь место вне зависимости от исхода проверки условия P_2', оператор $\mathbf{З}_1$ следует поместить после вычисления s и перед логическим оператором $\mathbf{P}_2'$.

3) Условие окончания взятия проб

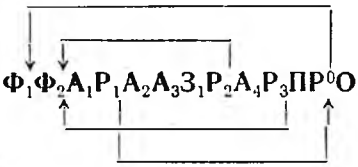
$$P_3 = P_3\{\Delta x_{k+1} \geqslant 2^{-N}\}$$

зависит от Δx, и поэтому, естественно, оператор $\mathbf{P}_3$ в логической схеме программы следует поместить после оператора, реализующего изменение Δx (D).

Введем обозначения:

Оператор $\mathbf{A}_1$ производит вычисления по формуле (A),
» $\mathbf{A}_2$ » » » » (B),
» $\mathbf{A}_3$ » » » » (C′),
» $\mathbf{A}_4$ » » » » (D_2),
» $\mathbf{Ф}_1$ формирует начальные данные x_0, $f(x)_0$,
» $\mathbf{Ф}_2$ » начальное значение $\Delta x_0 = \Delta$,
» $\mathbf{П}$ — печать x_i^* (с переводом в десятичную систему),
» $\mathbf{О}$ — останов.

Согласно с изложенным описанием алгоритма и отдельных операторов получим следующую логическую схему программы:



В этом примере мы ограничимся приведением схемы программы. Предоставляем читателю написание программы с предварительным изменением ее таким образом, чтобы используемое здесь значение функции в нулевой пробе ($f(x_0)$) предварительно вычислялось тем же оператором $\mathbf{A}_3$.

Заметим также, что изложенная методика определения корней легко может быть приспособлена к случаям определения одного наибольшего или одного наименьшего корня или для определения корней, расположенных на некотором подинтервале интервала (a, b).

3. Автоматический выбор шага по параметру. Рассмотрим задачу вычисления координат точек кривой, заданной параметрически,

$$\left.\begin{aligned} x &= x(t) \\ y &= y(t) \end{aligned}\right\} \quad (t_0 \leqslant t \leqslant T),$$

где x, y — непрерывные вычислимые функции от параметра t. Потребуем, чтобы последовательность определяемых, т. е. выдаваемых на печать, координат точек (x_k, y_k) рассматриваемой кривой удовлетворяла следующим условиям:

$$\begin{aligned} |\Delta x_k| &= |x_{k+1} - x_k| < \Delta, \\ |\Delta y_k| &= |y_{k+1} - y_k| < \Delta \end{aligned} \qquad (2)$$

и по крайней мере одна из величин Δx_k, Δy_k была больше δ:

$$|\Delta x_k| > \delta \quad \text{или} \quad |\Delta y_k| > \delta. \tag{3}$$

Здесь величины Δ и δ заданы и удовлетворяют, например, условию $4\,\delta < \Delta$.

Условие (2) означает, что точки, координаты которых выдаются на печать, не должны располагаться слишком редко на кривой; условие (3), наоборот, означает, что эти точки не должны располагаться на ней слишком близко друг от друга.

Оба требования являются вполне естественными, так как выполнение первого из них дает возможность получить последовательность точек, дающую достаточно полное представление о кривой; нарушение второго привело бы к излишнему загромождению объема выводимой информации.

Пусть на некотором шаге получены $x_k\,(t_k)$, $y_k\,(t_k)$. Тогда для вычисления x_{k+1}, y_{k+1} принимаем следующую схему счета:

1. $t^i_{k+1} = t_k + \Delta t^i_{k+1}$, где $\Delta t^i_{k+1} = \Delta t_k$; $i = 1, 2, \ldots$;
2. $x^i_{k+1} = x(t^i_{k+1})$;
   $y^i_{k+1} = y(t^i_{k+1})$;
3. $\Delta x^i_{k+1} = x^i_{k+1} - x_k$;
   $\Delta y^i_{k+1} = y^i_{k+1} - y_k$;
4. $\Delta t^{i+1}_{k+1} = \dfrac{\Delta t^i_{k+1}}{2}$, если $P_1 = 0$;
   $\Delta t^{i+1}_{k+1} = 2\Delta t^i_{k+1}$, если $P_2 = 0$;
   $\Delta t^{i+1}_{k+1} = \Delta t_{k+1}$, если $P_1 = 1$ и $P_2 = 1$;
5. $t_{k+1} = t_k + \Delta t_{k+1}$; $x_{k+1} = x(t_{k+1})$; $y_{k+1} = y(t_{k+1})$.
6. Вычисления прекращаются при $P_3 = 1$:

$$P_3 = P_3\{t \geqslant T\}.$$

Здесь через P_1 и P_2 обозначены условия

$$P_1 = P_1\{|\Delta x^i_{k+1}| < \Delta \quad \text{и} \quad |\Delta y^i_{k+1}| < \Delta\};$$
$$P_2 = P_2\{|\Delta x^i_{k+1}| > \delta \quad \text{или} \quad |\Delta x^i_{k+1}| > \delta\}.$$

Обозначим:

$\mathbf{A}_1$ — оператор вычисления по формуле 1,
$\mathbf{A}_2$ — оператор вычисления по формулам 2 — 3,
$\mathbf{A}_3$ — » » » формуле 4—I,
$\mathbf{A}_4$ — » » » формуле 4—II

(величина Δt^{i+1}_{k+1} помещается в ячейку, содержавшую ранее Δt^{i}_{k+1}).

З — оператор засылки:

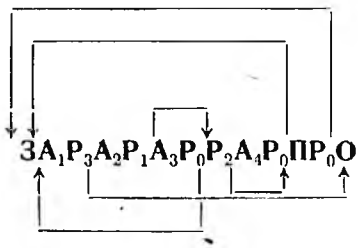
$$t_{k+1} \to t_k,$$

$$x_{k+1} \to x_k,$$

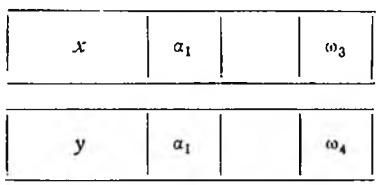
$$y_{k+1} \to y_k;$$

П — оператор вывода на печать x_{k+1} и y_{k+1};
О — оператор останова.

Получаем логическую схему программы



Обозначим через



обобщенные команды, которые осуществляют вычисление x^{i}_{k+1} и y^{i}_{k+1} с помещением результатов в ячейки ω_3 и ω_4 по заданному t^{i}_{k+1}, содержащемуся в ячейке α_1.

Программа может быть представлена в виде

Числа			Команды					
α	t_0		$N+1$	+	—	α_1	α	З
α_1		t_{k+1}	$N+2$	+	—	ω_3	ω_1	
ω_1			$N+3$	+	—	ω_4	ω_2	
ω_2			$N+4$	+	α	β	α_1	A_1
ω_3	x_0	x_{k+1}	$N+5$	$\leqslant$	δ_1	α_1	$N+21$	P_3
ω_4	y_0	y_{k+1}	$N+6$	x	α_1		ω_3	A_2
β	Δt_0	Δt	$N+7$	y	α_1		ω_4	
δ_1	T		$N+8$	—	ω_1	ω_3	γ_1	
δ_2	Δ		$N+9$	—	ω_2	ω_4	γ_2	
δ_3	2		$N+10$	$\lvert\leqslant\rvert$	δ_2	γ_1	$N+12$	P_1
δ_4	δ		$N+11$	$\lvert\leqslant\rvert$	γ_2	δ_2	$N+14$	
γ_1		Δx	$N+12$	:	β	δ_3	β	A_3
γ_2		Δy	$N+13$	$\leqslant$	—	—	$N+4$	P_0
			$N+14$	$\lvert\leqslant\rvert$	δ_4	γ_1	$N+18$	P_2
			$N+15$	$\lvert\leqslant\rvert$	δ_4	γ_2	$N+18$	
			$N+16$	$\times$	β	δ_3	β	A_4
			$N+17$	$\leqslant$	—	—	$N+4$	P_0
			$N+18$	$П$			ω_3	П
			$N+19$	$П$			ω_4	
			$N+20$	$\leqslant$	—	—	$N+1$	P_0
			$N+21$	*ост.*				О

Программирование двойных циклических процессов с одним параметром не представляет новых затруднений и может быть выполнено читателем самостоятельно (см. упражнение 2).

2. Двойной циклический процесс с двумя параметрами. 1. Решение уравнения теплопроводности. Уравнение теплопроводности

$$\frac{\partial U}{\partial t} = a^2 \frac{\partial^2 U}{\partial x^2}$$

в области $0 \leqslant t \leqslant T$, $0 \leqslant x \leqslant X$ с начальными и граничными условиями

$$U(0, x) = f(x); \quad U(t, 0) = \varphi(t); \quad U(t, X) = \psi(t)$$

можно решать по следующей схеме: положим

$$\Delta x = h; \quad \Delta t = \frac{h^2}{2a^2}$$

и обозначим

$$K = \frac{2a^2}{h^2} T; \quad J = \frac{X}{h},$$

$$U_{ki} = U(k\Delta t, ih) \qquad (k = 0, 1, \ldots, K; \quad i = 0, 1, \ldots, J).$$

Заменим

$$\frac{\partial U}{\partial t} \simeq \frac{1}{\Delta t}(U_{k+1, i} - U_{k, i}); \quad \frac{\partial^2 U}{\partial x^2} \simeq \frac{1}{h^2}(U_{k, i+1} - 2U_{k, i} + U_{k, i-1}).$$

Тогда уравнению теплопроводности будет соответствовать разностное уравнение

$$U_{k+1, i} = \frac{1}{2}(U_{k, i-1} + U_{k, i+1}), \qquad \begin{matrix} k = 0, 1, \ldots, K-1, \\ i = 0, 1, \ldots, J-1. \end{matrix}$$

Эти формулы определяют оператор счета $\mathbf{A}_{ik}$, зависящий от двух параметров: i и k. Схема программы соответствует двойному циклическому процессу с двумя параметрами:

$$\Phi_{(k)} \mathbf{A}_{ik} \mathbf{F}(i) \mathbf{P}_1 \overset{\mathbf{A}}{\uparrow} \mathbf{F}(k) \Phi(i) \mathbf{P}_2 \overset{\mathbf{A}}{\uparrow} !$$

Вид программы зависит от расположения чисел U_{ki} в ячейках $ЗУ$. Предположим, что числа U_{ki} помещаются в ячейках $\alpha + i + kJ$

$$(k = 0, 1, 2, \ldots, K; \quad i = 0, 1, 2, \ldots, J).$$

Для программирования оператора $\mathbf{A}_{ik}$ число $\frac{1}{2}$ поместим в ячейке β.

Оператор $\mathbf{A}_{ik}$ реализуется командами

Λ_{ik}					
	$A+1$	$+$	$\alpha+(i-1)+kJ$	$\alpha+(i+1)+kJ$	ω
	$A+2$	$\times$	ω	β	$\alpha+i+(k+1)J$

Начальный вид оператора $\mathbf{A}_{ik}$ при $i = k = 1$

$\mathbf{A}_{11}$				
$A+1$	$+$	$\alpha+0+J$	$\alpha+2+J$	ω
$A+2$	$\times$	ω	β	$\alpha+1+2J$

Зависимость адресов оператора $\mathbf{A}_{ik}$ от параметров определяется таблицей

$A+1$	$(i-1)+kJ$	$i+1+kJ$	
$A+2$			$i+(k+1)J$

Из таблицы следует, что для изменения параметра i на единицу необходимо иметь две константы переадресации. Одна содержит единицу в I и во II адресе, что мы условились обозначать

	1	1	

а другая содержит только единицу в III адресе, т. е. имеет вид

			1

Константы переадресации

	1	1	
			1

поместим в ячейках δ_1 и δ_2. Программа оператора переадресации примет вид

$\mathbf{F}(i)$				
$F+1$	$\textcircled{1}$	$A+1$	δ_1	$A+1$
$F+2$	$\oplus$	$A+2$	δ_2	$A+2$

Для программирования окончания внутреннего циклического процесса можно воспользоваться либо специальным счетчиком числа циклов, либо одной из переменных команд оператора $\mathbf{A}_{ik}$, например $A+2$.

Рассмотрим второй способ. Окончательный вид команды $A+2$ на последнем цикле по параметру i следующий:

$A+2$	$\times$	ω	β	$\alpha+J+(k+1)J$

Соответствующий код мы поместим в ячейке δ_3 и используем его в качестве константы для сравнения, отметив зависимость этого кода от параметра k:

δ_3			$(k+1)J$

Логическое условие P_1 тогда реализуется командой

P_1+1	$\leqslant$	$A+2$	δ_3	$A+1$

Для программирования $\mathbf{F}(k)$ и Φ (i) нужно знать конечный вид оператора $\mathbf{A}_{ik}$ после осуществления первичного циклического процесса по параметру i, т. е. $\mathbf{A}_{J,k}$, и начальный вид этого оператора $\mathbf{A}_{ik}$ перед осуществлением циклического процесса по параметру k, т. е. $\mathbf{A}_{1,k+1}$,

$\mathbf{A}_{1,k+1}$

$A+1$	$+$	$\alpha+0+(k+1)J$	$\alpha+2+(k+1)J$	ω
$A+2$	$\times$	ω	β	$\alpha+1+(k+2)J$

Сравнивая $\mathbf{A}_{1,k+1}$ и $\mathbf{A}_{J,k}$, находим, что для преобразования $\mathbf{A}_{J,k}$ в $\mathbf{A}_{1,k+1}$ нужно в команде $A+1$ изменить на единицу I и II адреса, а в команде $A+2$ изменить на единицу III адрес. Кроме того, нужно произвести изменение по параметру k константы δ_3, что осуществляется с помощью константы «J единицы в III адресе», которую поместим в ячейке δ_7. Программа операторов $\mathbf{F}(k)$, $\Phi(i)$

$F+1$	$\oplus$	$A+1$	δ_1	$A+1$
$F+2$	$\oplus$	$A+2$	δ_2	$A+2$
$F+3$	$+$	δ_3	δ_7	δ_3

Для программирования логического условия P_2 можно использовать конечный вид команды $A+2$ при $k=K$ и $i=J$:

$A+2$	$\times$	ω	β	$\alpha+(K+2)J$

Поместим константу

×	ω	β	$\alpha+(K+2)J$

в ячейке δ_3. Тогда логическое условие P_2 реализуется командой

⩽	$A+2$	δ_8	$A+1$

Остается составить программу для оператора $\Phi(k)$, восстанавливающего исходный вид оператора $\mathbf{A}_{ik}$ для повторного его применения. Переход от конечного вида оператора $\mathbf{A}_{ik}$ после окончания циклических процессов по параметрам i и k к начальному виду $\mathbf{A}_{11}$ можно осуществить либо с помощью пересылки исходных команд $A+1$ и $A+2$ из фиксированных ячеек $ЗУ$ δ_5 и δ_6 в соответствующие места программы, либо с помощью вычисления подходящих констант из команд $A+1$ и $A+2$, которые получим из сравнения $\mathbf{A}_{1,k}$ и $\mathbf{A}_{11}$, а именно, констант

$(k-1)J$	$(k-1)J$	

		$(k-1)J$

Эти константы поместим в ячейках $ЗУ$ δ_5 и δ_6. В соответствии с этим оператор $\Phi(k)$ может быть реализован командами

$\Phi+1$	⊕	δ_5		$A+1$
$\Phi+2$	⊕	δ_6		$A+2$

либо командами

$\Phi+1$	⊖	$A+1$	δ_5	$A+1$
$\Phi+2$	⊖	$A+2$	δ_6	$A+2$

Так как объем используемой памяти при этом одинаков, то первый вариант предпочтительнее второго. Впрочем, если бы константы вычитания δ_5 и δ_6 во втором случае образовывались в процессе счета по программе, то второй вариант оказался бы лучше.

Программа 2

Числа

$\alpha+i+kJ$ $(i=0, 1, \ldots, J)$ $(k=0, 1, \ldots, K)$	u_{ik}			
β	$\frac{1}{2}$			
δ_1		1	1	
δ_2				1
δ_3	$\times$	ω	β	$\alpha+3J-1$
δ_4	$\times$	ω	β	$\alpha+(k+1)J$
δ_5	$+$	$\alpha+J$	$\alpha+2+J$	ω
δ_6	$\times$	ω	β	$\alpha+1+2J$
δ_7				J
ω	рабочая			

			$(k+1)J$
–			

Команды

$k+1$	$\oplus$	δ_5		$k+3$
$k+2$	$\ominus$	δ_6		$k+4$
$k+3$	$+$	$\alpha+J$	$\alpha+2+J$	ω
$k+4$	$\times$	ω	β	$\alpha+1+2J$
$k+5$	$\ominus$	$k+3$	δ_1	$k+3$
$k+6$	$\ominus$	$k+4$	δ_2	$k+4$
$k+7$	$\leqslant$	$k+4$	δ_3	$k+3$
$k+8$	$\ominus$	$k+3$	δ_1	$k+3$
$k+9$	$\oplus$	$k+4$	δ_2	$k+4$
$k+10$	$+$	δ_3	δ_7	δ_3
$k+11$	$\leqslant$	$k+4$	δ_4	$k+3$
$k+12$	*ост.*			

ТЗП

$k+3$		$i-1+kJ$	$i+1+kJ$	
$k+4$		–	–	$i+(k+1)J$

Выше мы отмечали, что для программирования логических условий P_1 и P_2 необходимо вести счет числа циклов по параметрам i и k. Этот счет осуществляется при помощи переменных команд оператора $\mathbf{A}_{ik}$. Однако можно воспользоваться счетчиком числа циклов в специальной ячейке $ЗУ$ δ_3, в которой вначале помещается число нуль. В качестве единицы счета используем константу переадресации δ_2. Счетчик можно осуществить при помощи команды

$C+1$	$+$	δ_2	δ_3	δ_3

Число циклов по параметру i равно $J-1$. Следовательно, для выполнения логического условия P_1 необходимо иметь число «$J-1$ единиц в III адресе», которое поместим в ячейке δ_8. Тогда логическое условие P_1 можно реализовать командой

P_1+1	$\leqslant$	δ_3	δ_8	$A+1$

Для реализации логического условия P_2 нужно иметь константу «$(k+1)J$ единиц в III адресе», которую поместим в ячейке δ_4

P_2+1	$\leqslant$	δ_3	δ_4	$A+1$

Заметим, что теперь константа δ_1 зависит от параметра k и, следовательно, должна изменяться оператором $\mathbf{F}(k)$. Изменить программы операторов $\mathbf{F}(k)$ $\mathbf{\Phi}(i)$ в этом случае предоставляем читателю.

2. Разложение многочлена $f(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0$ по степеням $(x-a)$. Нам нужно определить коэффициенты b_k $(k=0, 1, 2, \ldots, n)$ разложения многочлена по степеням $(x-a)$

$$f(x)=b_n(x-a)^n+b_{n-1}(x-a)^{n-1}+\ldots+b_1(x-a)+b_0;$$

будем определять их последовательным применением схемы Горнера деления многочлена $f(x)$ на $(x-a)$. Коэффициенты в i-й строке схемы Горнера вычисляются по рекуррентной формуле

$$a_k^{(i)}=aa_{k+1}^{(i)}+a_k^{(i-1)},$$

где

$$k=n-1,\ n-2,\ \ldots,\ i-1;\quad i=1,\ 2,\ \ldots,\ n,$$

причем $a_k^{(0)}=a_k$ $(k=0,\ 1,\ 2,\ \ldots,\ n)$; $a_{i-1}^{(i)}=b_{i-1}$ — искомые числа. Оператор счета $\mathbf{A}_{ki}$ зависит от двух параметров: k и i; число внутренних циклов по параметру k зависит от i. Схема программы имеет вид

$$\mathbf{A}_{ki}\mathbf{F}(k)\,\mathbf{P}_1\overset{\mathbf{A}}{\uparrow}\mathbf{F}(i)\,\Phi(k)\,\mathbf{P}_2\overset{\mathbf{A}}{\uparrow}!$$

Исходные данные для счета оператора $\mathbf{A}_{ki}$ — числа $a_k=a_k^{(0)}$ $(k=0,1,\ \ldots,\ n)$ и число a — поместим в ячейках ЗУ $\alpha+k\,(k=0,1,\ \ldots,\ n)$ и β. Результаты вычислений $a_k^{(i)}$ также помещаются в ячейках $\alpha+k$. Оператор $\mathbf{A}_{kl}$ программируется командами

$\mathbf{A}_{kl}$

$A+1$	$\times$	$\alpha+n$	β	ω	$k+1$		
$A+2$	$+$	$\alpha+n-1$	ω	$\alpha+n-1$	k		k

Для реализации оператора $\mathbf{F}(k)$ необходимо иметь константы переадресации, которые поместим в ячейках δ_1 и δ_2:

δ_1	—	1	—	—
δ_2	—	1	—	1
F_1+1	$\ominus$	$A+1$	δ_1	$A+1$
F_2+2	$\ominus$	$A+2$	δ_2	$A+2$

Для программирования логического условия P_1 можно использовать команду $A+1$, которая в конце первичного цикла имеет вид

$A+1$	$\times$	$\alpha+0$	β	ω	$l-1$		

Константа для сравнения

×	$\alpha+0$		

зависящая от параметра i, помещается в ячейке δ_3:

δ_3	×	$\alpha+0$			$i-1$		

Для реализации оператора $\mathbf{F}(i)$ нужно изменить лишь константу δ_3, что осуществляется командой

F_2+1	+	δ_3	δ_1	δ_3

Так как вид оператора $\mathbf{A}_{kl}$ после циклического процесса по параметру k зависит от параметра i, то восстановление оператора $\mathbf{A}_{kl}$ удобно осуществлять пересылкой кодов

δ_4	×	$\alpha+n$	β	ω
δ_5	+	$\alpha+n-1$	ω	$\alpha+n-1$

в ячейки $A+1$ и $A+2$. Оператор $\Phi_{(k)}$ естественно поместить в начале программы. Для программирования логического условия P_2 можно использовать константу, содержащуюся в ячейке δ_3, конечный вид которой

×	$\alpha+n$	—	—

В ячейке δ_6 поместим константу

×	$\alpha+n-1$	—	—

Тогда логическое условие P_2 можно реализовать командой

$\leqslant$	δ_3	δ_6	$\Phi+1$

Программа 3

Числа

$\alpha+k$ $0 \leqslant k \leqslant n$	a_k	b_k
β	a	
δ_1	— | 1	— | —
δ_2	— | 1	— | 1
δ_3	$\times$ | $\alpha+0$	— | —
δ_4	$\times$ | $\alpha+n$	β | ω
δ_5	$+$ | $\alpha+n-1$	ω | $\alpha+n-1$
δ_6	$\times$ | $\alpha+n+1$	— | —
ω	раб.	

i	—	—	

Команды

$k+1$	$+$	δ_4	$+$	$k+3$
$k+2$	$+$	δ_5	$+$	$k+4$
$k+3$	$\times$	$\alpha+n$	β	ω
$k+4$	$+$	$\alpha+n-1$	ω	$\alpha+n-1$
$k+5$	$\ominus$	$k+3$	δ_1	$k+3$
$k+6$	$\ominus$	$k+1$	δ_2	$k+4$
$k+7$	$\leqslant$	$k+3$	δ_3	$k+3$
$k+8$	$+$	δ_3	δ_1	δ_3
$k+9$	$\leqslant$	δ_3	δ_6	$k+1$
$k+10$	*ост.*			

3. Тройной циклический процесс.

1. Решение систем линейных уравнений методом итераций. Дана система линейных уравнений $Ax = b$, где $= \{a_{ik}\}^m$ — квадратная матрица m-го порядка, $b = (b_1, b_2, \ldots, b_m)$ — m-мерный вектор. Если матрица A удовлетворяет известным условиям, то эта система может быть решена методом итераций. Примем следующую вычислительную схему:

$$\Delta x_k^{(n)} = x_k^{(n)} - x_k^{(n-1)} = \sum_{i=1}^{k-1} a_{ki} x_i^{(n)} + \sum_{i=k}^{m} a_{ki} x_i^{(n-1)} + b_k,$$

$$x_k^{(n)} = x_k^{(n-1)} + \Delta x_k^{(n)},$$

$$k = 1, 2, \ldots, m.$$

Итерации прекращаются при условии, что

$$\|\Delta x^{(n)}\| = \sqrt{\sum_{k=1}^{m} (\Delta x_k^{(n)})^2} < \varepsilon,$$

где ε — заданное число. Схема счета содержит три параметра: k, i и m. Введем операторы счета, учитывая их зависимости от параметров и стремясь к разделению параметров. Сначала действует оператор $\mathbf{A}_{ik}$

$$\sum_{i=1}^{k-1} a_{ki} x_i^{(n)} + \sum_{i=k}^{m} a_{ki} x_i^{(n-1)} = s_k^{(n)},$$

затем оператор $\mathbf{B}_k$

$$\Delta x_k^{(n)} = s_k^{(n)} + b_k; \quad x_k^{(n)} = x_k^{(n-1)} + \Delta x_k^{(n)};$$

$$d_0 = 0, \quad d_k = d_{k-1} + \left(\Delta x_k^{(n)}\right)^2, \quad k = 1, 2, \ldots, m.$$

Схема программы следующая:

$$\mathbf{A}_{ik}\mathbf{F}(i)\,\mathbf{P}_1 \uparrow \mathbf{B}_k\mathbf{F}_1(k)\,\mathbf{\Phi}_1(i)\,\mathbf{P}_2 \uparrow \mathbf{F}_2(n)\,\mathbf{\Phi}_2(k)\,\mathbf{P}_3 \uparrow !$$

(A) (A) (A)

Поместим коэффициенты системы a_{ik} в ячейки $\alpha + i + (k-1)m$ $(i = 1, 2, \ldots, m;\ k = 1, 2, \ldots, m)$, правые части уравнений b_k в ячейки $\beta + k$ $(k = 1, 2, \ldots, m)$, исходные значения неизвестных $x_k^{(0)}$ в ячейки $\gamma + k$ $(k = 1, 2, \ldots, m)$. В этих же ячейках будем помещать $x_k^{(n)}$. Для чисел $\Delta x_k^{(n)}$ отведем одну и ту же ячейку ω_1, которую необходимо очищать при изменении параметра k. Программа оператора $\mathbf{A}_{ik}$:

$\mathbf{A}_{ik}$

$A+1$	$\times$	$\alpha+1+0\cdot m$	$\gamma+1$	ω_2
$A+2$	$+$	ω_1	ω_2	ω_1

	$i+(k-1)m$	k	—

Легко составляется программа оператора $\mathbf{F}_1(i)$:

$\mathbf{F}_1(i)$

F_1+1	$\oplus$	$A+1$	δ_1	$A+1$

δ_1

—	1	—	—

Для программирования логического условия P_1 используем команду $A+1$ и константу, которая определяется конечным видом команды $A+1$ при $i = m+1$

$A+1$	$\times$	$\alpha+m+1$	$\gamma+1$	ω_2		$(k-1)m$	k	—

Эту константу поместим в ячейке δ_2, отметив ее зависимость от параметра k.

Программа оператора $\mathbf{B}_k$

$\mathbf{B}_k$

$B+1$	$+$	ω_1	$\beta+1$	ω_1
$B+2$	$+$	ω_1	$\gamma+1$	$\gamma+1$
$B+3$	$\times$	ω_1	ω_1	ω_1
$B+4$	$+$	ω_3	ω_1	ω_3

—	—	k	—
—	—	k	k

использует ячейку ω_3, в которой помещается $\|\Delta x^{(n)}\|$; следовательно, ω_3 зависит от параметра n. Операторы $\mathbf{F}_2(k)\,\mathbf{F}_1(i)$ изменяют

команду $A+1$, очищают ячейку ω_1 и изменяют константу δ_2. Константу для изменения команды $A+1$ получим, вычитая из $(A+1)$-й команды при значении параметров $i=1$ и $k=k+1$ ее значение при $i=m+1$ и $k=k$

	$1+km$	$k+1$	—
	$m+1+(k-1)\,m$	k	—

δ_3	—	—	1	—

Для изменения δ_2 используется константа

—	m	—	—

которая помещается в ячейке δ_4. Программа $\mathbf{F}_2(k)\,\Phi_1(i)$

	F_2+1	⊕	$A+1$	δ_3	$A+1$
$\mathbf{F}_2(k)\,\Phi_1(i)$	F_2+2	+	δ_2	δ_4	δ_2
	F_2+3	+			ω_1

Как и в предыдущем примере, благодаря связи между параметрами i и k операторы $\mathbf{F}_2(k)$ и $\Phi_1(i)$ программируются одновременно. Логическое условие P_2 реализуется сравнением δ_2 с константой

δ_5	×	$\alpha+m^2+1$	—	—

Операторы $\mathbf{F}_3(n)\ \Phi_2(k)$ очищают ячейку ω_3 и восстанавливают исходный вид операторов $\mathbf{A}_{ik}$ и $\mathbf{B}_k$ с помощью констант переадресации

δ_6	—	m^2-1	m	—
δ_7	—	—	m	—
δ_8	—	—	m	m

Логическое условие P_3 осуществляется сравнением константы $(\delta_9)=\varepsilon^2$ с $(\omega_3)=\|\Delta x^{(n)}\|^2$.

Очистку ω_3 лучше производить в начале программы.

Программа 4

Числа

$\alpha+l+(k-1)m$ $1\leqslant l\leqslant m$ $1\leqslant k\leqslant m$	a_{kl}
$\beta+k$ $1\leqslant k\leqslant m$	b_k
$\gamma+k$ $1\leqslant k\leqslant m$	$x_k^{(0)}$

δ_1	—	1	—	—
δ_2	×	$\alpha+m+1$	$\gamma+1$	ω_2
δ_3	—	—	1	—
δ_4	—	m	—	—
δ_5	×	$\alpha+m^2+1$	—	—
δ_6	—	m^2-1	m	—
δ_7	—	—	m	—
δ_8	—	—	m	m
δ_9	ε^2			
ω_1	0		$s_k^{(n)}$	
ω_2	раб.			
ω_3	0		$a_k^{(n)}$	

$(k-1)m$	k	—	

Команды

$k+1$	+	—	—	ω_3
$k+2$	+	—	—	ω_1
$k+3$	×	$\alpha+1$	$\gamma+1$	ω_2
$k+4$	+	ω_1	ω_2	ω_1
$k+5$	⊕	$k+3$	δ_1	$k+3$
$k+6$	⩽	$k+3$	δ_2	$k+3$
$k+7$	+	ω_1	$\beta+1$	ω_1
$k+8$	+	ω_1	$\gamma+1$	$\gamma+1$
$k+9$	×	ω_1	ω_1	ω_1
$k+10$	+	ω_3	ω_1	ω_3
$k+11$	⊕	$k+3$	δ_3	$k+3$
$k+12$	+	δ_2	δ_4	δ_2
$k+13$	⩽	δ_2	δ_5	$k+2$
$k+14$	⊖	$k+3$	δ_6	$k+3$
$k+15$	⊖	$k+7$	δ_7	$k+7$
$k+16$	⊖	$k+8$	δ_8	$k+8$
$k+17$	⩽	δ_9	ω_3	$k+1$
$k+18$	!			

$l+k\cdot m$	k	—

—	k	—
—	k	k

2. Решение задачи Дирихле для прямоугольника. Уравнение Лапласа

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0$$

с заданными граничными условиями на сторонах прямоугольника $0\leqslant x\leqslant a$, $0\leqslant y\leqslant b$ применением метода сеток можно свести к решению системы линейных уравнений

$$u(kh,\ ih)=\frac{1}{4}[u(kh,\ (i+1)h)+u(kh,\ (i-1)h)+$$
$$+u((k-1)h,\ ih)+u((k+1)h,\ ih)],$$

где $k=0,\ 1,\ 2,\ \ldots,\ K$, $i=0,\ 1,\ 2,\ \ldots,\ J$, которую удобно решать методом итераций:

$$u_{ki}^{(s+1)}=\frac{1}{4}\left[u_{k,\,i+1}^{(s)}+u_{k,\,i-1}^{(s)}+u_{k-1,\,i}^{(s)}+u_{k+1,\,i}^{(s)}\right],$$
$$\Delta_s=\max_{k,\,i}\left|u_{ki}^{(s+1)}-u_{k,\,i}^{(s)}\right|.$$

О степени точности решения можно судить по максимальной разности между двумя последовательными приближениями Δ_s. Исходные данные задачи (граничные значения) и начальные приближения в точках сети $u_{ki}^{(0)}$ расположим в ячейках

$$\alpha+k+iK,\quad k=0,\ 2,\ \ldots,\ K;\quad i=0,\ 1,\ \ldots,\ J.$$

Оператор счета по формулам $\mathbf{A}_{kis}$ зависит от трех параметров. Легко составляется схема программы

$$\mathbf{\Phi}_1(s)\,\mathbf{A}_{kis}\mathbf{F}_1(k)\,\mathbf{P}_1\uparrow^{\mathbf{A}}\mathbf{F}_2(i)\,\mathbf{\Phi}_2(k)\,\mathbf{P}_2\uparrow^{\mathbf{A}}\mathbf{F}_3(s)\,\mathbf{\Phi}_3(i)\,\mathbf{P}_3\uparrow^{\mathbf{A}}!$$

Расписать программу по командам предоставляем читателю.

3. Вычисление определителя невырожденной квадратной матрицы. Пусть $A=\{a_{ik}\}$ $(i=1,\ 2,\ \ldots,\ n;\ k=1,\ 2,\ \ldots,\ n)$ — квадратная матрица n-го порядка. Определитель матрицы D_A будем вычислять путем приведения матрицы A к диагональному виду. Вычитанием строк обращаем в нули все элементы матрицы, стоящие ниже главной диагонали. Предположим, что в процессе счета диагональные элементы не обращаются в нуль. Введем следующие операторы счета:

$\mathbf{C}_{ir}$ счет $C_{ir}=\dfrac{a_{ir}}{a_{rr}}$; $\quad r=2,\ 3,\ \ldots,\ n;\quad i=r+1,\ r+2,\ \ldots,\ n;$

$\mathbf{A}_{kir}$ счет $a_{ik}^{(r)}=a_{ik}^{(r-1)}-a_{rk}^{(r-1)}c_{ir}$; $\quad k=r,\ r+1,\ \ldots,\ n;$

$\mathbf{D}_r$ счет $d_r=d_{r-1}a_{rr}^{(r+1)}$, $\qquad d_0=1;\quad d_n=D_A.$

Схема программы имеет вид

$$\Phi_1(r)\,D_r C_{ir} A_{kir} F_1(k)\,P_1 \overset{A}{\uparrow} F_2(i,\ k)\,P_2 \overset{C}{\uparrow} F_3(r,\ i)\,P_3 \overset{D}{\uparrow} !$$

Заметим, что счет по параметру k можно проводить, начиная с $k = 1$, так как при $k < r$ будут происходить лишь вычитания нулей друг из друга, что не изменит результатов счета. Зато оператор формирования параметра k может упроститься. Начало счета по параметру i, начиная с $i = r + 1$, существенно, так как при $i = r$ произойдет вычитание r-й строки самой из себя, чего делать нельзя. Составление программы по данной схеме нам уже знакомо.

Упражнения

1. Составить программу решения дифференциального уравнения $\frac{dy}{dx} = f(x, y)$ по следующей схеме программы $\Phi A P_1 \overset{A}{\uparrow} B P_2 \overset{\Phi}{\uparrow} !$ (см. п. 1).

2. Составить схему программы и программу вычисления и печати таблицы значений многочлена $f(x + ih)$, $i = 0, 1, 2, \ldots, J$.

3. Составить программу разложения многочлена

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

по степеням $(x - a)$ с вычислением коэффициентов разложения b_k в ячейках $\beta + k$, $k = 0, 1, 2, \ldots, n$.

4. Как изменится программа упражнения 3, если коэффициенты многочлена a_k поместить в ячейках $\alpha + n - k$, $k = 0, 1, 2, \ldots, n$?

5. Составить программу умножения двух квадратных матриц n-го порядка

$$A = \{a_{ik}\}_1^n \quad \text{и} \quad B = \{b_{ik}\}_1^n.$$

4. Сложный циклический процесс.

1. Определение линии уровня функции двух переменных. Рассмотрим линию уровня непрерывной функции двух переменных

$$f(x, y) = c. \tag{4}$$

Предположим, что линия (4) разбивает плоскость на две связные области, в которых функция

$$\varphi(x, y) = f(x, y) - c \tag{5}$$

имеет разные знаки. В плоскости xOy проведем прямоугольную сетку координатных прямых с шагами Δx и Δy соответственно. Назовем прямоугольники сети, имеющие общие точки с линией (4), накрывающими. Заметим, что так как на любой *ЦАМ* нуль также имеет знак, то каждый из узлов, в том числе и те, для которых $\varphi(x, y) = 0$, имеет определенный знак.

Очевидно, что вопрос об определении линии уровня на машине дискретного действия состоит в отыскании вершин накрывающих прямоугольников при достаточно малых Δx и Δy.

Простейшим подходом к решению задачи на *ЦАМ* мог бы служить прием, при котором на каждой горизонтали (или вертикали) некоторого достаточно большого прямоугольника, размеры которого диктуются постановкой конкретной задачи, последовательно в каждом из узлов сети определяется знак φ, в случае, если при переходе от одного узла к другому знак узла изменился, считаем эти вершины искомыми и фиксируем их (печатаем). Затем после прохождения одной из вертикалей (горизонталей) можно предусмотреть автоматический переход к следующей и т. д. до полного исчерпания рассматриваемого прямоугольника, после чего расчеты заканчиваются. Разумеется, в машине с фиксированной запятой интересующая нас область значений переменных выбором масштабов преобразуется внутрь квадрата ($|x|<1$, $|y|<1$).

Однако объем всей выполняемой работы в таком случае значительно превосходит объем расчетов, связанных непосредственно с получением искомого результата (в квадрате с n шагами расчет производится в n^2 узлах, тогда как, например, для линии уровня, задаваемой функцией, однозначной по одной из координат, число пар соседних вершин накрывающих прямоугольников не превосходит n). Мы приведем два метода решения поставленной задачи, исключающие необходимость вычисления значений функции $\varphi(x, y)$ в каждом из узлов сети.

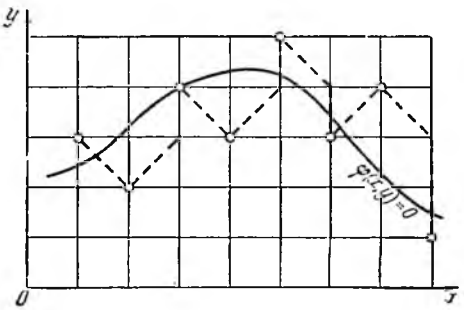


Рис. 16.

1) Пусть уравнение (4) определяет y как однозначную функцию от x на интервале (a, b). Кривая (4) определяет две области, в которых функция $z=\varphi(x, y)$ имеет противоположные знаки. Для определенности будем считать, что выше кривой (5) $z<0$, а ниже $z>0$.

Совокупность всех узлов накрывающих прямоугольников может быть полностью восстановлена, если на каждой из вертикалей будет определен один узел накрывающего прямоугольника, попеременно отрицательный или положительный (рис. 16). Для выделения соответствующих узлов накрывающих прямоугольников примем следующую схему вычис-

лений: вычисляем значения функции $\varphi(x, y)$ в одном из узлов прямоугольника и считаем «начальный» шаг Δy_0 вдоль оси Oy положительным, если выбранный узел положителен, и наоборот.

Дальнейшие вычисления значений функции $\varphi(x, y)$ производятся в вершинах, в которые попадает некоторая точка, движущаяся по системе узлов согласно правилам а) — д):

а) каждый последующий шаг Δx_{n+1} движущейся точки вдоль оси Ox равен нулю, если знаки данного n-го узла и предыдущего направления движения вдоль оси Oy совпадают; в противном случае этот шаг Δx_{n+1} равен Δx;

б) шаг Δy_{n+1} вдоль оси Oy равен предыдущему шагу, если шаг вдоль оси Ox равен нулю; этот шаг противоположен предыдущему по направлению и равен ему по величине в противном случае;

в) при достижении точкой границы $y = y_{max}$ и при отсутствии перемены знака узла точка перемещается в узел $(x + \Delta x, y_{min})$; Δy при этом сохраняет свой знак;

г) при пересечении вертикали $x = x_{max}$ расчеты прекращаются;

д) координаты каждого узла, из которого шаг вдоль оси Ox не равен нулю, фиксируются.

Нетрудно убедиться в том, что, начиная с одного из узлов рассматриваемого прямоугольника и продолжая вычисления согласно пп. а)—д), мы зафиксируем на каждой вертикали по одному узлу накрывающих прямоугольников.

Для составления программы проведем анализ условий движения.

Согласно а) — д) движение точки на n-м шаге должно определяться знаком предыдущего направления движения вдоль оси Oy и знаком функции $\varphi(x, y)$ в n-й точке $s_n = \operatorname{sign} \varphi(x_n, y_n)$. Для осуществления движения на n-м шаге необходимо определить величины Δx и Δy по значениям $\operatorname{sign} \Delta y_{n-1}$ и s_n в соответствии с а) — д).

Рассмотрим таблицу возможных вариантов движения:

$\operatorname{sign} \Delta y_{n-1}$	s_n	Δx_n	Δy_n	$\Delta y_{n-1} \cdot s_n$
+	+	0	Δy_{n-1}	+
−	−	0	Δy_{n-1}	+
+	−	Δx	$-\Delta y_{n-1}$	−
−	+	Δx	$-\Delta y_{n-1}$	−

Из таблицы видно, что значения Δx и Δy, а также фиксирование координат узлов накрывающих прямоугольников выбираются в соответствии со знаком произведения $d_n = \Delta y_{n-1} \cdot s_n$.

Разместим данные в ячейках *ЗУ*:

α_1	α_2	α_3	α_4	ω_1	ω_2	ω_3	ω_4	β_1
Δx_0	Δy_0	x_{max}	y_{max}	$x_0\,(x_n)$	$y_0\,(y_n)$	d_n	s_n	y_{min}

Блок-схема программы приведена на таблице 7.

Заметим, что в соответствии с условием для расчетов по данной программе требуется предварительный подсчет нулевого направления движения вдоль оси Oy, т. е. знака $\varphi(x_0, y_0)$. Этот предварительный подсчет легко может быть включен в программу (см. § 4).

2) Построение линии уровня по предыдущему методу позволяет определить в один прием лишь однозначную ветвь кривой. Естественно поэтому разработать прием, позволяющий определять линию уровня произвольной достаточно гладкой функции.

Предположим, что кривая $\varphi(x, y) = 0$ — непрерывная, не имеет точек самопересечения и разбивает плоскость на две связные области, в которых функция $z = \varphi(x, y)$ имеет противоположные знаки.

В плоскости xOy построим квадратную сеть с шагом под углом в 45° к осям координат. Из приведенных условий следует, что при достаточно малом h граница каждого накрывающего квадрата имеет только две точки пересечения с кривой (эти точки могут совпадать в узле сетки).

Рассмотрим следующие правила движения по сторонам накрывающих прямоугольников сети:

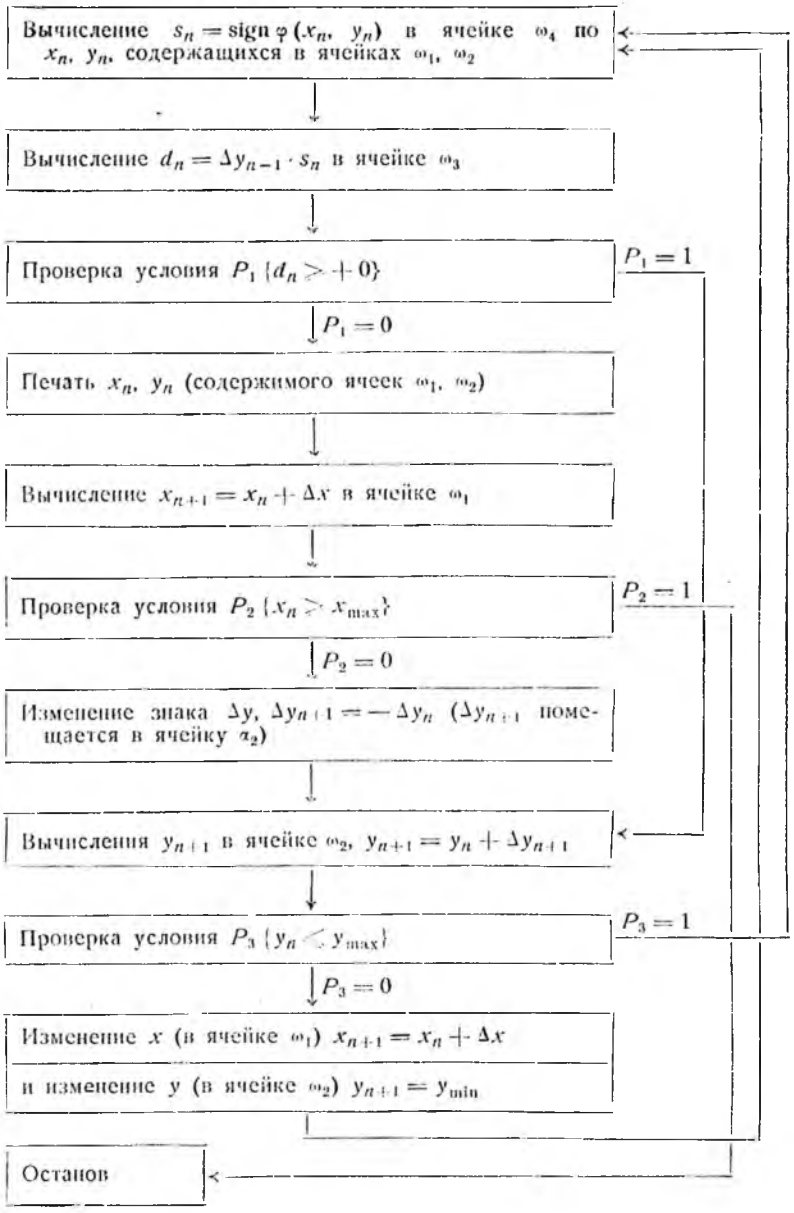
а) начальный шаг — произвольный переход по стороне квадрата сети с разнозначными узлами; направление движения на каждом последующем шаге изменяется на 90° по отношению к предыдущему направлению вправо или влево;

б) если на предыдущем шаге точка попала в узел со знаком +, то она поворачивает вправо;

в) если на предыдущем шаге точка перешла в узел со знаком —, то она поворачивает влево.

Нетрудно понять, что при движении по сторонам накрывающего квадрата точка обязательно перейдет на сторону соседнего накрывающего квадрата. Теперь заметим, что в силу связности областей, на которые разбивается плоскость заданной кривой при достаточно малом h, любые два узла одной из областей могут быть соединены между собой ломаной, проходящей по сторонам квадратов сети, лежащим в этой области. Такую совокупность узлов одинакового знака назовем *связной*.

Таблица 7


I
Вычисление $s_n = \operatorname{sign} \varphi(x_n, y_n)$ в ячейке ω_4 по x_n, y_n, содержащихся в ячейках ω_1, ω_2
II
Вычисление $d_n = \Delta y_{n-1} \cdot s_n$ в ячейке ω_3
III
Проверка условия $P_1 \{d_n \geqslant +0\}$
$P_1 = 1$
$P_1 = 0$
IV
Печать x_n, y_n (содержимого ячеек ω_1, ω_2)
V
Вычисление $x_{n+1} = x_n + \Delta x$ в ячейке ω_1
VI
Проверка условия $P_2 \{x_n \geqslant x_{\max}\}$
$P_2 = 1$
$P_2 = 0$
VII
Изменение знака Δy, $\Delta y_{n+1} = -\Delta y_n$ (Δy_{n+1} помещается в ячейку ω_2)
VIII
Вычисления y_{n+1} в ячейке ω_2, $y_{n+1} = y_n + \Delta y_{n+1}$
IX
Проверка условия $P_3 \{y_n \leqslant y_{\max}\}$
$P_3 = 1$
$P_3 = 0$
X
Изменение x (в ячейке ω_1) $x_{n+1} = x_n + \Delta x$
и изменение y (в ячейке ω_2) $y_{n+1} = y_{\min}$
XI
Останов

Можно доказать, что всякая связная совокупность отрицательных узлов, дополнение к которой также связно, может быть описана движением по условиям а) — в).

Для программирования движения вокруг связной совокупности отрицательных вершин проведем логический анализ условий а) — в).

Согласно условиям а) — в) движение точек на n-м шаге определяется:

1) предыдущим направлением движения вдоль Ox, $\operatorname{sign} \Delta x_{n-1}$;

2) предыдущим направлением движения вдоль Oy, $\operatorname{sign} \Delta y_{n-1}$;

3) знаком функции $\varphi(x, y)$ в n-й точке s_n.

Для определения движения на n-м шаге необходимо определить величины Δx_n и Δy_n по значениям $\operatorname{sign} \Delta x_{n-1}$, $\operatorname{sign} \Delta y_{n-1}$, s_n.

Рассмотрим таблицу возможных вариантов движения (см. таблицу 8).

Таблица 8

$\operatorname{sign} \Delta x_{n-1}$	$\operatorname{sign} \Delta y_{n-1}$	s_n	$\operatorname{sign} \Delta x_n$	$\operatorname{sign} \Delta y_n$	$\operatorname{sign} \Delta x_{n-1} \cdot \operatorname{sign} \Delta y_{n-1} \cdot s_n$
+	+	+	+	−	+
+	−	−	+	+	+
−	+	−	−	−	+
−	−	+	−	+	+
+	+	−	−	+	−
+	−	+	−	−	−
−	+	+	+	+	−
−	−	−	+	−	−

Из таблицы 8 получаем следующие формулы:

$$\operatorname{sign} \Delta x_n = (\operatorname{sign} \Delta x_{n-1} \cdot \operatorname{sign} \Delta y_{n-1} \cdot s_n) \operatorname{sign} \Delta x_{n-1} = \operatorname{sign} \Delta y_{n-1} \cdot s_n;$$

$$\operatorname{sign} \Delta y_n = -(\operatorname{sign} \Delta x_{n-1} \cdot \operatorname{sign} \Delta y_{n-1} \cdot s_n) \operatorname{sign} \Delta y_{n-1} = -\operatorname{sign} \Delta x_{n-1} \cdot s_n.$$

При составлении программы вычисления Δx_n и Δy_n по этим формулам заметим, что при каждом шаге изменяется знак только одной из величин Δx или Δy:

$$\Delta y_{n+1} = -\Delta y_n, \qquad \Delta x_{n+1} = \Delta x,$$

если $\operatorname{sign} \Delta x_{n-1} \cdot \operatorname{sign} \Delta y_{n-1} \cdot s_n$ имеет знак плюс, и

$$\Delta y_{n+1} = \Delta y_n \qquad \Delta x_{n+1} = -\Delta x_n$$

в противном случае.

Приведем здесь лишь часть программы, осуществляющую вычисления Δx_n и Δy_n.

	Числа		Команды				
α_1	Δx_{n-1}	$N+1$	$\times$	α_1	α_2	α_4	$\Delta x_{n-1} \cdot \Delta y_{n-1}$
α_2	Δy_{n-1}	$N+2$	$\times$	α_3	α_4	α_4	
α_3	s_n	$N+3$	$\leqslant$	α_4	γ_1	$N+6$	
α_4	$\operatorname{sign} \Delta x_{n-1} \cdot s_n$	$N+4$	—		α_1	α_1	
γ_1	-0	$N+5$	—		α_2	α_2	
		$N+6$	—		α_1	α_1	

Программа должна быть дополнена командами, осуществляющими достижение первого накрывающего квадрата (например, по методике, изложенной в п. 1), командами, выполняющими вычисление s_n, печать координат вершин накрывающих прямоугольников при пересечении кривой, и командами, определяющими прекращение вычислений. Составление операторной схемы предоставляем читателю.

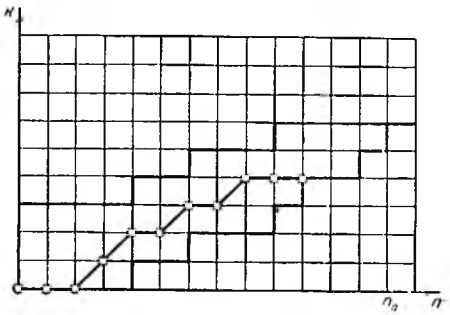


Рис. 17.

2. Решение одной краевой задачи для разностного уравнения параболического типа. Оптимальный метод приемочного статистического контроля состоит в следующем: задается контрольная область*) (рис. 17); если проверенное изделие оказывается годным, то откладывается единичный отрезок вправо, если оно оказывается негодным — вправо и вверх и т. д. Достижение верхней границы области означает необходимость браковки всей партии, достижение нижней — ее прием. При попадании во внутренние точки области испытания продолжаются.

*) Постановку задачи см. в статье Н. С. Михалевича, «Последовательные байесовские решения и оптимальные методы приемного статистического контроля», Теория вероятностей и ее применения, т. I, вып. 4, 1956, 437—465.

Граница области приемки обладает следующими геометрическими свойствами:

1) граница области — графики двух ступенчатых (нижняя и верхняя) монотонных неубывающих функций;

2) вертикальные скачки границ не превышают единицы;

3) правее вертикали n_0 внутренних точек нет;

4) при $n < n_0$ не могут существовать вертикали, на которых отсутствуют внутренние точки.

Расчеты таблиц для оптимальных методов статистического приемочного контроля сводятся к решению следующего неоднородного разностного уравнения:

$$\rho(k, n) = M(k, n)\,\rho(k+1, n+1) + (1 - M(k, n))\,\rho(k, n+1) + c \quad (6)$$

с неизвестной границей и к определению этой границы при условии, что на ней решение $\rho(k, n)$ уравнения (6) совпадает с известными решениями соответствующего однородного разностного уравнения:

$$\rho(k, n) = M(k, n) \text{ на нижней границе,}$$
$$\rho(k, n) = p_0 \text{ на верхней границе.}$$

Будем считать известными значения $M(k, n)$ при заданных k и n, а также $k_{\max}$ и n_0.

Функцию $\rho(k, n)$, определенную во внутренних точках искомой области и на ее границе, доопределим для точек прямоугольника Π $(0 \leqslant n \leqslant n_0;\ 0 \leqslant k \leqslant k_{\max})$, полагая

$$\rho(k, n) = M(k, n)$$

для «нижних» точек, расположенных под нижней границей, и

$$\rho(k, n) = p_0$$

для «верхних» точек, расположенных в прямоугольнике над верхней границей.

Введем в рассмотрение функцию $\rho_1(k, n)$, удовлетворяющую во всех точках прямоугольника Π уравнению

$$\rho_1(k, n) = M(k, n)\,\rho(k+1, n+1) + (1 - M(k, n))\,\rho(k, n+1) + c,$$

и обозначим через $m(k, n)$ величину

$$m(k, n) = \min(M(k, n), p_0). \quad (7)$$

Тогда очевидно, что для всех точек прямоугольника Π

$$\rho(k, n) = \min(m(k, n), \rho_1(k, n)). \quad (8)$$

При этом имеют место соотношения:

а) если $\rho = \rho_1 < m$, то точка — внутренняя,

б) если $\rho = m$, точка — внешняя и, кроме того,

в) если $\rho = M < p_0$, точка — нижняя,

г) если $\rho = p_0$, точка — верхняя.

Таким образом, задача состоит в определении координат (k, n) внутренних точек и значений функции $\rho(k, n)$ в них при условиях а) — г) с помощью рекуррентных соотношений:

1. $M(k, n)$,
2. $\rho_1(k, n) = M(k, n)\,\rho(k+1, n+1) + (1 - M(k, n))\,\rho(k, n+1) + c$,
3. $m(k, n) = \min(M(k, n), p_0)$,
4. $\rho(k, n) = \min(m(k, n), \rho_1(k, n))$.

Наиболее естественным подходом к выполнению указанных расчетов по этим формулам является вычисление в целочисленных точках по вертикалям снизу вверх функции $\rho(k, n)$ по $\rho(k, n+1)$ и $\rho(k+1, n+1)$ от точки $k = 0$, $n = n_0$ до первой верхней точки, после чего производится переход к вычислениям на следующей слева вертикали. Начало счета n_0 — вертикаль, на которой значения функции $\rho(k, n)$ задаются (или специально подсчитываются) и помещаются в специально отведенные ячейки

$$\alpha+0,\ \alpha+1,\ \alpha+2,\ \ldots,\ \alpha+k_{\max}.$$

Число ячеек, необходимых для запоминания ρ, определяется максимальной «высотой» области $(k_{\max}+1)$.

Цикличность вычислений обеспечивается помещением вычисляемых значений функции $\rho(k, n)$ в ячейку $\alpha + k$, в которой находилось уже использованное в расчетах значение величины $\rho(k, n+1)$.

Расчеты заканчиваются на нулевой вертикали $(n = 0)$.

В каждой точке на основании условий а) — г) выясняется, какой категории точек принадлежит точка (k, n):

если точка (k, n) — нижняя, запоминаем значение $\rho(k, n)$ для вычислений на $n - 1$ вертикали и переходим к вычислениям в точке $(k+1, n)$ (к «новой точке»);

если точка (k, n) — внутренняя, выдаем (печатаем) ее координаты и значение функции $\rho(k, n)$, которое, кроме того, также запоминается, и переходим к вычислениям в точке $(k+1, n)$ (к «новой точке»);

если точка (k, n) — верхняя, переходим к вычислениям в точке $(0, n-1)$ к «новой вертикали»; значение функции $\rho(k, n)$ при этом равно p_0 и не требует специального запоминания.

3. Экономия ячеек памяти и времени расчетов. Однако данная методика счета требует запоминания значений функции $\rho(k, n)$ во всех точках каждой вертикали до первых верхних и охватывает счет по не интересующей нас области нижних точек, расположенной между осью абсцисс и нижней границей искомой области.

Используя геометрические свойства границы области, можно предложить следующую экономную методику счета.

Счет на каждой n-й вертикали производится, начиная k^*-й точки (k^*, n), где k^* — ордината самой верхней из нижних точек на вертикали $n+1$, и продолжается до первой верхней точки, после чего производится переход на вертикаль $n-1$. Вычисления начинаются в точке (k_{max}, n_0) и заканчиваются в точке $(0, 0)$.

В таком случае для вычислений по рекуррентным формулам 1—4 необходимо запоминать значения функции $\rho(k, n)$ только во внутренних точках. Программа вычисления функции $\rho_1(k, n)$ в точке (k, n) строится на стандартных ячейках, в которые с помощью операторов засылки $З_1$ и $З_2$ пересылаются вычисленное на предыдущей вертикали значение $\rho(k+1, n+1)$ и использованное при вычислениях в предыдущей точке значение $\rho(k, n+1)$; последнее в начальной точке (k^*, n) каждой вертикали вычисляется по формуле

$$\rho(k^*, n+1) = M(k^*, n+1),$$

поскольку точка $(k^*, n+1)$ — нижняя.

Запоминание функции ρ на каждой вертикали производится, начиная с первой внутренней точки, в ячейке $\alpha + 0$ и т. д.

Количество ячеек для запоминания значений $\rho(k, n)$ теперь равно максимальной «ширине» области.

4. Сокращение объема выводимых данных. При расчетах на *ЦАМ* можно осуществить следующий сокращенный вывод данных, сохраняющий всю необходимую информацию:

1) если на вертикали нет внутренних точек, печатается условный знак «вертикаль»;

2) печатается ордината только первой внутренней точки;

3) на каждой вертикали во внутренних точках значения печатаются в порядке возрастания ординат точек;

4) вывод информации о каждой вертикали производится последовательно, начиная от вертикали n_0 и кончая нулевой вертикалью.

Как удовлетворить требованиям 3) — 4) очевидно. Поэтому остановимся лишь на выполнении требований 1) — 2).

Для осуществления первого требования необходимо выработать признак, позволяющий различать вертикали с внутренними точками от вертикалей без внутренних точек. Для второго — необходим признак для различения первой внутренней точки на данной вертикали от остальных внутренних точек.

Поскольку наличие на вертикали внутренних точек влечет за собой встречу с первой внутренней точкой и наоборот, то для осуществления передач управления согласно 1) и 2) достаточно в некоторую ячейку β помещать в начале каждой верти-

кали «— 0» и сохранять это значение неизменным до встречи с первой внутренней точкой, после чего в ячейку β помещать «+ 0». Тогда интересующие нас свойства точек будут определяться знаком ячейки β:

если данная точка — внутренняя первая, β содержит — 0;

если данная точка — внутренняя не первая, β содержит + 0;

если пройденная вертикаль не содержала внутренних точек, β содержит —0;

если пройденная вертикаль содержала внутренние точки, β содержит + 0.

Введем обозначения:

I. Операторы счета:

$\mathbf{A}_1$ — вычисление функции M в точках $(k^*, n+1)$,

$\mathbf{A}_2$ — вычисление функции M в точке (k, n).

$\mathbf{A}_3$ — вычисление функций $\rho_1(k, n)$, $m(k, n)$, $\rho(k, n)$,

$\mathbf{A}_4$ — подготовка к счету в следующей точке $(k+1)$,

$\mathbf{A}_5$ — подготовка k^* для начала счета на следующей $(n-1)$-й вертикали.

II. Операторы засылки:

$\mathbf{З}_1$ — засылка $\rho(k+1, n+1)$ в стандартную ячейку (для оператора $\mathbf{A}_3$ — вычисление $\rho_1(k, n)$),

$\mathbf{З}_2$ — запоминание $\rho(k, n)$ в стандартной ячейке для вычислений на следующей вертикали,

$\mathbf{З}_3$ — засылка «—0» в ячейку β,

$\mathbf{З}_4$ — засылка $\rho(k, n+1)$ в стандартную ячейку (для оператора $\mathbf{A}_3$).

III. Операторы переадресации:

Π_1 — переадресация оператора $З_1$;

Π_2 — переадресация оператора $З_2$.

IV. Логические условия:

1. Разделение точек на внутренние и внешние:

$$P_1 = \begin{cases} 1, \text{ если точка — внешняя } (\rho = m); \text{ управление передается оператору } \mathbf{P}_3; \\ 0, \text{ если точка — внутренняя; управление передается оператору } \mathbf{P}_2. \end{cases}$$

2. Разделение внутренних точек на первую и последующие:

$$P_2 = \begin{cases} 0, \text{ если внутренняя точка — первая; управление передается оператору } \mathbf{B}_1, \beta = -0; \\ 1, \text{ в противном случае; управление передается оператору } \mathbf{B}_2, \beta = +0. \end{cases}$$

3. Разделение внешних точек на верхние и нижние:

$$P_3 = \begin{cases} 0, \text{ если внешняя точка — верхняя, управление передается оператору } \mathbf{P}_4; \\ 1, \text{ в противном случае; управление передается оператору } \mathbf{A}_4. \end{cases}$$

4. Разделение вертикалей на имеющие и не имеющие внутренние точки:

$$P_4 = \begin{cases} 0, \text{ если на вертикали не было внутренних точек; управление передается оператору } \mathbf{B}_3,\ \beta = -0; \\ 1, \text{ в противном случае; управление передается оператору } \mathbf{P}_5,\ \beta = +0. \end{cases}$$

5. Определение окончания расчетов:

$$P_5 = \begin{cases} 0, \text{ если } n = 0; \text{ управление передается команде «останов»}; \\ 1, \text{ если } n > 0; \text{ управление передается оператору } \mathbf{\Phi}_2. \end{cases}$$

V. Операторы формирования:

$\mathbf{\Phi}_1$ — формирование начальных данных;

$\mathbf{\Phi}_2$ — формирование операторов $\mathbf{З}_2$ и $\mathbf{З}_4$ и β.

VI. Вывод данных:

$\mathbf{B}_1$ — печать k,

$\mathbf{B}_2$ — печать ρ,

$\mathbf{B}_3$ — печать условного знака «вертикаль без внутренних точек».

В принятых обозначениях схема программы имеет вид

$$\mathbf{\Phi}_1\mathbf{\Phi}_2\mathbf{A}_1\mathbf{A}_2\mathbf{З}_1\mathbf{A}_3\mathbf{З}_2\mathbf{P}_1 \underset{\mathbf{P}_3}{\downarrow} \mathbf{P}_2 \underset{\mathbf{B}_2}{\downarrow} \mathbf{B}_1\mathbf{B}_2\mathbf{B}_3\mathbf{П}_1\underbrace{\mathbf{П}_2\mathbf{З}_4\mathbf{A}_4\mathbf{P}^0}_{(2)} \underset{\mathbf{A}_2}{\downarrow} *$$

$$* \mathbf{P}_3 \underset{\mathbf{A}_4}{\downarrow} \mathbf{P}_4 \underset{\mathbf{P}_5}{\downarrow} \mathbf{B}_3\mathbf{P}_5 \underset{\mathbf{\Phi}_2}{\downarrow} \text{«ост.»}\ \mathbf{A}_5\underbrace{\mathbf{П}_2\mathbf{З}_4\mathbf{A}_4\mathbf{P}^0}_{(1)} \underset{\mathbf{A}_2}{\downarrow}$$

где * означает перенос формулы.

При рассмотрении схемы программы можно заметить, что подчеркнутая в конце группа оператора (1), оканчивающаяся оператором безусловного перехода к оператору $\mathbf{A}_2$, совпадает с подчеркнутой группой операторов (2), также заканчивающейся оператором безусловного перехода к оператору $\mathbf{A}_2$. Поэтому можно упростить схему программы, поместив после оператора $\mathbf{A}_5$ оператор безусловного перехода к оператору $\mathbf{П}_2$. Схема программы принимает вид

$$\mathbf{\Phi}_1\mathbf{\Phi}_2\mathbf{A}_1\mathbf{A}_2\mathbf{З}_1\mathbf{A}_3\mathbf{З}_2\mathbf{P}_1 \underset{\mathbf{P}_3}{\downarrow} \mathbf{P}_2 \underset{\mathbf{B}_2}{\downarrow} \mathbf{B}_1\mathbf{B}_2\mathbf{B}_3\mathbf{П}_1\mathbf{П}_2\mathbf{З}_4\mathbf{A}_4\mathbf{P}^0 \underset{\mathbf{A}_2}{\downarrow} \mathbf{P}_3 \underset{\mathbf{A}_5}{\downarrow} *$$

$$* \mathbf{P}_4 \underset{\mathbf{P}_5}{\downarrow} \mathbf{B}_3\mathbf{P}_5 \underset{\mathbf{\Phi}_2}{\downarrow} \text{«ост.»}\ \mathbf{A}_5\mathbf{P}^0 \underset{\mathbf{П}_2}{\downarrow}$$

§ 4. Логические условия в схемах программ

Изменение места логического условия в программе по отношению к операторам, работу которых это условие определяет, весьма существенно. Так, если логическое условие стоит после оператора, то последний будет действовать по крайней

мере один раз, если же логическое условие стоит перед оператором, то оператор может не применяться.

Рассмотрим схему с разветвлением. Пусть необходимо решить дифференциальное уравнение первого порядка с переменной правой частью:

$$\frac{dy}{dx} = \varphi(x),$$

$$\varphi(x) = \begin{cases} f_1(x), & \text{если } x \leqslant a, \\ f_2(x), & \text{если } x > a \end{cases} \quad (y(x_0) = y_0,\ x_0 \leqslant x < X).$$

Схема счета:

1) счет $\varphi(x_n)$ по первой $(f_1(x_n))$ или по второй $(f_2(x_n)$ формуле;

2) $y_{n+1} = y_n + h\varphi(x_n)$; $x_{n+1} = x_n + h$.

Для реализации этой схемы счета мы должны ввести логическое условие P_1, которое обеспечит работу оператора счета $\mathbf{A}_1$ (вычисление $f_1(x_n)$) или оператора счета $\mathbf{A}_2$ (вычисление $f_2(x_n)$). Введем логическое условие $P_1(x)$, принимающее два значения: «да» — обозначим числом 1 и «нет» — обозначим числом 0, по правилу:

$$P_1(x) = \begin{cases} 0, & x \leqslant a, \\ 1, & x > a. \end{cases}$$

$$\mathbf{P}_1$$

Тогда схема $\mathbf{P}_1(x)\ \mathbf{A}_1 \downarrow$ будет обеспечивать работу оператора $\mathbf{A}_1$ в соответствии со схемой счета. Для обеспечения работы оператора $\mathbf{A}_2$ в соответствии со схемой счета введем логическое условие $P_2(x)$ по правилу

$$P_2(x) = \begin{cases} 0, & x > a, \\ 1, & x \leqslant a. \end{cases}$$

Логическое условие $P_2(x)$ совпадает с отрицанием логического условия P_1, что мы запишем так: $P_2(x) = \overline{P}_1(x)$. Стрелку логического условия $\overline{P}_1$ будем помечать индексом 1.

Итак, схема программы данной задачи имеет вид

$$\begin{array}{cccccccc} 0 & \overline{\mathbf{P}}_1 & 1 & \mathbf{B} & \overline{1} & & \mathbf{P}_1 & \\ \downarrow \mathbf{P}_1 & \uparrow \mathbf{A}_1 & \downarrow \overline{\mathbf{P}}_1 & \uparrow \mathbf{A}_2 & \downarrow \mathbf{B} & \mathbf{P}_0 & \uparrow & !, \end{array}$$

где $\mathbf{B}$ — оператор счета x_{n+1} и y_{n+1}, P_0 — логическое условие, определяющее необходимость повторения циклов N раз

$$\left(N = \frac{X - x_0}{h}\right).$$

Схема программы может быть преобразована. Стрелку $\overset{1}{\downarrow}$ можно поставить после логического условия $\overline{P}_1$, так как при значении логического условия $P_1 = 0$ всегда $\overline{P}_1 = 1$. Но тогда логическое условие $\overline{P}_1$ можно заменить на тождественное логическое условие P_2^0, т. е. $P_2^0 \equiv 1$.

Таким образом, схема программы принимает вид

$$\begin{array}{l} 0 \quad \mathbf{A}_2 \quad \mathbf{B}\ 1 \quad 2 \quad \mathbf{P}_1 \\ \downarrow \mathbf{P}_1 \uparrow \mathbf{A}_1 \mathbf{P}_2^0 \uparrow \downarrow \mathbf{A}_2 \downarrow \mathbf{B} \mathbf{P}_0 \uparrow ! \end{array}$$

Составление программы по данной схеме несложно.

Программа 5

Числа

α_1	x_0	x_n	
α_2	y_0	y_n	
α_3	a		
α_4	h		
α_5	X		
ω	рабочая		

Команды

$k+1$	$\leqslant$	α_1	α_3	$k+4$	$\mathbf{P}_1$
$k+2$	$f_1(\alpha_1)$	—	—	ω	$\mathbf{A}_1$
$k+3$	$\lvert\leqslant\rvert$	—	—	$k+5$	$\mathbf{P}_2$
$k+4$	$f_2(\alpha_1)$	—	—	ω	$\mathbf{A}_2$
$k+5$	$\times$	α_4	ω	ω	
$k+6$	$+$	ω	α_2	α_2	$\mathbf{B}$
$k+7$	$+$	α_1	α_4	α_1	
$k+8$	$\leqslant$	α_1	α_5	$k+1$	$\mathbf{P}_0$
$k+9$	*ост.*				

Здесь команды $k+2$ и $k+4$ — символические команды вычисления функций f_1 и f_2 соответственно.

Рассмотрим решение на машине с фиксированной запятой дифференциального уравнения первого порядка

$$\frac{dy}{dx} = \frac{\varphi(x, y)}{\psi(x, y)}$$

с начальным условием $y(x_0) = y_0$ $(x_0 \leqslant x \leqslant X)$. Примем следующую схему счета.

I. Если $\varphi_k = \varphi(x_k, y_k) > \psi_k = \psi(x_k, y_k)$, то вычисления ведутся по формулам

$$A^1 \quad y_{k+1} = y_k + h, \quad x_{k+1} = x_k + 2h\frac{\psi_k}{2\varphi_k}.$$

II. Если $\varphi_k \leqslant \psi_k$, то вычисляем по формулам

$$A^{II} \quad x_{k+1} = x_k + h, \quad y_{k+1} = y_k + 2h\frac{\varphi_k}{2\psi_k}.$$

Введем логическое условие P_1:

$$P_1 = \begin{cases} 0, & \varphi_k \leqslant \psi_k, \\ 1, & \varphi_k > \psi_k. \end{cases}$$

Введем следующие обозначения: Φ — оператор счета $\varphi_k = \varphi(x_k, y_k)$; Ψ — оператор счета $\psi = \psi(x_k, y_k)$; P_2 — логическое условие, реализующее циклический процесс по параметру k на интервале $x_0 \leqslant x \leqslant X$:

$$P_2 = \begin{cases} 0, & x \leqslant X, \\ 1, & x > X. \end{cases}$$

Аналогично предыдущему получим схему программы

$$\begin{array}{cccccccc} 2 & & \mathbf{A}^{II} & & \mathbf{P}_2 1 & & 3 & \Phi \\ \downarrow \Phi\Psi\mathbf{P}_1 & \uparrow & \mathbf{A}^{I}\mathbf{P}_3^0 & \uparrow & \downarrow \mathbf{A}^{II} & \downarrow & \mathbf{P}_2 & \uparrow ! \end{array}$$

Составление программы предоставляем читателю.

Обратим внимание на то, что в данной задаче работа логического условия P_1 подготавливается в процессе счета по программе. Таким образом, логические условия, используя простые исходные данные о задаче, осуществляют управление сложным вычислительным процессом.

Рассмотрим вычисление определителя невырожденной квадратной матрицы с выбором максимальных элементов (см. § 3, стр. 202).

Если при вычислении определителя (см. стр. 206) элементы главной диагонали могут обращаться в нуль, или могут быть малыми по абсолютной величине, то вычисление определителя по программе § 3 невозможно или ведет к большой потере точности. Для устранения этого недостатка будем в каждой строке определять максимальный по модулю элемент и обращать в нуль все элементы, стоящие под ним, вычитая из последующих строк величины, пропорциональные элементам данной строки.

Введем оператор $\mathbf{H}_{kr}$, определяющий $m_{kr} = \max\{m_{k-1,r}, |a_{kr}|\}$, $m_{0r} = 0$ в ячейке v_1.

В исходной схеме счета изменяются операторы $\mathbf{D}_r$ и $\mathbf{C}_{ir}$.

Для реализации оператора $\mathbf{D}_r$ необходимо: 1) запомнить максимальный по модулю элемент строки a_{rk_r} в некоторой фиксированной ячейке v_2, 2) образовать число $(-1)^{k}$ и

запомнить $(-1)^{k_r}$ в некоторой ячейке v_3, 3) вычислить $d_0 = (-1)^{\frac{n(n+1)}{2}}$. Тогда оператор $\mathbf{D}_r$ считает

$$d_r = d_{r-1} a_{rk_r} \cdot (-1)^{k_r}; \quad d_0 = (-1)^{\frac{n(n+1)}{2}}; \quad d_n = D_A$$

и реализуется двумя командами:

$D+1$	$\times$	v_0	v_3	v_3
$D+2$	$\times$	v_2	v_3	v_0

в ячейке v_0 находится d_0.

Оператор $\mathbf{C}_{ir}$ ведет счет $C_{ik_r} = \frac{a_{ik_r}}{a_{rk_r}}$.

Для его реализации необходимо иметь число

—	k_r	—	—

в ячейке v_4. Программа оператора $\mathbf{C}_{ir}$ имеет вид

$c+1$	$\oplus$	$c+2$	v_4	$c+2$
$c+2$	:	$a+n+0$	v_2	ρ

Введем оператор $\mathbf{U}_{k_r}$, который запоминает a_{rk_r} в ячейке v_2, запоминает $(-1)^{k_r}$ в ячейке v_3 и запоминает

—	k_r	—	—

в ячейке v_4, и оператор $\mathbf{B}_k$, который вычисляет $(-1)^k$ и числа

—	k	—	—

в ячейках v_5 и v_6 соответственно.

Схема программы дополнится теперь схемой выбора максимальных элементов и подготовки работы операторов $\mathbf{D}_r$ и $\mathbf{C}_{ir}$:

$$\overset{5}{\downarrow} \mathbf{B}_k \mathbf{H}_{kr} \mathbf{U}_{k_r} \mathbf{F}(k) \mathbf{P}_5 \overset{\mathbf{B}}{\uparrow}.$$

Окончательно схема программы вычисления определителя невырожденной квадратной матрицы с выбором максимальных элементов имеет вид

$$\Phi_1(r)\overset{3}{\downarrow}\Phi_2(k)\overset{5}{\downarrow}\mathbf{B}_k\mathbf{H}_{kr}\mathbf{U}_{k_r}\mathbf{F}_1(k)\,\mathbf{P}_5\overset{\mathbf{B}}{\uparrow}\mathbf{D}_r\overset{2}{\downarrow}\mathbf{C}_{ir}\overset{1}{\downarrow}\mathbf{A}_{kir}*$$
$$*\,\mathbf{F}_2(k)\,\mathbf{P}_1\overset{\mathbf{A}}{\uparrow}\mathbf{F}_3(i,k)\,\mathbf{P}_2\overset{\mathbf{C}}{\uparrow}\mathbf{F}_4(r,i)\,\mathbf{P}_3\overset{\Phi_1}{\uparrow}! \qquad (9)$$

Составим программу только по дополнительной схеме. Введем оператор **Q**, определяющий модуль числа a, находящегося в ячейке α и пересылающий этот модуль в ячейку β. Предположим, что оператор **Q** можно реализовать одной командой

| |	α	—	β	

Тогда программа $\mathbf{H}_{kr}$ имеет вид

$H+1$	$\|\leqslant\|$	$\alpha+n+1$	v_2	$H+3$	$r\cdot n+k$	—	—
$H+2$	| |	$\alpha+n+1$	—	v_1	$r\cdot n+k$	—	—

Логическое условие $P_{\text{н}}$, осуществляемое первой командой оператора $\mathbf{H}_{kr}$, принимает значение по правилу

$$P_{\text{н}}=\begin{cases}0, & m_{kr}<|a_{k+1,\,r}|,\\ 1, & m_{kr}\geqslant|a_{k+1,\,r}|.\end{cases}$$

С помощью этого логического условия обеспечивается запоминание максимального по модулю элемента строки. Легко понять, что это же логическое условие $P_{\text{н}}$ можно использовать для запоминания a_{rk_r} в заданной ячейке v_2. Более того, программа оператора $\mathbf{U}_r$ полностью может быть построена с использованием логического условия $P_{\text{н}}$, а именно:

Числа

v_2	0	a_{rk_r}
v_3	—	$(-1)^{k_r}$
v_4	0	| — | k_r | — | — |
v_5	—1	$(-1)^k$
v_6	0	| — | k | — | — |

Команды

$k+1$	$\|\leqslant\|$	$\alpha+n+1$	v_2	$k+6$	$r\cdot n+k$	—	—
$k+2$	| |	$\alpha+n+1$	—	v_2	$r\cdot n+k$	—	—
$k+3$	+	$\alpha+n+1$	—	v_1	$r\cdot n+k$	—	—
$k+4$	+	v_5	—	v_3			
$k+5$	+	v_6	—	v_4			

Составление программы по схеме (9) легко завершается (см. программу 6).

Программа 6

Числа

v_0	d_0	
v_1	—	a_{rk_r}
v_2	0	$m^{k_r}_{rk_r}$
v_3	—	$(-1)^{k_r}$
v_4	—	— | k_r | — | —
v_5	1	$(-1)^k$
v_6	0	— | k | — | —
v_7	—1	
v_8	—1 — —	
v_9	$-n$ — —	

Команды

B	$k+1$	$\times$	v_5	v_7	v_5
	$k+2$	$+$	v_6	v_8	v_6
H	$k+3$	$\|\leqslant\|$	$a+n+1$	v_2	$k+8$
	$k+4$	$\|\ \|$	$a+n+1$	—	v_2
U	$k+5$	$+$	$a+n+1$	—	v_1
	$k+6$	$+$	v_5	—	v_3
	$k+7$	$+$	v_6	—	v_4
	$k+8$	$\oplus$	$k+3$	v_8	$k+3$
	$k+9$	$\oplus$	$k+4$	v_8	$k+4$
$F_1(k)$	$k+10$	$\oplus$	$k+5$	v_8	$k+5$
P_5	$k+11$	$\leqslant$	v_6	v_9	$k+1$

$r\cdot n+k$	—	—
$r\cdot n+k$	—	—
$r\cdot n+k$	—	—

Программа 27 (стр. 152) позволяет вычислять либо $\sin x$, либо $\cos x$, в зависимости от того, с какой команды начинается работа программы.

С помощью логического переменного можно составить программу счета $\sin x$ и $\cos x$ с одним входом. Пусть логическое условие P_1 принимает значения по правилу

$$P_2 = \begin{cases} 1, & \text{если нужно считать } \sin x, \\ 0, & \text{» » » } \cos x. \end{cases}$$

Схема программы

$$\begin{array}{c} \Phi_C \quad A\,2 \quad 0\,1 \quad A \\ P_2 \uparrow \Phi_s P_0 \uparrow \downarrow \Phi_C \downarrow \downarrow A P_1 \uparrow ! \end{array} \qquad (10)$$

имеет общее начало для счета $\sin x$ и $\cos x$.

Для реализации логического условия P_2 перед каждым обращением к подпрограмме счета $\sin x$ или $\cos x$ поместим в ячейке β число -0, если нужно считать $\sin x$, и $+0$, если нужно считать $\cos x$. Тогда логическое условие P_2 реализуется командой

P_2	$\leqslant$	$-$	β	Φ_C

Число в ячейке β однозначно определяет значение логического условия P_2. В дальнейшем содержание ячейки β, принимающее только два значения и определяющее значение логического условия, будем называть *логическим переменным*.

Упражнения

1. Построить логическое переменное в ячейке *ЗУ*, соответствующее логическому условию P_H программы.

2. Объединить счет $\sin x$ и $\cos x$ в программе без использования тождественного логического условия P_0.

3. Составить программу вычисления модуля числа, находящегося в ячейке α, с помещением результата в ячейку β, используя только команды арифметических операций и сравнение.

§ 5. Программирование логических операторов

Программы некоторых логических операторов состоят из отдельных команд передачи управления. Перечислим логические условия, программы которых представляют собой отдельные команды передачи управления, приведенные в главе III.

Логическое условие	*Программа*			
1. $P\{(a) \leqslant (b)\}$	$\leqslant$	a	b	A
2. $P\{\lvert(a)\rvert \leqslant \lvert(a)\rvert\}$	$\lvert\leqslant\rvert$	a	b	A
3. $P\{(a) \neq (b)\}$	$\neq$	a	b	A
4. $P\{(a) = (b)\}$	$=$	a	b	A
5. $P\{(a) \geqslant (b)\}$	$\geqslant$	a	b	A
6. $P\{(a) \geqslant +0\}$	*УПЧ*	a	A	B

Здесь A — код оператора, которому передается управление при выполнении соответствующего логического условия; B — код оператора, которому передается управление, если логическое условие не выполнено. При этом будем называть кодом оператора номер его начальной команды. Через (a) обозначено число, содержащееся по адресу a.

Как уже отмечалось, в различных машинах обычно существуют разные операции передачи управления. Однако можно составить программу любого логического оператора, ограничиваясь только одной из указанных операций. Воспользуемся, например, с этой целью операцией передачи управления по числу «*УПЧ*» и рассмотрим несколько примеров.

1. Программа логического условия $P_1\{(a) \leqslant -0\}$, т. е. отрицания логического условия $P\{(a) \geqslant +0\}$, имеет вид

УПЧ	a	B	A

2. Логическое условие $P_6\{(a) > -0\}$ равнозначно условию $P\{(a) \geqslant +0\}$ и реализуется программой

УПЧ	a	A	B

3. Логическое условие $P_2\{(a) \leqslant (b)\}$ равнозначно условию $P\{(a) - (b) \leqslant -0\}$, программа которого имеет вид

N	—	a	b	c
$N+1$	*УПЧ*	c	B	A

4. Логическое условие $P_4\{|(a)| \leqslant |(b)|\}$ равнозначно условию $P\{|(a)| - |(b)| \leqslant -0\}$, программа которого имеет вид

N	$\|-\|$	a	b	α
$N+1$	*УПЧ*	α	B	A

5. Логическое условие $P_5\{(a) \neq (b)\}$ равнозначно условию $P\{(a-b)(b-a) \not< -0\}$ и может быть реализовано программой

N	—	a	b	α
$N+1$	—	b	a	β
$N+2$	×	α	β	α
$N+3$	*УПЧ*	α	B	A

6. Логическое условие $P_6\{(a) > (b)\}$ является отрицанием логического условия $P\{(a) \leqslant (b)\}$ и программируется приказами

$N+1$	—	a	b	c
$N+2$	*УПЧ*	c	A	B

7. Логическое условие $P_7\{(a) = (b)\}$ равнозначно отрицанию условия $P\{(a) \neq (b)\}$ и реализуется программой

$N+1$	—	a	b	α
$N+2$	—	b	a	β
$N+3$	×	α	β	α
$N+4$	*УПЧ*	α	A	B

Логические операторы, определяемые логическим условием $P\{(\alpha) \leqslant -0\}$ или его отрицанием $P\{(\alpha) \geqslant +0\}$, где (α) задается арифметической формулой, будем называть *простыми логическими операторами*.

Программы простых логических операторов состоят из команд вычисления значения (α) по заданной арифметической

формуле и заканчиваются командой условной передачи управления по числу α оператору **A**, если выполнено логическое условие, либо оператору **B**, если логическое условие не выполнено.

Логические операторы, определяемые логической формулой, содержащей простые логические операторы, будем называть *сложными логическими операторами*.

Программирование сложных логических операторов может быть осуществлено двумя различными способами.

Первый способ состоит в составлении программы вычисления сложного логического условия, определяющего соответствующий логический оператор, по какой-либо логической формуле, содержащей простые логические условия. Программы вычисления логических условий строятся как арифметические программы. Рассмотрим примеры.

8. Логическое условие $P_8\{(\alpha_1) \leqslant -0$ или $(\alpha_2) \leqslant -0\}$ выражается формулой

$$\alpha = \operatorname{sign}(\alpha_1) \vee \operatorname{sign}(\alpha_2).$$

Программа

$\vee$	α_1	α_2	α
УПЧ	α	*B*	*A*

9. Логическое условие $P_9\{(\alpha_1) \geqslant +0$ и $(a) - (b) \leqslant -0\}$ выражается формулой

$$\alpha = -\operatorname{sign}(\alpha_1) \wedge \operatorname{sign}[(a) - (b)].$$

Программа

$k+1$	—	—	α_1	α
$k+2$	—	a	b	β
$k+3$	$\wedge$	α	β	α
$k+4$	*УПЧ*	α	*B*	*A*

Второй способ программирования сложных логических операторов основан на использовании следующих трех правил программирования.

I. Правило программирования отрицания логического оператора **P**. Пусть логический оператор **P** передает управление оператору **A** в случае истинности соответ-

ствующего логического условия и оператору **B** в случае его ложности. Программа такого оператора содержит команды передачи управления по числу операторам **A** и **B**. Программа отрицания этого оператора ($\overline{\mathbf{P}}$) получается из программы исходного оператора **P** заменой в командах передачи управления кода оператора **A** *) на код оператора **B** и наоборот.

Нетрудно убедиться, что программы отрицания логических операторов с двумя и более командами передачи управления получаются в соответствии с приведенным правилом.

Приведем несколько примеров.

10. Сложный логический оператор $\mathbf{P}_{10}$, определяемый программой

$k+1$	*УПЧ*	α_1	A	$k+2$
$k+2$	*УПЧ*	α_2	A	B

передает управление оператору **A** при условии, что хотя бы одно из чисел α_1 или α_2 имеет значение $\geqslant +0$, и оператору **B** в ином случае.

Программа отрицания данного оператора в соответствии со сформулированным правилом будет иметь вид

$\overline{\mathbf{P}}_{10}$

$k+1$	*УПЧ*	α_1	B	$k+2$
$k+2$	*УПЧ*	α_2	B	A

В соответствии с логическим смыслом знака отрицания оператор, определяемый этой программой, передает управление оператору **A** только тогда, когда и $\alpha_1 \leqslant -0$ и $\alpha_2 \leqslant -0$.

11. Логический оператор, определяемый программой

$\mathbf{P}_{11}$

$k+1$	*УПЧ*	α_1	$k+2$	B
$k+2$	*УПЧ*	α_2	A	B

передает управление оператору **A** при условии, что и $\alpha_1 \geqslant +0$ и $\alpha_2 \geqslant +0$, и оператору **B** в противном случае. Применяя

*) Код оператора, непосредственно следующего за данным оператором, можно принять равным числу, на единицу большему числа всех команд данного оператора.

правило отрицания оператора, получим программу оператора — **P**, передающего управление оператору **A** при условии, что $\alpha_1 \leqslant -0$ или $\alpha_2 \leqslant -0$,

$\overline{\mathbf{P}}_{11}$

$k+1$	*УПЧ*	α_1	$k+2$	A
$k+2$	*УПЧ*	α_2	B	A

12. Отрицание логического оператора $\mathbf{P}_{12}$ с программой

$k+1$	*УПЧ*	α_1	A	$k+2$
$k+2$	*УПЧ*	α_2	B	A

получим, применяя правило отрицания

$k+1$	*УПЧ*	α_1	B	$k+2$
$k+2$	*УПЧ*	α_2	A	B

II. Правило программирования суммы двух логических операторов $\mathbf{P}_1 \vee \mathbf{P}_2$. Программа логической суммы двух операторов $\mathbf{P}_1$ и $\mathbf{P}_2$, передающих управление одному и тому же оператору **A** в случае истинности одного из соответствующих логических условий и оператору **B** в случае ложности обоих логических условий, получается приписыванием к команде оператора $\mathbf{P}_1$ команд оператора $\mathbf{P}_2$. В командах передачи управления оператора $\mathbf{P}_1$ код оператора **B** заменяется на код оператора $\mathbf{P}_2$.

Рассмотрим примеры.

13. Даны два оператора:

$\mathbf{P}_1$

УПЧ	α_1	A	B

$\mathbf{P}_2$

УПЧ	α_2	A	B

Программа логической суммы этих операторов согласно правилу имеет вид

$\mathbf{P}_{13} = \mathbf{P}_1 \vee \mathbf{P}_2$

$k+1$	*УПЧ*	α_1	A	$k+2$
$k+2$	*УПЧ*	α_2	A	B

Непосредственной проверкой убедимся, что полученная программа передает управление оператору **A**, если выполнено либо условие $(\alpha_1) \geqslant +0$, либо условие $(\alpha_2) \geqslant +0$, и оператору **B** в противном случае.

14. Построим логическую сумму операторов

P_1	*УПЧ*	α_1	B	A

P_2	*УПЧ*	α_2	A	B

Объединяя программы, получим программу оператора, передающего управление оператору **A** при следующих условиях:

$P\{(\alpha_1) \leqslant -0 \quad \text{или} \quad (\alpha_2) \geqslant +0\}$

$k+1$	*УПЧ*	α_1	$k+2$	A
$k+2$	*УПЧ*	α_2	A	B

III. Правило построения программы логического произведения двух логических операторов $\mathbf{P}_1 \wedge \mathbf{P}_2$. Программа логического произведения двух логических операторов, передающих управление в случае истинности соответствующих логических условий одному и тому же оператору **A** и в случае ложности хотя бы одного из них — оператору **B**, может строиться по формуле

$$\mathbf{P}_1 \wedge \mathbf{P}_2 = \overline{(\overline{\mathbf{P}}_1 \vee \overline{\mathbf{P}}_2)},$$

т. е. построение программы логического произведения операторов сводится к применению правил I и II построения отрицания и логической суммы двух операторов.

Нетрудно убедиться, что программа логического произведения двух логических операторов $\mathbf{P}_1$ и $\mathbf{P}_2$ может строиться по следующему правилу: к командам оператора $\mathbf{P}_1$ приписываются команды оператора $\mathbf{P}_2$; в командах передачи управления оператора $\mathbf{P}_1$ код оператора **A** заменяется на код оператора $\mathbf{P}_2$. Легко понять, что оба сформулированных правила эквивалентны.

Рассмотрим примеры.

15. Даны два оператора:

P_1	*УПЧ*	α_1	A	B

P_2	*УПЧ*	α_2	A	B

Программа их отрицаний имеет вид

$\overline{P}_1$	*УПЧ*	α_1	B	A

$\overline{P}_2$	*УПЧ*	$-\alpha_2$	B	A

Программа их логического произведения получается следующей:

$\mathbf{P}_1 \wedge \mathbf{P}_2 = \overline{\overline{\mathbf{P}_1} \vee \overline{\mathbf{P}_2}}$

$k+1$	*УПЧ*	α_1	$k+2$	B
$k+2$	*УПЧ*	α_2	A	B

в соответствии со второй формулировкой правила программирования логического произведения.

16. Даны два оператора:

$\mathbf{P}_1$				
	УПЧ	α_1	A	B

$\mathbf{P}_2$				
	УПЧ	α_2	B	A

Программа логического произведения получает вид

$\mathbf{P}_1 \wedge \mathbf{P}_2$

$k+1$	*УПЧ*	α_1	$k+2$	B
$k+2$	*УПЧ*	α_2	B	A

Рассмотрим управление повторением работы операторов программы. В программах циклических процессов отдельные операторы или группы операторов повторно применяются обычно в соответствии со значением логического условия, аргумент которого является переменной величиной, изменяемой от цикла к циклу. Такое логическое условие, как известно, в логике называется одноместным предикатом. Например, циклическим процессом N-кратного применения некоторого оператора **A** можно управлять с помощью логического условия $P\{n \leqslant N\}$, где n — переменная величина, изменяющаяся от цикла к циклу на единицу, начиная с нуля. Такое логическое условие мы называли счетчиком числа циклов (см. главу III, § 4).

В общем случае управление повторным применением некоторых операторов при переходе от цикла к циклу осуществляется логическим условием $P(n)$, аргумент которого пробегает натуральный ряд чисел в соответствии с номером повторения цикла. Значения $P(n)$ могут быть заранее представлены в виде таблицы

n	0	1	2	...
$P(n)$	$P(0)$	$P(1)$	$P(2)$	...

Здесь $P(n)$ равно 0 или 1, в зависимости от чего управление передается тем или иным операторам. Тогда программа соответствующего логического оператора может быть построена при помощи логической шкалы, предложенной М. Р. Шура-Бура. В простейшем случае логическая шкала занимает две ячейки *ЗУ*: α и β. Одна ячейка α содержит таблицу значений логического условия $P(n)$, во второй ячейке β помещается единица в знаковом разряде и нули в остальных разрядах. Это — указатель шкалы. С помощью логического произведения этих двух ячеек определяется наличие нуля или единицы в соответствующем разряде логической шкалы и производится передача управления согласно полученному значению. После каждой передачи управления происходит сдвиг указателя шкалы на один разряд вправо, либо сдвиг шкалы на один разряд влево.

Рассмотрим примеры.

17. Логический оператор определяется условием $P(n)$, задаваемым таблицей

n	0	1	2	3	4	5	6	7	8	9	10
$P(n)$	0	1	1	1	0	1	1	1	1	0	1

Управление передается оператору **A**, если $P(n) = 0$, и оператору **B**, если $P(n) = 1$.

Программа логического оператора с применением логической шкалы имеет вид

Числа			Команды			
α_1	01110111101	$k+1$	$\wedge$	α_1	α_2	α
α_2	10000000000	$k+2$	->	1	α_1	α_1
		$k+3$	*УПЧ*	α	A	B

или *)

α_1	0 1110111101 ...	$k+1$	*Ц+*	α_1	α_1	α_1
		$k+2$	*УПЧ*	α_1	A	B

*) Во втором случае исходная шкала размещается, начиная с первого числового разряда, операция *Ц+* осуществляет сдвиг шкалы в знаковый разряд.

Если число разрядов одной ячейки недостаточно для содержания таблицы значений логического условия, то шкала может быть расположена в нескольких ячейках. При этом программа соответствующего оператора усложняется.

18. Логический оператор $P(n)$ определяется таблицей

n	0	1	2	3	4	5	6	7	...	57	58	59	60
$P(n)$	0	1	1	1	0	1	1	1		0	1	1	1

которую разместим в двух ячейках *ЗУ* (в шестнадцатеричной записи)

$\alpha+1$	777 · 777 · 777
$\alpha+2$	777 · 777 000

Программа оператора имеет вид

Числа

$\alpha+1$	777777777			
$\alpha+2$	777777000			
α_3	800000000			
α_4	800000000			
α_5	—	1	—	—

Команды

$k+1$	$\wedge$	$\alpha+1$	α_3	α
$k+2$	$\rightarrow$	1	α_3	α_3
$k+3$	—	α_3	—	β
$k+4$	—	α	—	α
$k+5$	*УПЧ*	β	$k+8$	$k+6$
$k+6$	$\oplus$	$k+1$	α_5	$k+1$
$k+7$	+	α_4	—	α_3
$k+8$	*УПЧ*	α	B	A

Логические шкалы могут быть использованы для передачи управления более чем в двух направлениях. В этом случае соответствующее логическое условие является не двузначным,

а многозначным, а следовательно, каждое значение логического условия кодируется при помощи многозначного двоичного числа. Например, для передачи управления в трех или четырех направлениях используется трехзначное или четырехзначное логическое условие; каждое значение такого логического условия кодируется двухразрядным двоичным числом. Соответственно указатель шкалы содержит столько единиц, сколько разрядов отведено для кодирования значений логического условия.

19. Оператор управления задается логическим условием

n	0	1	2	3	4	5	. . .
$P(n)$	00	01	11	00	01	11	. . .

где управление передается операторам $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, если $\mathbf{P}(n) = 00; 01; 11$ соответственно.

Программа оператора с использованием логической шкалы

α_1	000	111 000	111 . . .
α_2	1 1 0 . . .		
α_3	0 0 . . . 0		
α_4	0 1 0 . . .		

$k+1$	$\wedge$	α_1	α_2	α
$k+2$	$\leftarrow$	α_1	α_1	α_1
$k+3$	$\leqslant$	α	α_3	$\mathbf{A}_1$
$k+4$	$\leqslant$	α	α_4	$\mathbf{A}_2$
$k+5$	*БП*	—	—	$\mathbf{A}_3$

Если значения логического условия $P(n)$ повторяются периодически, то можно построить шкалу только на один период значений $P(n)$. При этом в программе соответствующего оператора нужно предусмотреть восстановление шкалы (или указателя шкалы) после окончания периода.

20. Логическое условие задается в виде

$$P(3n) = 1;$$

$$P(3n \pm 1) = 0.$$

Программа такого оператора передачи управления согласно

периодически повторяющимся значениям $P(n)$ составляется следующим образом:

α_1	001 . . .
α_2	001 . . .
α_3	001 . . .

$k+1$	$\wedge$	α_1	α_2	α
$k+2$	$\leqslant$	—	α_1	$k+5$
$k+3$	$+$	α_3	—	α_1
$k+4$	$\leqslant$	—	—	$k+6$
$k+5$	$\leftarrow$	1	α_1	α_1
$k+6$	*УПЧ*	α	A	B

§ 6. Содержательное преобразование схем программ

Как уже было ранее отмечено, программа по данной вычислительной схеме определяется не однозначно. По одной и той же вычислительной схеме могут быть составлены различные программы. Можно поставить вопрос о преобразовании программ к виду более простому и удобному для выполнения в машине, а также о построении наиболее экономной программы (по объему вводимой информации или по числу тактов работы машины и др.). Преобразование схем программ может осуществляться с учетом их содержания или чисто формально по заранее разработанному алгоритму (см. Фаддеева [1]).

В настоящем параграфе мы рассмотрим несколько примеров содержательного преобразования схем программ.

1. Преобразование схем программ с изменением операторов переадресации. Пусть дана схема программы двойного циклического процесса с двумя параметрами

$$\Phi_1(k)\overset{2}{\downarrow}\overset{1}{\downarrow}\mathbf{A}_i\mathbf{F}_1(i)\mathbf{P}_1\overset{\mathbf{A}}{\uparrow}\mathbf{B}_k\mathbf{F}_2(k)\Phi_2(i)\mathbf{P}_2\overset{\mathbf{A}}{\uparrow}! \qquad (11)$$

В конкретном примере решения уравнения теплопроводности (см. программу 2 § 3) программы операторов $\mathbf{F}_1(i)$ и операторов $\mathbf{F}_2(k)\Phi_2(i)$ содержат общую часть. Это обстоятельство может быть использовано для преобразования схемы программы. Так как программа операторов $\mathbf{F}_2(k)\Phi_2(i)$ состоит из двух команд, таких же, как и команды, представляющие оператор $\mathbf{F}_1(i)$, то эти две команды оператора $\mathbf{F}_2(k)$ можно исключить из программы, передав управление командам оператора $\mathbf{F}_1(i)$. Таким образом, преобразованная схема программы приобретает вид

$$\Phi_1(k)\overset{1}{\downarrow}\mathbf{A}_i\overset{2}{\downarrow}\mathbf{F}(i)\mathbf{P}_1\overset{\mathbf{A}}{\uparrow}\mathbf{B}_k\Phi(i)\mathbf{P}_2\overset{\mathbf{F}}{\uparrow}! \qquad (12)$$

Теперь оператор $\Phi(i)$ подготавливает работу логического условия P_1.

В программу решения задачи Дирихле для прямоугольника (стр. 206) также можно внести изменения благодаря наличию связи между параметрами k и i: изменению i на единицу соответствует изменение k на k единиц. Операторы $F_2(i)\,\Phi_2(k)$ могут быть реализованы двукратным применением оператора $F(k)$, что можно осуществить введением специального логического условия P_4. Схема программы будет такой:

$$\begin{matrix} & 3 & & 2\ 1 & & 4 & & & \mathbf{A} & \mathbf{F}_1 & \mathbf{A} & & & \mathbf{\Phi}_2(i) \\ \mathbf{\Phi}_1(s)\downarrow & \mathbf{\Phi}_2(i)\downarrow & \downarrow & \mathbf{A}_{kis}\downarrow & \mathbf{F}_1(k) & \mathbf{P}_1\uparrow & \mathbf{P}_4\uparrow & \mathbf{P}_2\uparrow & \mathbf{F}_3(s) & \mathbf{P}_3\uparrow & & ! \end{matrix} \qquad (13)$$

При составлении программы по этой схеме нужно обратить внимание на работу логического условия P_1.

Сложность переадресации в программе вычисления определителя невырожденной квадратной матрицы вызвана тем, что число циклов по параметру i — переменное. Этот недостаток можно устранить, если ввести логическое условие P_4, которое исключает из работы оператор счета $\mathbf{A}_{kir}$ при $i \leqslant r$, т. е.

$$P_4 = \begin{cases} 0, & i > r, \\ 1, & i \leqslant r. \end{cases}$$

Схема программы теперь будет иметь вид

$$\begin{matrix} & 3 & 2 & 1 & \mathbf{F}_2 & & \mathbf{P}_4\ 4 & & \mathbf{C} & & \mathbf{D} \\ \mathbf{\Phi}(r)\downarrow & \mathbf{D}_r\downarrow & \mathbf{C}_{ir}\downarrow & \mathbf{P}_4\uparrow & \mathbf{A}_{kir} & \mathbf{F}_1(k)\,\mathbf{P}_1\uparrow & \downarrow & \mathbf{F}_2(i,\,k)\,\mathbf{P}_2\uparrow & \mathbf{F}_3(r,\,i)\,\mathbf{P}_3\uparrow & ! \end{matrix} \qquad (14)$$

2. Преобразование операторов счета. Программа решения дифференциального уравнения первого порядка на машине с фиксированной запятой имеет два оператора счета: $\mathbf{A}^{\mathrm{I}}$ и $\mathbf{A}^{\mathrm{II}}$, из которых на каждом цикле действует только один оператор, в зависимости от логического условия P_1.

Сравнение программ операторов $\mathbf{A}^{\mathrm{I}}$ и $\mathbf{A}^{\mathrm{II}}$ показывает, что команды, реализующие эти операторы, отличаются лишь адресами чисел. Это позволяет преобразовывать оператор счета $\mathbf{A}^{\mathrm{I}}$ в оператор счета $\mathbf{A}^{\mathrm{II}}$ и наоборот.

Введем оператор $\mathbf{H}$ преобразования $\mathbf{A}^{\mathrm{I}}$ в $\mathbf{A}^{\mathrm{II}}$ и оператор $\mathbf{M}$ преобразования $\mathbf{A}^{\mathrm{II}}$ в $\mathbf{A}^{\mathrm{I}}$. Тогда схема программы может быть приведена к виду

$$\begin{matrix} 2 & & \mathbf{A} & 1 & & \mathbf{P}_2 & 3 & \mathbf{\Phi} \\ \downarrow & \mathbf{\Phi}\Psi\mathbf{P}_1\uparrow & \mathbf{H}\downarrow & \mathbf{A}^{\mathrm{II}}\mathbf{P}_3\uparrow & \mathbf{M}\downarrow & \mathbf{P}_2\uparrow & ! & \end{matrix} \qquad (15)$$

Программирование операторов преобразования $\mathbf{H}$ и $\mathbf{M}$ можно осуществить двумя способами: либо операторы осуществляют пересылку нужных данных в стандартные ячейки, номера

которых входят в адреса команд оператора **А**, либо операторы **Н** и **М** производят нужную переадресацию команд оператора $\mathbf{A}^{\mathrm{I}}$ и $\mathbf{A}^{\mathrm{II}}$ соответственно.

Приведем программы операторов преобразования **Н** и **М** в обоих случаях. В первом случае оператор **Н** должен поменять местами в ячейках *ЗУ* числа x_k и y_k, φ_k и ψ_k, что осуществляется командами

$H+1$	+	$\alpha+1$	—	ω
$H+2$	+	$\alpha+2$	—	$\alpha+1$
$H+3$	+	ω	—	$\alpha+2$
$H+4$	+	$\alpha+3$	—	ω
$H+5$	+	$\alpha+4$	—	$\alpha+3$
$H+6$	+	ω	—	$\alpha+3$

Программа оператора **М** восстанавливает исходное размещение этих данных в *ЗУ* для работы оператора **А** и имеет тот же вид, что и программа оператора **Н**.

Во втором случае для переадресации команд оператора **А** нужны следующие константы:

δ_1	—	1	—	—
δ_2	—	—	1	—
δ_3	—	—	1	1
δ_4	—	1	—	1

Программа оператора **Н**, осуществляющего переадресацию команд оператора $\mathbf{A}^{\mathrm{I}}$ в команды оператора $\mathbf{A}^{\mathrm{II}}$, имеет вид

$H+1$	$\oplus$	$A+1$	δ_1	$A+1$
$H+2$	$\ominus$	$A+2$	δ_2	$A+2$
$H+3$	$\ominus$	$A+4$	δ_3	$A+4$
$H+4$	$\oplus$	$A+5$	δ_4	$A+5$

Программа оператора **M**, восстанавливающего команды оператора **A**, имеет вид

$M+1$	$\ominus$	$A+1$	δ_1	$A+1$
$M+2$	$\oplus$	$A+2$	δ_2	$A+2$
$M+3$	$\oplus$	$A+4$	δ_3	$A+4$
$M+4$	$\ominus$	$A+5$	δ_4	$A+5$

Упражнения

1. Преобразовать схему программы 4 аналогично преобразованию схемы программы 2.
2. Ввести логическое условие в программу 3, устраняющее переменное число внутренних циклов по параметру k.

ГЛАВА V

АДРЕСНОЕ ПРОГРАММИРОВАНИЕ

Для сложных задач схемы счета и схемы программ существенно отличаются друг от друга. Удельный вес арифметических операторов, выполняющих счет по формулам, в схемах программ сложных задач весьма незначителен. Основное же место занимают операторы управления (переадресации, передачи управления и др.), которые обеспечивают выполнение определенной логической схемы счета.

В алгоритмах неарифметических задач арифметический счет почти полностью отсутствует. Описание таких алгоритмов дается в виде системы правил преобразования исходной информации и излагается обычно в словесной форме. Словесная формулировка алгоритма, как правило, не является достаточно четкой и не содержит указаний о способах его программного осуществления. Отсутствие таких указаний затрудняет программирование неарифметических задач для *ЦАМ*. Эффективный способ описания алгоритмических процессов, удобный для чтения и для выполнения его на машине, должен удовлетворять следующим требованиям:

1) он должен быть близок к словесному и принятому в математике символическому описанию алгоритмических процессов;

2) описание алгоритмов не должно зависеть от конкретных технических особенностей *ЦАМ* и в то же время должно допускать возможность автоматического перехода к программному описанию в командах конкретных *ЦАМ*.

В то же время программное описание алгоритма в командах *ЦАМ* содержит, помимо логической схемы алгоритма, и специфические технические особенности конкретных *ЦАМ*.

В настоящей главе излагается способ программирования, в основу которого положены две характерные особенности программного осуществления вычислительных процессов на *ЦАМ*: принцип адресности и принцип программного управления (см. Королюк В. С. [1] и Королюк В. С., Ющенко Е. Л. [1]).

§ 1. Понятие адресной программы

1. Вводные замечания. Различные машины имеют различные наборы операций, а также обладают рядом специфических технических особенностей, которые должны учитываться при программировании. Для одной и той же задачи программа решения ее имеет различный вид для разных *ЦАМ*. Поэтому обычно программирование задач излагается в командах той или иной конкретной *ЦАМ*. В то же время ясно, что основные логические принципы программирования не должны зависеть от конкретных технических особенностей той или иной *ЦАМ*.

Общим принципом построения программ для всех *ЦАМ* является *принцип адресности*. Информация о задаче кодируется и размещается в определенном порядке в ячейках *ЗУ* машины, имеющих адреса. Размещение исходных данных в ячейках *ЗУ* является начальным этапом программирования. Программа вычислительного процесса записывается в *адресном виде*, т. е. в программе указываются не сами числа, над которыми необходимо производить операции, а адреса ячеек *ЗУ*, содержащих эти числа. Выполнение программы на *ЦАМ* приводит к решению конкретной задачи при заданном заполнении ячеек *ЗУ*. Благодаря описанию программы в адресном виде имеется возможность по одной и той же программе решать различные конкретные задачи, меняя заполнение ячеек *ЗУ* (меняя параметры задач). Этим обеспечивается массовость алгоритма, описанного в виде программы.

Этот же принцип адресности лежит в основе и других методов описания вычислительных процессов. Например, при описании вычислительного процесса в буквенном виде подразумевается, что при конкретном осуществлении этого процесса вместо букв будут подставлены соответствующие им числа. Буквы являются обозначениями, метками, адресами соответствующих им чисел.

Другой общей для всех *ЦАМ* особенностью программного описания вычислительных процессов является *принцип программного управления*. Порядок выполнения отдельных этапов вычислений строго определен в программе порядком команд и командами передачи управления, учитывающими результаты промежуточных вычислений.

Эти принципы программирования мы сохраняем в определении понятия адресной программы, отказываясь в описании вычислительных процессов от других ограничений, связанных с техническими особенностями конструкций *ЦАМ*. При этом, конечно, язык адресного программирования по возможности сохраняется машинным, в связи с чем перевод адресных программ в программы для конкретных *ЦАМ* легко может быть

автоматизирован. Заметим также, что техническое усовершенствование машин может привести к значительному облегчению перевода адресной программы для конкретных *ЦАМ*.

2. Система объектов кодирования. Исходным материалом для кодирования информации о задаче служит система объектов S, между которыми установлены некоторые соотношения. Отдельные объекты системы S будем называть *кодами*. Соотношения между кодами, установленные в системе кодов S, назовем *операциями над кодами*. Для обозначения кодов и операций будем использовать различные символы. Как правило, коды обозначают буквами (латинскими или греческими) и числами. Для обозначения операций используются специальные символы, принятые в математике:

арифметические операции $+, -, \times, :$;

функциональные операции $\sqrt{}$, sin, ln и т. п.;

логические операции $\vee$, $\wedge$, $\neg$ (операция отрицания);

операции отношения (они называются также предикатными операциями) $=, <, \leqslant$ и др.

Примеры.

1. Конечная совокупность S объектов (символов), для которых установлено только отношение отличия: для любых двух объектов из S определено, равны они друг другу или нет. Такая система объектов обычно называется алфавитом.

2. Натуральный ряд чисел образует систему $S(N, 0, ')$. N — множество натуральных чисел, 0 — элемент множества N, $'$ — операция следования натуральных чисел.

3. Множество D n-разрядных двоичных чисел, в котором заданы одноместные (например, отрицание) и двуместные (например, арифметические) операции*), образует систему объектов, используемую для кодирования информации о задачах в *ЦАМ*.

Соотношения между кодами, установленные в системе кодов, позволяют строить по обычным правилам выражения (функции), значениями которых также являются коды, полученные в результате выполнения указанных в выражении операций. Заметим, что результатом операции отношения могут быть только два кода, которые мы будем обозначать 1, если отношение выполнено, и 0, если отношение не выполняется. Иначе говоря, операции отношения определяют функции, принимающие лишь два значения (1 или 0). Такие функции называются предикатами. Условимся записывать предикаты в виде $P\{\ \}$, где в фигурных скобках записывается соответствующее выражение, определяющее предикат.

---

*) Одноместными называются функции одного аргумента, двуместные — двух аргументов.

3. Алгоритмические операции в системе кодов. В системе кодов S, помимо операций, которыми определяются арифметические и предикатные функции, введем операцию *выделения содержимого адреса* или, коротко, *операцию по адресу*, обозначаемую символом $'$, который ставится слева вверху перед кодом. Содержание этой операции следующее: *каждому коду a из S ставится в соответствие код $'a$ из S. Код a называется адресом кода $'a$. Код $'a$ называется содержимым адреса a.* Последовательное применение операции выделения содержимого адреса приводит к понятию *ранга адреса*. Пусть b является содержимым адреса a, т. е. $'a \equiv b$, а c есть содержимое адреса b, т. е. $'b \equiv c$; тогда a есть адрес адреса кода c.

Назовем *a адресом второго ранга кода c* или, коротко, *фиксатором кода c* и будем писать:

$$^2a \equiv '('a) \equiv 'b \equiv c.$$

Аналогично определяются *адреса k-го ранга* (k — целое положительное число). Содержимое адреса a k-го ранга определяется из соотношения

$$^ka \equiv '(^{k-1}a).$$

Задание в системе кодов S операции выделения содержимого адреса определяет состояние системы кодов S, которые мы будем называть *распределением адресов* в S (аналогично тому, как заполнение ячеек *ЗУ* машины определяет состояние *ЗУ* машины). Введем теперь алгоритмическую операцию *переноса по адресу* (с вытеснением), которая изменяет содержимое адресов, т. е. преобразует состояние системы S. Для обозначения операции переноса по адресу воспользуемся символом $\Rightarrow$.

Применение операций переноса по адресу к кодам a и b из S

$$a \Rightarrow b$$

означает, что $'b \equiv a$, т. е. содержимым адреса b становится a (вытесняя предыдущее содержимое b).

Так как значения функции f кодов S также есть коды из S, то имеет смысл операция по адресу, которая является значением функции $'f$. При этом $'f$ означает код, являющийся содержимым адреса, который есть значение функции f. В операции $a \Rightarrow b$ можно считать a и b функциями. Тогда результат операции $a \Rightarrow b$ можно сформулировать так: значение функции a становится содержимым адреса, который является результатом вычисления значения функции b ($'b \equiv a$).

Выражения $a \Rightarrow b$ назовем *формулами преобразования*, или *операторами действия*.

Для построения программ будем пользоваться предикатными формулами, которые определяем следующим образом.

Пусть P — предикатная функция в S, т. е. функция, принимающая только два значения (1 или 0). Пусть α и β — формулы (или обозначения формул).

Предикатной формулой называется выражение $P\alpha\beta$.

Обратим внимание, что теперь α и β в определении предикатной формулы могут быть либо формулами преобразования, либо предикатными формулами.

Предикатные формулы не изменяют состояния S (т. е. не меняют содержимое адресов), а определяют порядок действий формул по следующему правилу: *если предикатная функция* P *принимает значение* 1, *то применяется формула* α; если P *принимает значение* 0, *то применяется формула* β*).

Предикатные формулы называются еще *операторами распознавания.*

Отметим еще, что если для обозначения формул в предикатной формуле пользоваться кодами из S, то есть возможность определить порядок применения формул по адресу. Например, запись $P'\alpha^2\beta$ означает, что когда значение P есть 1, то действует формула $'\alpha$, т. е. формула, обозначение которой есть содержимое адреса α; если же P принимает значение 0, то действует формула, обозначение которой есть содержимое адреса второго ранга β. Если предикатная функция P тождественно равна 1, то в соответствующей предикатной формуле второй индекс опускается.

Условимся опускать второй индекс и тогда, когда при значении $P = 0$ управление передается следующей по порядку формуле.

Безусловную передачу управления формуле α будем записывать $P\alpha$.

Наконец, условимся указания об окончании работы программы обозначать *«ост.»*.

Перейдем теперь к определению понятия адресной программы.

Адресная программа задается исходным распределением адресов в S *(состоянием системы кодов* S*) и последовательностью адресных формул с указанием порядка их применения.*

Указания порядка действия формул задаются либо предикатными формулами, либо обычным порядком следования: при записи в строку — слева направо, при записи в столбец — сверху вниз.

Может случиться, что в программе некоторые формулы могут действовать в произвольном порядке. Тогда при записи программы в столбец будем такие формулы писать в строку, а при записи программы в строку запишем их в столбец.

---

*) Иногда предикатные формулы мы будем записывать в форме P_{β}^{α}.

Проиллюстрируем определение адресной программы на простейших конкретных примерах.

Пример 1. Вычисление суммы n чисел: $S_n = \sum_{k=1}^{n} a_k$.

Введем адреса $\alpha_k = \alpha_0 + k$ для слагаемых ($'\alpha_k = a_k$), суммы S ($'S = 0$) и адреса α ($'\alpha = \alpha_0 + 1$), β ($'\beta = \alpha_0 + n$).

Адресная программа имеет следующий вид:

Распределение адресов

α		β	
$\alpha_0 + 1$	. . .	$\alpha_0 + n$	S
a_1		a_n	0

k_1 $\quad {}'S + {}^2\alpha \Rightarrow S$
$\quad {}'\alpha + 1 \Rightarrow \alpha$
$\quad P\{{}'\alpha \leqslant {}'\beta\}\ k_1$ *ост.*

Пример 2. Вычислить и запомнить N членов геометрической прогрессии

$$a_{k+1} = a_k \cdot q, \quad k = 0, 1, 2, \ldots, N.$$

Введем адреса $\alpha_k = \alpha_0 + k$ для членов прогрессии $'\alpha_0 = \alpha_0$ (исходное содержимое адресов α_k, где $1 \leqslant k \leqslant N$, произвольно) и адреса δ ($'\delta = q$).

Адресная программа имеет следующий вид:

Распределение адресов

α			β	
α_0	$\alpha_0 + 1$	. . .	$\alpha_0 + N$	δ
a_0				q

k_1 $\quad {}^2\alpha \times {}'\delta \Rightarrow {}'\alpha + 1$
$\quad {}'\alpha + 1 \Rightarrow \alpha$
$\quad P\{{}'\alpha \leqslant {}'\beta\}\ k_1$ *ост.*

4. Граф-схемы программ. Если ввести обозначение «$\to$» для определения переходов от одной формулы к другой и при этом предикатные формулы изображать в виде $P\begin{smallmatrix}1\to\alpha\\0\to\beta\end{smallmatrix}$, то мы придем к определению граф-схемы программы (см. Калужнин Л. А. [1]), дающей наглядное представление о порядке действий в программе.

Пусть $D_1, D_2, \ldots, D_m$ — система операторов действий, преобразующих информацию о задаче, и $P_1, P_2 \ldots, P_r$ — система операторов распознавания. $D - P$ граф-схемой называется система точек, соединенных стрелками, имеющих следующие особенности. Существует точка, которая называется входом

граф-схемы и из которой выходит только одна стрелка. Обозначим эту точку буквой B. Существует точка, которая называется выходом граф-схемы; обозначим ее знаком Я, иногда эту точку будем обозначать *«ост.»*. Остальные точки схемы могут быть либо D-точками, либо P-точками. Из каждой D-точки выходит только одна стрелка, из каждой P-точки выходят две стрелки: одна с отметкой 1, другая с отметкой 0. В любую точку граф-схемы может входить любое число стрелок. Каждой D-точке ставится в соответствие некоторый оператор действия $D_k (1 \leqslant k \leqslant m)$; каждой P-точке ставится в соответствие некоторый распознаватель $P_s (1 \leqslant s \leqslant r)$.

Действие $D - P$ граф-схемы, как алгоритма, осуществляется следующим образом: из точки входа по стрелке поступает сигнал действия в следующую точку схемы (обозначим ее T); если точка T есть D-точка, то информация о задаче преобразуется согласно описанию соответствующего ей оператора действия, после чего сигнал действия поступает в следующую точку схемы по стрелке, которая выходит из точки T. Если точка T есть P-точка, то выясняется наличие у информации свойства согласно описанию соответствующего ей распознавателя; если искомое свойство есть в информации, то сигнал действия поступает по стрелке с отметкой единица, если информация не обладает искомым свойством, то сигнал действия передается по стрелке с отметкой 0, и т. д. Алгоритм кончает работу, когда сигнал действия поступает в точку выхода схемы Я. $D - P$ граф-схема определяет только порядок действий без каких-либо ограничений на сложность схемы или характер операторов и распознавателей.

5. Программирование адресных формул для *ЦАМ*. Превращение адресных программ в программы для конкретных типов *ЦАМ* с учетом их технических особенностей не связано с большими затруднениями и может быть автоматизировано.

Рассмотрим примеры программ простейших адресных формул, содержащих адреса второго ранга в кодах трехадресных *ЦАМ* с набором операций, описанным в главе II, а именно:

$$\text{а) } {}^2\alpha \Rightarrow \beta, \quad \text{б) } {}'\alpha \Rightarrow {}'\beta, \quad \text{в) } {}^2\alpha \Rightarrow {}'\beta.$$

Программы этих формул, очевидно, зависят от способа кодирования. Мы рассмотрим простейший случай, когда фиксаторы кодов содержат в первых адресах номера ячеек *ЗУ*, хранящих эти коды, при условии, что в каждой ячейке *ЗУ* хранится только один код. В этих предположениях программы адресных формул строятся весьма просто.

а) Пусть коды a и b содержатся в ячейках с номерами γ и ρ соответственно, т. е. ${}'\gamma = a$, ${}'\rho = b$, а номера ячеек γ и ρ содер-

жатся в первом адресе ячеек α и β соответственно, причем в остальных разрядах этих ячеек помещены нули.

Таким образом, $'\alpha = \gamma$, $'\beta = \rho$. Тогда программа переноса $^2\alpha \Rightarrow \beta$ имеет следующий вид:

$k+1$	$\wedge$	δ_1	$k+3$	$k+3$
$k+2$	$+$	α	$k+3$	$k+3$
$k+3$	$+$			β

Здесь ячейка δ_1 содержит константу, которая имеет единицы в разрядах кода операции и в третьем адресе и нули в разрядах первого и второго адресов. Условно мы ее изобразим в виде

δ_1	1 . . . 1	0 . . . 0	0 . . . 0	1 . . . 1

Команда $k+1$ приводит команду $k+3$ к виду, указанному в программе. После выполнения команды $k+2$ команда $k+3$ принимает вид

$+$	γ		β

б) Программа переноса $'\alpha \Rightarrow '\beta$ имеет следующий вид:

$k+1$	$\wedge$	δ_2	$k+4$	$k+4$
$k+2$	$\to 2A$	β		c
$k+3$	$+$	c	$k+4$	$k+4$
$k+4$	$+$	α		

где

δ_2	1 . . . 1	1 . . . 1	0 . . . 0	0 . . . 0

Команда $k+1$ приводит команду $k+4$ к виду, указанному в программе. Команда $k+2$ переносит содержимое I адреса ячейки β в III адрес ячейки c (происходит сдвиг на два адреса вправо содержимого ячейки β). После выполнения команды $k+3$ команда $k+4$ имеет вид

$k+4$	$+$	α		ρ

в) Программа переноса ${}^{2}\alpha \Rightarrow {}'\beta$ записывается в следующем виде:

$k+1$	$\wedge$	δ_3	$k+5$	$k+5$
$k+2$	$+$	a	$k+5$	$k+5$
$k+3$	$\to 2A$	β		c
$k+4$	$+$	c	$k+5$	$k+5$
$k+5$	$\div$			

Здесь

δ_3	1 . . . 1	0 . . . 0	0 . . . 0	0 . . . 0

Команда $k+1$ приводит команду $k+5$ к виду, указанному в программе. После выполнения команд $k+2 \div k+4$ команда $k+5$ имеет вид

$k+5$	$+$	γ		ρ

Напомним, что ${}'\alpha = \gamma$, ${}'\beta = \rho$. Программирование более сложных адресных программ сводится к программированию рассмотренных операторов переноса и обычному программированию.

Введение в *ЦАМ* специальных операций может существенно упростить программирование адресных формул (см. описание машины *Киев*).

§ 2. Схемы обозревания кодов

1. Понятие обозревания информации. Как отмечалось в начале предыдущего параграфа, в программах задач для *ЦАМ* указываются адреса ячеек *ЗУ*, содержимым которых являются коды информации о задаче. Однако лишь для очень простых задач программа содержит адреса всех кодов информации о задаче. Например, программа вычисления по данной формуле может содержать адреса всех ячеек, в которых хранятся числа, участвующие в счете. Как правило, только часть адресов кодов информации входит в программу; адреса других кодов используются в программе в процессе ее работы. В машине, описанной в главе II, это достигается за счет команд переадресации.

Будем говорить, что *данный код обозревается программой, если его адрес некоторого ранга содержится в программе.*

Применение адресов высших рангов расширяет возможность обозревания кодов программы.

Адрес a второго ранга кода c ($^2a \equiv c$) будем называть *фиксатором* кода c.

В программах широкого круга задач коды исходной информации обозреваются в определенной последовательности.

Схемой обозревания последовательности кодов

$$a_0,\ a_1,\ \ldots,\ a_n \tag{1}$$

назовем циклическую адресную программу, на i-м цикле которой обозревается i-й элемент последовательности.

Для построения программ обозревания кодов введем операцию следования C в последовательности кодов (1)

$$Ca_i = a_{i+1}.$$

Код a_{i+1} называется *следующим* за кодом a_i; код a_i называется *предыдущим* (предшествующим) для кода a_{i+1}. Очевидно, что операция следования C имеет обратную $\overline{C}$:

$$\overline{C}a_{i+1} = a_i.$$

Последовательность кодов (1) упорядочивается операцией следования C.

Примеры.

1. В последовательности целых чисел

$$n+1,\ n+2,\ \ldots,\ n+m$$

операцией следования является прибавление единицы

$$C(n+k) = (n+k)+1.$$

2. В последовательности чисел, образующих арифметическую прогрессию с разностью d, операция следования определяется прибавлением числа d; для чисел, которые образуют геометрическую прогрессию со знаменателем q, операция следования определяется умножением на q.

Чтобы упорядочить произвольную последовательность кодов (1), расположим их по адресам

$$\alpha_0,\ \alpha_1,\ \ldots\ \alpha_n, \tag{2}$$

для которых определена операция следования C:

$$C\alpha_i = \alpha_{i+1} \qquad (0 \leqslant i \leqslant n), \tag{3}$$

причем $'\alpha_i = a_i$ $(0 \leqslant i \leqslant n)$.

Тогда операция следования C_1 кодов (1) задается так:

$$C_1 a_i = {}'(C\alpha_i),$$

так как

$${}'(C\alpha_i) = {}'\alpha_{i+1} = a_{i+1}.$$

Для построения программы обозревания последовательности кодов (1) введем адрес α для адреса первого кода:

$$'\alpha = \alpha_0,$$

т. е. α есть фиксатор кода a_0:

$$^2\alpha = a_0.$$

Для наглядности исходное распределение адресов представим таблицей

α			
α_0	α_1	. . .	α_n
a_0	a_1		a_n

(4)

В последней строке размещены коды последовательности (1); в каждой предыдущей строке указаны адреса кодов, расположенных в следующей строке.

Адресная программа обозревания кодов (1) имеет следующий вид:

Распределение адресов

α			
α_0	α_1	. . .	α_n
a_0	a_1		a_n

Программа 1

k_1. $C\ '\alpha \Rightarrow \alpha$;

k_2. $P\{'\alpha \neq \alpha_n\}\ k_1$ *ост.*

Первая формула изменяет содержимое фиксатора α в соответствии с (3) так, что коды (1) последовательно обозреваются программой 1. В таблице распределения адресов действие формулы k_1 можно образно представить себе как сдвиг фиксатора α по строке вправо на один адрес. В дальнейшем формулы вида k_1 будем называть *формулами сдвига фиксатора.*

Рассмотрим различные видоизменения программы 1. Введем адрес β, где $'\beta = \alpha_n$; тогда вид программы будет стандартным для любой последовательности кодов (1) с любым распределением их по адресам.

Распределение адресов

α		. . .	β
α_0	α_1	. . .	α_n
a_0	a_1	. . .	a_n

Программа 2

k_1. $C\ '\alpha \Rightarrow \alpha$.

k_2. $P\{'\alpha \neq '\beta\}\ k_1$ *ост.*

Как уже упоминалось выше, операция следования C может иметь различную интерпретацию, в зависимости от которой будет определяться вид формулы k_1.

Примеры.

1. Если

$$C\alpha_i = \alpha_i + 1, \tag{5}$$

то

$$k_1.\quad {}'\alpha + 1 \Rightarrow \alpha; \tag{5'}$$

2. если

$$\alpha_{i+1} = C\alpha_i = \alpha_i + d \tag{6}$$

(d — шаг переадресации), то

$$k_1.\quad {}'\alpha + d \Rightarrow \alpha; \tag{6'}$$

3. если

$$\alpha_{i+1} = C\alpha_i = \alpha_i + {}'\delta, \tag{7}$$

где ${}'\delta = d$, т. е. шаг переадресации содержится в адресе δ, то

$$k_1.\quad {}'\alpha + {}'\delta \Rightarrow \alpha. \tag{7'}$$

Введем адрес a в программу 2 следующим образом:

Распределение адресов

α			β
α_0	α_1	. . .	α_n
a_0	a_1	. . .	a_n

Программа 3

$k_1.$ ${}^2\alpha \Rightarrow a$

$k_2.$ $C'\alpha \Rightarrow \alpha$

$k_3.$ $P\{{}'\alpha \neq {}'\beta\}\, k_1$ *ост.*

Коды последовательности (1) обозреваются программой 3: содержимым адреса a последовательно от цикла к циклу являются коды (1). Адрес a называется *стандартным*, а программа 3 — *программой обозревания кодов в стандартном адресе.*

В результате выполнения программы 2 адрес α_0 начального кода a_0 обозреваться не будет.

Во многих задачах желательно сохранить фиксирование начального и конечного кодов последовательности (1). Это можно осуществить введением дополнительного рабочего фиксатора γ.

Распределение адресов

α			β
α_0	α_1	. . .	α_n
a_0	a_1	. . .	a_n

Программа 4

$k_1.$ ${}'\alpha \Rightarrow \gamma$

$k_2.$ $C'\gamma \Rightarrow \gamma$

$k_3.$ $P\{{}'\gamma \neq {}'\beta\}\, k_2$ *ост.*

Для формулы сдвига фиксатора α, вида $C\ '\alpha = '\alpha + 1$, можно предложить вариант программы обозревания кодов (1) со счетчиком. Введем адрес δ (вначале $'\delta = 0$).

Распределение адресов

α				
α_0	$\alpha_0 + 1$	. . .	$\alpha_0 + n$	δ
a_0	a_1	. . .	a_n	0

Программа 5

k_1. $'('\alpha + '\delta) \Rightarrow a$

k_2. $'\delta + 1 \Rightarrow \delta$

k_3. $P('\delta \leqslant n)\ k_1$ *ост.*

В этой программе фиксатор начала последовательности (1) α не сдвигается. Адрес δ играет роль счетчика числа циклов.

2. Примеры программ со схемами последовательностей обозревания кодов. Рассмотренная программа обозревания последовательности кодов часто встречается в программах различных задач. Рассмотрим некоторые примеры.

Пример 1. Вектор с компонентами

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

умножить на скаляр b.

Распределение адресов

α			α_1	
$\alpha_0 + 1$	$\alpha_0 + 2$	. . .	$\alpha_0 + n$	β
a_1	a_2	. . .	a_n	b

Программа 6

k_1. $^2\alpha \times '\beta \Rightarrow '\alpha$

k_2. $'\alpha + 1 \Rightarrow \alpha$

k_3. $P\{'\alpha \leqslant '\alpha_1\}\ k_1$ *ост*

Пример 2. Вычислить значение функции $f(x)$ для $x = x_0 + kh$ $(0 \leqslant k \leqslant n)$ и разместить по адресам $\alpha_0 + k$ $(0 \leqslant k \leqslant n)$.

Распределение адресов

α			α_1		
α_0	$\alpha_0 + 1$	. . .	$\alpha_0 + n$	β	δ
$f(x_0)$	$f(x_1)$	. . .	$f(x_n)$	x_0	h

Программа 7

k_1. $f('\beta) \Rightarrow '\alpha$

k_2. $'\beta + '\delta \Rightarrow \beta$

k_3. $\alpha + 1 \Rightarrow \alpha$

k_4. $P\{'\alpha \leqslant '\alpha_1\}\ k_1$ *ост.*

Пример 3. Выделить положительные числа из последовательности

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

и расположить их в последовательность адресов, начиная с адреса β_0.

Распределение адресов

α			α_1	β		
α_0+1	α_0+2	. . .	α_0+n	β_0	β_0+1	. . .
a_1	a_2	. . .	a_n	a_{i_1}	a_{i_2}	. . .

Программа 8

k_1. $P\{^2\alpha > 0\}\ k_2 k_3$

k_2. $^2\alpha \Rightarrow {}'\beta$

$'\beta+1 \Rightarrow \beta$

k_3. $'\alpha+1 \Rightarrow \alpha$

k_4. $P\{'\alpha \leqslant {}'\alpha_1\}\ k_1$ *ост.*

Приведем пример использования адресов третьего ранга для обозревания последовательности кодов.

Пример 4. Пусть коды

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

расположены в адресах

$$\alpha_1,\ \alpha_2,\ \ldots,\ \alpha_n,$$

для которых операция следования не определена.

Тогда введем следующее распределение адресов:

γ			γ_1	
$\beta+1$	$\beta+2$	. . .	$\beta+n$	
α_1	α_2	. . .	α_n	a
a_1	a_2	. . .	a_n	

Программа обозревания кодов в стандартном адресе a имеет следующий вид:

Программа 9

k_1. $^3\gamma \Rightarrow a$

k_2. $'\gamma+1 \Rightarrow \gamma$

k_3. $P\{'\gamma \leqslant {}'\gamma\}\ k_1$ *ост.*

Пример 5. Вычисление суммы $s = \sum_{i=1}^{n}\sum_{j=i}^{n} a_{ij}$.

Предполагается, что $a_{ij} = {}'(\alpha + (i-1)n + j)$ (размещение элементов матрицы по строкам). Логический смысл задачи состоит в просмотре (и суммировании) «наддиагональных» элементов прямоугольной матрицы, т. е. элементов, расположенных на диагонали и над ней. Введем адреса: φ — для обозревания диагональных элементов матрицы, ψ — для обозревания «наддиагональных» элементов строк, χ — для элементов последнего

столбца матрицы. Для начала положим $'\varphi = \alpha + 1$, $'\chi = \alpha + n$, $'\gamma = 0$, $'\delta = n$, $'\bar{\chi} = \alpha + n^2$.

Распределение адресов

$'(\alpha + (i-1)n + j) = a_{ij}$

$'\varphi = \alpha + 1$

$'\chi = \alpha + n$

$'\gamma = 0 \mid s$

$'\delta = n$

$'\bar{\chi} = \alpha + n^2$

ψ — рабочая

Программа 10

k_1. $'\varphi \Rightarrow \psi$;

k_2. $^2\psi + '\gamma \Rightarrow \gamma$;

$'\psi + 1 \Rightarrow \psi$;

$P\{'\psi \leqslant '\chi\}\, k_2$;

$'\varphi + '\delta + 1 \Rightarrow \varphi$; $'\chi + '\delta \Rightarrow \chi$;

$P\{'\chi \leqslant '\bar{\chi}\}\, k_1$ *ост.*

Пусть теперь требуется максимальный по модулю наддиагональный элемент данной матрицы (a_{ij}) поместить в ячейку γ. В приведенной программе достаточно формулу k_2 заменить формулой $\max\{'\gamma, {}^2\psi\} \Rightarrow \gamma$.

Пример 6. m-ю степень подстановки $i_1, i_2, \ldots, i_k$ — чисел $1, 2, \ldots, k$, размещенную по ячейкам $'(\alpha + j) = i_j$ $(j = 1, 2, \ldots, k)$, поместить соответственно в ячейки $\beta + 1$, $\beta + 2$, $\ldots$, $\beta + k$ $(\beta + 1 > \alpha + k)$. Введем адреса: φ — для обозревания элементов исходной подстановки, ψ — для элементов ее m-й степени. Положим для начала $'\varphi = \alpha + 1$, $'\bar{\varphi} = \alpha + n$, $'\psi = \beta + 1$.

Распределение адресов

$'(\alpha + j) = i_j$ $(j = 1, 2, \ldots, k)$ исходная подстановка

$'\varphi = \alpha + 1$

$'\bar{\varphi} = \alpha + n$

$'\psi = \beta + 1$

$'(\beta + 1)$, $\ldots$, $'(\beta + k)$ — m-я степень исходной подстановки

Программа 11

k_1. $\underbrace{'(\ldots'('(}_{m \text{ раз}}{}^2\varphi + \alpha) + \alpha) + \ldots \ldots + \alpha) \Rightarrow '\psi$;

$'\varphi + 1 \Rightarrow \varphi$; $'\psi + 1 \Rightarrow \psi$;

$P\{'\varphi \leqslant '\bar{\varphi}\}\, k_1$ *ост.*

§ 3. Программы задач линейной алгебры

В этом параграфе на ряде примеров мы иллюстрируем методику построения адресных программ для решения задач линейной алгебры.

К схемам обозревания информации, характерным для задач линейной алгебры, может быть сведено обозревание информации в самых разнообразных задачах.

Входными и выходными данными в задачах линейной алгебры могут служить скаляры, векторы или матрицы. В случае векторных данных будем считать, что их компоненты заданы в виде последовательностей, т. е. размещены в последовательностях адресов. В случае матричных данных будем считать, что элементы матриц размещены по строкам или столбцам. В первом случае адресом элемента, размещенного на i-й строке в j-м столбце, будет служить величина

$$\alpha+(i-1)n+j,$$

где n — число столбцов в матрице, α — некоторая константа, определяющая начальный адрес массива. Во втором случае этот адрес определится как

$$\alpha+(j-1)m+i,$$

где m — число строк в матрице.

Поскольку решение сложных задач линейной алгебры может быть сведено к использованию подпрограмм, представляющих собой программы для решения более элементарных задач, целесообразно к последним предъявить некоторые стандартные требования. Такими требованиями являются уже сформулированные требования о размещении данных о векторах и матрицах. Потребуем еще, чтобы и рабочие фиксаторы, используемые в отдельных подпрограммах, были для них общими: ϕ_1, ϕ_2, ϕ_3, ...

Например, если входными являются

1-й массив: $\alpha+1$, $\alpha+2$, ...
2-й массив: $\beta+1$, $\beta+2$, ...

и выходным

3-й массив: $\gamma+1$, $\gamma+2$, ...,

то для работы подпрограммы будем считать, что фиксаторы ϕ_1, ϕ_2 и ϕ_3 должны фиксировать начала этих массивов:

$$'\phi_1=\alpha+1,$$
$$'\phi_2=\beta+1,$$
$$'\phi_3=\gamma+1.$$

В частном случае, когда один из массивов представляет собой одну ячейку (входной или выходной величиной является скаляр), будем считать, что эта величина непосредственно хранится по адресу ϕ_i.

Таким образом, для входных и выходных массивов мы задаем лишь стандартные фиксаторы или адреса (в случае скалярных величин); сами же массивы, фиксируемые этими

фиксаторами, могут быть размещены в произвольных последовательностях.

Для параметров, характеризующих размерности векторов или матриц, введем также стандартные ячейки: ξ_1, ξ_2, ...

Пример 1. Вычитание векторов

$$X - Y = Z.$$

Пусть ψ_1 фиксирует первую компоненту X, ψ_2 фиксирует первую компоненту Y, ψ_3 — содержит адрес, отведенный для первой компоненты Z, $'\xi = n$ — размерность векторов X, Y, Z.

Программа 1

$$k_{11}.\quad 0 \Rightarrow \delta;$$
$$k_{12}.\quad '('\psi_1 + '\delta) - '('\psi_2 + '\delta) \Rightarrow '\psi_3 + '\delta;$$
$$'\delta + 1 \Rightarrow \delta;$$
$$P\{'\delta < '\xi\}\, k_{12}\, Я.$$

Пример 2. Умножение вектора на скаляр

$$Xa = Y.$$

Пусть ψ_1 фиксирует первую компоненту X; $'\psi_2 = a$; $'\xi = n$ — размерность векторов X, Y; ψ_3 содержит адрес, отведенный для первой компоненты Y.

Программа 2

$$k_{21}.\quad 0 \Rightarrow \delta$$
$$k_{22}.\quad '('\psi_1 + '\delta)\, '\psi_2 \Rightarrow '\psi_3 + '\delta;$$
$$'\delta + 1 \Rightarrow \delta;$$
$$P\{'\delta < '\xi\}\, k_{22}\, Я.$$

Пример 3. Вычисление квадрата модуля вектора

$$\|X\|^2 = a.$$

Пусть ψ_1 — фиксирует первую компоненту вектора X, ψ_2 — адрес, по которому помещается результат, $'\xi = n$ — размерность вектора X.

Программа 3

$$k_{31}.\quad 0 \Rightarrow \delta;\quad 0 \Rightarrow \psi_2;$$
$$k_{32}.\quad '('\psi_1 + '\delta)^2 + '\psi_2 \Rightarrow \psi_2;$$
$$'\delta + 1 \Rightarrow \delta;$$
$$P\{'\delta < '\xi\}\, k_{32}\, Я.$$

Пример 4. Умножение n-квадратной матрицы A на вектор X.

$$AX = Y.$$

Пусть ψ_1 фиксирует первый элемент матрицы A, заданной по строкам; ψ_2 фиксирует первый элемент вектора X; ψ_3 содержит адрес, отведенный для первой компоненты Y; $'\xi = n$.

Программа 4

$$k_{41}.\quad 0 \Rightarrow \delta_1;\quad 0 \Rightarrow \delta_3;$$
$$k_{42}.\quad 0 \Rightarrow \delta_2;\quad 0 \Rightarrow {}'\psi_3 + {}'\delta_3;$$
$$k_{43}.\quad {}'({}'\psi_1 + {}'\delta_1) \times {}'({}'\psi_2 + {}'\delta_2) + {}'({}'\psi_3 + {}'\delta_3) \Rightarrow {}'\psi_3 + {}'\delta_3;$$
$${}'\delta_1 + 1 \Rightarrow \delta_1;\quad {}'\delta_2 + 1 \Rightarrow \delta_2;$$
$$P\{{}'\delta_2 < {}'\xi\}\, k_{43};$$
$${}'\delta_3 + 1 \Rightarrow \delta_3;$$
$$P\{{}'\delta_3 < {}'\xi\}\, k_{42}\, Я.$$

Пример 5. Матрица A (n-квадратная) задана по строкам. Умножить транспонированную к ней матрицу A^* на вектор X:

$$A^*X = Y.$$

Воспользуемся обозначениями примера 4.

Программа 5

$$k_{52}.\quad 0 \Rightarrow \delta_1;\quad 0 \Rightarrow \delta_3;$$
$$k_{52}.\quad 0 \Rightarrow \delta_2;\quad 0 \Rightarrow {}'\psi_3 + {}'\delta_3;$$
$$k_{53}.\quad {}'({}'\psi_2 + {}'\delta_1) \times {}'({}'\psi_2 + {}'\delta_2) + {}'({}'\psi_3 + {}'\delta_3) \Rightarrow {}'\psi_3 + {}'\delta_3;$$
$${}'\delta_1 + {}'\xi \Rightarrow \delta_1;\quad {}'\delta_2 + 1 \Rightarrow \delta_2;$$
$$P\{{}'\delta_2 < {}'\xi\}\, k_{53};$$
$${}'\delta_3 + 1 \Rightarrow \delta_3;\quad \delta_1 - {}'\xi\,({}'\xi - 1) + 1 \Rightarrow \delta_1;$$
$$P\{{}'\delta_3 < {}'\xi\}\, k_{52}\, Я.$$

Во всех приведенных примерах обозревание информации осуществлялось с использованием счетчиков; содержимое фиксаторов, используемых в программе, не изменялось.

Приведем теперь пример более сложной задачи, решение которой почти полностью сводится к использованию приведенных программ 1 — 5.

Пример 6. Решение системы линейных алгебраических уравнений методом Фридмана (см. Фридман В. М. [1]).

Система линейных алгебраических уравнений

$$AX = F,$$

где $A=(a_{ij})$ — матрица коэффициентов системы, $F=(f_1, \ldots, f_n)$ — вектор свободных членов, X — искомый вектор решений, может быть решена путем следующего итерационного процесса.

Пусть X^0 — некоторый (произвольный) вектор нулевых приближений. Тогда вектор $(k+1)$-х X^{k+1} приближений определяется по формулам

$$X^{k+1}=X^k-a_{k+1}A^*(AX^k-F),$$

где

$$a_{k+1}=\frac{\|AX^k-F\|^2}{\|A^*(AX^k-F)\|^2},$$

A^* — транспонированная матрица A. Итерационный процесс прекращается, когда $\|AX^{k+1}-F\|$ становится меньше некоторого заданного числа ε (пусть $'\omega=\varepsilon$), и значения X^{k+1} принимаются в качестве решений.

Пусть исходная матрица A задана по строкам

$$'(\alpha+(i-1)n+j)=a_{ij} \qquad (i, j=1, 2, \ldots, n);$$

вектор правых частей F задан в виде последовательности

$$'(\vartheta+i)=f_i \qquad (i=1, 2, \ldots, n).$$

Вектор нулевых приближений X^0 задан также в виде последовательности

$$'(\beta+i)=x_i^0.$$

Выделим n рабочих ячеек $\delta+1, \ldots, \delta+n$, в которые в процессе счета будут помещаться компоненты векторов

$$AX^k, \quad AX^k-F, \quad A^*(AX^k-F).$$

Для обозревания информации введем следующие адреса второго ранга:

$$'\gamma_1=\alpha+1; \quad '\gamma_4=\vartheta+1;$$

$$'\gamma_2=\beta+1; \quad '\varphi=\delta+n$$

$$'\gamma_3=\delta+1; \quad \text{кроме того,} \quad '\zeta=n; \quad '\omega=\varepsilon.$$

Введем понятие *адресной формулы вхождения*

$$\pi\alpha\beta.$$

Под *формулами вхождения* будем понимать переход к выполнению адресной формулы, обозначением которой служит индекс α, с возвратом при наличии знака Я к выполнению формулы, обозначением которой служит индекс β. Другими словами, формула вхождения означает переход к выполнению

подпрограммы, начальной командой которой служит команда α, с возвратом после ее выполнения на команду β.

Условимся, кроме того, индекс β опускать, если он означает следующую команду программы.

В принятых обозначениях для сформулированной задачи получаем следующую программу.

Программа 6

		Заполнение рабочих фиксаторов подпрограмм			Передача управления подпрограмме
		ψ_1	ψ_2	ψ_3	
N_1.	$'\gamma_1 \Rightarrow \psi_1;\ '\gamma_2 \Rightarrow \psi_2;\ '\gamma_3 \Rightarrow \psi_3$ πk_{41}	$'\gamma_1$	$'\gamma_2$	$'\gamma_3$	$AX = Y$
	$'\gamma_3 \Rightarrow \psi_1;\ '\gamma_4 \Rightarrow \psi_2$ πk_{11}	$'\gamma_3$	$'\gamma_4$	$'\gamma_3$	$X - Y = Z$
N_2.	$P\{\lvert {}^2\psi_3 \rvert > {}'\omega\}\ N_3$ $'\psi_3 + 1 \Rightarrow \psi_3$ $P\{'\psi_3 \leqslant {}'\overline{\varphi}\}\ N_2 N_4$				
N_3.	πk_{31}	$'\gamma_3$	$\lVert\ \rVert^2$		$\lVert X \rVert^2$
	$'\psi_2 \Rightarrow r$ $'\gamma_1 \Rightarrow \psi_1;\ '\gamma_3 \Rightarrow \psi_2;\ '\gamma_3 \Rightarrow \psi_3$ πk_{51}	$'\gamma_1$	$'\gamma_3$	$'\gamma_3$	$A^* X = Y$
	$'\gamma_3 \Rightarrow \psi_1$ πk_{31}	$'\gamma_3$	$\lVert\ \rVert^2$		$\lVert X \rVert^2$
	$'r : '\psi_2 \Rightarrow \psi_2$ πk_{21}	$'\gamma_3$	a_k	$'\gamma_3$	$Xa = Y$
	$'\gamma_2 \Rightarrow \psi_1;\ '\gamma_3 \Rightarrow \psi_2;\ '\gamma_2 \Rightarrow \psi_3$ $\pi k_{11} N_1$	$'\gamma_2$	$'\gamma_3$	$'\gamma_2$	$X - Y = Z$
N_4.	πk_{61} Я	$'\gamma_2$			*Подпрограмме группового перевода и печати*

При наличии стандартных подпрограмм для основных операций линейной алгебры программирование задач линейной алгебры сводится к построению программ, управляющих работой подпрограмм. Как видно из примера, управляющая программа состоит из передачи управления на подпрограмму. При этом подготовка к работе подпрограмм состоит в соответствующем заполнении рабочих фиксаторов подпрограмм.

Здесь предполагается, что все подпрограммы используют ячейку ξ, содержащую порядок исходной системы n.

Помимо приведенных нами программ 1—5, в программе 6 используется подпрограмма группового перевода и печати с начальной командой k_{61}.

Рассмотрим пример, встречающийся в задачах линейной алгебры, арифметический счет в котором полностью отсутствует.

Пример 7. Найти минор $C = (C_{ij})$ элемента a_{ks} квадратной матрицы $A = (a_{ij})$. Предполагается, что матрица задана по строкам

$$'(\alpha + (i-1)n + j) = a_{ij} \qquad (i, j = 1, 2, \ldots, n)$$

числа k и s заданы. Получаемый минор помещается по строкам в ячейки $\beta + 1, \beta + 2, \ldots, \beta + (n-1)^2$.

Пусть φ обозревает элементы матрицы, первоначально $'\varphi = \alpha + 1$; $\bar{\varphi}$ обозревает элементы последнего столбца матрицы, первоначально $'\bar{\varphi} = \alpha$; $\bar{\bar{\varphi}}$ фиксирует последний элемент матрицы:

$$'\bar{\bar{\varphi}} = \alpha + n^2;$$

χ обозревает элементы выбрасываемого столбца, первоначально $'\chi = \alpha + s - n$; ω фиксирует первый элемент выбрасываемой строки:

$$'\omega = \alpha + (k-1)n + 1;$$

ψ обозревает адреса минора матрицы.

Примем следующий порядок работы алгоритма: последовательно просматриваются элементы данной матрицы по строкам; элементы, не принадлежащие выбрасываемым строке и столбцу, переносятся в соответствующую последовательность.

Распределение адресов

$'(\alpha + (i-1)n + j) = a_{ij}$
$(i, j = 1, 2, \ldots, n)$

$'\varphi = \alpha + 1$

Программа 7

R_1. $P\{'\varphi \neq '\omega\}\, R_2$

$'\varphi + n \Rightarrow \varphi$; $'\bar{\varphi} + n \Rightarrow \bar{\varphi}$;
$'\chi + n \Rightarrow \chi$.

$$'\bar{\varphi} = \alpha \qquad R_2.\ P\{'\varphi > '\bar{\varphi}\}\ \text{В}$$

$$'\bar{\bar{\varphi}} = \alpha + n^2 \qquad '\bar{\varphi} + n \Rightarrow \bar{\varphi};\quad '\chi + n \Rightarrow \chi;$$

$$'\chi = \alpha + s - n \qquad R_3.\ P\{'\varphi = '\chi\}\ R_4$$

$$'\omega = \alpha + (k-1)n + 1 \qquad {}^2\varphi \Rightarrow '\psi$$

$$'\psi = \beta + 1 \qquad '\psi + 1 \Rightarrow \psi$$

$$'(\beta + (i-1)n + j) = c_{ij} \qquad R_4.\ '\varphi + 1 \Rightarrow \varphi$$

$$(i, j = 1, 2, \ldots, n-1) \qquad P\{'\varphi > '\bar{\varphi}\}\ R_1 R_3.$$

Упражнения

Составить адресные программы:

1. Вычисление косинуса угла между двумя векторами.
2. Умножение матриц.
3. Вычисление следа матрицы.
4. Вычисление квадрата матрицы.
5. Решение системы линейных уравнений методом Зейделя.
6. Решение системы методом простой итерации.
7. Решения системы в случае симметричной матрицы методом квадратных корней (в памяти хранятся по строкам лишь элементы главной диагонали и наддиагональные элементы).
8. Решение системы линейных алгебраических уравнений методом исключений с выбором главного элемента (см. Фаддеева В. Н. [1]).
9. Вычисление определителя путем образования нулей в столбце, содержащем максимальный по модулю элемент (см. Фаддеева В. Н. [1]).

§ 4. Неарифметические задачи

В настоящем параграфе рассматриваются логические задачи, схемы решения которых хорошо известны или легко определяются, но уточнение этих схем в виде программы требует значительных усилий. Если известна логика решения задачи, которую может осуществить человек, то это еще не означает, что алгоритм решения задачи определен в такой степени, чтобы его можно было осуществить на машине.

Только точная формализация всех этапов решения задачи может обеспечить успешное программирование для *ЦАМ*. Такая степень формализации процесса решения задач достигается в адресном программировании. При этом, как было показано в предыдущей главе, переход от адресных программ к программам для конкретной *ЦАМ* не представляет серьезных затруднений и может быть автоматизирован. В этом параграфе рассматриваются следующие задачи:

1. Перевод арифметических формул в запись без скобок.
2. Выполнение нормальных алгоритмов (алгоритмов А. А. Маркова) на *ЦАМ* (см. Марков А. А. [1]).

3. Дифференцирование выражений в элементарных функциях.

4. Приведение формул исчисления высказываний к нормальной форме.

1. Перевод арифметических формул с двуместными операциями в запись без скобок. Постановка задачи. Рассмотрим выражения, содержащие величины, которые мы будем обозначать малыми латинскими буквами a, b, ..., и двуместные (бинарные) операции (например, сложение, вычитание, умножение и т. п.), которые обозначаем буквой 0, или символами $+$, $-$, $\times$ и т. д.

Формулы определяются в записи со скобками — открывающими (знак открывающей скобки «(») и закрывающими (знак закрывающей скобки «)»), следующим образом:

Определение 1. Элементарной формулой называется выражение $(a)0(b)$, где a и b — величины. Формулой называется выражение $(A)0(B)$, где A и B — формулы.

В бесскобочной записи формулы определяются так:

Определение 2. Элементарной формулой называется выражение $0ab$, где a и b — величины. Формулой называется выражение $0AB$, где A и B — формулы.

Например, арифметической формулой с двуместными операциями является выражение

$$(((a)+(b))\times(c))-((a)\times((b)+(c))).$$

Эта же формула в записи без скобок имеет вид

$$-\times+abc\times a+bc.$$

Порядок применения операций в бесскобочной записи формулы удобно описывается словесно. В нашем примере формула в бесскобочной записи может быть прочитана так: разность между произведением суммы a и b на c и произведением a на сумму b и c. Заметим, что принятое нами определение записи формул со скобками приводит к более громоздким выражениям, чем обычные правила записи арифметических формул. Зато формулы по определению, сформулированному выше, более просты для анализа и построения алгоритма перевода в запись без скобок.

Переход от записи формул со скобками к бесскобочной записи можно осуществить двумя способами.

Первый способ состоит в последовательном отыскании в формуле, записанной со скобками, первой слева элементарной формулы вида $(a)0(b)$ и запись её без скобок $0ab$. Затем процесс повторяется до тех пор, пока вся формула не будет записана без скобок.

В приведенном примере последовательность преобразований будет выглядеть так:

1-й шаг $((+ab)\times(c))-((a)\times((b)+(c)))$
2-й шаг $(\times+abc)-((a)\times((b)+(c)))$
3-й шаг $(\times+abc)-((a)\times(+bc))$
4-й шаг $(\times+abc)-(\times a+bc)$
5-й шаг $-\times+abc\times a+bc$ — искомый результат.

Второй способ состоит в том, что данная формула в записи со скобками рассматривается как выражение $(A)0(B)$; отыскивается операция, соответствующая такому представлению формулы, и эта сложная формула переписывается в запись без скобок $0AB$. Затем каждое выражение A и B, являющееся в свою очередь формулой вида $(A_1)0(A_2)$ и $(B_1)0(B_2)$, переписывается в запись без скобок, и т. д.

Приведеный пример формулы в этом случае преобразуется в запись без скобок следующим образом:

1-й шаг $-((a)+(b))\times(c)(a)\times((b)+(c))$
2-й шаг $-\times(a)+(b)\,c\times a\,(b)+(c)$
3-й шаг $-\times+abc\times a+bc$ — искомый результат.

Построим алгоритм перевода формул в запись без скобок по второму правилу. Прежде всего, необходимо уточнить содержание работы алгоритма. Для этого нужно выяснить, как найти операцию в рассматриваемом выражении, которая определяет его как формулу $(A)0(B)$. Правило нахождения операции, определяющей формулу, непосредственно следует из определения формулы в записи со скобками: *перед операцией, определяющей формулу, число открывающих скобок равно числу закрывающих скобок.*

Кодирование информации и программирование. Исходная информация для алгоритма перевода формул в запись без скобок состоит из следующих элементов: открывающие скобки «(», закрывающие скобки «)», величины a, b, ..., операции 0. Код скобок должен содержать два признака: признак скобки, (знак /) и признак вида скобки, открывающей или закрывающей (знаки «(» и «)» соответственно). Коды величин и коды операций также содержат по два признака. Коды величин содержат признак величины и признак порядкового номера величины; коды операций — признак операции и порядковый номер операции.

Для стирания скобок введем «пустой» код Λ. Исходным объектом для алгоритма является формула, т. е. последовательность кодов элементов формулы, расположенных в определенном порядке. Как это было показано в § 2 настоящей главы,

порядок следования кодов элементов формулы фиксируем распределением их по адресам

$$\alpha_1,\ \alpha_2,\ \alpha_3,\ \ldots,$$

для которых определена операция следования C,

$$C\alpha_i = \alpha_{i+1} \qquad (i = 1,\ 2,\ \ldots).$$

Начало формулы фиксируем адресом второго ранга (фиксатором начала) $\varphi_н$. В конце формулы ставим знак !, отмечающий конец формулы. Введем еще счетчик в адресе α числа открывающих и числа закрывающих скобок. Таким образом, исходное распределение адресов имеет следующий вид:

Распределение адресов

$\varphi_н$					
α_1	α_2	α_3	. . .		α
Коды элементов формулы				!	0

Алгоритм состоит из двух частей:

1. Поиск операции, определяющей формулу.
2. Переход к бесскобочной записи формулы.

Программа 1. Перевод формул в запись без скобок.

k_1. $'\varphi_н \Rightarrow \varphi_0$

k_2. $P\{^2\varphi_0 = (\}\ k_3$
$C'\varphi_0 \Rightarrow \varphi_0$
$P\{^2\varphi_0 \neq !\}\ k_2k_9$

k_3. $'\varphi_0 \Rightarrow \varphi_1$

k_4. $P\{^2\varphi_1 = /\}\ k_6$

k_5. $C'\varphi_1 \Rightarrow \varphi_1$
Pk_4

k_6. $P\{^2\varphi_1 = (\}\ k_7$
$'\alpha - 1 \Rightarrow \alpha$
$P\{'\alpha \neq 0\}\ k_5$
$C'\varphi_1 \Rightarrow \varphi_1$
Pk_8

k_7. $'\alpha + 1 \Rightarrow \alpha$
Pk_5

Начало работы алгоритма — поиск и фиксирование фиксатором φ_0 первой открывающей скобки в формуле. Фиксатором φ_0 последовательно обозреваются элементы формулы до первой открывающей скобки. После фиксирования первой открывающей скобки управление передается оператору k_3. Второй этап алгоритма состоит в счете числа открывающих и закрывающих скобок, начиная с $^2\varphi_0$. Фиксатором φ_1 последовательно обозреваются элементы формулы после φ_0 и отыскиваются скобки. Если φ_1 фиксирует открывающую скобку, то управление передается оператору k_7 — в α прибавляется 1. Если φ_1 фиксирует закрывающую скобку, то из α вычитается 1 и содержимое α сравнивается с нулем. Если $'\alpha \neq 0$, то продолжается счет скобок; если же $'\alpha = 0$, то происходит переход ко второй части алгоритма, программа которого начинается с оператора k_8.

Прежде чем строить программу второй части алгоритма, заметим, что переход от записи $(A)0(B)$ к записи $0AB$ означает, что знак операции нужно перенести на место первой открывающей скобки, а скобки, стоящие слева и справа от знака операции, стереть. Кроме того, нужно найти *последнюю* закрывающую скобку в формуле и ее также стереть. Этот этап работы алгоритма — стирание последней закрывающей скобки в формуле — является довольно сложным, так как требует, по существу, повторения поиска, аналогичного поиску операции, определяющей формулу. Но заметим, что если оставить последнюю закрывающую скобку в формуле, то при этом не изменится правило поиска операции, определяющей формулу, поэтому можно упростить алгоритм, осуществляя переход к записи без скобок в два приема. На первом шагу происходит переход от выражения $(A)0(B)$ к выражению $0AB)$. После записи всех операций в бесскобочном виде происходит стирание всех оставшихся закрывающих скобок. Признаком окончания записи всех операций в бесскобочном виде может служить отсутствие открывающих скобок в формуле, что определяется предикатом $P\{{}^2\varphi_0 = !\}$.

Итак строим программу перехода от записи $(A)0(B)$ к записи $0AB$.

k_8. $\quad {}^2\varphi_1 \Rightarrow {}'\varphi_0$
$\quad C\,{}'\varphi_1 \Rightarrow \varphi_2$
$\quad \Lambda \Rightarrow {}'\varphi_2$
$\quad \bar{C}\,{}'\varphi_1 \Rightarrow \varphi_2$
$\quad \Lambda \Rightarrow {}'\varphi_2$
$\quad \Lambda \Rightarrow {}'\varphi_1$
$\quad Pk_2$

k_9. $\quad {}'\varphi_n \Rightarrow \varphi$

k_{10}. $\quad P\{{}^2\varphi = !\}$ ост.
$\quad P\{{}^2\varphi \neq /\}\, k_{11}$
$\quad \Lambda \Rightarrow {}'\varphi$

k_{11}. $\quad C\,{}'\varphi \Rightarrow \varphi$
$\quad Pk_{10}$

Код операции, фиксированный φ_1, переносится на место первой открывающей скобки, фиксированной φ_0. Затем стираются скобки, соседние с кодом операции, и код операции. Оператором k_9 начинается заключительный этап алгоритма — стирание оставшихся закрывающих скобок. Дополнительных пояснений к этому этапу не требуется.

Рекомендуем читателю составить программу, переписывающую полученную формулу в бесскобочной записи в новую последовательность адресов без (пустого) кода Λ.

2. Реализация нормальных алгоритмов А. А. Маркова на ЦАМ. Выясним сначала смысл понятия *нормального алгоритма*, введенного А. А. Марковым (за подробностями мы

отсылаем читателя к монографии А. А. Маркова, Теория алгоритмов, Труды Матем. ин-та АН СССР, т. 42, 1954 г.). Исходной информацией для нормального алгоритма служат слова — последовательности элементарных символов, называемых буквами некоторого алфавита. Алгоритм перерабатывает исходное слово в некоторое другое слово, которое является результатом действия алгоритма. В качестве элементарной операции преобразования слов принимается операция *подстановки*. Пусть дано некоторое слово X и слово A. Говорят, что слово A *входит* в слово X, если X представимо в виде $X = YAC$. Для данных слов X и A такое представление может быть, конечно, неоднозначным. *Вхождение* слова A в слово X называется *первым*, если слово Y в представлении $YAC = X$ является наикратчайшим. Операция подстановки теперь определяется так: пусть дано слово X и подстановка $A \to B$. Если $X = YAC$, то *применение операции подстановки* $A \to B$ к слову X переводит слово X в слово YBC.

Нормальный алгоритм определяется схемой, состоящей из последовательности формул типа

$$(\mathfrak{N})\quad \begin{array}{l} A_1 \to B_1 \\ A_2 \to B_2 \\ \ldots\ldots \\ A_n \to B_n. \end{array}$$

Процесс последовательных подстановок, применяемых на основании данной схемы алгоритма $(\mathfrak{N})$ к слову X, осуществляется по следующему правилу: (1) находится первая (сверху) формула подстановки в схеме $(\mathfrak{N})$ такая, что ее левая часть входит в рассматриваемое слово X, например подстановка $A_k \to B_k$; (2) вместо первого вхождения A_k в слово X подставляется B_k, получается в результате слово X'; (3) к слову X' применяется снова сформулированное только что правило, и т. д. Если на некотором этапе ни одна из левых частей формул схемы не входит в X, то процесс прекращается, и полученное слово на этом этапе считается результатом применения нормального алгоритма с рассматриваемой схемой $(\mathfrak{N})$. Окончание алгоритма может произойти и в другом случае, когда применена подстановка схемы $(\mathfrak{N})$, отмеченная как заключительная формула. Такие заключительные формулы записываются со стрелкой с отметкой точкой $A \to \cdot B$.

Рассмотрим, например, нормальный алгоритм для вычисления абсолютного значения разности двух натуральных чисел. Натуральные числа n и m записываются в алфавите, содержащем один знак 1, т. е. каждое натуральное число представляется числом знаков, равным числу единиц в числе. Пара чисел n и m

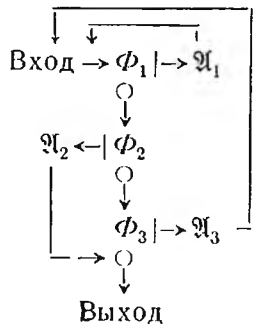
записывается с разделительным знаком $*$: $n * m$. Соответствующий нормальный алгоритм задается схемой

$$|*| \to *$$
$$* \to$$

Словесная формулировка содержания работы нормального алгоритма может быть уточнена при помощи граф-схемы. Введем операторы $\mathfrak{A}_k$, осуществляющие подстановку $A_k \to B_k$, т. е. в применении к слову X заменяющие первое вхождение A_k в слово X словом B_k. Введем распознаватели Φ_k, определяющие наличие вхождения слова A_k в рассматриваемое слово X. Тогда граф-схема нормального алгоритма

$$A_1 \to B_1$$
$$A_2 \to \cdot B_2$$
$$A_3 \to B_3$$

выглядит так:



Теперь перейдем к построению адресной программы для нормального алгоритма А. А. Маркова. Исходной информацией служит рассматриваемое слово X, состоящее из упорядоченной последовательности букв, и формулы подстановок $\{A_k \to B_k\}$, заданные в определенном порядке. Согласно граф-схеме нормального алгоритма он состоит из распознавателей Φ_k вхождения слова A_k в рассматриваемое слово X и операторов подстановки $\mathfrak{A}\{A_k \to B_k\}$ вместо первого вхождения слова A_k в слово X левой части формулы B_k.

Алгоритм распознавания вхождения слова A в слово X осуществляется следующим образом: ищется в слове X первая буква слова A, затем проверяется совпадение слова A со словом, состоящим из последовательности букв X, начинающейся первой буквой слова A; если такое совпадение имеется, то найдено вхождение слова A в слово X, если же совпадения нет, то в слове X снова ищется следующее вхождение первой буквы слова A и т. д.

Для программного описания алгоритма распознавания первого вхождения слова A в слово X введем фиксаторы:

$\varphi_{\text{н}}$ — фиксирует начало слова X (в конце слова, как обычно ставится знак !)

$\psi_{\text{н}}$ — фиксирует начало слова A.

Коды букв слова X и коды подстановки $A \to B$ располагаем в адресах, упорядоченных операцией следования C.

Программа 2.1. Распознавание первого вхождения слова A в слово X.

k_1. $'\psi_{\text{н}} \Rightarrow \psi_1$

$'\varphi_{\text{н}} \Rightarrow \varphi_1$

k_2. $P\,\{{}^2\varphi_1 = {}^2\psi_1\}\ k_3$

$C\,'\varphi_1 \Rightarrow \varphi_1$

$P\,\{{}^2\varphi_1 \neq !\}\ k_2\ ост.$

k_3. $'\varphi_1 \Rightarrow \varphi_2$

$'\psi_1 \Rightarrow \psi_2$

k_4. $C\,'\varphi_2 \Rightarrow \varphi_2$

$C\,'\psi_2 \Rightarrow \psi_2$

$P\,\{{}^2\psi_2 = \to\}\ k_6$

$P\,\{{}^2\varphi_2 = {}^2\psi_2\}\ k_4$

k_5. $C\,'\varphi_1 \Rightarrow \varphi_1$

$P\,\{{}^2\varphi_1 = {}^2\psi_1\}\ k_3$

$P\,\{{}^2\varphi_1 \neq !\}\ k_5\ ост.$

Начало работы алгоритма — поиск и фиксирование φ_1 первой буквы слова A в слове X. Если в слове X есть первая буква слова A, то управление передается оператору k_3, которым начинается проверка (побуквенно) вхождения слова A в слово X. Если слово A входит в X, то управление передается следующей программе (оператору k_6), фиксируя ψ_2 знак $\to$, стоящий после слова A. Если на какой-то букве ${}^2\varphi_2 \neq {}^2\psi_2$, то переходим к оператору k_5, которым начинается поиск в слове X после фиксированной буквы φ_1 первой буквы слова A. Если таковая есть в слове X, то управление передается оператору k_3 — снова проверяется вхождение слова A в слово X. Если кроме первой буквы слова A нет в слове X, то алгоритм останавливается.

Программа подстановки вместо первого вхождения слова A в слово X слова B строится теперь с учетом результатов работы программы 1:

ψ_2 — фиксирует знак $\to$, после которого стоит слово B.

φ_1 — фиксирует начало слова A в слове X,

φ_2 — следующую букву после A в слове X.

Таким образом, программа выполнения подстановки $\{A \to B\}$ должна в слове X вместо букв от ${}^2\varphi_1$ до ${}^2\varphi_2$ подставить буквы слова B, стоящие после ${}^2\psi_2$ и заканчивающиеся знаком, стоящим в конце слова B, например знаком $*$. Так как слова A и B, вообще говоря, произвольной длины, то результат подстановки будем располагать в новой последовательности адресов, начало которой обозревается фиксатором $\omega_{\text{н}}$. В соответствии с этими замечаниями краткое содержание работы программы, выполняющей подстановку $\{A \to B\}$, следующее: переписывается начало слова X от буквы ${}^2\varphi_{\text{н}}$ до буквы ${}^2\varphi_1$ в последовательность адре-

сов по фиксатору $\omega_н$; затем переписывается слово B, буквы которого расположены после ${}^2\psi_2$ до знака $*$; переписывается конец слова X от ${}^2\varphi_2$ до знака !; меняются ролями фиксаторы $\varphi_н$ и $\omega_н$.

Теперь нетрудно построить соответствующую адресную программу.

Программа 2.2. Выполнение подстановки $\{A \to B\}$.

k_6. $'\omega_н \Rightarrow \omega$
$'\varphi_н \Rightarrow \varphi$

Переписывается начало слова X от ${}^2\varphi_н$ до ${}^2\varphi_1$ (оператор $k_7 \cdot {}^2\varphi \Rightarrow '\omega$).

k_7. ${}^2\varphi \Rightarrow '\omega$
$C\,'\varphi \Rightarrow \varphi$
$C\,'\omega \Rightarrow \omega$
$P\{'\varphi \neq '\varphi_1\}\, k_7$
$C\,'\psi_2 \Rightarrow \psi$

Переписывается слово B (после ${}^2\psi_2$) по фиксатору ω.

k_8. $P\{{}^2\psi = *\}\, k_9$
${}^2\psi \Rightarrow '\omega$
$C\,'\omega \Rightarrow \omega$
$C\,'\psi \Rightarrow \psi$
Pk_8

k_9. $'\varphi_2 \Rightarrow \varphi$

Оператором k_{10} начинается программа переписывания конца слова X от ${}^2\varphi_2$ до знака !.

k_{10}. ${}^2\varphi \Rightarrow '\omega$
$C\,'\varphi \Rightarrow \varphi$
$C\,'\omega \Rightarrow \omega$
$P\{'\varphi = !\}\, k_{10}$

k_{11}. ${}^2\varphi \Rightarrow '\omega$

Программа перемены ролями фиксаторов $\varphi_н$ и $\omega_н$. При этом в конце слова приписывается знак ! (оператор ${}^2\varphi \Rightarrow '\omega$).

$'\omega_н \Rightarrow \varphi$
$'\varphi_н \Rightarrow \omega_н$
$'\varphi \Rightarrow \varphi_н$
ост.

Теперь нетрудно составить программу реализации данного нормального алгоритма в соответствии с граф-схемой данного алгоритма, имея программы всех распознавателей вхождения левых частей формул подстановки в рассматриваемое слово и программы, выполняющие подстановки $\{A_k \to B_k\}$. В то же время можно построить универсальную программу, обеспечивающую реализацию на *ЦАМ* любого нормального алгорита А. А. Маркова. Ведь программа распознавания вхождения левой части подстановки в рассматриваемое слово может быть общей для всех формул подстановки, а также общей может быть программа реализации подстановки $\{A \to B\}$. Нужно только соответствующим образом организовать управление этими программами.

Исходной информацией для такой универсальной программы служит рассматриваемое для переработки слово X, начало которого фиксировано фиксатором $\varphi_н$, и последовательность формул подстановки

$$A_1 \to B_1 * A_2 \to B_2 * \ldots * A_n \to B_n !,$$

в которой знак $*$ разделяет соседние формулы подстановки. Начало формул подстановки фиксировано фиксатором $\psi_н$.

Для построения универсальной программы реализации нормальных алгоритмов на *ЦАМ* нужно построить программу перехода от одной формулы подстановки к следующей формуле подстановки. Эта программа очень проста.

Программа 2.3. Переход к следующей формуле подстановки.

k_{13}. $C\,'\psi_1 \Rightarrow \psi_1$
$P\,\{{}^2\psi_1 = !\}$ *ост.*
$P\,\{{}^2\psi_1 \neq *\}\, k_{13}$
$C\,'\psi_1 \Rightarrow \psi_1$
ост.

Напомним, что фиксатор ψ_1 в программе 2.1 фиксирует первую букву первого слова формулы подстановки. Для фиксирования первой буквы первого слова следующей формулы подстановки нужно осуществлять поиск знака *, после которого находится искомая буква.

В заключение отметим, что останов алгоритма может произойти по одному из двух признаков: либо фиксатор ψ_1 фиксирует заключительный знак ! последовательности формул подстановки, либо выполненная формула подстановки является заключительной, т. е. содержит знак $\to *$. Вспоминая (см. программу 2.1), что знак, разделяющий первое и второе слова формулы подстановки, фиксируется ψ_2, приходим к останову алгоритма по предикату

$$P\,\{{}^2\psi_2 = \to .\}$$

Общую программу приводим без пояснений.

Универсальная программа 2.4 выполнения нормального алгоритма на *ЦАМ*

k_1. $'\psi_н \Rightarrow \psi_1$
k_2. $'\varphi_н \Rightarrow \varphi_1$
k_3. $P\,\{{}^2\varphi_1 = {}^2\psi_1\}\, k_4$
$C\,'\varphi_1 \Rightarrow \varphi_1$
$P\,\{{}^2\varphi_1 \neq !\}\, k_3 k_7$
k_4. $'\varphi_1 \Rightarrow \varphi_2$
$'\psi_1 \Rightarrow \psi_2$
k_5. $C\,'\varphi_2 \Rightarrow \varphi_2$
$C\,'\psi_2 \Rightarrow \psi_2$
$P\,\{{}^2\psi_2 = \to\}\, k_8$
$P\,\{{}^2\varphi_2 = {}^2\psi_2\}\, k_5$

k_6. $C\,'\varphi_1 \Rightarrow \varphi_1$
$P\,\{{}^2\varphi_1 = {}^2\psi_1\}\, k_4$
$P\,\{{}^2\varphi_1 \neq !\}\, k_6 k_7$

k_7. $C\,'\psi_1 \Rightarrow \psi_1$
$P\,\{{}^2\psi_1 = !\}$ *ост.*
$P\,\{{}^2\psi_1 \neq *\}\, k_7$
$C\,'\psi_1 \Rightarrow \psi_1$
$P k_2$

k_8. $'\omega_н \Rightarrow \omega$
$'\varphi_н \Rightarrow \varphi$

k_9. $^2\varphi \Rightarrow {}'\omega$
$C\,'\varphi \Rightarrow \varphi$
$C\,'\omega \Rightarrow \omega$
$P\,\{'\varphi = {}'\varphi_1\}\ k_9$
$C\,'\psi_2 \Rightarrow \psi$
k_{10}. $P\,\{^2\psi = *\}\ k_{11}$
$^2\psi \Rightarrow {}'\omega$
$C\,'\omega \Rightarrow \omega$
$C\,'\psi \Rightarrow \psi$
Pk_{10}

k_{11}. $'\varphi_2 \Rightarrow \varphi$
k_{12}. $^2\varphi \Rightarrow {}'\omega$
$C\,'\varphi \Rightarrow \varphi$
$C\,'\omega \Rightarrow \omega$
$P\,\{^2\varphi \neq 1\}\ k_{12}$
$^2\varphi \Rightarrow {}'\omega$
$'\omega_{\text{н}} \Rightarrow \varphi$
$'\varphi_{\text{н}} \Rightarrow \omega_{\text{н}}$
$'\varphi \Rightarrow \varphi_{\text{н}}$
$P\,\{^2\psi_2 \neq \rightarrow\}\ k_1$ *ост.*

3. Дифференцирование выражений в элементарных функциях. Рассмотрим выражения в элементарных функциях, представленные в записи без скобок. Пусть эти выражения содержат: двуместные операции — сложение, вычитание, умножение, и одноместные операции — exp, ln, sin, cos, ... и т. п.; одноместной операцией является возведение в целую (положительную или отрицательную) степень, которую мы обозначим в виде χ_n, где индекс n означает показатель степени. Выражение в элементарных функциях в записи без скобок содержит формулы вида αAB, где α — знак двуместной операции, и формулы вида βA, где β — знак одноместной операции.

Например, выражение

$$(a + \sin y)^n \times \ln (b + y^2) \tag{1}$$

в бесскобочной записи имеет вид

$$\times \chi_n + a \sin y \ln + b\chi_2 y. \tag{2}$$

Операцию дифференцирования обозначим буквой D. Запись выражений, содержащих операцию дифференцирования, производится по следующему правилу: знак операции дифференцирования относится к формуле, которая определяется операцией, стоящей непосредственно после знака дифференцирования. Так, запись $D\beta A$ означает, что нужно продифференцировать формулу βA. Запись $D\alpha AB$ означает, что нужно продифференцировать формулу αAB. Дифференцирование выражения (2) записывается так:

$$D \times \chi_n + a \sin y \ln + b\chi_2 y. \tag{3}$$

Теперь для построения алгоритма дифференцирования необходимо сформулировать правила дифференцирования, известные из курса анализа, в форме, обеспечивающей их формальное осуществление.

Выпишем правила дифференцирования формул с одноместными и двуместными операциями в записи без скобок:

1. $D \pm uv = \pm\, Du\, Dv$
2. $D \times uv = + \times Duv \times u\, Dv$
3. $D\chi_n u = \times \times n\chi_{n-1} u\, Du$
4. $D \exp u = \times \exp u\, Du$
5. $D \ln u = \times \chi_{-1} u\, Du$
6. $D \cos u = \times - 0 \sin u\, Du$
7. $D \sin u = \times \cos u\, Du$

и т. п.

8. $Dy = 1$
9. $Da = 0.$

Заметим, что приведенные формулы дифференцирования не являются формулами подстановки, так как u и v в свою очередь могут быть формулами, и притом сколь угодно сложными. Приведенные формулы определяют алгоритмы дифференцирования соответствующих двуместных или одноместных операций. Легко понять, что алгоритм дифференцирования суммы и разности один и тот же, а алгоритм дифференцирования произведения отличается от алгоритма дифференцирования суммы и разности. В то же время все алгоритмы дифференцирования одноместных операций могут быть сделаны одинаковыми, если ввести специальную таблицу формул подстановок:

$$\chi_n \to \times \times n\chi_{n-1} * \exp \to \times \exp * \ln \to \times \chi_{-1} * \sin \to \times \cos * \cos \to \times - 0 \sin *; \qquad (4)$$

знак $\to$ при кодировании, очевидно, можно выбросить.

Наконец, применение формул дифференцирования независимой переменной и величин констант также объединяется в отдельный алгоритм.

Итак, алгоритм дифференцирования выражений в элементарных функциях должен содержать следующие алгоритмы:

1) алгоритм дифференцирования суммы и разности,
2) алгоритм дифференцирования произведения,
3) алгоритм дифференцирования одноместных операций,
4) алгоритм дифференцирования независимой переменной и константы.

Алгоритмы дифференцирования суммы, разности и произведения обычно называют правилами дифференцирования, а

алгоритм дифференцирования одноместных операций, т. е. дифференцирования элементарных функций, называется применением таблицы дифференцирования. Понятно, что алгоритм дифференцирования элементарных функций называется табличным потому, что он содержит таблицу формул подстановок (4).

1. Классификация и кодирование информации. В соответствии с изложенным выше можно привести классификацию элементов информации по их свойствам. Выражения в элементарных функциях содержат операции, числа, независимую переменную величину и знак дифференцирования D. Различаются двуместные и одноместные операции. Двуместные операции имеются двух видов: сложение (вычитание) и умножение. Одноместные операции имеются стольких видов, сколько их содержится в таблице подстановок (4).

2. Содержание работы алгоритма.

1. Отыскивается знак дифференцирования.

2. Выясняется вид операции, стоящей после знака дифференцирования D, и в соответствии с видом операции применяется тот или иной алгоритм дифференцирования формулы, которая определяется операцией, стоящей после D.

Выражение каждый раз просматривается от начала до конца, и алгоритмы дифференцирования применяются столько раз, сколько знаков D содержится в выражении. Алгоритм прекращает свою работу при отсутствии D в выражении.

Дифференцирование в примере (2) производится следующим образом:

1-й просмотр выражения — знак D стоит перед операцией умножения. Применяем правило дифференцирования произведения

$$+\times\underline{D}\chi_n+a\sin y\ln+b\chi_2 y\times\chi_n+a\sin y\underline{D}\ln+b\chi_2 y.$$

2-й просмотр выражения — знак D стоит перед операцией χ_n возведения в n-ю степень выражения $+a\sin y$. Применяем правило дифференцирования степени

$$+\times\times\times n\chi_{n-1}+a\sin y\underline{D}+a\sin y\ln+b\chi_2 y\times\chi_n+$$
$$+a\sin y\underline{D}\ln+b\chi_2 y.$$

Далее знак D стоит еще перед операцией ln. Применяем алгоритм дифференцирования

$$+\times\times\times n\chi_{n-1}+a\sin yD+a\sin y\ln+b\chi_2 y\times\chi_n+$$
$$+a\sin y\times\chi_{-1}+b\chi_2 y\underline{D}+b\chi_2 y.$$

3-й просмотр выражения — знак D стоит перед операцией сложения выражений a и $\sin y$ и перед операцией сложения выражений b и $\chi_2 y$. Применяем правило дифференцирования

суммы

$$+\times\times\times n\chi_{n-1}+a\sin y+Da\,D\sin y\ln+b\chi_2 y\times\chi_n+a\sin y\times\chi_{-1}+b\chi_2 y+Db\,D\chi_2 y.$$

4-й просмотр выражения — знак D стоит перед числом a, перед числом b, перед операцией sin и перед операцией возведения в степень независимой переменной. После работы алгоритма дифференцирования получим выражение

$$+\times\times\times n\chi_{n-1}+a\sin y+0\times\cos y\,Dy\ln+b\chi_2 y\times\chi_n+a\sin y\times\chi_{-1}+b\chi_2 y+0\times\times 2\chi_1 y\,Dy.$$

5-й просмотр выражения — знак D стоит перед независимой переменной y. В результате получим выражение

$$+\times\times\times n\chi_{n-1}+a\sin y+0\times\cos y1\ln+b\chi_2 y\times\chi_n+a\sin y\times\chi_{-1}+b\chi_2 y+0\times\times 2\chi_1 y1.$$

Данное выражение может быть записано проще, если учесть наличие тривиальных операций, таких, как сложение с нулем, умножение на единицу, возведение в первую степень и т. п. Однако такая задача упрощения является новым алгоритмом. Составление соответствующего алгоритма упрощения результата предоставляется читателю. Упростив выражение и переходя к обычной записи, получим в результате дифференцирования выражения (2) ответ:

$$+\times\times\times n\chi_{n-1}+a\sin y\cos y\ln+b\chi_2 y\times\chi_n+a\sin y\times\chi_{-1}+b\chi_2 y\times 2y=n(a+\sin y)^{n-1}\cos y\ln(b+y^2)+(a+\sin y)^n(b+y^2)^{-1}2y.$$

3. Программирование дифференцирования выражений в элементарных формулах. Алгоритм дифференцирования состоит из следующих этапов:

$\mathfrak{A}_1$ — поиск знака D в выражении,

$\mathfrak{A}_2$ — определение вида операции после знака D.

Составной частью алгоритма дифференцирования являются этапы:

$\mathfrak{A}_3$ — определение формулы,

$\mathfrak{A}_4$ — переписывание формулы,

$\mathfrak{A}_5$ — дифференцирование одноместных операций,

$\mathfrak{A}_6$ — дифференцирование суммы и разности,

$\mathfrak{A}_7$ — дифференцирование произведения,

$\mathfrak{A}_8$ — применение формул подстановки.

Перейдем к построению программ каждого этапа алгоритма.

Программа $\mathfrak{A}_1$. Поиск знака D в выражении. Пусть начало выражения задается фиксатором φ_0, конец — заключительным знаком !. Тогда поиск осуществляется по следующей программе:

Программа $\mathfrak{A}_1$.

k_{11}. $'\varphi_0 \Rightarrow \varphi_1$

k_{12}. $P\{^2\varphi_1 = D\}\, k_{21}$

$C'\varphi_1 \Rightarrow \varphi_1$

$P\{^2\varphi_1 \neq !\}\, k_{12}\mathfrak{Я}$

Оператор k_{12} передает управление следующему этапу, если есть знак D в выражении. Если знака D в выражении нет, то выход в программе дает последний предикат.

Программа $\mathfrak{A}_2$. Определение вида операции после знака D.

k_{21}. $C'\varphi_1 \Rightarrow \varphi_2$

$P\{^2\varphi_2 = \alpha\}\, k_{22}$

$P\{^2\varphi_2 = \beta\}\, \mathfrak{A}_5\mathfrak{A}_8$

k_{22}. $P\{^2\varphi_2 = \times\}\, \mathfrak{A}_7\mathfrak{A}_6$

Эта программа имеет четыре выхода в зависимости от того, какой элемент стоит после знака D: одноместная операция, число или независимое переменное, знак умножения, знак сложения или вычитания.

Программа $\mathfrak{A}_3$. Определение формулы, стоящей после данной операции. Операция, фиксированная после знака D, определяет некоторую формулу, к которой должно быть применено соответствующее правило дифференцирования. Если операция — двуместная, то после нее стоят две формулы u и v, к которым она применяется. Если операция — одноместная, то после нее идет формула u, к которой применяется эта операция. Конец формулы u определяется на основании следующего правила: в формуле число двуместных операций (коды α) на единицу меньше числа величин (коды a, y), входящих в нее. Для определения конца формулы введен счетчик в адресе δ (вначале $'\delta = 0$), определяющий разность между числом величин и числом двуместных операций, входящих в данное выражение. Появление в счетчике δ единицы определяет конец формулы.

Программа $\mathfrak{A}_3$.

k_{31}. $0 \Rightarrow \delta$

$C'\varphi_2 \Rightarrow \varphi_3$

k_{32}. $P\{^2\varphi_3 = \alpha\}\, k_{34}$

$P\{^2\varphi_3 = a \vee y\}\, k_{35}$

k_{33}. $C'\varphi_3 \Rightarrow \varphi_3$

Pk_{32}

k_{34}. $'\delta - 1 \Rightarrow \delta$

Pk_{33}

k_{35}. $'\delta + 1 \Rightarrow \delta$

$P\{'\delta \neq 1\}\, k_{33}\mathfrak{Я}$.

Если фиксатор φ_3 фиксирует двуместную операцию, то в δ вычитается единица. Если φ_3 фиксирует величину, то в δ прибавляется единица, а затем выясняется, равно ли содержимое δ единице. При выполнении этого условия программа заканчивает работу (выход $\mathfrak{Я}$)

Программу $\mathfrak{A}_4$ переписывания формулы из одной последовательности адресов в другую мы не приводим, так как нам приходилось встречаться с ней в предыдущих задачах этого параграфа.

Программа $\mathfrak{A}_5$. Дифференцирование одноместных операций. Эта программа состоит из нескольких элементов.

Подпрограмма $\mathfrak{A}_{5.1}$. Поиск в таблице подстановок операции, аналогичной фиксированной φ_2.

k_{51}. $'\rho_0 \Rightarrow \rho$
k_{52}. $P\{^2\varphi_2 = {}^2\rho\}\, k_{54}$
k_{53}. $C'\rho \Rightarrow \rho$
$P\{^2\rho \neq *\}\, k_{53}$
$C'\rho \Rightarrow \rho$
Pk_{52}

Начало таблицы задается фиксатором ρ_0. При совпадении элементов, фиксированных φ_2 в выражении u ρ в таблице, поиск заканчивается передачей управления следующему этапу (оператору k_{54}). Операторы, начиная с k_{53}, фиксируют следующую подстановку в таблице.

Подпрограмма $\mathfrak{A}_{5.2}$. Применение подстановки.

k_{54}. $C'\rho \Rightarrow \rho$
k_{55}. $^2\rho \Rightarrow {}'\psi$
$C'\rho \Rightarrow \rho$
$C'\psi \Rightarrow \psi$
$P\{^2\rho \neq *\}\, k_{55}\mathfrak{A}_3$

Предыдущая программа фиксирует (фиксатором ρ) операцию, которая определяет подстановку. Фиксатор ψ обозревает адрес, начиная с которого должна быть перенесена подстановка. По окончании переноса управление передается программе $\mathfrak{A}_3$, определяющей конец формулы u, стоящей после знака одноместной операции.

Программа дифференцирования одноместной операции содержит подпрограммы переписывания формулы u в новую последовательность, приписывание знака D после формулы и еще раз переписывание формулы u. Напомним правило дифференцирования одноместной операции

$$D\beta u = \beta' u\, Du.$$

Здесь $\beta \to \beta'$ — соответствующая подстановка из таблицы (4). Условимся в фигурных скобках, стоящих после обозначения программы, указывать фиксаторы, с которыми имеет дело программа. Тогда схема программы дифференцирования одноместных операций имеет следующий вид:

Программа $\mathfrak{A}_5$.

$\mathfrak{A}_{5.1}$ $\{\varphi_2, \rho_0, \rho\}$
$\mathfrak{A}_{5.2}$ $\{\psi, \rho\}$
$\mathfrak{A}_3$ $\{\varphi_2, \varphi_3\}$
$\mathfrak{A}_4$ $\{\varphi_2, \varphi_3, \psi\}$
$D \Rightarrow {}'\psi$
$C'\psi \Rightarrow \psi$
$\mathfrak{A}_4$ $\{\varphi_2, \varphi_3, \psi\}$

Осуществляется поиск в таблице подстановок (фиксатор ρ_0) операции (фиксатор ρ), совпадающей с $^2\varphi_2$ в выражении. Затем применяется подстановка, т. е. подстановка переписывается в последовательность (фиксатор ψ). Далее определяется конец формулы (фиксатор φ_3), стоящий после знака операции (фиксатор φ_2). Эта формула переписывается в последовательность (ψ); записывается туда же знак D и снова переписывается формула.

Программа $\mathfrak{A}_6$. Дифференцирование суммы и разности.

k_{61}. $^2\varphi_2 \Rightarrow {}'\psi$
$C'\psi \Rightarrow \psi$
$D \Rightarrow {}'\psi$
$C'\psi \Rightarrow \psi$
$\mathfrak{A}_3$ $\{\varphi_2, \varphi_3\}$
$\mathfrak{A}_4$ $\{\varphi_2, \varphi_3, \psi\}$
$D \Rightarrow {}'\psi$
$C'\psi \Rightarrow \psi$
$\mathfrak{A}_3$ $\{\varphi_3, \varphi_4\}$
$\mathfrak{A}_4$ $\{\varphi_3, \varphi_4, \psi\}$

Содержание работы программы в соответствии с правилом дифференцирования

$$D \pm uv = \pm Du\,Dv$$

следующее: переносится знак операции $(^2\varphi_2)$ и знак D в новую последовательность адресов (ψ); определяется формула $u(\varphi_2, \varphi_3)$ и в последовательность (ψ) приписывается знак D. Определяется формула $v(\varphi_3, \varphi_n)$ и переписывается в последовательность (ψ).

Программа $\mathfrak{A}_7$. Дифференцирование произведения.

k_{71}. $+ \Rightarrow {}'\psi$
$C'\psi \Rightarrow \psi$
$\times \Rightarrow {}'\psi$
$C'\psi \Rightarrow \psi$
$D \Rightarrow {}'\psi$
$C'\psi \Rightarrow \psi$
$\mathfrak{A}_3$ $\{\varphi_2, \varphi_3\}$
$\mathfrak{A}_4$ $\{\varphi_2, \varphi_3, \psi\}$
$\mathfrak{A}_3$ $\{\varphi_3, \varphi_4\}$
$\mathfrak{A}_4$ $\{\varphi_3, \varphi_4, \psi\}$
$\times \Rightarrow {}'\psi$
$C'\psi \Rightarrow \psi$
$\mathfrak{A}_4$ $\{\varphi_2, \varphi_3, \psi\}$
$D \Rightarrow {}'\psi$
$C'\psi \Rightarrow \psi$
$\mathfrak{A}_4$ $\{\varphi_3, \varphi_4, \psi\}$

Содержание работы программы в соответствии с правилом дифференцирования

$$D \times uv = + \times Duv \times uDv$$

следующее: переписываются в последовательность (ψ) знаки $+$, $\times$ и D. Определяется первая формула $u(\varphi_2, \varphi_3)$ и переписывается в последовательность (ψ). Определяется вторая формула $v(\varphi_3, \varphi_4)$ и переписывается в последовательность ψ. Приписываются последовательно знак $\times$, формула u, знак D и формула v.

Программа $\mathfrak{A}_8$. Применение формул подстановок $Dy \Rightarrow 1$ и $Da \Rightarrow 0$. Содержание работы очевидно.

Кроме построенных программ, алгоритм дифференцирования выражений в элементарных функциях содержит: программу переписывания начала выражения от φ_0 до знака D; программу переписывания выражения от φ_3 до следующего знака D после применения алгоритма дифференцирования одноместной операции; программу переписывания выражения от φ_4 до следующего знака D после применения алгоритма дифференцирования двуместных операций. Если после применения алгоритма дифференцирования в выражении далее нет знака D, то

переписывается конец выражения в последовательность после φ_3 или после φ_4.

Общая схема программы дифференцирования выражений в элементарных функциях приведена на рис. 18 (напомним, что начало выражения фиксирует φ_0; начало последовательности адресов, в которые переписывается результат дифференцирования, фиксирует ω_0; начало таблицы подстановок фиксирует ρ_0).

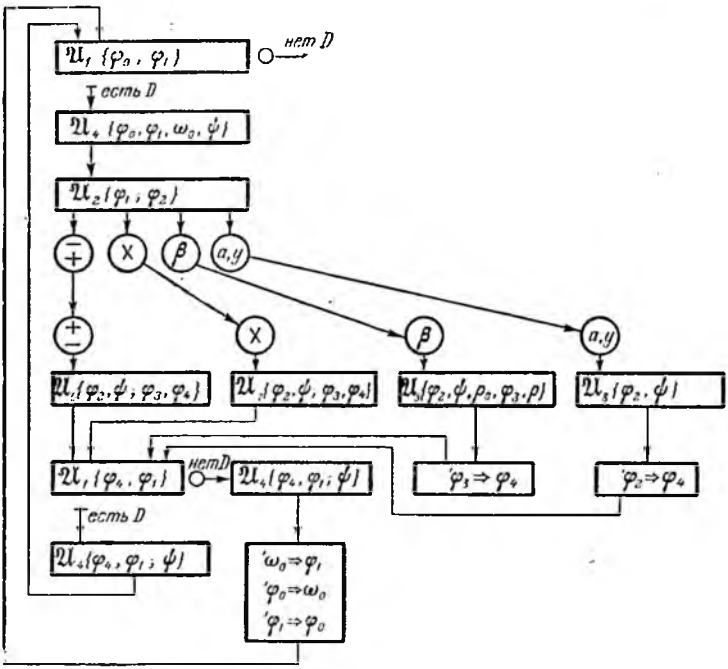


Рис. 18. Общая схема дифференцирования.

4. Приведение формул исчисления высказываний к нормальной форме. В главе I было дано определение нормальной формы выражения сложных высказываний через элементарные высказывания при помощи основных логических операций — логического сложения (знак $\vee$), логического умножения (знак $\wedge$) и отрицания.

Представление сложного высказывания в нормальной форме было нами использовано для построения формулы по таблице значений истинности сложного высказывания. Очевидно, что и обратная задача — построение таблицы значений истинности сложного высказывания по данной формуле — проще всего решается при записи формулы в нормальной форме, в частности,

по нормальной форме просто решается важная задача теории высказываний: проверка тождественной истинности сложного высказывания. Для решения этой задачи, как легко понять, достаточно проверить тождественную истинность каждого конъюнктивного члена нормальной формы. Каждый конъюнктивный член в свою очередь тождественно истинен тогда и только тогда, когда в него входит хотя бы одно элементарное высказывание вместе со своим отрицанием.

Для построения алгоритма приведения формул исчисления высказываний к нормальной форме примем бесскобочную запись формул. Нам уже известно (см. дифференцирование выражений в элементарных функциях), что в бесскобочной записи каждая двуместная операция определяет формулу вида $0AB$, где A и B — формулы. Так как операция отрицания является одноместной операцией и всегда применяется к некоторой формуле, то отрицание формулы можно отмечать отрицанием операции, которая данную формулу определяет. Например, запись $\overline{\wedge}AB$ означает, что отрицается формула $\wedge AB$, т. е. запись $\overline{\wedge}AB$ эквивалентна следующей записи со скобками:

$$\overline{[A]\wedge[B]}.$$

Аналогично запись $\overline{\vee}AB$ эквивалентна записи со скобками $\overline{[A]}\vee[B]$. Итак, формулы исчисления высказываний в бесскобочной записи с принятым условием выражения отрицания записываются через элементарные высказывания при помощи двух бинарных двуместных операций $\wedge$ и $\vee$ и этих же операций с отрицаниями, что означает отрицание соответствующей формулы, которую определяет данная двуместная операция. Рассмотрим пример сложного высказывания

$$\wedge\,\overline{\wedge}\,\overline{\vee}\,xy\vee x\bar{y}\vee\bar{x}\bar{y}.$$

В обычной записи со скобками эта формула имеет следующий вид:

$$\overline{((\overline{x\vee y})\wedge(x\vee\bar{y}))}\wedge(\bar{x}\vee\bar{y}).$$

Приведение формул для сложных высказываний к нормальной форме основано на использовании следующих правил преобразования:

1. $\overline{(\overline{A})}\equiv A$.
2. $\overline{\wedge}\,AB\equiv\vee\,\bar{A}\bar{B}$.
3. $\overline{\vee}\,AB\equiv\wedge\,\bar{A}\bar{B}$.
4. $\vee\,A\wedge BC\equiv\wedge\vee AB\vee AC$.
5. $\vee\wedge ABC\equiv\wedge\vee AC\vee BC$.

Правила 2 и 3 есть так называемые правила де-Моргана; правила 4 и 5 — дистрибутивность логического умножения относительно сложения.

В только что рассмотренном примере сложного высказывания приведение к нормальной форме по этапам выглядит следующим образом:

$$\text{исходная формула } \wedge \underline{\overline{\wedge \vee xy \vee x\bar{y}}} \vee \bar{x}\bar{y};$$

применим правило 2 к подчеркнутой формуле

$$\wedge \vee \vee xy \underline{\overline{\vee x\bar{y}}} \vee \bar{x}\bar{y}.$$

Применим правило 3 к подчеркнутой формуле

$$\wedge \underline{\vee \vee xy \wedge \bar{x}\bar{y}} \vee \bar{x}\bar{y}.$$

Применение правил 2 и 3 приводит формулу к записи, в которой отрицание относится только к элементарным высказываниям. Теперь применяем правило 4 к подчеркнутой формуле ($A = \vee xy$, $B = \bar{x}$, $C = y$)

$$\wedge \wedge \vee \vee xy\bar{x} \vee \vee xyy \vee \bar{x}\bar{y},$$

формула приведена к нормальной форме.

Перейдем теперь к описанию алгоритма приведения сложных высказываний к нормальной форме. Исходной информацией служит формула в записи без скобок с двумя бинарными операциями $\vee$ и $\wedge$ с отрицанием или без отрицания над элементарными высказываниями и над операциями. Формула определяется по следующему свойству: число высказываний в формуле на единицу больше числа операций $\wedge$ или $\vee$.

Содержание работы алгоритма:

I. Применяются правила 2 и 3 (правила де-Моргана) до тех пор, пока все знаки операций не останутся без отрицаний. При этом учитывается правило 1.

II. Применяется правило дистрибутивности либо в форме 4, если перед $\wedge$ стоит формула, либо в форме 5, если перед $\wedge$ стоит знак $\vee$.

Более подробно основные этапы работы алгоритма следующие:

I. 1. Поиск первого элемента с отрицанием от начала формулы.

I. 2. Определение вида фиксированного элемента с отрицанием. Если фиксировано высказывание с отрицанием, то выполняется п. I. 3; если фиксирована операция с отрицанием, то выполняется п. I. 4.

I. 3. Поиск следующего элемента с отрицанием.

I. 4. Применение правил 2 и 3 преобразования формулы и переход к п. I. 3.

Если в пп. I. 1. и I. 3 не обнаружен элемент с отрицанием в формуле, то п. I алгоритма закончен и переходим к п. II алгоритма.

II. 1. Поиск знака $\wedge$ после $\vee$. Если есть знак $\wedge$ после $\vee$, то выполняется п. II. 2; если нет знака $\wedge$ после $\vee$, то формула приведена к нормальной форме, алгоритм закончил работу.

II. 2. Выбор правила 4 или 5 дистрибутивности.

II. 3. Применение правила 4 и переход к п. II. 1.

II. 4. Применение правила 5 и переход к п. II. 1.

В приведенном описании основных этапов работы алгоритма легко построить адресные программы для пп. I. 1, I. 2, I. 3, II. 1, II. 2. Пусть начало формулы отмечено фиксатором φ_0, конец формулы, как обычно, определяется по знаку !.

Программа I. 1. Поиск первого элемента с отрицанием.

k_{11}. $'\varphi_0 \Rightarrow \varphi_1$

k_{12}. $P\{^2\varphi_1 = \bar{e}\}\, k_{13}$

$C\,'\varphi_1 \Rightarrow \varphi_1$

$P\{^2\varphi_1 \neq !\}\, k_{12}$Я

Знаком e обозначается произвольный элемент в формуле. Если нет элемента с отрицанием в формуле, то алгоритм прекращает работу.

Программа I. 2. Определение вида элемента, фиксированного φ_1.

k_{13}. $P\{^2\varphi_1 = \bar{A}\}\, k_{14}k_{15}$

Здесь знак A обозначает высказывание. Программа имеет два выхода: когда φ_1 фиксирует высказывание с отрицанием (переход к k_{14}), когда фиксирован знак операции с отрицанием (переход к k_{15}).

Программа I. 3. Поиск следующего после φ_1 элемента с отрицанием.

k_{14}. $C\,'\varphi_1 \Rightarrow \varphi_1$

$P\{^2\varphi_1 = \bar{e}\}$ Я$_{\bar{e}}$

$P\{^2\varphi_1 \neq !\}\, k_{14}$Я$_e$

Эта программа имеет два выхода: есть $\bar{e}$, нет $\bar{e}$.

Программа I. 4. Применение правил преобразования 2-го и 3-го.

k_{15}. $P\{^2\varphi_1 = \overline{\wedge}\}\,(\vee \Rightarrow '\varphi_1)(\wedge \Rightarrow '\varphi_1)$

$C\,'\varphi_1 \Rightarrow \varphi_2$

k_{16}. $P\{^2\varphi_2 = \bar{e}\}\,(e \Rightarrow '\varphi_2)(\bar{e} \Rightarrow '\varphi_2)$

$0 \Rightarrow \delta$

$'\varphi_2 \Rightarrow \varphi_3$

Оператор k_{15} выполняет замену знака $\overline{\wedge}$ на $\vee$ и знака $\overline{\vee}$ на $\wedge$. Оператор k_{16} заменяет $\bar{e}$ на e и наоборот, т. е. выполняется отрицание первой формулы после знака операции.

k_{17}. $P\{^2\varphi_3 = A\}('\delta + 1 \Rightarrow \delta)('\delta - 1 \Rightarrow \delta)$
$C'\varphi_3 \Rightarrow \varphi_3$
$P\{'\delta \neq 1\}\, k_{17}$

Введем в адресе δ счетчик разности между числом элементарных высказываний и числом операций в формуле. Как уже отмечалось ранее, если известно начало формулы, то конец ее определяется условием $'\delta = 1$.

k_{18}. $P\{^2\varphi_3 = \bar{e}\}(e \Rightarrow '\varphi_3)(\bar{e} \Rightarrow '\varphi_3)$

Счет ведется оператором k_{17}. Оператор k_{18} производит отрицание второй формулы после знака операции.

Программа II.1. Поиск первого знака $\wedge$ после знака $\vee$.

k_{21}. $'\varphi_0 \Rightarrow \varphi_1$
k_{22}. $P\{^2\varphi_1 = \vee\}\, k_{23}$
$C'\varphi_1 \Rightarrow \varphi_1$
$P\{^2\varphi_1 \neq !\}\, k_{22}$ ост.
k_{23}. $'\varphi_1 \Rightarrow \varphi_2$
k_{24}. $P\{^2\varphi_2 = \wedge\}\, k_{25}$
$C'\varphi_2 \Rightarrow \varphi_2$
$P\{^2\varphi_2 \neq !\}\, k_{24}$ ост.

Эта программа имеет три выхода, соответствующих случаям: когда в формуле нет знака $\vee$, когда нет знака $\wedge$ после $\vee$, когда есть знак $\wedge$ после знака $\vee$.

В последнем случае управление передается программе II.2 (оператору k_{25}).

Программа II.2. Выбор правил 4 или 5.

k_{25}. $\overline{C}'\varphi_2 \Rightarrow \varphi_3$
$P\{^2\varphi_3 = A\}\, k_{26}k_{27}$

Если перед знаком $\wedge$ стоит высказывание, то применяется правило 4 (оператор k_{26}), если перед $\wedge$ стоит $\vee$, то применяется правило 5 (оператор k_{27}).

При построении программ II.3 и II.4 нам придется переписывать формулу, так как применение правил дистрибутивности расширяет формулу (увеличивает количество элементов в ней). Введем для программы переноса последовательности элементов с началом, отмеченным фиксатором φ_0, до элемента, отмеченного фиксатором φ_1, в последовательность адресов, начало которой отмечено ψ, сокращенное обозначение $\Pi\{\varphi_0, \varphi_1, \psi\}$. Соответствующую программу, после выполнения которой фиксатор ψ фиксирует элемент $C'\varphi_1$, предоставляем читателю.

Перенос происходит, включая элемент, фиксированный φ_0, и исключая элемент, фиксированный φ_1. Заметим еще, что после выполнения программы II.2 φ_2 фиксирует знак $\wedge$, перед которым имеется знак $\vee$. Для построения программы II.4 нужно

определить знак $\vee$ в формуле, к которой применяется правило 4. Легко понять, что знак $\vee$ в формуле $\vee A \wedge BC$ определяется по условию: разность между числом элементарных высказываний и числом операций, стоящих перед знаком $\wedge$, равна нулю. В самом деле, A является формулой, в которой эта разность равна 1, наличие знака $\vee$ перед A делает эту разность равной нулю.

Программа II.4. Применение правила 4.

k_{26}. $0 \Rightarrow \delta$

$'\varphi_3 \Rightarrow \varphi_1$

k_{27}. $P\{^2\varphi_1 = A\}\,('\delta + 1 \Rightarrow \delta)\,('\delta - 1 \Rightarrow \delta)$

$P\{'\delta = 0\}\,k_{28}$

$\overline{C}\,'\varphi_1 \Rightarrow \varphi_1$

Pk_{27}

k_{28}. $'\psi_0 \Rightarrow \psi$

$\Pi\{\varphi_0, \varphi_1, \psi\}$

$C\,'\varphi_1 \Rightarrow \varphi_3$

$C\,'\varphi_2 \Rightarrow \varphi_4$

$0 \Rightarrow \delta$

$'\varphi_4 \Rightarrow \varphi_5$

k_{29}. $P\{^2\varphi_5 = A\}\,('\delta + 1 \Rightarrow \delta)\,('\delta - 1 \Rightarrow \delta)$

$C\,'\varphi_5 \Rightarrow \varphi_5$

$P\{'\delta \neq 1\}\,k_{29}$

$C\,'\varphi_5 \Rightarrow \varphi_5$

$\wedge \Rightarrow '\psi$

$C\,'\psi \Rightarrow \psi$

$\vee \Rightarrow '\psi$

$C\,'\psi \Rightarrow \psi$

$\Pi\{\varphi_3, \varphi_2, \psi\}$

$\Pi\{\varphi_4, \varphi_5, \psi\}$

$\vee \Rightarrow '\psi$

$C\,'\psi \Rightarrow \psi$

$\Pi\{\varphi_3, \varphi_2, \psi\}$

$'\varphi_5 \Rightarrow \varphi$

$k_{2.10}$. $^2\varphi \Rightarrow '\psi$ $\qquad$ $'\psi_0 \Rightarrow \varphi$

$C\,'\varphi \Rightarrow \varphi$ $\qquad$ $'\varphi_0 \Rightarrow \psi_0$

$C\,'\psi \Rightarrow \psi$ $\qquad$ $'\varphi \Rightarrow \varphi_0$

$P\{^2\varphi \neq !\}\,k_{2.10}k_{2.11}$ $\qquad$ Pk_{21}

Вначале происходит поиск знака $\vee$, определяющего формулу $\vee A \wedge BC$ перед фиксированным φ_3 последним элементом формулы A (до оператора k_{28}).

Затем начало формулы (φ_0, φ_1) переписывается в новую последовательность адресов по фиксатору ψ; φ_3, φ_4, φ_5 фиксируют начала формул A, B, C. В новую последовательность адресов (ψ) вписывается результат применения правила 4:

$$\vee A \wedge BC = \wedge \vee AB \vee AC.$$

Построение программы применения правила 5 предоставляется читателю.

ГЛАВА VI

АВТОМАТИЗАЦИЯ ПРОГРАММИРОВАНИЯ

Трудности подготовки задач для их решения на *ЦАМ* привели к развитию методов программирования, допускающих автоматизацию, т. е. возможность бо́льшую часть работы по программированию производить с использованием той же машины.

В настоящей главе приводится краткий обзор методов автоматизации программирования. Более подробно рассматриваются вопросы автоматизации, связанные с операторным методом программирования.

§ 1. Общий обзор методов

Процесс подготовки задач для решения на универсальной вычислительной машине с программным управлением естественно расчленить на следующие этапы.

1. Выбор численного метода, оценка погрешности и выбор способа контроля правильности и точности результата.

2. Составление схемы счета, т. е. разбиение вычислительного процесса на этапы и определение порядка их выполнения.

3. Программирование — представление процесса вычислений в виде последовательности элементарных операций, выполняемых отдельными командами машины.

Первые два этапа требуют высокой квалификации от специалиста, ведущего подготовку задачи для решения на *ЦАМ*. Последний этап — программирование — является более или менее технической работой, весьма трудоемкой и громоздкой и может быть выполнен значительно менее квалифицированным работником. Непосредственное составление программ сопряжено с большой затратой человеческого труда, требует исключительного внимания и аккуратности. При расписывании схемы счета по командам, как правило, неизбежны ошибки технического характера: описки, неправильное использование запоминающего устройства, ошибки в программировании передач управления и т. п. При использовании подготовленных заранее

стандартных подпрограмм приходится последние согласовывать с общей программой задачи; здесь также вероятны технические ошибки. При обнаружении ошибок в программе или при необходимости ее дополнения приходится вносить соответствующие изменения и дополнения, сдвигая те или иные части программы в *ЗУ*, перераспределяя ячейки *ЗУ*. Исправление ошибок в программе тоже требует внесения в нее соответствующих изменений. Часто при этом опять допускаются ошибки. Чтобы обеспечить безошибочное решение задачи на *ЦАМ*, приходится производить предварительно отладку программы, проводя пробные расчеты. Вся подготовительная работа: составление программы по схеме счета, отыскание и исправление ошибок в программе, изменение и внесение дополнений в программы, пробный счет на машине — требует больших затрат времени работы программистов, вычислителей и машинного времени. Часто получается так, что подготовка задачи для решения на *ЦАМ* занимает во много раз больше времени, чем непосредственное ее решение на машине. Для обеспечения работы *ЦАМ* с полной нагрузкой приходится иметь большой штат квалифицированных научных сотрудников, программистов и вычислителей (100—200 человек для обеспечения работы *ЦАМ* средней мощности).

По мере накопления опыта программирования удается разработать приемы и методы, облегчающие и упорядочивающие работу по подготовке программ для *ЦАМ*. Систематизация этих правил программирования делает возможной автоматизацию процесса составления программ.

В настоящее время получили наибольшее распространение следующие методы, совершенствующие работу по составлению программ и допускающие ее автоматизацию:

1) метод библиотечных подпрограмм,
2) метод символических адресов,
3) операторный метод, разработанный под руководством проф. А. А. Ляпунова,
4) метод интерпретирующих программ,
5) метод адресного программирования.

В настоящей главе мы приведем краткое описание первых двух методов и подробно остановимся лишь на вопросах автоматизации программирования, основанных на операторном методе составления программ; кратко излагается также метод интерпретирующих программ и метод адресного программирования.

1. Метод библиотечных подпрограмм. Значительно сокращает работу по программированию использование стандартных программ, разработанных для вычислительных схем, часто встречающихся при решении различных задач. При наличии достаточно богатой библиотеки стандартных программ имеется

возможность почти полностью составлять программы многих сложных задач из таких стандартных блоков. В этом случае программирование задачи состоит в расчленении вычислительного процесса на стандартные этапы, программы которых уже имеются, и в согласовании этих стандартных программ между собой. Программирование переходов от одних стандартных участков программы к другим является чисто технической работой и может быть формализовано. Можно сформулировать достаточно простые и вполне определенные правила, применение которых позволяет автоматизировать работу по включению стандартных программ в общую программу задачи.

При включении стандартных программ в общую программу задачи могут встретиться следующие случаи:

1. Стандартная программа помещается во внутреннюю часть *ЗУ* на заранее определенное место. Такие стандартные программы называются *замкнутыми*.

2. Стандартная программа (размещаемая во внешней части *ЗУ*) для работы помещается во внутреннюю часть *ЗУ* на место, определяемое программистом. Такие стандартные программы называются *открытыми*.

В первом случае (при использовании замкнутых стандартных программ) в основной программе необходимо учитывать, какие ячейки *ЗУ* являются входными, выходными и рабочими ячейками стандартной программы.

В основной программе должны быть предусмотрены команды переноса данных, необходимых для работы стандартной программы, во входные ячейки и команды переноса результатов вычислений по стандартной программе из выходных ячеек в заданные ячейки, используемые в основной программе. Переход на стандартную программу кодируется в основной программе отдельной командой (или несколькими командами) с помощью специальных кодов. Специальная интерпретирующая программа расшифровывает этот код, заменяя его передачей управления первой команде стандартной программы, и в случае необходимости подготавливает возврат после работы стандартной программы к соответствующему месту основной программы.

Во втором случае (при использовании открытых стандартных программ) в основной программе, помимо указанного выше, необходимо стандартную программу приводить в соответствие с местом, на котором она будет выполняться. Если стандартная программа содержит внутри себя команды передачи управления, то эти команды должны быть изменены. Исходной информацией для изменения команд передач управления стандартной программы служит число, равное разности между действительным номером первой команды подпрограммы и некоторым условным номером этой команды, на которой была рассчи-

тана стандартная программа. Это число задается программистом в команде перехода на стандартную программу.

Помимо рассмотренной работы, интерпретирующая программа может автоматически производить изменение других параметров, характеризующих подпрограммы (число циклов, коэффициенты, точность вычислений и др.). Данные для таких изменений задаются в основной программе.

Рассмотренный способ использования стандартных программ приводит, конечно, к увеличению объема программы задачи и времени ее работы. Поэтому во многих случаях более целесообразно включать стандартную программу в основную программу в качестве ее составной части. При этом меняется не только место подпрограммы, но и адреса входных, выходных и рабочих ячеек.

2. Метод символических адресов. Суть этого метода состоит в том, что программа задачи полностью составляется вручную, однако вместо действительных адресов в командах и номеров самих команд используются условные числа — символические коды. При этом вовсе не требуется, чтобы условные числа имели порядковый смысл. Важно только, чтобы различным адресам и командам приписывались различные символические коды. Необходимо также, чтобы действительные коды отличались от условных чисел. Символические коды выбираются таким образом, чтобы можно было легко и просто вносить добавления и исправления в программу. Только после того, как программа тщательно проверена и может быть введена в машину для работы, производится замена символических кодов действительными. Преобразование условных кодов в действительные номера команд и действительные адреса ячеек *ЗУ* может быть осуществлено автоматически специальной преобразующей программой. Информацией для такой программы служит программа задачи в символических кодах и таблица распределения ячеек *ЗУ*. При распределении действительных номеров команд преобразующая программа размещает команды данной программы в порядке их поступления в машину. Исключение составляют команды, имеющие с самого начала действительный номер — они помещаются в *ЗУ* по указанному номеру. Следующие за ними команды с символическими кодами помещаются подряд в порядке поступления в машину. По таблице распределения ячеек *ЗУ* в командах передачи управления происходит замена символических кодов команд действительными номерами. Замена символических адресов чисел производится по таблице распределения *ЗУ* следующим образом. Преобразующая программа просматривает данную программу и выделяет символический адрес числа в команде, отыскивает его в таблице распределения действительных адресов и заменяет соответствующим

действительным адресом, указанным в таблице. Если символический код числа в таблице отсутствует, то он заменяется очередным свободным действительным номером ячейки *ЗУ* и в таблицу распределения адресов заносится символический код и соответствующий ему действительный адрес. Машина выдает в конечном итоге преобразованную программу и таблицу распределения ячеек *ЗУ*, по которой определяется соответствие между символическими и действительными кодами.

Метод символических адресов может быть применен в сочетании с методом библиотечных стандартных программ. При этом подпрограммы могут быть составлены в символических кодах, что расширяет возможности их применения.

Преобразующая программа, осуществляющая замену символических кодов действительными адресами, довольно сложна. В то же время эта программа, по существу, является составной частью программирующей программы, рассмотренной ниже.

3. Операторный метод автоматизации программирования. Исходной информацией для составления рабочей программы задачи при помощи программирующей программы служит операторная схема программы, которая состоит из того или иного набора операторов. При составлении схемы программы обычно используются следующие виды операторов: арифметические, логические, переадресации, засылки, формирования (восстановления), циркуляции. Эти операторы называются *стандартными* операторами. Если оказывается, что для данной задачи невозможно составить удовлетворительную схему программы в стандартных операторах, то вводятся так называемые *нестандартные* операторы, имеющие произвольное функциональное назначение. Программа для нестандартных операторов строится вручную. В частности, нестандартным оператором в схеме программы может служить некоторая разработанная ранее стандартная подпрограмма.

Информация об операторах, входящих в схему программы, обычно кодируется (в условных числах) таким образом, чтобы обеспечить ее автоматический перевод программирующей программой в программу задачи.

Работа программирующей программы может осуществляться в несколько этапов, автоматически следующих один за другим.

1. Операторы, входящие в схему программы, программируются в символических кодах. Однотипные операторы программируются подряд в определенной очередности, например, арифметические операторы, логические операторы, операторы засылки, циркуляции, переадресации и восстановления, или в порядке их следования в схеме.

2. Производится экономия рабочих ячеек. При программировании арифметических формул промежуточные результаты

вычислений помещаются в рабочих ячейках *ЗУ*, число которых заранее неизвестно. Однако практически это может привести к нерациональному использованию оперативных ячеек *ЗУ*. Поэтому возникает необходимость производить программирование вычислений по формулам с возможно меньшим использованием оперативных ячеек *ЗУ* для хранения промежуточных результатов. Экономию рабочих ячеек удобнее всего производить после программирования арифметических и других операторов, в которых встречаются промежуточные результаты счета.

3. Расставляются программы отдельных операторов в соответствии с очередностью их в схеме программы.

4. Символические коды чисел и команд заменяются на действительные адреса в соответствии с распределением ячеек запоминающего устройства и т. д.

Каждый этап работы программирующей программы осуществляется специальным ее блоком. Блоки программирующей программы работают независимо друг от друга, однако результат обработки одного блока может служить исходной информацией для работы другого. С учетом этого и устанавливается очередность работы блоков программирующей программы:

А — арифметический блок,

Р — логический блок,

Г — блок переадресации,

З — блок засылки,

Ф — блок восстановления,

Н — блок обработки нестандартных операторов,

Э — блок экономии рабочих ячеек,

N — блок расстановки операторов по их порядковым номерам в схеме программы,

Д — блок присвоения действительных адресов числам и командам составленной программы.

В связи с большим объемом как программирующей программы, так и обрабатываемой информации программы блоков программирующей программы помещаются во внешней памяти машины и вызываются поочередно во внутреннюю память для обработки информации. Последняя может также частями вводиться через вводное устройство. Если объем программы заранее не известен, а значит, нельзя произвести присвоение действительных адресов, т. е. замену символических адресов истинными, то можно предусмотреть вывод программы в символических адресах для составления таблицы распределения действительных кодов.

Для того чтобы было легче ориентироваться в составленной программе, блок расстановки операторов в соответствии со схемой программы составляет таблицу характеристик, в которой указывается, на каких местах стоит каждый оператор

схемы. Таким образом, в результате работы программирующей программы может быть получена программа задачи в соответствии со схемой программы в символических и действительных кодах, таблица соответствия между символическими и действительными кодами и таблица характеристик.

Подготовка схемы программы для программирования по ней программирующей программой в значительной степени отличается от соответствующей подготовки при ручном программировании. При составлении рабочей программы программистом схема программы играет вспомогательную роль, ориентируя программиста в схеме счета и позволяя расчленять на части работу по составлению программы задачи. Неполнота схемы программы при ручном программировании может быть восполнена программистом, знающим схему счета и особенности программирования на данной машине. Ошибки в схеме программы при расписывании ее по командам также могут быть замечены программистом и исправлены в процессе работы.

При автоматическом программировании с использованием программирующей программы схема программы должна включать исчерпывающую информацию о задаче, по которой при помощи вполне определенных алгоритмов может быть составлена программа. Роль схемы программы, таким образом, существенно повышается, а значит, возрастает ответственность специалиста за правильность составления схемы счета. Ошибка, допущенная в схеме программы при подготовке ее для обработки программирующей программой, не будет исправлена и перейдет в программу задачи. Для уменьшения числа ошибок при подготовке схемы программы и повышения надежности работы программирующей программы разрабатываются правила кодировки информации, обеспечивающие достаточную ее простоту и удобства в использовании программистами. Количество информации, вводимое в машину, должно быть по возможности минимальным. Полная информация о задаче составляется по частям — отдельно для каждого оператора, входящего в схему программы, кроме того, указывается информация, относящаяся ко всей программе в целом.

Приведем описание информации, которую следует указывать для различных типов операторов.

1. Для арифметического оператора указываются формулы, по которым должен быть произведен счет. Формулы указываются в том порядке, в котором по ним нужно считать.

2. Для логического оператора указываются логическая формула

$$f(P_1, P_2, \ldots, P_n)$$

и номера операторов, которым передает управление данный ло-

гический оператор. Здесь P_1, P_2, ..., P_n — элементарные логические условия.

3. Для оператора переадресации указываются: параметр, по которому осуществляется переадресация, и шаг переадресации; величины, зависящие от данного параметра, и номера операторов, содержащих эти величины.

4. Информация об операторах засылки содержит величины, которые должны быть засланы в стандартные ячейки, или величины, которые должны быть перенесены из стандартных ячеек в последовательность, и номера операторов, которые должны использовать указанные величины в стандартных ячейках.

5. Информация об операторе восстановления содержит номера операторов, которые должны быть приведены к первоначальному состоянию.

6. Информация о нестандартном операторе может иметь произвольное значение, но записывается в форме программы.

Информация о каждом операторе содержит условные числа, по которым определяются тип оператора и его порядковый номер в схеме программы.

Информация, относящаяся ко всей составляемой программе в целом, оформляется в виде таблиц.

1. Таблица содержания элементарных логических условий. Эта таблица может быть общей для многих задач, но может быть и конкретизирована для данной задачи.

2. Таблица запаса рабочих ячеек. В этой таблице указываются массивы ячеек *ЗУ*, которые могут быть использованы в качестве рабочих.

3. Таблица распределения запоминающего устройства для чисел. Таблица содержит номера ячеек, в которых помещаются величины, участвующие в счете; в частности, таблица содержит номера ячеек, содержащих дополнительные константы, численное значение которых определено заранее, до вычислений.

4. Таблица распределения запоминающего устройства для команд. Таблица содержит номера ячеек *ЗУ*, в которых помещается действующая программа.

После того как вся информация о схеме программы составлена и закодирована в надлежащем виде, эта информация вводится в машину в определенном порядке. Сначала вводится информация об операторах, входящих в схему программы. Затем вводятся перечисленные выше таблицы. Управление передается программирующей программе, которая в результате работы выдает рабочую программу задачи и таблицу условных чисел с соответствующими им адресами.

Описанный здесь операторный метод автоматического программирования обладает важными преимуществами перед

рассмотренными ранее. Метод библиотечных подпрограмм и метод символических адресов обладают значительно меньшими возможностями. Эти методы охватывают лишь отдельные этапы работы универсальной программирующей программы. В частности, блок присвоения действительных адресов выполняет, по существу, ту же работу, что и программа в методе символических адресов. Стандартные программы могут применяться при работе программирующей программы в качестве нестандартных операторов.

Операторная программирующая программа значительно облегчает и ускоряет подготовку программы. Программисту приходится иметь дело не с программой, содержащей, быть может, сотни команд, а со схемой программы, состоящей из небольшого числа операторов. Кодирование информации может выполняться параллельно, что ускоряет работу и создает возможность надежного контроля.

Таким образом, применение программирующей программы повышает производительность труда, экономит время по составлению, откладке и проверке программы.

Операторный метод автоматизации программирования создает предпосылки к дальнейшему совершенствованию процесса программирования. Следующим этапом автоматизации является составление схемы программы по заданной схеме счета с учетом возможностей данной машины.

В заключение следует сказать, что работа по автоматизации подготовки задач для решения их на *ЦАМ* ведется в направлении максимального упрощения и сокращения предварительной работы по подготовке задач, затрачиваемой человеком. Возможности *ЦАМ* позволяют ожидать больших успехов в этой области.

4. Метод интерпретирующих программ. Сочетание перечисленных выше методов, допускающих автоматизацию программирования решения задач, позволяет максимально сократить работу по подготовке задач для решения на *ЦАМ*. При использовании библиотеки стандартных программ для общеупотребительных вычислительных схем все же значительная часть работы по подготовке исходной информации о задаче для решения ее на *ЦАМ* производится человеком. Например, для вычисления значения многочлена

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

в *ЦАМ* обычно имеется специальная стандартная программа. Чтобы применить эту стандартную программу, нужно соответствующим образом задать исходную информацию для ее работы: разместить коэффициенты многочлена $a_0, a_1, \ldots, a_n$, сте-

пень многочлена n и аргумент x в соответствующих ячейках *ЗУ*, на которых предусмотрена работа программы.

Значительно более сложную работу должен осуществить программист по подготовке исходных данных для решения системы обыкновенных дифференциальных уравнений, например, по методу Адамса — Штермера. Помимо распределения содержания *ЗУ*, нужно составить подпрограмму вычисления значений функций, входящих в правые части уравнений системы, и включить их в общую стандартную программу решения системы дифференциальных уравнений. Наконец, если стандартная программа решения целого класса вычислительных задач предусматривает возможность варьирования самой схемы этой программы в зависимости от исходных данных задачи, то для использования такой программы нужно подготовить условия для автоматического выбора схемы вычислений.

Во всех случаях процесс подготовки исходных данных задачи для работы стандартной программы, решающей данную задачу, может быть сделан алгоритмически вполне определенным, а значит, допускающим автоматизацию. Программу алгоритма подготовки исходных данных для работы стандартной программы назовем *интерпретирующей программой*. Остается только заметить, что интерпретирующие программы могут строиться не только для численных (арифметических) схем, но и для неарифметических (логических) схем.

Интерпретирующие программы, используя исходную информацию о задаче, управляют автоматическим процессом решения задачи на *ЦАМ*, в который входят программирование отдельных этапов вычисления, анализ исходной и получающейся в процессе вычислений информации и выбор схемы вычислений.

Наряду с программами, интерпретирующими исходные данные задачи для выполнения стандартных процессов вычислений, нужно строить программы, интерпретирующие результаты этих процессов вычислений для использования их в дальнейших вычислениях или для выдачи из *ЦАМ* в виде, привычном для использования человеком.

§ 2. Построение алгоритмов операторной программирующей программы

Рассмотрим более подробно процесс построения алгоритмов программирующей программы. При этом мы ограничимся рассмотрением принципиальных вопросов построения алгоритмов на примере разработки алгоритма программирования арифметических формул.

Разработка алгоритма начинается с анализа информации и выбора способа кодирования информации на основании

данного содержания работы алгоритма. Ясно, что от способа кодирования информации будет зависеть алгоритм ее обработки. В то же время при построении алгоритма могут быть дополнительно выявлены необходимые требования к кодированию информации.

При выборе способов задания и кодирования информации нужно исходить из следующих требований:

1. Информация о задаче должна иметь по возможности такую форму, которой обычно пользуется программист при ручном программировании.

2. Количество задаваемой информации должно быть по возможности минимальным.

3. Алгоритм программирования должен быть как можно проще.

Алгоритм программирования арифметических формул:

1. Анализ информации и выбор способа кодирования. Арифметические формулы содержат числа, операции и скобки. Числа могут быть двух видов: константы, значение которых известно до вычислений по формулам, и переменные величины, численное значение которых определяется в процессе счета. Для простоты задания информации мы не будем делать различия между константами и переменными величинами, считая, что все величины, входящие в формулу, являются равноправными числами. Результаты, которые получаются в процессе вычислений и не фиксируются, называются промежуточными. Промежуточные результаты вычислений помещаются в рабочих ячейках *ЗУ*.

Операции, входящие в формулы, могут быть одноместными, например ln, exp, sin и т. д., и двуместными, например сложение, вычитание, умножение, деление и т. п. Фиксирование результата вычислений осуществляется при помощи знака равенства.

Для простоты алгоритма и записи информации мы рассмотрим запись формул без скобок. В дальнейшем будем пользоваться следующими обозначениями: числа будем обозначать буквой a с индексом, одноместные операции — буквой β и двуместные операции — буквой α.

Определение формул в бесскобочной записи:

а) элементарными формулами являются выражения

$$\beta a_1, \quad \alpha a_1 a_2,$$

где a_1 и a_2 — числа;

б) формулами являются выражения

$$\beta A_1, \quad \alpha A_1 A_2,$$

где A_1, A_2 — числа, элементарные формулы или формулы.

Очевидно, что любая формула в обычной записи (со скобками) может быть переписана в бесскобочную запись. Нужно только соблюдать очередность выполнения операций. Например, формула

$$(a_1 + a_2) \times \frac{a_3}{\sin a_4} = a_5$$

в бесскобочной записи имеет вид

$$= \times + a_1 a_2 : a_3 \sin a_4 a_5.$$

Итак, мы пришли к классификации элементов информации в арифметических формулах (рис. 19).

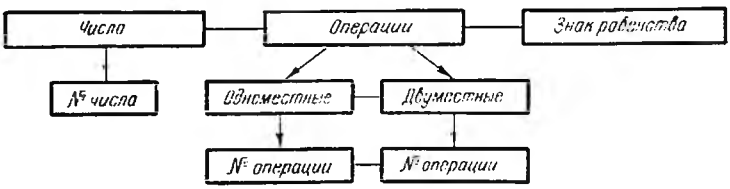


Рис. 19.

В соответствии с этим теперь можно сформулировать правило кодирования элементов информации, входящих в арифметические формулы: код каждого элемента информации состоит из двух частей; первая часть содержит код вида информации — число, операцию или знак равенства; вторая часть кода числа содержит порядковый номер числа; вторая часть кода операции в свою очередь разделяется на две части: одна содержит код одноместной (β) или код двуместной операции (α), вторая — номер операции; вторая часть знака равенства свободна. Таким образом, код числа состоит из двух групп цифр, что мы будем обозначать *ап*. Код операции состоит из трех групп цифр, что обозначим *0αп* или *0βп*.

2. Формулировка алгоритма программирования арифметических формул. Для построения данного алгоритма, прежде всего, нужно четко сформулировать содержание его работы. Поступим следующим образом: вначале постараемся до предела упростить содержание работы алгоритма, а затем, постепенно дополняя это содержание, займемся соответствующим его уточнением.

Для программирования вычислений по формуле алгоритм должен выполнять следующую работу: программировать вычисление элементарных формул, входящих в данную формулу, с фиксированием результатов в свободных рабочих ячейках *ЗУ* и заменять элементарные формулы в информации результатами

вычислений по ним. При этом мы вначале для простоты исключаем из рассмотрения знак равенства.

В соответствии с этим алгоритм программирования формул можно расчленить на следующие этапы:

1) поиск первой элементарной формулы,

2) составление программы вычисления по элементарной формуле и замена элементарной формулы результатом вычисления по ней.

Циклическое повторение работы алгоритма приведет к составлению программы счета по заданной формуле.

Рассмотрим пример: формула имеет вид

$$\times + \times + a_1 a_2 : a_3 \sin a_4 a_5 + a_1 a_2.$$

Применяем сформулированный алгоритм программирования формулы. Первой элементарной формулой является $+ a_1 a_2$. Программа вычислений по ней

+	a_1	a_2	r_1

После замены этой формулы результатом в информации получим формулу

$$\times + \times \ldots r_1 : a_3 \sin a_4 a_5 + a_1 a_2.$$

Снова применяем алгоритм. Первая элементарная формула $\sin a_4$, программа ее

sin	a_4	—	r_2

Преобразованная информация

$$\times + \times \ldots r_1 : a_3 \ldots r_2 a_5 + a_1 a_2;$$

первая элементарная формула $: a_3 r_2$; программа ее

:	a_3	r_2	r_3

Преобразованная информация

$$\times + \times \ldots r_1 \ldots r_3 a_5 + a_1 a_2$$

и т. д.

Сформулированный выше алгоритм программирования формул удобен для применения его человеком, но приведенная формулировка не дает представления, как этот алгоритм реализуется на машине. Прежде всего, непонятно, как осуществляется поиск первой элементарной формулы. В описании алгоритма не указано, как по данной элементарной формуле

строится программа вычисления по ней и как заменяется эта формула результатом вычислений.

Анализируя рассмотренный пример, мы приходим к более точному описанию работы алгоритма.

Назовем операцию, входящую в элементарную формулу, *выполнимой операцией*. Тогда можно формулировать следующие правила: одноместная операция в формуле выполнима, если после нее непосредственно стоит число; двуместная операция выполнима, если после нее непосредственно стоят подряд два числа.

Прежде чем перейти к составлению адресных программ, заметим, что в рассматриваемом алгоритме понятия адреса в адресной программе и в программе для трехадресной *ЦАМ* не всегда совпадают. Так, нам понадобится обозначение адресов отдельных частей ячеек *ЗУ*, соответствующих коду операции и первому, второму и третьему адресам. Введем для них следующее обозначение: $(a)_i$, где $i = 0, I, II, III$. Таким образом $(a)_0$ есть адрес части ячейки с адресом a, соответствующей коду операций; $(a)_I$, $(a)_{II}$ и $(a)_{III}$ — адреса частей ячейки a, соответствующие первому, второму и третьему адресам.

Адресная программа I. Поиск первой выполнимой операции.

k_{11}. $'\varphi_0 \Rightarrow \varphi_1$

k_{12}. $P\{{}^2\varphi_1 = !\}$ *ост.*
$C\,'\varphi_1 \Rightarrow \varphi_1$
$P\{{}^2\varphi_1 \neq 0\}\, k_{12}$
$'\varphi_1 \Rightarrow \varphi_2$

k_{13}. $C\,'\varphi_2 \Rightarrow \varphi_2$
$P\{{}^2\varphi_2 = 0\}\, k_{13}$
$P\{{}^2\varphi_2 \neq a\}\, k_{12}$
$P\{{}^2\varphi_1 = \beta\}\, k_{21}$
$'\varphi_2 \Rightarrow \varphi_3$

k_{14}. $C\,'\varphi_3 \Rightarrow \varphi_3$
$P\{{}^2\varphi_3 = 0\}\, k_{14}$
$P\{{}^2\varphi_3 \neq a\}\, k_{12}k_{22}$

φ_0 фиксирует начало формулы; ψ обозревает адрес очередной свободной ячейки в последовательности, отведенной для составляемой рабочей программы; ω обозревает адрес очередной свободной рабочей ячейки.

Вначале происходит поиск первого слева кода операции (фиксирует φ_1).

После этого отыскивается первый ненулевой элемент в формуле. Если он не число, то операция невыполнима, управление передается оператору k_{12}. Если после кода операции стоит число, то выясняется вид операции. Если операция — одноместная, то она выполнима и передается управление второму этапу (оператору k_2). Если операция — двуместная, то отыскивается следующий ненулевой элемент в формуле. Если этот элемент не число, то снова передается управление k_{12} — поиску следующего кода операции. Если же этот элемент — число, то двуместная операция выполнима, управление передается второму этапу k_{22}.

Здесь ! — признак конца формулы; 0 — признак операции; β — признак одноместной операции; a — признак величины.

Адресная программа II. Программирование элементарной формулы и замена ее результатом вычисления по ней.

k_{21}. $^2\varphi_1 \Rightarrow (\psi)_0$
$^2\varphi_2 \Rightarrow (\psi)_I$
$'\omega \Rightarrow (\psi)_{III}$
$0 \Rightarrow {}'\varphi_1$
$'\omega \Rightarrow {}'\varphi_2$
Pk_{23}.

k_{22}. $^2\varphi_1 \Rightarrow (\psi)_0$
$^2\varphi_2 \Rightarrow (\psi)_I$
$^2\varphi_3 \Rightarrow (\psi)_{II}$
$'\omega \Rightarrow (\psi)_{III}$
$0 \Rightarrow {}'\varphi_1$
$0 \Rightarrow {}'\varphi_2$
$'\omega \Rightarrow {}'\varphi_3$

k_{23}. $C\,'\omega \Rightarrow \omega$
$C\,'\varphi \Rightarrow \psi$
Pk_{11}.

Заметим, что код операции в выполнимой формуле фиксирует φ_1, код числа фиксирует φ_2. Если выполнимая операция — двуместная, то код второго числа фиксирует φ_3.

Оператором k_{21} начинается программа составления команды для одноместной выполнимой операции и замены соответствующей элементарной формулы адресом рабочей ячейки. Оператором k_{22} начинается программа составления команды для двуместной выполнимой операции и замены соответствующей формулы адресом свободной рабочей ячейки.

Оператор k_{23} сдвигает фиксаторы ω и ψ, т. е. подготавливает адрес свободной рабочей ячейки и адрес очередной команды рабочей программы.

В заключение заметим, что при усложнении алгоритма программирования арифметических формул не возникает серьезных затруднений. Например, пусть формула содержит знак равенства. Очевидно, что знак равенства может рассматриваться как двуместная операция, определяющая формулу вида

$$=ab.$$

Программа построения команды

+	a	—	b

для такой формулы может быть легко составлена читателем.

§ 3. Метод адресного программирования

Адресное программирование, изложению которого была посвящена глава V, также представляет собой один из методов автоматизации программирования.

Использование адресов высших рангов и схем обозревания информации по ним, особенно для логически сложных задач, как это было показано на приведенных в главе V примерах,

значительно упрощает алгоритмический язык и позволяет записывать алгоритмы в весьма обозримом виде. Адресное программирование использует общепринятый язык математических и логических формул, чем объясняется простота его применения.

Проверка адресных алгоритмов облегчается тем, что в процессе своей работы адресные программы сохраняют свою запись неизменной.

Существенным преимуществом адресного языка на современном уровне развития машинной математики является его практическая пригодность в качестве средства для обмена информацией в виде адресных программ между организациями, использующими машины различных типов. Действительно, этот язык является универсальным алгоритмическим языком, поскольку в нем не учитываются индивидуальные особенности машин, а принимаются во внимание лишь общие принципы машинного решения задач — принцип адресности и принцип автоматического (программного) управления.

Вместе с тем адресное программирование может служить основой для построения алгоритмов автоматического перевода адресных программ в программы конкретных машин — программирующих программ для конкретных машин, информацией для которых служит адресный алгоритм (см. Ющенко Е. Л., Быстрова Л. П. [1]).

Для построения таких программ могут быть использованы идеи, изложенные при описании программирующих программ, основанных на операторном методе. При этом появится необходимость включения в программирующую программу специальных блоков, например блока понижения ранга адреса и программирования операций с адресами второго ранга *), если речь идет о программировании для машины с набором операций, описанным в главе II, т. е. о машинах, выполняющих операции лишь по адресам первого ранга. Программирующие программы, основанные на адресном программировании, по мере развития этого языка могут совершенствоваться, как и программирующие программы, основанные на операторном программировании.

*) Можно показать, что адресная программа, включающая адреса выше второго ранга, может быть преобразована в эквивалентную программу, которая содержит только адреса не выше второго ранга (см. Ющенко Е. Л. [1]).

ГЛАВА VII

ОРГАНИЗАЦИЯ РАБОТЫ НА *ЦАМ*

§ 1. Оформление программ на бланках

При решении задач на *ЦАМ* огромное значение имеет организация работ по составлению, контролю и вводу программ в *ЗУ* и по контролю правильности выполнения программ машиной.

Уже оформление программ на бумаге при ручном программировании должно выполняться с максимальной аккуратностью и точностью. Точно так же при программировании задач с помощью программирующих программ особенно тщательно должна оформляться информация о задачах.

После составления программ в буквенно-численной записи память машины распределяется между программой и числами, переписывается в истинных адресах и тем самым приводится к рабочему виду.

Следует иметь в виду, что в машинах адреса ячеек *ЗУ* нумеруются в двоичной системе счисления. Для сокращения записи обычно используются восьмеричная или шестнадцатеричная системы счисления. В связи с этим на бланках коды операций, адреса в командах и условные числа (константы переадресации, коды команд сравнений и др.) записываются также в восьмеричной или шестнадцатеричной системе. Код при этом разбивается на группы восьмеричных или шестнадцатеричных цифр, одна из которых соответствует коду операции, остальные — адресам; при этом отсутствующие старшие разряды в каждой группе не опускаются, а изображаются нулями. Десятеричная система счисления используется только для записи десятеричных чисел. Для записи рабочих программ или информации о задаче при автоматическом программировании удобно пользоваться специальными бланками, где для каждой команды или числа отводится отдельная строка. Помимо колонок для записи кодов команд и чисел, на бланках отводится специальная колонка для буквенной записи величин в числовой части программы (в обозначениях, принятых в схеме счета) и результатов операций в командах. В командах передачи управления

Таблица 9. Примерная форма стандартного бланка для трехадресной машины

Шифр задачи	Лист №	Всего листов	Номер отсека	Составил	Проверил	Утвердил

Номер ячейки	Код операции	IA	IIA	IIIA	Печать на контрольном устройстве	Контрольный результат	Буквенное обозначение	Зависимость от параметров	Пояснения

здесь могут записываться соответствующие логические условия. Далее, отводится графа, используемая при отладке программ для указания результата вычислений, например при первом выполнении некоторых «узловых» команд, а также графа для указания зависимости адресов команд от параметров. Переменные команды, кроме того, можно выделять особо, например обвести номер команды красным карандашом.

При наличии печатающего контрольного устройства на бланках отводится колонка, которая заполняется программой на клавишном устройстве в процессе ее пробивки с бланков на перфоленты или перфокарты.

На бланках сверху отводится строка для указания шифра задачи, номера бланка и указания общего числа бланков, составляющих программу, а также номера отсека внешнего *ЗУ*, на который записывается программа.

Проверка правильности программ должна производиться в «две руки» программистом и дублером, эти лица на каждом бланке ставят свою подпись.

Подготовленные бланки программ вместе со схемой счета скрепляются и вкладываются в папку. Всякие исправления в программе в связи с обнаружением в ней ошибок целесообразно записывать на отдельных бланках, а в программу вносить лишь соответствующие ссылки.

§ 2. Ввод программ в *ЗУ* и контроль ввода

Как уже отмечалось, на большинстве современных *ЦАМ* для ввода программы в оперативную память или во внешнее *ЗУ* используются перфорированные ленты или перфорированные карты. Перфорирование рабочей программы на карты или ленты производится на специальных перфораторах. Для избежания внесения ошибок при перфорировании программ разработан ряд специальных приемов. Так, на большинстве машин перфорирование выполняется двумя лицами, независимо друг от друга, на разных перфораторах. Затем закодированные комплекты программ сличаются на специальном устройстве — контрольнике. Обнаруженные несовпадения устраняются. Кроме этого, существуют печатающие устройства, пропуская через которые перфорированную программу, получают ее отпечатанной на бумаге и сверяют с исходной записью на бланках. На некоторых перфораторах имеется специальное контрольное устройство: бланк с программой вставляется в каретку перфоратора и на клавишах набирается содержимое первой строки; затем нажимается специальная клавиша «печать на контрольном устройстве» и подготовленный набор печатается на бланк в соответствующем столбце; содержимое бланка и отпечатан-

ного набора сличается, после выяснения совпадения нажимается кнопка «исполнение», в результате чего совершается соответствующая пробивка на ленте или бланке. Вместе с этим на бланке печатается условный знак, свидетельствующий о том, что данное число уже зашифровано; далее переходят к перфорации следующей строки и т. д.

При переносе кодов на перфокарты или ленты восьмеричные (шестнадцатеричные) или десятеричные коды чисто механическим путем преобразуются в двоичные или десятично-двоичные. К моменту ввода команд в память номера ячеек, входящие в их адреса, и коды операций таким образом оказываются уже записанными в двоичной системе счисления. Десятично-двоичные коды чисел либо в процессе ввода, либо с помощью специальной подпрограммы переводятся в двоичную систему для расчетов.

Для того чтобы различать числа и команды в процессе ввода, последние группируются в отдельные массивы и вводятся с помощью различных команд «ввод чисел» или «ввод команд» или при их кодировке предусматривается специальный знак «признак десятеричного числа».

При вводе программы в машину возможны сбои в работе читающего устройства, в устройстве управления или *ЗУ*. В результате возможны искажения программы уже на этом этапе. Для проверки правильности ввода чаще всего пользуются операцией циклического сложения, которая в связи с этим еще называется операцией контрольного суммирования. При наличии двух комплектов одной и той же программы последняя вводится отдельно с каждого из них и оба раза все коды программы суммируются при помощи операции циклического сложения. Контрольные суммы затем сличаются. Вероятность равенства контрольных сумм при несовпадении программ весьма ничтожна, поэтому равенство этих сумм является гарантией того, что программа введена правильно. Если же контрольные суммы не совпадают, производится проверка работы ввода. Если программа имеется в одном экземпляре, ее вводят дважды и точно так же сличают контрольные суммы. Дополнительная затрата времени на организацию такого контроля оправдывается повышением надежности ввода.

§ 3. Проверка правильности работы машины

1. Проверка машины в профилактических целях. Для исключения влияния систематических ошибок, вызванных неисправностями отдельных узлов машины или их неустойчивой работой, машина систематически подвергается профилактическому контролю. С этой целью в специально ухудшенных контрольных режимах решаются так называемые *тестовые задачи* или *тесты*.

Проверке на тестах, как правило, машина подвергается перед отладкой или перед решением каждой новой задачи.

Тестовые задачи подбираются таким образом, чтобы при их решении были охвачены все звенья машины. Чаще всего для проверки отдельных устройств машины составляются специальные тесты.

Тестовые программы удобно помещать во внешнее *ЗУ*, а во внутренней памяти хранить начальные команды в специальных ячейках (не занимаемых другими командами), поскольку обращение к тестам может понадобиться при появлении систематической ошибки или при неустойчивой работе в процессе решения какой-либо задачи. Это позволяет переключать машину на работу по тестам без затраты времени на их ввод.

Только после устранения неисправностей, выявленных при решении тестовых задач, приступают к отладке программы.

Например, для проверки сумматора арифметического устройства тест-программа может быть построена следующим образом. Начиная с некоторых чисел $x_0, \ldots$ и $a_0, \ldots$ формируются последовательности чисел $x_1, x_2, \ldots$ и $a_1, a_2, \ldots$ (например, с помощью команды циклического сложения) и выполняются операции умножения и деления $a_k x_k$ и $\frac{a_k x_k}{x_k}$. При этом формирование чисел x_k и a_k производится так, чтобы при умножении не было выхода из разряда (для машины с плавающей запятой). Если $\left| a_k - \frac{a_k x_k}{x_k} \right| > \varepsilon$, где в ε входят ошибки округления при умножении и делении, то происходит останов, который говорит о том, что в сумматоре арифметического устройства есть неисправность. Дальнейшая локализация места неисправности производится с помощью выполнения более простых действий (например, сложения) со специально подобранными кодами; если это не дает результата, то проверка производится с помощью технических средств.

Тест-программа для проверки *ЗУ* осуществляет засылку различных чисел во все ячейки *ЗУ* с последующим сравнением их с такими же числами, которые формируются в некоторой стандартной ячейке.

В машинах с памятью на электронно-лучевых трубках ошибки могут появляться в результате многократного обращения к одной ячейке или к соседним ячейкам. Соответственно этому строятся программы для обнаружения неисправностей такого сорта.

2. Проверка правильности работы машины в процессе решения задач. Даже при проверенной, отлаженной и правильно введенной в *ЗУ* программе и при устойчивой работе машины на тестах нельзя гарантировать безошибочность решения задач.

В результате случайных сбоев в машине могут появляться случайные ошибки, искажающие результаты.

Поэтому для исключения влияния случайных ошибок на ход вычислений и на их результаты важное значение имеет применение автоматического (программного) контроля вычислений. Такой контроль вычислений необходимо предусматривать в процессе решения любой более или менее сложной задачи.

Как и при контроле ввода программ в машину, для контроля хода вычислительного процесса может быть использована операция циклического сложения.

С этой целью в программах на определенных этапах предусматривается циклическое суммирование кодов команд и чисел или некоторых результатов вычислений. Полученная контрольная сумма запоминается, и вычисления данного этапа повторяются. Контрольные суммы первого и второго вычислений сличаются на полное совпадение. Если совпадение имеет место, совершается переход к следующему этапу и т. д. В случае несовпадения этих сумм предусматривается останов или повторение вычислений. В последнем случае третья контрольная сумма сравнивается с каждой из двух предыдущих. При совпадении одной из пар контрольных сумм совершается переход к дальнейшим вычислениям; несовпадение свидетельствует о том, что произошел сбой и необходимо принять меры для устранения ошибки.

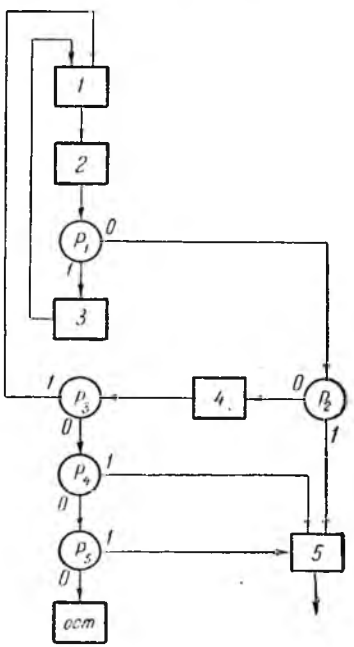


Рис. 20.

Блок-схема вычислений с дублированием представлена на рис. 20.

На рис. 20 введены следующие обозначения: *1* — вычислительный этап; *2* — контрольное суммирование; $P_1 = P\{n = 1\}$, где n — номер выполнения вычислительного этапа; *3* — запоминание контрольной суммы S_1; $P_2 = P\{S_1 = S_2\}$ — условие совпадения первых двух контрольных сумм; *4* — запоминание контрольной суммы S_2;

$$P_3 = P\{n = 2\}; \quad P_4 = P\{S_3 = S_1\}; \quad P_5 = P\{S_3 = S_2\};$$

5 — продолжение вычислений. В схемах только с двойным просчетом вместо *4* ставится останов, а P_3, P_4, P_5 опускаются.

Разумеется, реализация автоматического дублирования вычислений по этапам сопряжена с увеличением числа команд программы и с дополнительным использованием ячеек оперативной или внешней памяти для хранения исходных данных на каждом этапе. Можно применять аналогичные схемы автоматического дублирования вычислений по этапам, после которых вместо контрольного суммирования сличаются на полное совпадение некоторые результаты вычислений. При появлении случайного сбоя, исказившего какую-либо команду программы, последующие вычисления могут давать совпадающие результаты при контрольном дублировании. В связи с этим метод контрольного дублирования можно несколько усложнить, например, выполняя отдельно контрольное суммирование команд, скажем, в тот момент, когда команды принимают свой исходный вид.

Описанная выше схема контроля может не дать результата, если при сбое появилось несколько ошибок, компенсирующих одна другую. Кроме того, в процессе решения может появиться систематическая неисправность, которая также не будет обнаружена. Поэтому, если есть возможность, необходимо применять методы контроля, использующие содержательный смысл результатов вычислений, например проводить вычисления двумя различными методами или проверять различные соотношения, которым должны удовлетворять результаты.

а) Например, при вычислении таблицы синусов и косинусов можно проводить проверку, удовлетворяют ли получаемые машиной числа соотношению $\sin^2 x + \cos^2 x = 1$. Равенство, разумеется, проверяется с точностью до ε, где ε — допустимая погрешность вычислений.

В некоторых случаях целесообразно при решении алгебраических и дифференциальных уравнений предусматривать в программе проверку решений путем подстановки полученных численных значений искомых величин (и их производных в случае дифференциальных уравнений) в исходные уравнения; вычисления продолжаются лишь при удовлетворительных результатах подстановки.

б) В некоторых случаях удается вводить вспомогательные контрольные величины, связь которых с результатами заранее известна. Контрольные величины вычисляются одновременно с другими величинами, и на определенных вычислительных этапах производится проверка удовлетворения связи между искомыми и вспомогательными величинами, что предусматривается программой.

Например, при решении системы линейных алгебраических уравнений методом Гаусса, помимо исходной системы, рассматривается система с той же матрицей коэффициентов и свободными членами, равными суммам элементов строк матрицы коэффициентов, включая и свободные члены. Каждый корень вспомогательной системы будет на единицу больше соответствующего корня исходной системы, что и используется для контроля.

Иногда о правильном ходе вычислений можно судить по монотонности или отсутствию резких изменений отдельных результатов счета.

Контроль, если только он достаточно полный, т. е. охватывает все этапы вычислений, почти всегда дает возможность фиксировать момент сбоя. При обнаружении сбоя необходимо вторично ввести программу в машину и продолжать вычисления по обновленной программе. Очевидно, целесообразно продолжать вычисления с того момента, когда последний раз совпали контрольные суммы (или выполнились контрольные условия).

Для этого необходимо при совпадении контрольных сумм предусмотреть в программе запоминание значений переменных параметров программы, определяющих дальнейший ход вычислений.

Решение задач с большим количеством вариантов при автоматическом переходе от варианта к варианту сопряжено с трудностями определения, при каких значениях параметров задачи, определяющих варианты, произошел сбой. В этих случаях можно предусмотреть запоминание значений параметров, определяющих следующий вариант после верного просчета предыдущего варианта. В случае сбоя программа вводится заново, а указанные значения параметров используются для продолжения вычислений. Запоминание значений параметров можно производить выдачей на перфокарты или перфоленты. На машине *Стрела* перфокарту, содержащую набор значений параметров, определяющих вариант, просчитанный со сбоем, принято называть *картой сбоя*.

§ 4. Отладка программ

Опыт решения задач на цифровых автоматических машинах показывает, что даже после самой тщательной проверки в программе остаются ошибки. Отыскание этих ошибок и окончательная подготовка программы к счету, а также определение времени, необходимого для полного решения задачи, — все это входит в задачу отладки программы на машине. Кроме того, во время отладки могут решаться специальные для данной

программы вопросы, например выбор шага интегрирования, обеспечивающего необходимую точность, если автоматический выбор его встречает затруднения, или выбор масштабов.

Как правило, отладка состоит в последовательном выполнении вычислений по отдельным участкам программы и в выдаче после каждого этапа вычислений промежуточных результатов и содержания некоторых ячеек памяти. Сравнивая результаты, полученные на машине, с вычислениями вручную, можно судить о правильности работы арифметических операторов. Содержание различных счетчиков и переменных команд может указать на ошибки в логике программы. Вывод на печать результатов можно делать с пульта управления после контрольных остановов, предусмотренных в конце каждого контролируемого участка. Контрольные остановы осуществляются либо с помощью специального признака (контрольного сигнала), если таковой предусмотрен в машине, либо с пульта управления (*Урал*), либо другими способами.

Можно автоматизировать процесс отладки путем введения в программу команд, позволяющих без остановов проверять последовательно участки программы и выдачу результатов. Составление такой программы зачастую требует большого дополнительного труда.

Поэтому целесообразно для отладки применять универсальные контролирующие программы.

Идея универсальной программы отладки состоит в следующем. Из контролируемой программы последовательно извлекаются команды и выполняются внутри программы отладки. После выполнения некоторых заранее зафиксированных команд производится сравнение промежуточных результатов, и в случае несовпадения исправление ошибочного результата, печать ошибок и переход к контролю следующих команд. Кроме того, в программе отладки есть возможность пропускать куски контролируемой программы, сокращать некоторые циклы, печатать команды и т. п.

Порядок отладки задается контролирующей программе в виде специально закодированной информации.

Хорошо продуманный порядок отладки имеет огромное значение для быстрого обнаружения ошибок и экономии машинного времени.

ПРИЛОЖЕНИЕ

БЭСМ

Быстродействующая электронная счетная машина *БЭСМ*, сконструированная в 1951 г. под руководством акад. С. А. Лебедева в АН СССР, оперирует с 39-разрядными кодами в системе с плавающей запятой. При этом для кодирования порядка числа используются 6 разрядов (первый разряд — знак порядка) и для мантиссы — 33 разряда (первый разряд — знак мантиссы), как это показано на рис. 21.

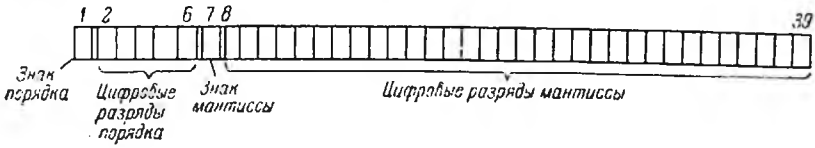


Рис. 21.

Для команд используется трехадресная система, 6 разрядов отводятся для кодирования вида операции; на каждый адрес отводится по 11 разрядов.

Быстродействие машины определяется в 8—10 тысяч операций в секунду. Объем внутренней памяти 2047 ячеек, внешняя память на двух магнитных барабанах и четырех магнитных лентах. Емкость одного барабана 5120 кодов; емкость магнитной ленты 30 000 кодов (ленты — сменные). Ввод — с перфокарт и перфолент, вывод — на перфокарты или бумажную ленту.

В отличие от ряда других советских машин (*Стрела, Киев, Урал*) машина *БЭСМ* является машиной с двумя управлениями — центральным и местным, что представляет известные удобства при использовании подпрограмм.

В машине *БЭСМ*, помимо регистра команд K, имеются два счетчика команд *):

C_1 — счетчик команд центрального управления;

C_2 — счетчик команд местного управления.

*) Здесь введены обозначения лишь тех регистров и переключателей, которые нам нужны для описания элементарных операций машины.

Кроме того, имеется триггер управления T. Наличие единицы в триггере T соответствует работе центрального управления, наличие нуля — местного.

Отдельный такт машины состоит в выполнении команды $'K$, т. е. команды, код которой содержится на регистре команд K, и в засылке в этот регистр кода следующей команды.

Для описания операций машины *БЭСМ* выделим части регистра команд K, которые соответственно обозначим (см. таблицу 10).

Таблица 10

Разряды регистра команд *БЭСМ* (нумерация слева направо)	Обозначение и название групп разрядов (регистров)
1—6	K_0 — регистр кода операции
7—17	K_1 — регистр I адреса
18—28	K_2 — регистр II адреса
29—39	K_3 — регистр III адреса

Кроме того, при необходимости i-й разряд регистра K будем обозначать через ε_i.

Рисунок 22 показывает это деление на части регистра команд (и ячейки, в которой записана команда).

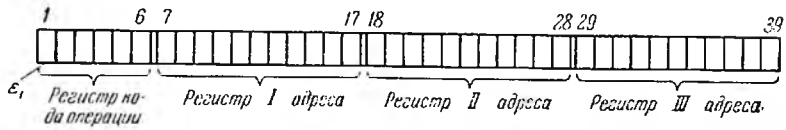


Рис. 22.

Для записи $'K_0$ будем пользоваться двоичной системой. Допустимое множество значений $'K_0$ будет 000000, 000001, ..., 111111. Обозначим через T_1 имеющийся в машине тумблер условного останова.

Обычное выполнение команды (арифметической, логической и вспомогательных команд вычислительного назначения) состоит из:

1) собственно преобразования кодов и засылки их по адресу, т. е. выполнение операции вида

$$f(^2K_1,\ ^2K_2) \Rightarrow {}'K_3;$$

2) прибавления единицы в счетчик C_1, если $'T = 1$, или в счетчик C_2, если $'T = 0$, т. е.

$$'C_1 + 'T \Rightarrow C_1;$$
$$'C_2 + (1 - 'T) \Rightarrow C_2;$$

3) перехода к следующей команде, номер которой стоит на счетчике C_1, если $'T = 1$, или на счетчике C_2, если $'T = 0$:

$$'('C_1 'T + 'C_2 (1 - 'T)) \Rightarrow K.$$

Таким образом, особыми операциями машины *БЭСМ* являются операции, изменяющие состояние триггера управления T.

I. Операции передачи управления с местного на центральное и с центрального на местное

1. $'K_0 = 011000$ (*ПМУК*) — передача на местное управление. По этой команде:

а) $0 \Rightarrow T$;

б) $^2C_2 \Rightarrow K$.

Таким образом передается управление на местное; выполняются команды, начиная с номера, хранящегося на счетчике местного управления C_2.

2. $'K_2 = 011001$ (*ПЦУК*) — переход на центральное управление команд.

По этой команде:

а) $1 \Rightarrow T$;

б) $^2C_1 \Rightarrow K$.

Таким образом передается управление на центральное; выполняются команды, начиная с адреса, хранящегося на счетчике центрального управления C_1.

При этом содержание регистров K_1, K_2, K_3 безразлично.

3. $'K_0 = 011010$ (*ИМУК*) — изменение счетчика местного управления:

а) $0 \Rightarrow T$;

б) $'K_3 \Rightarrow C_2$;

в) $^2C_2 \Rightarrow K$.

Таким образом, если работа производилась на центральном управлении, то она переходит на местное и выполняются команды, начиная с адреса $'K_3$. Если работа производилась на местном управлении, то она продолжается, но начиная

с адреса $'K_3$. Содержание регистров K_1 и K_2 на результат операции не влияет.

4. $'K_0 = 011011$ (*ИЦУК*) — изменение счетчика центрального управления командами:

а) $'C_1{}'T + {}'C_2(1 - {}'T) + 1 \Rightarrow ('K_2)_3$; $011011 \Rightarrow ('K_2)_0$;

б) $1 \Rightarrow T$;

в) $'K_3 \Rightarrow C_1$;

г) ${}^2C_1 \Rightarrow K$.

Операция выполняется аналогично предыдущей; однако по адресу $'K_2$ засылается команда, имеющая код операции *ИЦУК* *), а в третьем адресе — увеличенный на единицу адрес текущей команды (пункт а)). Содержимое регистра K_1 на результат операции не влияет.

Операция *ИМУК* удобна для выполнения перехода на подпрограммы. С этой целью основная программа пишется на центральном управлении, а подпрограммы — на местном. Командой *ИМУК* кодируется переход на подпрограмму — с центрального на местное управление. При этом в ІІІ*А* команды *ИМУК* заносится адрес начальной команды подпрограммы.

Для возврата к основной программе с подпрограммы существует операция *ПЦУК*, которая позволяет вернуться к прерванному месту основной программы (к команде, следующей за командой *ИМУК*, с которой был выполнен переход на подпрограмму). При этом очевидно, что в процессе выполнения подпрограммы выполнение команд, переводящих управление с местного на центральное, должно учитываться особо. Так, если при выполнении некоторой подпрограммы нужно будет перейти к программе, может быть использована команда *ИЦУК*. С помощью этой команды в ІІІ адресе ячейки, адрес которой указан в K_2, может быть зафиксировано показание счетчика центрального управления команд (места ухода с основной программы на подпрограмму).

Приведем теперь список остальных операций машины *БЭСМ*.

ІІ. Операции преобразования и пересылки кодов

1. $'K_0 = 000001$ (+) — сложение:

а) ${}^2K_1 + {}^2K_2 \Rightarrow 'K_3$;

б) $'C_1 + {}'T \Rightarrow C_1$; $'C_2 + (1 - {}'T) \Rightarrow C_2$;

в) $'('C_1{}'T + {}'C_2(1 - {}'T)) \Rightarrow K$.

---

*) Напомним, что запись $(\ )_3$ означает разряды, соответствующие третьему адресу ячейки, адрес которой указан в скобках ( ).

Перед сложением чисел выравниваются их порядки. Результат нормализуется и округляется.

2. $'K_0 = 100001$ (, +) — то же, что и $'K_0 = 000001$, но нормализация блокируется.

Аналогично выполняются команды:

3. $'K_0 = 000010$ (—) — вычитание.

4. $'K_0 = 100010$ (, —) — вычитание с блокировкой нормализации.

5. $'K_0 = 000011$ ($\times$) — умножение.

6. $'K_0 = 100011$ (, $\times$) — умножение с блокировкой нормализации.

7. $'K_0 = 000100$ (:) — деление.

8. $'K_0 = 000101$ ($+П$) — сложение порядков. Порядок числа $^2K_2$ прибавляется к числу $^2K_1$, и нормализованный результат засылается по адресу $'K_3$.

9. $'K_0 = 100101$ (, $+П$) — то же, что и предыдущая операция, но с блокировкой нормализации.

10 и **11.** $'K_0 = 000110$ и $'K_0 = 100110$ ($—П$) и (, $—П$) — вычитание порядков. То же, что и при $'K_0 = 000101$ и $'K_0 = 100101$, только выполняемая операция является вычитанием.

12. $'K_0 = 000111$ ($ИПА$) — изменение порядка по адресу.

К порядку числа $^2K_1$ прибавляется число $'K_2$, и нормализованный результат засылается по адресу $'K_3$. Шестой разряд K_2 воспринимается как знаковый.

13. $'K_0 = 100111$ ($,ИПА$) — то же, что и при $'K_0 = 000111$, но с блокировкой нормализации.

14 и **15.** $'K_0 = 001000$ ($\times а$) и $'K_0 = 001001$ ($\times б$) — умножение с выводом удвоенного количества разрядов. Операция выполняется двумя командами — командой $'K_0 = 001000$ — умножение чисел $^2K_1$ и $^2K_2$ без округления, нормализация и вывод первых 32 разрядов результата с их порядком по адресу $'K_3$ и командой (следующей за ней) $'K_0 = 001001$ — нормализация 32 младших разрядов сумматора (с оставшимся от предыдущей команды порядком) и вывод по адресу $'K_3$ ($'K_1$ и $'K_2$ в последней команде не используются).

16 и **17.** $'K_0 = 101000$ и $'K_0 = 101001$ — то же, что и 14 и 15, но с блокировкой нормализации.

18 и **19.** $'K_0 = 001010$ ($: а$) и $'K_0 = 001011$ ($: б$) — аналогично предыдущей операции выполняется деление с выводом остатка.

20. $'K_0 = 001100$ ($ПЧ$) — передача числа нормальная. Число $^2K_1$ нормализуется и пересылается по адресу $'K_3$.

а) $(^2K_1)^н \Rightarrow 'K_3$,

пп. б) и в) как и при сложении; $'K_2$ — не используется. Через $(^2K_1)^н$ обозначен нормализованный код $^2K_1$.

21. $'K_0 = 101100$ (, *ПЧ*) — передача числа нормальная с блокировкой нормализации.

22. $'K_0 = 101100$ и $'\varepsilon_{19} = 1$ (, *ПЧТ*) — передача числа на печать. $^2K_1$ — печатается.

23. $'K_0 = 001100$ и $'\varepsilon_{26} = 1$ или $'\varepsilon_{27} = 1$, или $'\varepsilon_{28} = 1$ (*ПЧР*) — передача числа с регистров пульта управления.

Если $'\varepsilon_{26} = 1$, передается код с триггерного контрольного регистра; если $'\varepsilon_{27} = 1$ — со второго диодного регистра; если $'\varepsilon_{28} = 1$ — с первого диодного регистра.

24. $'K_0 = 101100$ и $'\varepsilon_{26} = 1$, или $'\varepsilon_{27} = 1$, или $'\varepsilon_{28} = 1$ (, *ПЧР*) — передача числа с регистров пульта управления с блокировкой нормализации.

25. $'K_0 = 001101$ (*ПЧ*—) — передача числа с изменением знака. Нормализованное число $^2K_1$ с обратным знаком передается по адресу $'K_3$. $'K_2$ не используется:

$$(-{}^2K_1)_н \Rightarrow 'K_3.$$

26. $'K_0 = 101101$ (, *ПЧ*—) — передача числа с изменением знака и с блокировкой нормализации.

27. $'K_0 = 001110$ ($|ПЧ|$) — передача модуля числа. Нормализованное число $^2K_1$ по модулю пересылается по адресу $'K_3$

$$|{}^2K_1|^н \Rightarrow 'K_3.$$

28. $'K_0 = 101110$ (, $|ПЧ|$) — передача числа по модулю с блокировкой нормализации.

29. $'K_0 = 001111$ (*ПЧ* ±) — передача числа с изменением знака в зависимости от знака другого числа. Передаваемое число $^2K_1$ нормализуется; знак этого числа изменяется на обратный, если число $^2K_2$ отрицательно, и сохраняется в противном случае; результат посылается по адресу $'K_3$.

30. $'K_0 = 101111$ (, *ПЧ* ±) — передача числа с изменением знака в зависимости от знака другого числа с блокировкой нормализации.

31. $'K_0 = 010000$ ($\downarrow$) — передача порядка числа. Порядок числа $^2K_1$ представляется в виде нормализованного числа со своим порядком и передается по адресу $'K_3$. $'K_2$ не используется.

32. $'K_0 = 110000$ (, $\downarrow$). Передача порядка числа с блокировкой нормализации.

33. $'K_0 = 010001$ ($\leftarrow$) — сдвиг с блокировкой порядков. Сдвиг числа $^2K_1$ на число разрядов, равное числу $'K_2$, если sign $'K_2 = 0$ — влево и вправо, если sign $'K_2 = 1$. При этом в качестве знакового разряда принимается $'\varepsilon_{22}$ (7-й разряд регистра K_2) (код порядка не сдвигается и не передается); результат засылается по адресу $'K_3$.

34. $'K_0 = 110001$ (, $\leftarrow$) — сдвиг по всем разрядам. То же, что и при $'K_0 = 010001$, но сдвиг происходит по всем разрядам.

35. $'K_0 = 011101$ (*Л*) — логическое умножение. Поразрядное логическое умножение кодов $^2K_1$ и $^2K_2$ с помещением результата по адресу $'K_3$.

36. $'K_0 = 010010$ (*СК*) — сложение команд. Выполняется операция сложения цифровых частей чисел $^2K_1$ и $^2K_2$ (включая знаки чисел); порядок второго числа не учитывается. Результат (без нормализации) засылается по адресу $'K_3$.

37. $'K_0 = 110010$ (, *СК*) — циклическое сложение. Коды $^2K_1$ и $^2K_2$, рассматриваемые как одно целое, складываются циклически (с переносом из разряда знака числа в 1-й разряд порядка и из разряда знака порядка в младший разряд цифровой части числа); результат помещается по адресу $'K_3$.

38. $'K_0 = 010011$ или 110011 (*ЦЧ*) — выделение целой части. Целая часть числа $^2K_1$ в виде числа с фиксированной запятой после младшего разряда и со своим знаком засылается по адресу $'K_3$; дробная его часть, приведенная к нулевому порядку, засылается по адресу $'K_2$.

III. Операции передачи управления

Помимо приведенных вначале операций передачи управления, в машине *БЭСМ* существуют следующие операции, сохраняющие режим работы на местном или центральном управлении.

1. $'K_0 = 010100$ ($<$) — сравнение с учетом знаков:

а) $P\{^2K_1 < {}^2K_2\}$ $\begin{matrix} {}'K_3(1-'T)+{}'C_2'T \Rightarrow C_2;\ {}'K_3'T+{}'C_1(1-'T) \Rightarrow C_1; \\ {}'C_1+{}'T \Rightarrow C_1;\quad {}'C_2+(1-'T) \Rightarrow C_2; \end{matrix}$

б) $'({}'C_1'T+{}'C_2(1-'T)) \Rightarrow K.$

Таким образом, если $^2K_1 < {}^2K_2$, управление передается по адресу $'K_3$; если $^2K_1 \geqslant {}^2K_2$, управление передается следующей по номеру команде. Аналогично выполняются следующие две операции.

2. $'K_0 = 110100$ (, $<$) — сравнение на равенство кодов:

а) $P\{^2K_1 \neq {}^2K_2\}$ $\begin{matrix} {}'K_3(1-'T)+{}'C_2'T \Rightarrow C_2;\ {}'K_3'T+{}'C_1(1-'T) \Rightarrow C_1; \\ {}'C_1+{}'T \Rightarrow C_1;\quad {}'C_2+(1-'T) \Rightarrow C_2; \end{matrix}$

б) $'({}'C_1'T+{}'C_2(1-'T)) \Rightarrow K.$

3. $'K_0 = 010101$ ($|<|$) — сравнение по модулю:

а) $P\{|^2K_1| < |^2K_2|\}$ $\begin{matrix} {}'K_3(1-'T)+{}'C_2'T \Rightarrow C_2;\ {}'K_3'T+{}'C_1(1-'T) \Rightarrow C_1; \\ {}'C_1+{}'T \Rightarrow C_1;\quad {}'C_2+(1-'T) \Rightarrow C_2; \end{matrix}$

б) $'({}'C_1'T+{}'C_2(1-'T)) \Rightarrow K.$

IV. Операции обращения к внешним устройствам

1 и **2.** $'K_0 = 010110$ (*МЗа*) и $'K_0 = 010111$ (*МЗб*) — команды обращения к внешним запоминающим устройствам (магнитная запись). Операции выполняются в две команды. В адресах команд указываются начальный и конечный номера группы кодов внешнего *ЗУ*, на которые распространяется операция, и начальный адрес ячейки внутреннего *ЗУ*, с которым связана операция. Характер операции дополнительно определяется первым адресом команды *МЗа* (запись, считывание или перемотка; барабан или лента; номер барабана или ленты). При этом предусматривается обращение к перфоленте, магнитному барабану и магнитной ленте.

3. $K_0 = 011100$ (*ост. усл.*) — останов условный (по тумблеру). Происходит останов, если специальный тумблер T_1 включен ($'T_1 = 1$), в противном случае выполняется следующая команда, т. е.

а) $P\{'T_1 = 1\}$ *ост.* $'C_1 + 'T \Rightarrow C_1$; $'C_2 + (1 - 'T) \Rightarrow C_2$;

б) $'('C_1'T + 'C_2(1 - 'T)) \Rightarrow K$.

4. $K_0 = 011111$ (*ост.*) — останов.

Производится останов с соответствующей сигнализацией.

СТРЕЛА

Универсальная цифровая автоматическая машина *Стрела* оперирует с 43-разрядными кодами. Числа представляются в системе с плавающей запятой, причем для двоичной мантиссы отведено 36 разрядов (первый из них — знак мантиссы), для двоичного порядка — 7 разрядов (первый — знак порядка) (рис. 23).

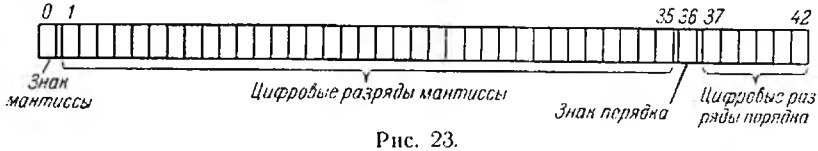


Рис. 23.

Разряды с 0 по 35 будем называть регистром мантиссы, с 36 по 42 — регистром порядка и обозначать номером ячейки с индексом м или п.

При хранении чисел в двоично-десятичном коде для мантиссы отводится 37 разрядов (один знаковый), для порядка — 6 (один знаковый). 36 цифровых разрядов мантиссы разбиваются на 9 тетрад, по одной на каждый десятичный знак.

Система команд принята трехадресная с естественным порядком выполнения. При записи команды в ячейке памяти каждый из адресов занимает 12 разрядов, а код операции — 6 разрядов, один разряд отведен для контрольного знака. На рис. 24 показано распределение разрядов ячейки, в которой записана команда.

Быстродействие машины *Стрела* составляет 2—3 тысячи операций в секунду. Внутренняя память, объемом в 2048 ячеек,

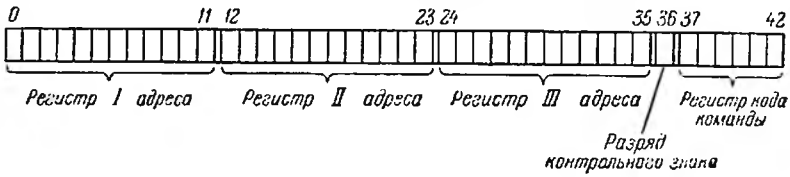


Рис. 24.

построена на электронно-лучевых трубках. Помимо оперативной внутренней памяти, машина имеет постоянную. Сюда входит так называемое устройство выдачи констант (*УВК*), в котором постоянно хранятся некоторые часто встречающиеся константы. Этим ячейкам присвоены номера от 7400 до 7777. В постоянную память входит также накопитель стандартных подпрограмм (*НСП*), в котором хранятся подпрограммы вычисления элементарных функций ($\frac{1}{x}$, e^x, $\sin x$, $\ln x$ и др.).

Внешняя память расположена на двух бобинах с ферромагнитной лентой. Магнитные ленты разбиваются на зоны, на одной ленте допустимо до 511 зон. В каждой зоне может быть записано до 2048 чисел. Зоны магнитных лент имеют соответственно номера от 4001 до 4777 и от 5001 до 5777. Считывание кодов с каждой зоны допустимо любыми группами, начинающимися первым кодом.

Ввод осуществляется с перфокарт, вывод — на перфокарты. Имеется устройство для перевода информации с перфокарт на печать. Каждый код (число или команда) на перфокарте пробивается в виде одной строки; всего на карту наносится 12 кодов.

При пробивке команд каждая восьмеричная цифра пробивается в виде тройки двоичных цифр; контрольный знак пробивается в виде одной двоичной цифры. При пробивке десятичных чисел каждая десятичная цифра пробивается в виде тетрады, знак — в виде одного двоичного числа. Преобразование чисел в двоичную систему — автоматическое.

Для описания операций нам понадобятся следующие обозначения:

1) c — счетчик команд,
2) K — регистр команд,
3) Γ — регистр команд групповой операции,
4) A — регистр модификации адресов,
5) ω — триггер признака для условного перехода,
6) T — тумблер контрольного режима.

Регистр команд K разделим в соответствии с принятым размещением команд (см. рис. 24) на группы разрядов (регистры) (см. таблицу 11).

Таблица 11

Группа разрядов регистра команд	Обозначение и название групп разрядов (регистров)
0—11	k_1 — регистр I адреса
12—23	k_2 — регистр II адреса
24—35	k_3 — регистр III адреса
36	α — разряд контрольного знака
37—42	k_0 — регистр кода команды

На такие же регистры разбивается при надобности регистр команд групповой операции Γ (и ячейки). Они обозначаются буквой «Γ» (или номером ячейки) с соответствующим индексом (0, 1, 2, 3).

Указанные регистры могут хранить восьмеричные коды:

$$'k_1,\ 'k_2,\ 'k_3 = 0000;\ 0001;\ \ldots;\ 7777;$$
$$'\alpha = 0;\ 1;$$
$$'k_0 = 00;\ 01;\ \ldots;\ 77.$$

Однако $'k_1$, $'k_2$, $'k_3$ во всех случаях, кроме специально оговоренных, меньше 4000 (2048 в 10-й системе), что соответствует объему внутреннего $ЗУ$ машины.

В машине предусмотрен так называемый режим контрольных остановов, для перехода на который включается тумблер $T('T = 1)$. В этом случае после выполнения каждой команды, в которой $'\alpha = 1$, машина останавливается. Иначе говоря, после выполнения каждой команды дополнительно выполняется действие:

$$P\{'\alpha \times 'T = 1\}\ ост.$$

В зависимости от кода $'k_0$ выполняются следующие операции.

I. Арифметические операции

1. $'k_0 = 01$ — сложение. Машина производит следующие действия:

а) $^2k_1 + {}^2k_2 \Rightarrow {}'k_3$;

б) $P\{{}^2k_3 \leqslant -0\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'c + 1 \Rightarrow c$;

г) $^2c \Rightarrow K$.

Пункт а) означает сложение кодов $^2k_1$, $^2k_2$ с нормализацией результата и засылкой его по адресу $'k_3$; пункт б) — выработку сигнала ω (используемого в командах условной передачи управления, см. далее); пункты в) и г) — переход к следующей команде — одинаковы для всех арифметических операций. Кроме этого, происходит останов при переполнении, т. е. если порядок результата больше 77; если же он меньше минимально допустимого — 77, то результатом считается нуль. Поэтому действие а) здесь, как и в других арифметических операциях, более подробно должно быть записано в виде

а) $P\{(^2k_1 + {}^2k_2)_п > 77\}$ *ост.*;

$P\{(^2k_1 + {}^2k_2)_п > -77\}$ $\begin{matrix} {}^2k_1 + {}^2k_2 \Rightarrow {}'k_3; \\ 0 \Rightarrow k_3. \end{matrix}$

2. $'k_0 = 03$ — вычитание.

3. $'k_0 = 04$ — вычитание модулей.

Эти операции производятся так же, как и сложение, но с соответствующей заменой знака операции в а).

4. $'k_0 = 05$ — умножение:

а) $^2k_1 \times {}^2k_2 \Rightarrow {}'k_2$;

б) $P\{|{}^2k_3| \geqslant 1\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) и г).

5. $'k_0 = 06$ — сложение порядков и

6. $'k_0 = 07$ — вычитание порядков:

а) $'('k_1)_м \Rightarrow ('k_3)_м$;

$'('k_1)_п \pm {}'('k_2)_п \Rightarrow ('k_3)_п$;

б) $P\{('k_3)_п \geqslant 1\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) и г).

7. $'k_0 = 10$ — перенос числа с присвоенным знаком другого числа:

а) $|{}^2k_1| \vee \operatorname{sign} {}^2k_2 \Rightarrow {}'k_3$;

б) $P\{'('k_3)_n \geqslant 1\} \begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) и г).

8. $'k_0 = 12$ — сложение чисел без округления:

а) ${}^2k_1 + {}^2k_2 \Rightarrow {}'k_3$;

б) $P\{{}^2k_3 = 0\} \begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) и г).

9. $'k_0 = 17$ — контрольное суммирование кодов ${}^2k_1$ и ${}^2k_2$, понимаемых как единые двоичные коды, с переносом из старшего разряда в младший:

а) ${}^2k_1\,(Ц+)\,{}^2k_2 \Rightarrow {}'k_3$;

б) $0 \Rightarrow \omega$;

в) и г).

10. $'k_0 = 02$ — специальное сложение (для команд) и

11. $'k_0 = 15$ — специальное вычитание (для команд):

а) $'('k_1)_i \pm '('k_2)_i \Rightarrow ('k_3)_i$, где $i = 1, 2, 3$;
$'('k_1)_0 \Rightarrow ('k_3)_0$;

б) $0 \Rightarrow \omega$;

в) и г).

II. Логические операции

1. $'k_0 = 11$ — поразрядное логическое умножение:

а) $({}^2k_1) \wedge ({}^2k_2) \Rightarrow {}'k_3$ (поразрядно);

б) $P\{{}^2k_3 = 0\} \begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) и г).

2. $'k_0 = 13$ — поразрядное логическое сложение:

а) $({}^2k_1) \vee ({}^2k_2) \Rightarrow {}'k_3$ (поразрядно);

б) $P\{{}^2k_3 = 0\} \begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) и г).

3. $'k_0 = 14$ — сдвиг. Возможны две модификации этой операции. Если $'k_2 < 4000$, т. е. является адресом оперативной ячейки, то код $^2k_1$ сдвигается на число разрядов, равное порядку числа $^2k_2$, если знак порядка + влево, если знак порядка — вправо. Если $'k_2 = 4000 + l$ или $'k_2 = 5000 + l$, то происходит сдвиг влево на l разрядов. Если же $'k_2 = 4100 + l$ или $'k_2 = 5100 + l$, то сдвиг происходит вправо на l разрядов ($l < 77$).

Обозначив знаком $\beta \leftrightarrow \alpha$ сдвиг кода β на α разрядов (влево, если $\alpha > 0$, вправо, если $\alpha < 0$), запишем операцию в адресном виде:

а) 1) $P\{'k_2 < 4000\}$ $^2k_1 \leftrightarrow {}'('k_2) \Rightarrow 'k_3$;

2) $P\{'k_2 < 4100\}$ $^2k_1 \leftrightarrow ('k_2 - 4000) \Rightarrow 'k_3$;

3) $P\{'k_2 < 5000\}$ $^2k_1 \leftrightarrow (4100 - 'k_2) \Rightarrow 'k_3$;

4) $P\{'k_2 < 5100\}$ $^2k_1 \leftrightarrow ('k_2 - 5000) \Rightarrow 'k_3$;

$^2k_1 \leftrightarrow (5100 - 'k_2) \Rightarrow 'k_3$;

б) $P\{^2k_3 = 0\}$ $1 \Rightarrow \omega$;

$0 \Rightarrow \omega$;

в) и г).

4. $'k_0 = 16$ — поразрядная операция «неравнозначно» (см. главу II):

а) $(^2k_1) \cong (^2k_2) \Rightarrow 'k_3$ (поразрядно);

б) $P\{^2k_3 \neq 0\}$ $1 \Rightarrow \omega$;

$0 \Rightarrow \omega$;

в) и г).

5. $'k_0 = 26$ — сравнение и останов при несовпадении:

а), б), г) те же, что и в 4;

в) $P\{\omega = 1\}$ *ост.*

$'c + 1 \Rightarrow c$;

III. Команды условного перехода и предварительные команды групповых операций

1. $'k_0 = 20$ — условный переход первого типа:

а) $P\{'\omega = 0\}$ $'k_1 \Rightarrow c$;

$'k_2 \Rightarrow c$;

б) $0 \Rightarrow 'k_3$;

в) $^2c \Rightarrow K$.

2. $'k_0 = 27$ — условный переход второго типа:

а) $P\{'\omega = 0\}$ $\begin{matrix} 'k_1 \Rightarrow c; \\ 'k_2 \Rightarrow c; \end{matrix}$

б) $'c + 1 \Rightarrow ('k_3)_1;$

$'c + 1 \Rightarrow ('k_3)_2;$

$'k_3 \Rightarrow ('k_3)_3;$

$20 \Rightarrow ('k_3)_0.$

Пункт а) — условный переход по признаку ω. Пункт б) — засылка команды безусловного перехода по адресу $('k_3)$.

в) $^2c \Rightarrow K.$

3. $'k_0 = 30, 31, 32, 33, 34, 35, 36, 37$ — предварительные команды групповых операций.

Групповые операции состоят из двух команд: предварительной и исполнительной (код последней — произвольный код арифметической или логической операции).

В предварительной команде во II адресе указывается число, на единицу меньшее числа выполнений исполнительной команды.

Код операции предварительной команды содержит информацию о правиле модификации адресов исполнительной команды. Кроме того, в III адресе предварительной команды указывается адрес, по которому засылается нуль.

Обозначим через ε_1, ε_2, ε_3 соответственно 40-й, 41-й и 42-й разряды регистра K (т. е. три младшие разряда регистра кода операции k_0). Действие групповых операций описывается следующей программой:

а) $0 \Rightarrow A; \quad 0 \Rightarrow 'k_3;$

б) $'('c + 1) \Rightarrow \Gamma;$

в) $'('\Gamma_1 + '\varepsilon_1 \cdot 'A)\, 0\, '('\Gamma_2 + '\varepsilon_2 'A) \Rightarrow '\Gamma_3 + '\varepsilon_3 'A;$

г) $'A + 1 \Rightarrow A;$

д) $P\{'A \leqslant 'k_2\}\ b);$

е) $'c + 2 \Rightarrow c;$

ж) $^2c \Rightarrow K.$

Здесь через 0 обозначена любая арифметическая или логическая операция.

Пункт а) — очистка регистра модификации A и очистка некоторой ячейки $'k_3$ (если последняя очистка не нужна, в III адресе ставится нуль).

Пункт б) — засылка исполнительной команды в регистр групповых операций Γ.

Пункт в) — выполнение команды, содержащейся в регистре Γ с модификацией адресов согласно трем последним разрядам кода операции предварительной команды, адрес Γ_l, для которого $\varepsilon_l = 1 (i = 1, 2, 3)$, увеличивается на содержимое регистра A (поскольку при первом выполнении этого пункта регистр A содержит нуль, то первый раз команда выполняется в том виде, как она была закодирована);

Пункт г) — увеличение содержимого регистра A на единицу.

Пункт д) — проверка окончания групповой операции; в результате действие в) будет повторено $('k_2 + 1)$ раз.

Пункты е) и ж) — переход к выполнению команды, следующей за исполнительной.

IV. Команды обращения к внешним устройствам

1. $'k_0 = 25$ — подвод под считывающую головку зоны $'k_1$ магнитной ленты.

2. $'k_0 = 43$ — перенос кодов с магнитной ленты в ячейки памяти. $('k_2 + 1)$ код из зоны $'k_1$ магнитной ленты переносится в ячейки $'k_3$, $'k_3 + 1$, ..., $'k_3 + 'k_2$. Условно это запишем в виде

$$'[('k_1)\,МЛ] \Rightarrow \begin{pmatrix} 'k_3 \\ 'k_3 + 'k_2 \end{pmatrix}.$$

3. $'k_0 = 46$ — перенос кодов внутреннего $ЗУ$ на магнитную ленту:

$('k_2 + 1)$ код

$$'k_1,\ 'k_1 + 1,\ \ldots,\ 'k_1 + 'k_2$$

из $ЗУ$ переносятся в зону $'k_3$ магнитной ленты.

Условно обозначим через

$$' \begin{pmatrix} 'k_1 \\ 'k_1 + 'k_2 \end{pmatrix} \Rightarrow ('k_3)\ МЛ.$$

Здесь через $('k_3)МЛ$ обозначена зона магнитной ленты номер $'k_3$.

4. $'k_0 = 41$ — перенос кодов с перфокарт во внутреннюю память. $('k_2 + 1)$ код с перфокарт переносится в ячейки

$$'k_3,\ 'k_3 + 1,\ \ldots,\ 'k_3 + 'k_2;$$

$$'ПК \Rightarrow \begin{pmatrix} 'k_3 \\ 'k_3 + 'k_2 \end{pmatrix}.$$

5. $'k_0 = 44$ — перенос кодов из ячеек *ЗУ* на перфокарты. $('k_2 + 1)$ код из ячеек $'k_1$, $'k_1 + 1$, ..., $'k_1 + 'k_2$ переносится на перфокарты:

$$'\begin{pmatrix} 'k_1 \\ 'k_1 + 'k_2 \end{pmatrix} \Rightarrow ПК.$$

6. $'k_0 = 45$ — перенос кодов из одних ячеек *ЗУ* в другие. $('k_2 + 1)$ код из ячеек $'k_1$, $'k_1 + 1$, ..., $'k_1 + 'k_2$ переносится в ячейки $'k_3$, $'k_3 + 1$, ..., $'k_3 + 'k_2$:

$$'\begin{pmatrix} 'k_1 \\ 'k_1 + 'k_2 \end{pmatrix} \Rightarrow \begin{pmatrix} 'k_3 \\ 'k_3 + 'k_2 \end{pmatrix}.$$

7. $'k_0 = 60$ — групповая передача с контролем. Возможны различные модификации этой операции. Если

$$'k_1 = 0000,\ 0001 \leqslant 'k_3 \leqslant 3777,$$

то производится перенос с перфокарт в память. Если

$$0001 \leqslant 'k_1 \leqslant 3777,\ 'k_3 = 0000$$

— из памяти на перфокарты. Если

$$0001 \leqslant 'k_3 \leqslant 3777,$$
$$4001 \leqslant 'k_1 \leqslant 4777 \text{ или } 5001 \leqslant 'k_1 \leqslant 5777$$

— с магнитной ленты в память. Если

$$0001 \leqslant 'k_1 \leqslant 3777,$$
$$4001 \leqslant 'k_3 \leqslant 4777 \text{ или } 5001 \leqslant 'k_3 \leqslant 5777,$$

— из памяти на магнитную ленту. Если

$$0001 \leqslant 'k_1 \leqslant 3777,\ 0001 \leqslant 'k_3 \leqslant 3777$$

— из одних ячеек памяти в другие. Одновременно с выполнением переноса производится циклическое суммирование $('k_0 = 17)$ кодов до передачи и после нее. Получаемые суммы сравниваются, и в случае их несовпадения машина останавливается.

8. Операция останова. $'k_0 = 40$. Числа $^2k_1$ и $^2k_2$ выдаются на пульт.

V. Стандартные программы постоянной памяти

Описанные далее команды являются групповыми операциями обращения к стандартным подпрограммам, хранящимся в постоянной памяти. После выполнения этих подпрограмм управление передается очередной команде основной программы. По-

следняя не должна быть командой условного перехода (коды 20 и 27), так как по этим операциям не вырабатывается признак ω.

1. $'k_0 = 62$ — вычисление обратной величины:

$$\frac{1}{'('k_1 + i)} \Rightarrow 'k_3 + i, \quad \text{где} \quad i = 0, 1, \ldots, 'k_2.$$

2. $'k_0 = 63$ — вычисление квадратного корня:

$$\sqrt{'('k_1 + i)} \Rightarrow 'k_3 + i, \; i = 0, 1, \ldots, 'k_2.$$

Аналогично выполняются операции:

3. $'k_0 = 64$ — вычисление показательной функции.
4. $'k_0 = 66$ — вычисление логарифма.
5. $'k_0 = 67$ — вычисление синуса.
6. $'k_0 = 73$ — вычисление функции arctg.
7. $'k_0 = 74$ — вычисление функции arcsin.

Во всех этих операциях происходит останов в случае переполнения ячейки $'k_3 + i$ и в случае неприменимости функции к коду $'k_1 + i$ в поле вещественных чисел.

8. $'k_0 = 70$ — перевод кодов из двоичной системы счисления в двоично-десятичную:

$$'('k_1 + i)_{\text{дес.}} \Rightarrow 'k_3 + i, \quad \text{где} \quad i = 0, 1, \ldots, 'k_2.$$

9. $'k_0 = 72$ — обратная операция:

$$'('k_1 + i)_{\text{двоичн.}} \Rightarrow 'k_3 + i, \quad i = 0, 1, \ldots, 'k_2.$$

Производятся также действия:

$$'c + 1 \Rightarrow c; \quad ''c \Rightarrow K.$$

УРАЛ

Универсальная цифровая автоматическая машина *Урал* оперирует с 36-разрядными полными или 18-разрядными неполными кодами. Для команд используется одноадресная система,

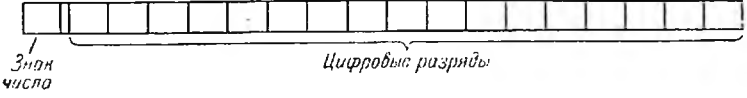


Рис. 25.

команды кодируются в неполных ячейках; пять разрядов отводятся для кода операции, 12 — для адреса, один разряд — признак изменяемости (модифицируемости) команды.

Числа представляются в системе с фиксированной запятой, первый разряд — знаковый.

При хранении чисел в коротких ячейках их разряды распределяются, как указано на рис. 25.

Распределение разрядов при хранении чисел в так называемых длинных (т. е. полных) ячейках показано на рис. 26 для двоичных кодов и на рис. 27 — для двоично-десятичных.

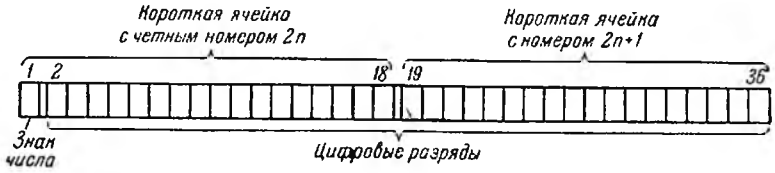


Рис. 26.

Быстродействие машины — 100 операций в секунду. Объем внутренней памяти (на магнитном барабане) 2048 коротких ячеек или 1024 полных. Полные ячейки имеют только четные

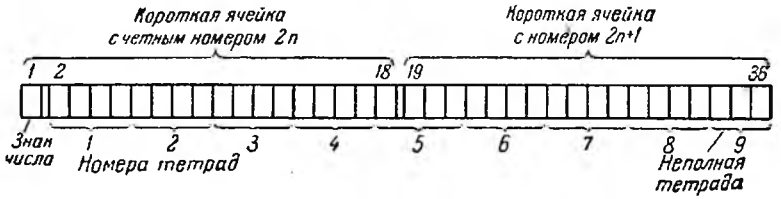


Рис. 27.

адреса. Внешняя память — на магнитных лентах. Ввод — с перфоленты, вывод — на перфоленту или бумажную ленту.

Специальные регистры и ключи машины *Урал*:

1) S — сумматор;
2) r — регистр арифметического устройства;
3) C — счетчик команд;
4) K — регистр команд;
5) ω — триггер признака для условной передачи управления;
6) $Ц$ — регистр циклов;
7) $K_i (i = 1, 2, \ldots, 7)$ — ключи;
8) T_1 — тумблер 1;
9) T_2 — тумблер 2.

Приняв указанные буквенные обозначения, перейдем к описанию элементарных операций машины *Урал*.

Отдельный цикл работы машины *Урал* состоит из:

1) выполнения команды K, т. е. команды, код которой хранится в данный момент в регистре команд;
2) засылки кода очередной команды в регистр команд.

Для описания набора элементарных операций *Урала* выделим в регистре команд группы разрядов (регистры) (см. таблицу 12).

Таблица 12

Разряды регистра команд *Урала* (нумерация слева направо)	Обозначение и название групп разрядов (регистров)
1	ε_0 — разряд признака изменяемости (модифицируемости) команды
2—6	k_0 — регистр кода операции
7	ε — разряд признака полной ячейки
8—18	k — регистр адреса команды

На рис. 28 показано разбиение регистра команд (и ячейки при хранении в ней команды) на группы разрядов.

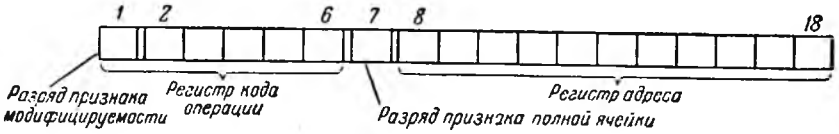


Рис. 28.

Множества допустимых для хранения на указанных регистрах кодов (в восьмеричной записи) следующие:

$$'\varepsilon_0,\ '\varepsilon = 0;\ 1;$$
$$'k_0 = 00,\ 01,\ \ldots,\ 37;$$
$$'k = 0000,\ 0001,\ \ldots,\ 3777.$$

I. Арифметические операции

1. $'k_0 = 01$ — сложение. По этой команде выполняются действия

а) $'S + '('k - '\varepsilon_0'Ц) \Rightarrow S;$
$\operatorname{sign} 'S \vee |'C'k - '\varepsilon_0'Ц)| \Rightarrow r;$

б) $P\{\operatorname{sign} 'S = 1\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C;$

г) $^2C \Rightarrow K.$

Если $'\varepsilon_0 = 0$, т. е. признак изменяемости команды отсутствует, выполняется операция

$$'S + {}^2k \Rightarrow S.$$

Если же $'\varepsilon_0 = 1$, адрес модифицируется, т. е. уменьшается на величину содержимого регистра циклов.

При этом, если $'\varepsilon = 0$, выбирается содержимое неполной ячейки с адресом $'k$; если $'\varepsilon = 1$ — полной. В последнем случае $'k$ должно быть четным числом.

б) означает засылку в триггер признака для условной передачи управления ω единицы или нуля в зависимости от знака числа в сумматоре.

в) и г) соответствуют подготовке к выполнению следующей по номеру команды.

Действия б), в) и г) производятся и при выполнении всех других команд данной группы. То же самое можно сказать и для разряда ε. В дальнейшем при описании операций мы не будем этого оговаривать.

2. $'k_0 = 02$ — засылка на сумматор:

а) $'('k - '\varepsilon_0'Ц) \Rightarrow S$; $'('k - '\varepsilon_0'Ц) \Rightarrow r$;

б), в) и г), как и при $'k_0 = 01$.

3. $'k_0 = 03$ — вычитание:

а) $'S - '('k - '\varepsilon_0'Ц) \Rightarrow S$;

$\operatorname{sign} 'S \vee |'('k - '\varepsilon_0'Ц)| \Rightarrow r$.

4. $'k_0 = 04$ (то же, что и при $'k_0 = 03$, только выполняемая операция является операцией вычитания модулей):

а) $|'S| - |'('k - '\varepsilon_0'Ц)| \Rightarrow S$;

$\operatorname{sign} 'S \vee |'('k - '\varepsilon_0'Ц)| \Rightarrow r$.

5. $'k_0 = 05$ — умножение с накоплением в сумматоре:

а) $'r \times '('k - '\varepsilon_0'Ц) + 'S \Rightarrow S$; $0 \Rightarrow r$;

содержимое ячейки умножается на содержимое регистра сумматора и результат прибавляется к содержимому сумматора.

6. $'k_0 = 06$ — умножение:

а) $'S \times '('k - '\varepsilon_0'Ц) \Rightarrow S$;

$0 \Rightarrow r$.

7. $'k_0 = 07$ — деление:

а) $'S : '('k - '\varepsilon_0'Ц) \Rightarrow S$, $0 \Rightarrow r$.

В машине, кроме указанных в списке регистров, имеются регистр переполнения разрядной сетки φ и связанные с ним тумблеры φ_1 и φ_2.

При выполнении арифметических операций 01, 03, 04, 05, 06, 07, если

$$|'S| < 1,$$

в триггер φ засылается 1:

$$1 \Rightarrow \varphi,$$

если $('S) < 1$, в φ засылается 0:

$$0 \Rightarrow \varphi.$$

При этом, если $'\varphi = 1$ и

а) $'\varphi_1 = 1$,

то выполняется действие

$$'C + 1 \Rightarrow C$$

(что означает переход к следующей команде программы); если

б) $'\varphi_1 = 0$, $'\varphi_2 = 1$,

то машина останавливается; если

в) $'\varphi_1 = 0$, $'\varphi_2 = 0$,

то выполняется действие $'C + 2 \Rightarrow C$, т. е. пропускается одна команда.

8. $'k_0 = 26$. Операция циклического сложения содержимого сумматора с содержимым ячейки $'k - '\varepsilon_0$ $'Ц$ (с переносом из старшего в младший разряд). Роль разрядов ε_0 и ε та же, что и при предыдущих операциях. Так же выполняются пп. б), в), г).

II. Логические операции

9. $'k_0 = 10$. Присвоение знака числа содержимому сумматора:

а) $|'S| \vee \operatorname{sign}'('k - '\varepsilon_0 'Ц) \Rightarrow S (\Rightarrow r)$;

б) $P\{\operatorname{sign}'S = 1\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

10. $'k_0 = 11$. Сдвиг содержимого регистра r сумматора (включая знаковый разряд) на число разрядов, соответствую-

щее модулю числа, расположенного в разрядах с 13-го по 18-й сумматора S:

а) влево, если sign $'S = 0$, и вправо, если sign $'S = 1$. Результат сдвига помещается в сумматор; $0 \Rightarrow r$;

б) $P\{'S = 0\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

Операция не зависит от $'\varepsilon_0$, $'\varepsilon_1$, $'k$.

11. $'k_0 = 12$. Операция поразрядного логического умножения:

а) $'S \wedge '('k - '\varepsilon_0'Ц) \Rightarrow S (\Rightarrow r)$,

при этом $'k$ — число четное;

б) $P\{'S = 0\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

12. $'k_0 = 13$. То же, что и при $'k_0 = 12$, только выполняемая операция является операцией поразрядного логического сложения.

13. $'k_0 = 14$ — поразрядная логическая операция «неравнозначно $\cong$»; разряды результата определяются таблицей 13.

Таблица 13

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

а) $['S \cong '('k - '\varepsilon_0'Ц)] \Rightarrow$
$\Rightarrow S (\Rightarrow r)$;

б) $P\{'S = 0\}$ $\begin{matrix} 0 \Rightarrow \omega; \\ 1 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

14. $'k_0 = 15$ — операция нормализации:

а) содержимое сумматора сдвигается влево на число разрядов, равное числу нулей между запятой и первой значащей цифрой, если $|'S| < \frac{1}{2}$, и вправо на один разряд, если $|'S| > 1$. После сдвига число записывается по адресу $'k - '\varepsilon_0'Ц$, а в сумматоре фиксируется порядок числа — число, соответствующее количеству сдвигов, в первом случае отрицательное, во втором — положительное. Знак порядка в сумматоре фиксируется в знаковом

разряде, младшим разрядом порядка является 19-й разряд сумматора.

б) $P\{'('k - '\varepsilon_0'Ц) = 0\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

15. $'k = 16$. Посылка из сумматора по адресу:

а) $'S \Rightarrow 'k - '\varepsilon_0'Ц$;

б) $P\{\operatorname{sign} 'S = 1\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

16. $'k_0 = 17$. Посылка на регистр сумматора:

а) $'('k - '\varepsilon_0'Ц) \Rightarrow r$;

б) $P\{'r = 0\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

17. $'k_0 = 20$. Посылка на сумматор:

а) $'k - '\varepsilon_0'Ц \Rightarrow S (\Rightarrow r)$.

В сумматор (с 7-го по 18-й разряды) заносится число $'k - '\varepsilon_0'Ц$, а не содержимое этого адреса, как в операции 02. Знаком числа $'k - '\varepsilon_0'Ц$ считается содержимое ε, и $'\varepsilon$ посылается в знаковый разряд сумматора;

б) $P\{\operatorname{sign} 'S = 1\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

III. Операции передачи управления

Операции передачи управления не меняют содержимого адресов машины и касаются лишь содержимого некоторых специальных регистров.

18. $'k_0 = 21$. Условная передача управления:

а) $P\{'\omega = 1\}$ $\begin{matrix} 'k - '\varepsilon_0'Ц \Rightarrow C; \\ 'C + 1 \Rightarrow C; \end{matrix}$

б) $^2C \Rightarrow K$.

Если в триггере ω находится 1, совершается переход к выполнению команды, хранящейся по адресу $'k - '\varepsilon_0'Ц$, в противном случае — к следующей по номеру команде.

19. $'k_0 = 22$ — безусловная передача управления:

а) $'k - '\varepsilon_0'Ц \Rightarrow C$;

б) $^2C \Rightarrow K$.

20. $'k_0 = 23$ — операция передачи управления по ключу. Если ключ $'k$ включен, пропускается одна команда, если он выключен, совершается переход к следующей команде, т. е.

а) $P\{'K_{'k} = 1\}$ $\begin{matrix} 'C + 2 \Rightarrow C; \\ 'C + 1 \Rightarrow C; \end{matrix}$

б) $^2C \Rightarrow K$.

21. $'k_0 = 25$ — начало групповой операции. По этой операции подготавливается содержимое регистра циклов $Ц$:

а) $'k \Rightarrow Ц$;

б) $'C + 1 \Rightarrow C$;

в) $^2C \Rightarrow K$.

22. $'k_0 = 24$ — конец групповой операции:

а) $P\{'Ц \neq 0\}$ $\begin{matrix} 'k - '\varepsilon_0'Ц \Rightarrow C; \quad 'Ц - '\varepsilon - 1 \Rightarrow Ц; \\ 'C + 1 \Rightarrow C; \end{matrix}$

б) $^2C \Rightarrow K$.

а) означает передачу управления циклу (команде, хранящейся по адресу $'k - '\varepsilon_0'Ц$), если $'Ц \neq 0$, и уменьшение содержимого регистра циклов $Ц$ на единицу, если $'\varepsilon = 0$ (для групповой операции по коротким ячейкам), и на два, если $'\varepsilon = 1$ (для групповой операции по полным ячейкам); в противном случае — выход из цикла.

23. $'k_0 = 30$. Операция изменения команд. По этой команде:

а) $'k - '\varepsilon_0'Ц \Rightarrow K$;

б) $'C + 1 \Rightarrow C$;

в) $'K + ^2C \Rightarrow K$.

Таким образом, на регистр команд поступает «следующая» команда, измененная на величину содержимого адреса $'k - '\varepsilon_0'Ц$.

24. $'k_0 = 37$. По этой команде машина останавливается и, кроме того, выполняется засылка на сумматор содержимого ячейки $'k - '\varepsilon_0'Ц$:

$$'('k - '\varepsilon_0'Ц) \Rightarrow S.$$

IV. Операции обращения к внешним устройствам

25. $'k_0 = 31$ — подготовительная команда для операции передачи кодов; $'k$ — адрес первой ячейки внутреннего *ЗУ* (магнитного барабана), с которой начинается операция. Операции обращения к внешним устройствам кодируются последовательностью из трех команд:

$$31\alpha;$$

$$\beta_1\beta;$$

$$00\gamma.$$

Здесь 31, β_1, 00 — коды, соответствующие регистру k_0; α, β, γ — коды, соответствующие регистру k. При этом для β_1 допустимы следующие значения:

$$\beta_1 = 01;\ 02;\ 03,$$

которые и определяют вид операции; β — номер зоны магнитной и перфорированной лент.

1) $\beta_1 = 01$, выполняется операция передачи кодов из вводного устройства (перфорированная лента) во внутреннее *ЗУ* (магнитный барабан).

2) $\beta_1 = 02$, выполняется операция передачи кодов из внешнего (магнитная лента) во внутреннее *ЗУ*;

3) $\beta_1 = 03$, выполняется операция передачи кодов из внутреннего во внешнее *ЗУ*.

В каждом из этих случаев код γ определяет адрес последней ячейки внутреннего *ЗУ*, до которой распространяется операция.

После операции передачи кодов выполняется следующая по номеру команда.

26. $'k_0 = 32$. Печать содержимого сумматора. При этом, если $'T_1 = 1$, печать происходит на бумажную ленту, при $'T_1 = 0$ — на перфоленту; при $'T_2 = 1$ выполняется печать восьмеричных кодов (вывод программы), при $'T_2 = 0$ — десятеричных (вывод числового материала).

27. $'k_0 = 34$ — пропуск одной строки на бумажной ленте.

М-3

Цифровая автоматическая машина *М-3* оперирует с 31-разрядными кодами. Числа представляются в системе с фиксированной запятой. Один разряд — знаковый, 30 разрядов — цифровые (рис. 29).

При записи чисел в двоично-десятичной системе два последних разряда не используются, а в остальных 28 помещаются семь десятичных цифр.

Система команд *М-3* — двухадресная с последовательным выполнением. Код операции занимает шесть разрядов, адреса —

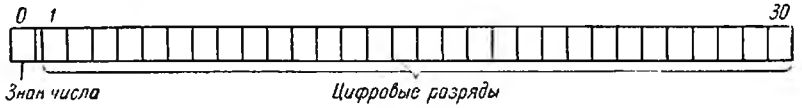


Рис. 29.

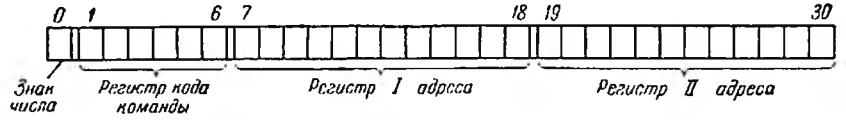


Рис. 30.

по 12 разрядов. Схема распределения разрядов ячейки при хранении в ней команды приведена на рис. 30.

Таблица 14

Группы разрядов регистра команд	Обозначение и название групп разрядов (регистров)
1—3	k_{01} — первый регистр кода команды } k_0 — регистр кода
4—6	k_{02} — второй регистр кода команды }
7—18	k_1 — регистр I адреса
19—30	k_2 — регистр II адреса

Оперативная память машины состоит из 2048 ячеек (с номерами от 0000 до 3777 в восьмеричной системе) и размещена на магнитном барабане. Быстродействие машины — 30 операций в секунду. Арифметические устройства и устройство управления являются быстродействующими. Поэтому замена магнитного барабана запоминающим устройством из ферритовых сердечников увеличивает быстродействие машины до 1500 операций в секунду. Ввод — с перфорированной бумажной ленты, вывод — на печать. Ячейка 0000 в отличие от машин *Стрела, БЭСМ, Киев* является обычной ячейкой (не содержащей код $+0$).

Таблица 15

Вид операции	$/k_{01}$
Сложение	0
Вычитание	1
Умножение	3
Деление	2
Поразрядное логическое умножение	6

Для описания операций введем обозначения:

1) C — счетчик команд,
2) K — регистр команд,
3) r — регистр арифметического устройства,
4) ω — триггер признака условного перехода.

Регистр команд K разделим на группы разрядов (регистры) (см. таблицу 14). Значение $'k_{02}$ определяет вид операции (см. таблицу 15).

Значение $'k_{01}$ определяет особенности выполнения операций. Пусть 0 обозначает любую из операций, приведенных в таблице. В зависимости от $'k_{01}$ машина выполняет следующие операции.

I. Арифметические операции

1. $'k_{01} = 0$.

а) $^2k_1 0 ^2k_2 \Rightarrow r$; $'r \Rightarrow 'k_2$;

б) $P\{'r \leqslant -0\}$ $\begin{matrix} 1 \Rightarrow \omega; \\ 0 \Rightarrow \omega; \end{matrix}$

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

Другими словами, в этом случае адреса аргументов операции кодируются в адресах команды; результат операции содержится по второму адресу $'k_2$ и в r — регистре $АУ$. Пункт б) — выработка признака ω, используемого в командах условной передачи управления.

Пункты б), в) и г) — переход к выполнению следующей команды — одинаковы для всех команд этой группы.

2. $'k_{01} = 1$.

а) $^2k_1 0 ^2k_2 \Rightarrow r$.

(Содержимое адреса $'k_2$, в отличие от случая $'k_{01} = 0$, операция не меняет.)

3. $'k_{01} = 2$.

а) $'r 0 ^2k_1 \Rightarrow r$;

$'r \Rightarrow 'k_2$.

(Операция 0 выполняется над содержимым r — регистра $АУ$ и ячейки $'k_1$; результат помещается в регистр r и по адресу $'k_2$.)

4. $'k_{01} = 3$.

а) $'r 0 ^2k_1 \Rightarrow r$.

5. $'k_{01} = 4$.

В этом случае выполняются следующие операции:

а) $^2k_1 0^2k_2 \Rightarrow r$;

$'r \Rightarrow 'k_2$;

печать $'r$.

($'r$ — печатается на бумажную ленту).

6. $'k_{01} = 5$.

а) $|^2k_1|0|^2k_2| \Rightarrow r$.

7. $'k_{01} = 6$.

$'r0^2k_1 \Rightarrow r$;

$'r \Rightarrow 'k_2$;

печать $'r$.

8. $'k_{01} = 7$.

$|'r|0|^2k_1| \Rightarrow r$.

II. Команды передачи управления

1. $'k_0 = 24$. Безусловный переход по первому адресу:

а) $'r \Rightarrow 'k_2$;

б) $'k_1 \Rightarrow C$;

в) $^2C \Rightarrow K$.

(Одновременно содержимое r — регистра $АУ$ пересылается по адресу $'k_2$.)

2. $'k_0 = 64$. Безусловный переход по первому адресу с печатью содержимого регистра r:

а) $'r \Rightarrow 'k_2$;

б) *печать* $'r$;

в) $'k_1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

3. $'k_0 = 74$. Безусловный переход по второму адресу ($'k_1 = 0$):

а) $|'r| \Rightarrow r$;

б) $'k_2 \Rightarrow C$;

в) $^2C \Rightarrow K$.

4. $'k_0 = 34$. Условный переход:

а) $P\{'\omega = 1\}$ $'k_1 \Rightarrow C$; $'k_2 \Rightarrow C$;

б) $^2C \Rightarrow K$.

5. $'k_0 = 37$. *Останов:*

а) $^2k_1$ — *выдается на пульт;*

б) *ост.*

III. Операции обращения к вводу (выводу) и переноса

1. $'k_0 = 07, 27$. *Ввод с перфоленты* ($'k_1 = 0$):

а) $'ПЛ \Rightarrow 'k_2$;

б) $'C + 1 \Rightarrow C$;

в) $^2C \Rightarrow K$.

Один код с перфоленты переносится в ячейку памяти $'k_2$.

2. $'k_0 = 05, 15$. *Перенос:*

а) $^2k_1 \Rightarrow 'k_2$;

б) $'C + 1 \Rightarrow C$;

в) $^2C \Rightarrow K$.

3. $'k_0 = 45, 55$. *Перенос с печатью:*

а) $^2k_1 \Rightarrow 'k_2$;

б) *печать* $^2k_1$;

в) $'C + 1 \Rightarrow C$;

г) $^2C \Rightarrow K$.

КИЕВ

Универсальная цифровая автоматическая машина *Киев**) оперирует с 41-разрядными кодами. Для команд в принятой трехадресной системе это дает пять разрядов для кодирования

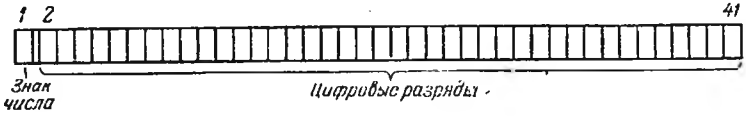


Рис. 31.

операций и по 12 разрядов для каждого из адресов. Первый разряд каждого адреса отведен для признака модифицируемости (изменяемости) адреса. Числа представляются в системе с фиксированной запятой, первый разряд — знаковый.

Распределение разрядов ячейки при записи двоичного числа показано на рис. 31.

---

*) См. Гнеденко Б. В., Глушков В. М., Ющенко Е. Л. [1] и Дашевский Л. Н., Погребинский С. В., Шкабара Е. А. [1].

Быстродействие машины составляет 15—20 тысяч операций в секунду типа сложения и 3—4 тысячи операций в секунду типа умножения и деления, в среднем около 9 тысяч операций в секунду. Объем внутренней памяти (*ВЗУ*) — 2048 ячеек, внешняя память — на магнитных барабанах и лентах. Внутренняя память содержит ячейки оперативные и пассивные. Пассивные ячейки имеются трех видов:

а) постоянно-спаянные,
б) сменно-спаянные,
в) сменно-наборные.

Постоянно-спаянная память предназначена для хранения наиболее часто встречающихся констант и подпрограмм. Сменно-спаянная память выполнена в виде спаянных блоков, предназначенных для хранения различных стандартных подпрограмм, которые по мере надобности могут включаться для работы. Восемь ячеек сменно-наборной памяти представляют собой наборы тумблеров, с помощью которых вручную могут вводиться те или иные коды. Перевод кодов чисел из десятичной системы в двоичную совмещается с вводом и не занимает дополнительного времени. Ввод осуществляется на перфоленте, вывод — на перфоленту и печать; предусмотрена возможность перехода на перфокарты.

Помимо ячеек *ВЗУ* с адресами 0000 + 3777 в восьмеричной системе, в машине *Киев* существуют следующие специальные регистры:

1) *С* — счетчик команд (11-разрядный),
2) *К* — регистр команд (41-разрядный),
3) *Р* — регистр возврата (11-разрядный),
4) *Ц* — регистр циклов (10-разрядный),
5) *А* — регистр модификации адресов (10-разрядный),
6) *Т* — регистр-триггер аварийного останова (1-разрядный).

При описании операций, включающих работу этих регистров, мы ограничимся их буквенными обозначениями — адресами, которые будем предполагать отличными от кодов адресов *ВЗУ*.

Согласно принципу программного управления (см. главу II), отдельный цикл работы машины состоит в выполнении команды *К*, т. е. команды, код которой хранится в данный момент на регистре команд, и в засылке на регистр *К* кода следующей команды.

Выполнение команды *К* зависит в свою очередь от ее содержания (кода).

Для хранения номера (адреса) «очередной» команды в машине *Киев*, как и во многих других машинах, используется счетчик команд *С*.

Для удобства описания набора элементарных операций, выполняемых отдельными командами машины *Киев*, в регистре

команд K выделим отдельные группы разрядов (регистры), как указано в таблице 16. Такое же деление производится при необходимости и в ячейках *ВЗУ*.

Таблица 16

Разряды регистра команд *Киева* (нумерация слева направо)	Обозначение и название групп разрядов (регистров)
1—5	k_0 — регистр кода операции
6	ε_1 — разряд признака модифицируемости I адреса
7—17	k_1 — регистр I адреса
18	ε_2 — разряд признака модифицируемости II адреса
19—29	k_2 — регистр II адреса
30	ε_3 — разряд признака модифицируемости III адреса
31—41	k_3 — регистр III адреса

Регистр команд (и ячейка памяти при хранении в ней команды) разбит, как это показано на рис. 32.

Рис. 32.

На указанных регистрах могут храниться соответственно восьмеричные коды:

$$'k_0 = 00,\ 01,\ 02,\ \ldots,\ 37;$$

$$'\varepsilon_1,\ '\varepsilon_2,\ '\varepsilon_3 = 0,1;$$

$$'k_1,\ 'k_2,\ 'k_3 = 0000,\ 0001,\ \ldots,\ 3777.$$

В зависимости от $'k_0$ — кода операции, машина выполняет следующие операции.

I. Арифметические операции

1. $'k_0 = 01$. Операция сложения «+». При этой операции машина выполняет следующие действия:

а) сложение чисел, являющихся содержимым адресов $'k_1 + '\varepsilon_1 'A$ и $'k_2 + '\varepsilon_2 'A$, и засылку результата сложения по адресу $'k_3 + '\varepsilon_3 'A$:

$$'('k_1 + '\varepsilon_1 'A) + '('k_2 + '\varepsilon_2 'A) \Rightarrow 'k_3 + '\varepsilon_3 'A;$$

б) увеличение содержимого счетчика команд C на единицу:

$$'C + 1 \Rightarrow C;$$

в) засылку следующей команды на регистр команд K:

$$^2C \Rightarrow K.$$

Пункт а) означает, что при $'\varepsilon_i = 1$ ($i = 1, 2, 3$) соответствующий адрес модифицируется, а именно, увеличивается на величину содержимого регистра модификации адресов A, и при $'\varepsilon_i = 0$ операция выполняется непосредственно по адресу. Пункты б) и в) означают подготовку к выполнению следующей по номеру команды.

Эти особенности будут относиться ко всем операциям данной группы.

2. $'k_0 = 02$. Операция вычитания «—». То же, что и «+», только выполняемая операция является вычитанием, т. е.

а) $'('k_1 + '\varepsilon_1'A) - '('k_2 + '\varepsilon_2'A) \Rightarrow 'k_3 + '\varepsilon_3'A;$

б) $'C + 1 \Rightarrow C;$

в) $^2C \Rightarrow K.$

3. $'k_0 = 03$. Сложение команд «$+_1$»:

а) $|'('k_1 + '\varepsilon_1'A) + |'('k_2 + '\varepsilon_2'A)|| \vee \operatorname{sign} '('k_2 + '\varepsilon_2'A) \Rightarrow$
$\Rightarrow 'k_3 + '\varepsilon_3'A;$

б) $'C + 1 \Rightarrow C,$

в) $^2C \Rightarrow K.$

Операция «$+_1$» отличается от операции сложения «+» тем, что здесь к содержимому ячейки $'('k_2 + '\varepsilon_2'A)$, рассматриваемому как код команды, т. е. к его модулю, прибавляется содержимое ячейки $'('k_1 + '\varepsilon_1'A)$ — константа переадресации, и результату присваивается знак команды $'('k_2 + '\varepsilon_2A)$.

4. $'k_0 = 06$. Операция вычитания модулей «|—|». То же, что и «—», но уменьшаемое и вычитаемое являются модулями соответствующих кодов.

5. $'k_0 = 07$. Циклическое сложение «$Ц+$» — сложение кодов с переносом из старшего разряда в младший разряд. Как и в предыдущих операциях, допускается возможность модификации адресов. Аналогично выполняются:

6. $'k_0 = 10$. Умножение без округления «$\times$».

7. $'k_0 = 11$. Умножение с округлением «$\otimes$».

8. $'k_0 = 12$. Деление «:».

Помимо изложенного, при выполнении операций «+», «—», «:» в регистр T засылается 1, если результат операции по модулю оказывается большим единицы, и 0 в противном случае.

При соответствующей установке тумблера аварийного останова засылка

$$1 \Rightarrow T$$

означает останов.

II. Логические операции

9. $'k_0 = 35$. Нормализация.

а) Число $'('k_1 + '\varepsilon_1 'A)$ нормализуется, порядок числа помещается в младшие шесть разрядов ячейки $('k_2 + '\varepsilon_2 'A)$, мантисса — по адресу $k_3 + '\varepsilon_3 'A$;

б) $'C + 1 \Rightarrow C$;

в) $^2C \Rightarrow K$.

10. $'k_0 = 13$. Сдвиг логический.

а) Код $'('k_1 + '\varepsilon_1 'A)$ (включая знаковый разряд) сдвигается на число разрядов, указанное в III адресе ячейки $'k_2 + '\varepsilon_2 'A$, вправо или влево в зависимости от знака этого числа; результат помещается по адресу $'k_3 + '\varepsilon_3 'A$;

б) $'C + 1 \Rightarrow C$;

в) $^2C \Rightarrow K$.

11. $'k_0 = 14$. Операция поразрядного логического сложения «$\vee$» (см. главу II):

а) $'('k_1 + '\varepsilon_1 'A) \vee '('k_2 + '\varepsilon_2 'A) \Rightarrow ('k_3 + '\varepsilon_3 'A)$;

б) $'C + 1 \Rightarrow C$;

в) $^2C \Rightarrow K$.

Аналогично выполняются следующие две операции.

12. $'k_0 = 15$. Операция поразрядного логического «умножения» «$\wedge$».

13. $'k_0 = 17$. Поразрядная логическая операция «неравнозначно $\simeq$».

III. Операции передачи управления

Все операции передачи управления не изменяют содержание ячеек *ВЗУ*; результат их действия касается лишь некоторых из специальных регистров.

14. $'k_0 = 16$. Операции условной передачи управления *по равенству модулей* «=».

Операция «=» реализует адресную программу:

а) $P\{'('k_1+'\varepsilon_1'A)='('k_2+'\varepsilon_2'A)\}$ $\begin{matrix}'k_3+'\varepsilon_3'A\Rightarrow C;\\ 'C+1\Rightarrow C\end{matrix}$

и

б) $^2C\Rightarrow K$.

Следующие три операции выполняются аналогично.

15. $'k_0=04$. Условная передача управления по соотношению «меньше или равно $\leqslant$»:

а) $P\{'('k_1+'\varepsilon_1'A)\leqslant'('k_2+'\varepsilon_2'A)\}$ $\begin{matrix}'k_3+'\varepsilon_3'A\Rightarrow C;\\ 'C+1\Rightarrow C;\end{matrix}$

б) $^2C\Rightarrow K$.

16. $'k_0=05$. Условная передача управления по соотношению «меньше или равно» без учета знаков «$|\leqslant|$»:

а) $P\{|'('k_1+'\varepsilon_1'A)|\leqslant|'('k_2+'\varepsilon_2'A)|\}$ $\begin{matrix}'k_3+'\varepsilon_3'A\Rightarrow C;\\ 'C+1\Rightarrow C;\end{matrix}$

б) $^2C\Rightarrow K$.

17. $'k_0=31$. Условная передача управления по знаку числа «*УПЧ*»:

а) $P\{'('k_1+'\varepsilon_1'A)\leqslant-0\}$ $\begin{matrix}'k_3+'\varepsilon_3'A\Rightarrow C;\\ 'k_2+'\varepsilon_2'A\Rightarrow C;\end{matrix}$

б) $^2C\Rightarrow K$.

18. $'k_0=32$. Условный переход на подпрограмму «*УПП*»:

а) $P\{'('k_1+'\varepsilon_1'A)\leqslant-0\}$ $\begin{matrix}'k_2+'\varepsilon_2'A\Rightarrow P;\quad 'k_3+'\varepsilon_3'A\Rightarrow C;\\ 'C+1\Rightarrow C;\end{matrix}$

б) $^2C\Rightarrow K$.

При этом $'k_3+'\varepsilon_3'A$ является адресом начальной команды подпрограммы, $'k_2+'\varepsilon_2'A$ — номер команды, которой должно быть передано управление после выполнения подпрограммы. При невыполнении этого условия производится переход к следующей по номеру команде.

Соответствующая подпрограмма заканчивается специальной командой.

19. $'k_0=30$. Переход по регистру возврата. По этой команде выполняются действия:

а) $'P\Rightarrow C$;

б) $^2C\Rightarrow K$.

При этом содержимое регистров k_1, k_2, k_3 на результат операции не влияет.

IV. Операции обращения к внешним устройствам

Все операции этой группы являются групповыми, т. е. относятся к последовательностям кодов.

20. $'k_0 = 20$. Ввод чисел *«ВЧ»* с перфоленты (*ПЛ*). По этой команде в ячейки оперативного запоминающего устройства (*ОЗУ*) с адресами

$$'k_1,\ 'k_1+1,\ \ldots,\ 'k_2$$

вводятся коды с перфоленты, предварительно переведенные из десятично-двоичной системы в двоичную. Условно запишем эту групповую операцию в виде

а) $'(ПЛ)_{\text{перев. в двоичн. код}} \Rightarrow \begin{pmatrix} 'k_1 \\ 'k_2 \end{pmatrix}$

Кроме того, по этой команде выполняются засылки:

б) $'C+1 \Rightarrow C$;

в) $^2C \Rightarrow K$.

21. $'k_0 = 21$. Ввод команд с перфоленты *«ВК»*. То же, что и *«ВЧ»*, только передача кодов производится без преобразования:

а) $'(ПЛ) \Rightarrow \begin{pmatrix} 'k_1 \\ 'k_2 \end{pmatrix}$;

б) $'C+1 \Rightarrow C$;

в) $^2C \Rightarrow K$.

22. $'k_0 = 22$. Вывод кодов на *ПЛ «ВПЛ»*.

Обратная операция: содержимое ячеек

$$'k_1,\ 'k_1+1,\ \ldots,\ 'k_2$$

выводится на *ПЛ*

а) $'\begin{pmatrix} k_1 \\ 'k_2 \end{pmatrix} \Rightarrow ПЛ$;

б) $'k_3 \Rightarrow C$;

в) $^2C \Rightarrow K$.

23. $'k_0 = 23$. Обмен кодами *ОЗУ* и внешним *ЗУ* (*МБ*) в режиме «запись» — *«МБЗ»*. Коды, содержащиеся в последовательности ячеек *ОЗУ*

$$'k_1,\ 'k_1+1,\ \ldots,\ 'k_2,$$

переносятся на *МБ*:

а) $'\binom{'k_1}{'k_2} \Rightarrow МБ$;

кроме того,

б) $'k_3 \Rightarrow C$;

в) $^2C \Rightarrow K$.

24. $'k_0 = 24$. Обмен кодами *ОЗУ* и *МБ* в режиме «считка» — «*МБЧ*». Коды с *МБ* передаются в последовательность ячеек *ОЗУ* $'k_1$, $'k_1 + 1$, ..., $'k_2$, т. е.

а) $'(МБ) \Rightarrow \binom{'k_1}{'k_2}$;

кроме того,

б) $'k_3 \Rightarrow C$;

в) $^2C \Rightarrow K$.

25. $'k_0 = 25$. Подготовительная операция для операций «*МБЗ*» и «*МБЧ*», обеспечивающая надлежащую подводку магнитного барабана. Здесь

$$'k_1 = \begin{cases} 0 \text{ для операции } 23, \\ 1 \quad » \qquad » \quad 24, \end{cases}$$

$'k_2$ — номер дорожки *МБ*, с которой производится обмен кодами;
$'k_3$ — номер числа на дорожке *МБ*, с которого необходимо начать соответствующую операцию.

В этом случае также выполняются действия

$$'C + 1 \Rightarrow C; \quad ^2C \Rightarrow K.$$

26. $'k_0 = 33$. Останов.

V. Операции модификации адресов

Операции модификации адресов являются групповыми в том смысле, что формируемое ими содержимое регистра модификации может быть использовано группой команд.

27. $'k_0 = 26$. «Начало групповой операции» *НГО*. По операции *НГО*:

а) на регистр циклов *Ц* засылается число, характеризующее число циклов в циклическом процессе,

$$'k_1 \Rightarrow Ц;$$

б) на регистр модификации адресов A засылается константа переадресации

$$'k_2 \Rightarrow A;$$

в) реализуется предикатная формула

$$P\{'Ц = 'A\} \begin{matrix} 'k_3 \Rightarrow C; \\ 'C + 1 \Rightarrow C; \end{matrix}$$

г) $^2C \Rightarrow K$.

Таким образом, команда *НГО* подготавливает содержимое регистра переадресации и тем самым обеспечивает надлежащую модификацию переменных адресов.

Если число циклов N заранее известно и p — шаг переадресации, то полагаем:

$$'k_1 = 'k_2 + Np.$$

При каждом повторении цикла в этом случае содержимое регистра A увеличивается на величину p. Пока $'Ц \neq 'A$, продолжаются вычисления по циклу; при $'Ц = 'A$ совершается выход с цикла — на команду с номером $'k_3$.

Увеличение содержимого регистра A на шаг переадресации p может быть получено за счет переадресации соответствующей команды *НГО* (увеличение ее второго адреса на p) и повторного выполнения этой команды или с помощью следующей команды *КГО*.

28. $'k_0 = 27$. Конец групповой операции «*КГО*». По команде *КГО*:

а) содержимое регистра A увеличивается на шаг переадресации $'k_1 + 'A \Rightarrow A$ ($'k_1 = p$ — шаг переадресации);

б) реализуется предикатная формула

$$P\{'Ц = 'A\} \begin{matrix} 'k_3 \Rightarrow C; \\ 'k_2 \Rightarrow C; \end{matrix}$$

в) $^2C \Rightarrow K$.

Здесь $'k_3$ — номер команды, которой передается управление по окончанию цикла; $'k_2$ — номер команды, которой передается управление при продолжении вычислений по циклу. При этом заметим, что $'k_2$ не является номером команды *НГО*, так как последняя теперь не повторяется при переходе от цикла к циклу, поскольку ее повторение привело бы к восстановлению начального заполнения регистра A.

29. $'k_0 = 34$. Заполнение регистра A по фиксатору (вызов по фиксатору) «*Ф*».

Помимо операции «*НГО*», обеспечивающей заполнение регистра модификации адресов, в машине реализована

операция «Φ», которая также выполняет эту функцию. Однако, в то время как в команде *НГО* величина, засылаемая на регистр A, задавалась в явном виде (как $'k_2$), в команде Φ эта величина задается лишь своим адресом. По команде Φ:

а) $'('k_1 + 'ε_1'A)_2 \Rightarrow A$;
б) $'('('k_1 + 'ε_1'A)_2) \Rightarrow 'k_3$;
в) $'C + 1 = C$;
г) $^2C \Rightarrow K$,

$'k_2$ — при выполнении команды не используется.

Обозначим через $'α_2$ содержимое II адреса ячейки $α$. Тогда, если

$$'k_1 = α; \quad 'α_2 = β,$$

по операции Φ имеем:

а) $β \Rightarrow A$;
б) $'β \Rightarrow 'k_3$;
в) $'C + 1 \Rightarrow C$;
г) $^2C \Rightarrow K$.

Удобства использования операции Φ можно уяснить из главы V. Можно показать, что наличие операции Φ позволяет для произвольных циклических параметрических процессов составлять программы, не изменяющиеся в процессе своей работы (без команд переадресации).

ЦИТИРОВАННАЯ ЛИТЕРАТУРА

[1] Сессия Академии наук СССР по научным проблемам автоматизации производства 15—20 октября 1956 г., тт. 1—7, Изд-во АН СССР, 1957, особенно т. 1.

[1] Бут Э. и Бут К., Автоматические цифровые машины, перев. с англ., под ред. В. М. Курочкина, Физматгиз, 1959.

[1] Быстродействующая вычислительная машина М-2, под ред. И. С. Брука, Гостехиздат, 1957.

[1] Быстродействующие вычислительные машины, перев. с англ., под ред. Д. Ю. Панова, ИЛ, 1952.

[1] Гильберт Д. И., Аккерман В., Основы теоретической логики, ИЛ, 1947.

[1] Гнеденко Б. В., Глушков В. М., Ющенко Е. Л., Математичні параметри універсальної цифрової автоматичної машини «*Київ*», Сб. ВЦ АН УССР, т. II, 1960.

[1] Дашевский Л. Н., Погребинский С. В., Шкабара Е. Л., Технічні параметри універсальної цифрової автоматичної машини «*Київ*», Сб. ВЦ АН УССР, т. II, 1960.

[1] Ершов А. П., Программирующая программа для быстродействующей электронной счетной машины, Изд-во АН СССР, 1958.

[2] Ершов А. П., Об операторных алгорифмах, Докл. АН СССР 122, № 6 (1958), 967—970.

[1] Каган Б. М. и Тер-Микаэлян Г. М., Решение инженерных задач на автоматических цифровых вычислительных машинах, Госэнергоиздат, 1958.

[1] Калужнин Л. А., Об алгоритмизации математических задач, Проблемы кибернетики, вып. 2, Физматгиз, 1959, 51—67.

[1] Китов А. П. и Криницкий Н. А., Электронные цифровые машины и программирование, Физматгиз, 1959.

[1] Королюк В. С., Про один спосіб програмування, Доп. АН УРСР, № 12 (1958), 1292—1295.

[1] Королюк В. С. и Ющенко Е. Л., Вопросы теории и практики программирования, Сб. ВЦ АН УССР, т. I, 1960.

[1] Ляпунов А. А., О логических схемах алгоритмов, Проблемы кибернетики, вып. I, Физматгиз, 1958, 46—74.

[1] Ляпунов А. А. и Шестопал Г. А., Начальные сведения о решении задач на электронных вычислительных машинах, Сб. Математическое просвещение, вып. 1, Гостехиздат, 1957, 57—74.

[2] Ляпунов А. А. и Шестопал Г. А., Об алгоритмическом описании процессов управления, Сб. Математическое просвещение, вып. 2, 1957, 81—95.

[1] Лебедев С. А. и Мельников В. А., Общее описание *БЭСМ* и методика выполнения операций, Физматгиз, 1959.

[1] Марков А. А., Теория алгорифмов, Тр. Матем. ин-та АН СССР, вып. 42, 1954.

[1] Новиков П. С., Элементы математической логики, Физматгиз, 1959.
[1] Ричардс Р. К., Арифметические операции на цифровых вычислительных машинах, ИЛ, 1957.
[1] Система стандартных подпрограмм, под ред. М. Р. Шура-Бура, Физматгиз, 1958.
[1] Трахтенброт Б. А., Алгорифмы и машинное решение задач, Гостехиздат, 1957.
[1] Уилкс М., Уилер Д., Гилл С., Составление программ для электронных счетных машин, пер. с англ., под ред. Д. Ю. Панова, ИЛ, 1953.
[1] Фаддеев Д. К. и Фаддеева В. Н., Вычислительные методы линейной алгебры, Физматгиз, 1960.
[1] Фридман В. М., Новые методы решения линейного операторного уравнения, Докл. АН СССР 128, № 3 (1959), 482—484.
[1] Ющенко Е. Л., Адресні алгоритмы та цифрові автоматичні машини, Сб. ВЦ АН УССР, т. II, 1960.
[1] Ющенко Е. Л., Быстрова Л. П., Програмуюча програма, інформацією для якої служить адресний алгоритм, Сб. ВЦ АН УССР, т. III, 1960.
[1] Янов Ю. И., О логических схемах алгоритмов, Проблемы кибернетики, вып. 1, Физматгиз, 1958, 75—127.